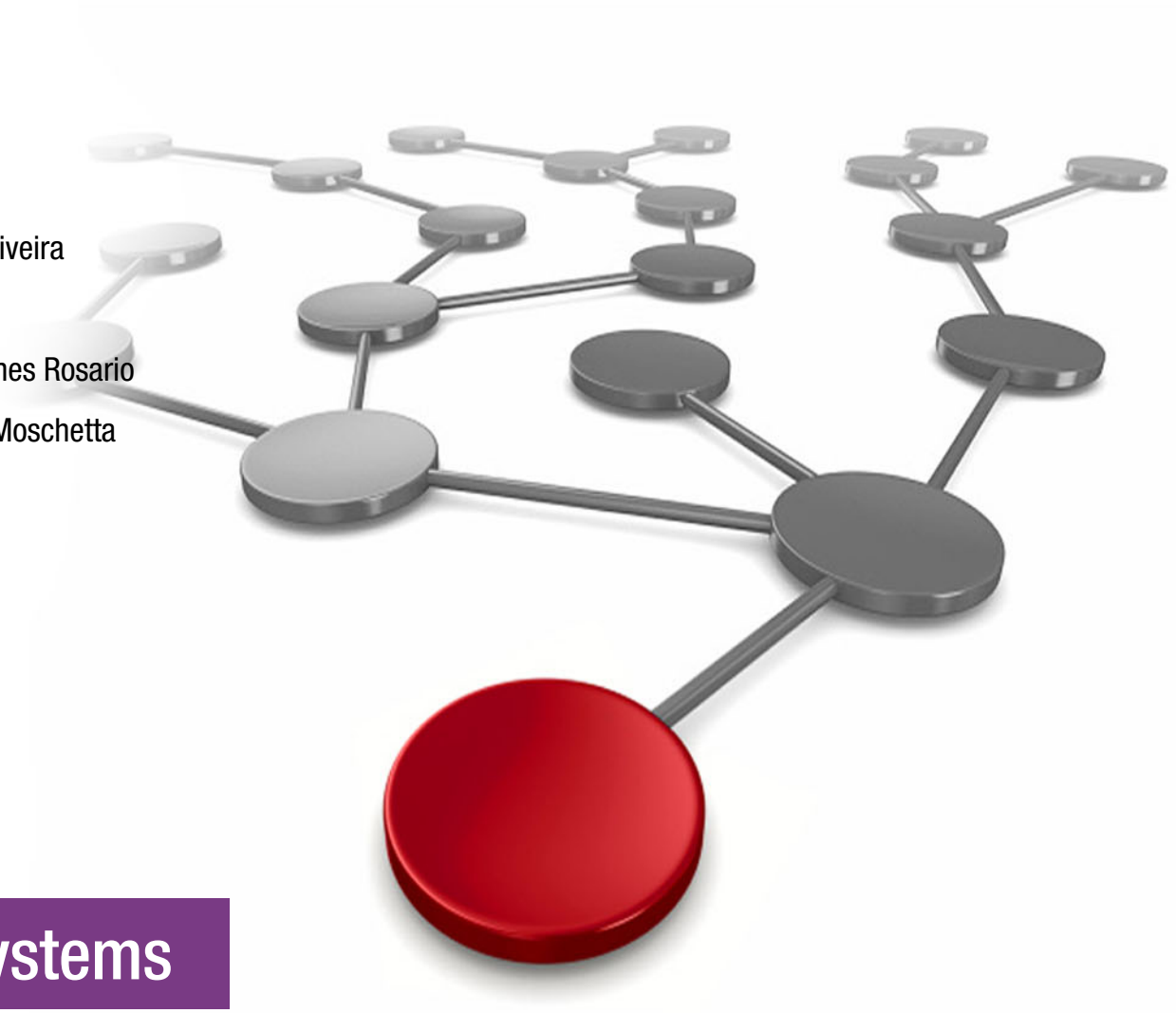
Redbooks
ibm.com/redbooks

IBM POWER8 High-Performance Computing Guide

IBM Power System S822LC (8335-GTB) Edition

Dino Quintero
Joseph Apuzzo
John Dunham
Mauricio Faria de Oliveira
Markus Hilger
Desnes Augusto Nunes Rosario
Wainer dos Santos Moschetta
Alexander Pozdneev



Power Systems

IBM

Redbooks

International Technical Support Organization

IBM POWER8 High-Performance Computing Guide: IBM Power System S822LC (8335-GTB) Edition

May 2017

SG24-8371-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (May 2017)

This edition applies to:
IBM Platform LSF Standard 10.1.0.1
IBM XL Fortran v15.1.4 and v15.1.5 compilers
IBM XLC/C++ v13.1.2 and v13.1.5 compilers
IBM PE Developer Edition version 2.3
Red Hat Enterprise Linux (RHEL) 7.2 and 7.3 in little-endian mode

Contents

Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Trademarks

Preface

This IBM® Redbooks® publication documents and addresses topics to provide step-by-step customizable application and programming solutions to tune application and workloads to use IBM Power Systems™ hardware architecture. This publication explores, tests, and documents the solution to use the architectural technologies and the software solutions that are available from IBM to help solve challenging technical and business problems.

This publication also demonstrates and documents that the combination of IBM high-performance computing (HPC) solutions (hardware and software) delivers significant value to technical computing clients who are in need of cost-effective, highly scalable, and robust solutions.

First, the book provides a high-level overview of the HPC solution, including all of the components that makes the HPC cluster: IBM Power System S822LC (8335-GTB), software components, interconnect switches, and the IBM Spectrum™ Scale parallel file system. Then, the publication is divided in three parts: Part 1 focuses on the developers, Part 2 focuses on the administrators, and Part 3 focuses on the evaluators and planners of the solution.

The IBM Redbooks publication is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) who are responsible for delivering cost-effective HPC solutions that help uncover insights from vast amounts of client's data so they can optimize business results, product development, and scientific discoveries.

Note: Throughout the book, the $ sign is used to indicate the beginning of a shell command. This sign was chosen to make it easier for readers to distinguish comments (indicated by a # sign) from commands.

The authors do not explicitly differentiate between super user or normal user privileges. Instead, the authors assume that the reader knows that most administration task are done as a super user. Other tasks, such as compiling and debugging code or job submission, are normally done as a non-privileged user.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is a Complex Solutions Project Leader and an IBM Level 3 Certified Senior IT Specialist with the ITSO in Poughkeepsie, New York. His areas of expertise include enterprise continuous availability, enterprise systems management, system virtualization, technical computing, and clustering solutions. He also is an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Joseph Apuzzo is an Administrator of HPC clusters at the IBM Poughkeepsie Lab in New York. He has been with IBM since 1998, but has been working with Linux since 1993 (starting with Yggdrasil Linux with kernel 0.98.1). His tenure at IBM has been focused on HPC software, which started with PSSP, IBM GPFS™ and most recently xCAT. This book is his second IBM Redbooks publication for IBM. He holds a Master's degree in Computer Science, with an undergraduate degree in Electrical Engineering. He is the Inventor of code coverage methods for testing of modular software.

John Dunham is a Software Developer for Platform Computing & IBM Power Systems Software™ in Poughkeepsie, New York. He joined IBM full time in 2015 after an internship for nearly three years in his undergraduate program. He has acted as a System Administrator for the Poughkeepsie Lab and worked as a C/C++ programmer. He holds a Master's degree in Game Design and Development from RIT (2015) and Bachelor's Degree in Computer Science from Marist (2012).

Mauricio Faria de Oliveira is an Advisory Software Engineer at the Linux Technology Center at IBM Brazil. His areas of expertise include Linux performance analysis and optimization, Debian and Ubuntu for IBM PowerPC® 64-bit Little-Endian, and Multipath I/O on IBM Power and OpenPower Systems. He also worked with official benchmark publications for Linux on IBM Power Systems and early development (bootstrap) of Debian on PowerPC 64-bit Little-Endian. Mauricio holds a Master of Computer Science and Technology degree and a Bachelor of Engineering degree in Computer Engineering from Federal University of Itajuba, Brazil.

Markus Hilger is an IT Specialist for HPC and HPSS Systems in Germany. He was a corporate student with IBM before he joined IBM in 2014. His areas of expertise include system administration and monitoring and energy-efficiency characterization of large HPC clusters. He also works for the Leibniz Supercomputing Centre, servicing the warm-water cooled SuperMUC. He holds a degree in Business Information Technology. This book is his second IBM Redbooks publication.

Desnes Augusto Nunes Rosario is a Staff Software Engineer at the Linux Technology Center, Brazil. He has three years of experience working on Embedded Linux development and customization for multiple architectures at IBM. He holds a Bachelor degree in Computer and Automation Engineering, an Engineering Specialist degree in Industrial Automation, and a Master degree in Electrical and Computer Engineering from Universidade Federal do Rio Grande do Norte (UFRN). His areas of expertise include software development and packaging for Linux, parallel CPU/GPU algorithms design, descriptive hardware in FPGAs, and basic electronics and industrial automation.

Wainer dos Santos Moschetta is a Staff Software Engineer in the IBM Linux Technology Center, Brazil. He initiated and formerly led the IBM Software Development Kit (SDK) project for the IBM PowerLinuxTM project. He has more than seven years of experience with designing and implementing software development tools for Linux on IBM Power Systems. Wainer holds a Bachelor degree in Computer Science from the University of São Paulo. He co-authored several IBM Redbooks publications: *IBM Parallel Environment (PE) Developer Edition*, SG24-8075, *Performance Optimization and Tuning Techniques for IBM Power Systems processors, including IBM POWER8*, SG24-8171, and *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8264. He also has published articles and videos for the IBM developerWorks® website, and contributes to the IBM Linux on Power technical community blog.

Alexander Pozdneev is a Research Software Engineer at IBM Science and Technology Center, Moscow, Russia. He has 12 years of experience in HPC. He holds a Ph.D. degree in Mathematical Modeling, Numerical Methods, and Software from Lomonosov Moscow State University. His areas of expertise include parallel computing and application performance optimization.

Thanks to the following people for their contributions to this project:

Wade Wallace
International Technical Support Organization, Poughkeepsie Center

Joan McComb
Fernando Pizzano
Duane Witherspoon
Nina Wilner Volgl
IBM US

Luis Bolinches
IBM Finland

Rafael Peria De Sene
Roberto Guimaraes Dutra De Oliveira
Alisson Linhares de Carvalho
IBM Brazil

Ettore Tiotto
IBM Canada

Now you can become a published author, too!

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

IBM Power System S822LC for HPC server overview

This chapter describes the IBM Power System S822LC for HPC server and the IBM high-performance computing (HPC) solution. This chapter also includes the following topics:

For more information about IBM Systems hardware and software for high-performance computing, see the IBM Power Systems website.

1.1 IBM Power System S822LC for HPC server

IBM Power System S822LC for HPC server (8355-GTB) includes the following features:

- Powerful IBM POWER8® processors that offer 16 cores at 3.259 GHz with 3.857 GHz turbo performance or 20 cores at 2.860 GHz with 3.492 GHz turbo.
- A 19-inch rack-mount 2U configuration.
- NVIDIA NVLink technology for exceptional processor-to-accelerator intercommunication.
- Four dedicated connectors for the NVIDIA Tesla P100 GPU.

This system is the first IBM Power Systems offering with the NVIDIA NVLink Technology, which removes graphics processing unit (GPU) computing bottlenecks by using the high-bandwidth and low-latency NVLink interface from CPU-to-GPU and GPU-to-GPU. This feature unlocks new performance and new applications for accelerated computing.

The Tesla P100 is a powerful and architecturally complex GPU accelerator architecture build. It features a 15.3 billion transistor GPU, a new high-performance interconnect that greatly accelerates GPU peer-to-peer and GPU-to-CPU communications, new technologies to simplify GPU programming, and exceptional power efficiency.

The Tesla P100 includes the following key features:

- Extreme performance

  Powering high-performance computing, deep learning, and many more GPU computing areas.
- NVLink

  NVIDIA's new high-speed, high-bandwidth interconnect for maximum application scalability.
- HBM2

  Fast, high-capacity, extremely efficient chip-on-wafer-on-substrate (CoWoS) stacked memory architecture.
- Unified memory, compute preemption, and new artificial intelligence (AI) algorithms

  Significantly improved programming model and advanced AI software that is optimized for the Pascal architecture.
- 16 nm FinFET

  Enables more features, higher performance, and improved power efficiency.

The Power System S822LC is ideal for clients that need more processing power and increased workload density with reduced data center floor space requirements. It offers a modular design to scale from a single rack to hundreds of racks, simplicity of ordering, and a strong innovation road map for GPUs.

1.1.1 IBM POWER8 processor

The POWER8 processor that is used in the 8335-GTB is manufactured by using the IBM 22 nmsilicon-on-insulator (SOI) technology. Each chip is 65 mm2 and contains over 4.2 billion transistors. The POWER8 chip can contain up to 12 cores, 2 memory controllers, PCIe Gen3 I/O controllers, and an interconnection system that connects all components within the chip.

Each core features 512 KB of L2 cache, and all cores share 96 MB of L3 (8 MB of L3 cache per core) that is embedded DRAM (eDRAM). The interconnect also extends through module and system board technology to other POWER8 processors, DDR4 memory, and various I/O devices.

POWER8 processor-based systems use memory buffer chips to interface between the POWER8 processor and DDR4 memory. Each buffer chip also includes an L4 cache to reduce the latency of local memory accesses.

The following features augment the performance of the POWER8 processor:

- Support for DDR4 memory through memory buffer chips that offload the memory support from the POWER8 memory controller.
- An L4 cache within the memory buffer chip that reduces the memory latency for local access to memory behind the buffer chip. The operation of the L4 cache is not apparent to applications that are running on the POWER8 processor. Up to 128 MB of L4 cache can be available for each POWER8 processor.
- Hardware transactional memory.
- On-chip accelerators, including on-chip encryption, compression, and random number generation accelerators.
- CAPI, which allows accelerators that are plugged into a PCIe slot to access the processor bus by using a low-latency, high-speed protocol interface.
- Adaptive power management.

The Linux kernel sees a thread as an equivalent CPU. An Intel server with two processors has approximately 36 cores (with two threads per core), which yields only 72 threads. An IBM POWER8 server with two processors can have up to 24 cores (with eight threads per core), which yields a staggering 192 threads. When you run the Linux command **`lscpu`**, you see the output that is shown in Example 1-1.

Example 1-1 Linux `lscpu` command output

```
$ lscpu
Architecture:          ppc64le
Byte Order:            Little Endian
CPU(s):                192
On-line CPU(s) list:   0-191
Thread(s) per core:    8
Core(s) per socket:    12
Socket(s):             2
NUMA node(s):          2
Model:                 1.0 (xxx)
Model name:            POWER8NVL (raw), altivec supported
L1d cache:             64K
L1i cache:             32K
L2 cache:              512K
L3 cache:              8192K
```

```
NUMA node0 CPU(s):     0-95
NUMA node1 CPU(s):     96-191
```

Hardware transactional memory

Transactional memory is an alternative to lock-based synchronization. It attempts to simplify parallel programming by grouping read and write operations and running them as a single operation. Transactional memory is similar to database transactions, where all shared memory accesses and their effects are committed together or discarded as a group.

All threads can enter the critical region simultaneously. If there are conflicts in accessing the shared memory data, threads try accessing the shared memory data again or are stopped without updating the shared memory data. Therefore, transactional memory is also called a *lock-free synchronization*.

Transactional memory can be a competitive alternative to lock-based synchronization. Transactional memory provides a programming model that simplifies parallel programming. A programmer delimits regions of code that access shared data and the hardware runs these regions atomically and in isolation, buffers the results of individual instructions, and attempts to run again if isolation is violated. Generally, transactional memory allows programs to use a programming style that is close to coarse-grained locking to achieve performance that is close to fine-grained locking.

Most implementations of transactional memory are based on software. The POWER8 processor-based systems provide a hardware-based implementation of transactional memory that is more efficient than the software implementations and requires no interaction with the processor core. This configuration allows the system to operate in maximum performance.

1.1.2 NVLink

NVLink is NVIDIA's high-speed interconnect technology for GPU-accelerated computing. Supported on SXM2-based Tesla P100 accelerator boards, NVLink significantly increases performance for GPU-to-GPU communications and for GPU access to system memory.

Multiple GPUs are common in workstations, as are in the nodes of high-performance computing clusters and deep-learning training systems. A powerful interconnect is extremely valuable in multiprocessing systems. NVLink creates an interconnect for GPUs that offers higher bandwidth than PCI Express Gen3 (PCIe) and are compatible with the GPU ISA to support shared memory multiprocessing workloads.

Support for the GPU ISA allows programs that are running on NVLink-connected GPUs to run directly on data in the memory of another GPU and on local memory. GPUs can also perform atomic memory operations on remote GPU memory addresses, which enable much tighter data sharing and improved application scaling.

NVLink uses NVIDIA's new High-Speed Signaling interconnect (NVHS). NVHS transmits data over a differential pair that is running at up to 20 Gbps. Eight of these differential connections form a Sub-Link that sends data in one direction, and two sublinks (one for each direction) form a Link that connects two processors (GPU-to-GPU or GPU-to-CPU).

A single Link supports up to 40 GBps of bidirectional bandwidth between the endpoints. Multiple Links can be combined to form gangs for even higher-bandwidth connectivity between processors. The NVLink implementation in Tesla P100 supports up to four Links, which allows for a gang with an aggregate maximum theoretical bandwidth of 160 GBps bidirectional bandwidth.

Although NVLink primarily focuses on connecting multiple NVIDIA Tesla P100s, it can also connect Tesla P100 GPUs with IBM Power CPUs with NVLink support. In this configuration, each GPU has 80 GBps bidirectional bandwidth to the other connected GPU and 80 GBps bidirectional bandwidth to the connected CPU.

1.2 HPC system hardware components

An HPC system includes multiple servers and can run parallel programs on these machines. In this context, a *parallel program* is a piece of software that is specifically designed to run simultaneously on multiple servers. For more information about the nature of parallel programs, see 2.6, “Development models” on page 54.

An HPC system typically consists of the following server components[1]:

- Login nodes
- Management nodes
- Compute nodes
- Parallel file system

Server components of an HPC system are coupled with the following networks:

- Operating system (OS) level network
- Node and hardware management network
- High-performance interconnect

Note: A storage area network (SAN) is not an intrinsic component of an HPC system. Typically, the SAN is hidden at the level of the parallel file system.

A simplified logical overview of an HPC system is shown in Figure 1-1.

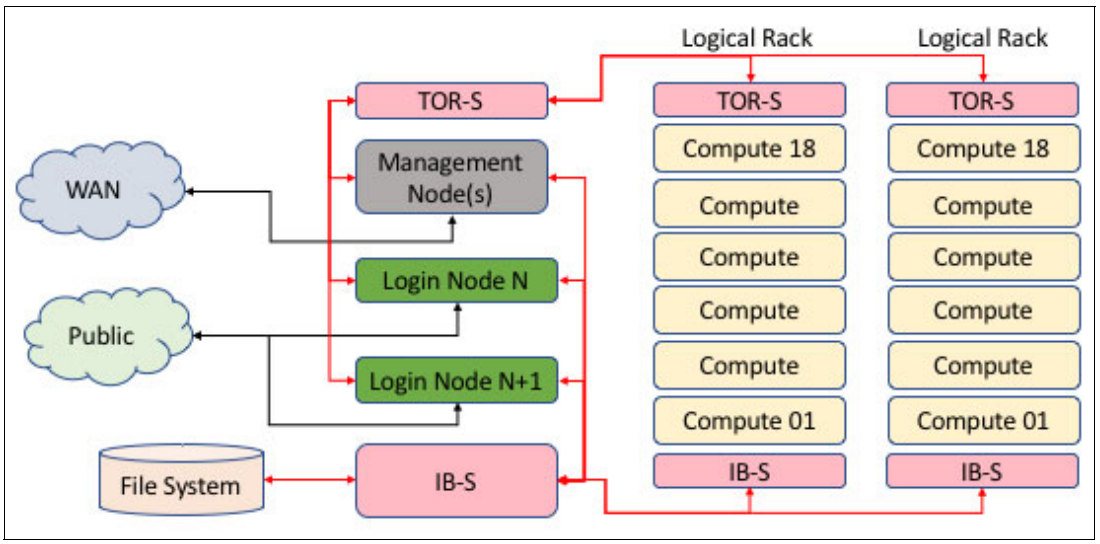


Figure 1-1 HPC system overview

The following sections describe each of these components. For more information about specific hardware offerings, see Chapter 7, “Hardware components” on page 319.

[1] In the IBM Blue™ Gene solution, *login nodes* were known as *frontend nodes*, and *management nodes* were known as *service nodes*.

1.2.1 Login nodes

The login node is a point of entry for users of an HPC system. Login nodes are typically made accessible from external networks (see Figure 1-1 on page 5). However, login nodes are typically hidden behind a firewall for security reasons. Login nodes can be made available only through virtual private network (VPN) connection. Other components of an HPC system are typically made inaccessible from external networks.

Note: The login node is the only component of an HPC system that is directly accessible by a user. Only system administrators have direct access to other components of an HPC system.

The file system that contains user data is typically kept physically separated from the login node hardware. This file system often is mounted to the login node as the `/home` directory. This file system is also mounted on the compute nodes and accessed by way of the high-speed InfiniBand network.

A user interacts with a login node according to the following typical scenario:

1. Copies data from some other system (local workstation, another remote system) to an HPC system.
2. Logs in to the HPC system.
3. Works with the HPC system in an interactive mode:
   - Edit source code files
   - Build, debug, and profile applications
   - Prepare input data for applications
   - Submit computing jobs (see “Workload management software” on page 15)
   - Post-process results of computations of previously completed jobs
4. Logs out of the HPC system.
5. Copies data from the HPC system to some other system (local workstation, another remote system).

1.2.2 Management nodes

Often, application users and developers are not aware of the existence of management nodes. These nodes are for use by system administrators only (see Figure 1-1 on page 5). The following system software components are typically on management nodes:

- System management software (see “System management software” on page 12)
- Workload management software (see “Workload management software” on page 15)

Therefore, management nodes are mainly used for the following purposes:

- Hardware and Software management tasks
- Deployment and update of compute nodes
- Managing resources and scheduling jobs

1.2.3 Compute nodes

Compute nodes constitute most of the server components of an HPC system (see Figure 1-1 on page 5). The purpose of compute nodes is to run parallel compute-intensive user tasks. Typically, all compute nodes of an HPC system have an identical hardware and software configuration. This configuration is used to ensure that the execution of a computation with some specific data takes the same amount of time regardless of a compute node on which it is run.

Compute nodes often are not directly accessible by a user. The user puts the compute tasks for execution through a job scheduler (see "Workload management software" on page 15).

Compute node of an HPC system include the following features:

- A processor is installed in each socket of a server or two servers.
- All memory slots of a server are populated with memory modules.
- The server has four GPUs (or a minimum of two).
- The server has a high-performance network adapter (see 1.2.5, "High-performance interconnect" on page 8).

Note: If you must choose between a hardware configuration with higher memory bandwidth and a server option with higher volume of memory, the choice often is made in favor of *higher* memory bandwidth.

1.2.4 Compute racks

As shown in Figure 1-1 on page 5, each compute rack is a logical unit that is self-contained. Each rack can hold up to 18 compute nodes. Each compute node is network-attached to the Top of Rack Switch (TOR-S), which is in turn connected to the management rack TOR-S. Each compute node also has a high-performance IB, which is connected to a rack InfiniBand switch. This switch then has mutable connections to a core InfiniBand switch.

In this example, a compute rack is a logical unit. Each rack can physically contain 0 - 18 nodes. Administrators tend to reference a physical node as being in a rack instead of a node number. Nodes are counted starting at the bottom of the physical rack and then counting up. This method is used because a rack must be bottom heavy. Therefore, you start at the bottom-most position and populate upwards when adding nodes to a rack.

A programmer can know a node only by its compute number by way of IBM Spectrum LSF®. Thus, node c100f05n01 (which is in frame 5 bottom most position as n01 indicates) can also be c100c062 to Spectrum LSF.

Therefore, each compute rack is self-contained, having networking connections back to the management rack and core InfiniBand switch.

1.2.5 High-performance interconnect

Technical computing workloads that involve interprocess communication are often characterized by a large volume of data that is transferred between processes. A delay between the request for data transfer and the actual data transfer affects the performance of applications that frequently send and receive small chunks of data. This delay means that the network interconnect between compute nodes of an HPC system must have the following features:

- High bandwidth
- Low latency

Modern high-performance interconnects typically implement the Remote Direct Memory Access (RDMA) feature. RDMA helps to minimize the processor overhead by allowing remote processor to directly access system memory with no OS involvement.

In addition to connecting compute nodes with each other, the high-performance interconnect provides access to the parallel file system (see Figure 1-1 on page 5 and 1.2.7, “Parallel file system” on page 10). This configuration allows high throughput operations on files.

High-performance interconnect includes the following hardware components:

- Host channel adapters (HCA)
- Network switches and the corresponding cables

The high-performance interconnect network is also known as *application network* because application processes that run on compute nodes use this network to communicate with each other.

1.2.6 Management and operating system

The following networks are used for hardware control, operating system deployment, and system management:

- Management network
- Operating system network

The hardware infrastructure for these networks includes the following components:

- Local area network (LAN) adapters built-in into servers
- Network switches
- Cables

Management network

The management network provides access to the service processors of the hardware (see Figure 1-1 on page 5). The management node uses this network to control the network attached devices in an *out-of-band* manner. For example, the IBM Power System S822LC server can be controlled through a baseboard management controller (BMC). The service network allows you to perform the following actions remotely:

- Updating the firmware
- Boot process troubleshooting by way of interacting with the system terminal
- Installation of operating system

Operating system network

The operating system network is used by the management node to support all management tasks that do not involve service processors of the hardware (see Figure 1-1 on page 5). This network has network interface controllers (NICs) as endpoints of the management node, login node, and compute nodes.

The management network allows management node (see 1.2.2, “Management nodes” on page 6) to perform the following tasks:

- Deployment of compute nodes:
  - Installing the operating system to the nodes
  - Managing the operating system of the nodes
  - Installing and configuring drivers and applications
- Managing resources and scheduling jobs

The following network services often are set up in the management network[2]:

- Domain name server (DNS)
- Hypertext Transfer Protocol (HTTP)
- Dynamic Host Configuration Protocol (DHCP)
- Trivial File Transfer Protocol (TFTP)
- Network file system (NFS)
- Network Time Protocol (NTP)

Networking technology considerations

The most common type of interconnect for management and service networks is the Gigabit Ethernet (1 GigE). This choice is dictated by the type of network that is supported by managed devices. For example, the typical network interface that is provided by a service processor of a server is 1 GigE.

If you plan a large data transfer between an HPC system and external world, you can consider a 10 Gigabit Ethernet (10 GigE) option for the site (public) network. However, 1 GigE often is enough for the site (public) network.

Security considerations

The access to an HPC system often is guarded externally by the following security technologies:

- Firewalls
- VPNs

Keep the login node (see 1.2.1, “Login nodes” on page 6) as a single point of entry to an HPC system. In this scenario, all the networks (application, management, and service) are configured only within an HPC system and are not visible from the outside. The management node (see 1.2.2, “Management nodes” on page 6) becomes accessible only through a login node.

One option to isolate networks is to use dedicated network switches and dedicated network interfaces for each network. However, a server built-in network port is often shared between the service processor network interface (BMC) and the OS. This configuration means that the separation of networks can be achieved within one network switch by using virtual local area network (VLAN) technology.

[2] The services are network services that are needed by xCAT.

1.2.7 Parallel file system

A shared file system that is accessible from the compute nodes and from a login node is an essential component of an HPC system (see Figure 1-1 on page 5). It is challenging to access the data that must be available for every node if no shared file system exists within an HPC system. For more information about the implications, see “Distributed execution environment” on page 14.

Technical computing workloads often require multiple processes of a distributed application to operate simultaneously on the same file. The parallel file system middleware hides the complexity of such operations and implements it in a performance efficient way.

When multiple processes access a file system at the same time, the file system performance can become a bottleneck. Parallel file systems use distributed storage servers and a high-performance interconnect. Therefore, parallel file systems help to minimize the performance implications of parallel input and output operations.

A parallel file system is simultaneously mounted on multiple nodes and provides the following features for an HPC system:

- Shared file system with common space of file names
- Simultaneous access to a file from different processes
- High bandwidth of input and output operations

Most portions of the following software components can be in a parallel file system:

- Distributed execution environment (see “Distributed execution environment” on page 14)
- Application development software (see 1.3.2, “Application development software” on page 16)
- Application software (see 1.3.3, “Application software” on page 18)

1.3 HPC system software components

An HPC system solution features the following main groups of software components:

- System software
- Application development software
- Application software

Figure 1-2 shows a simple extension of Figure 1-1 on page 5 with the mapping of software components to hardware.

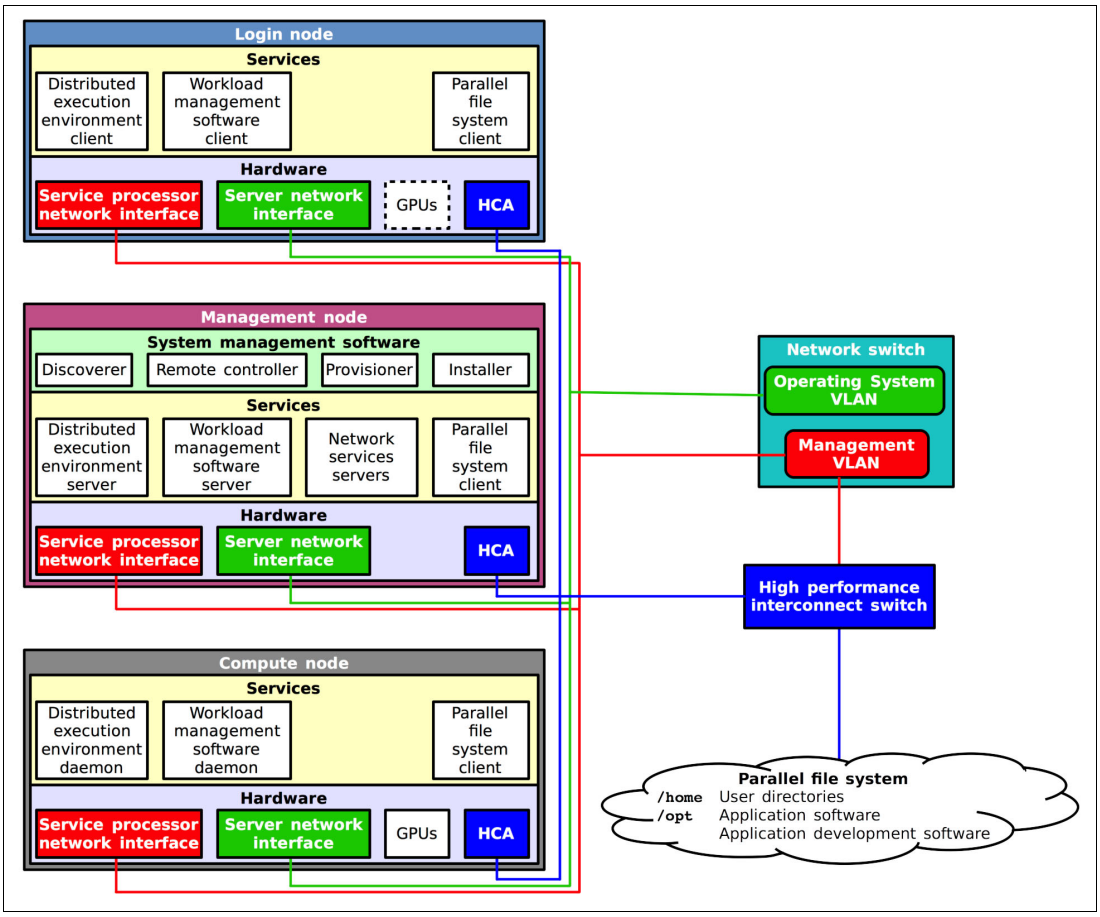


Figure 1-2 Software components of an HPC system that is mapped to the hardware

The following sections describe each of these components. For more information about specific software offerings, see Chapter 8, “Software stack” on page 351.

1.3.1 System software

The system software lies at the lowest level of software stack. It consists of the components that are responsible for the following tasks:

- System deployment
- Basic system functionality
- Operation of the high-performance computing pieces of hardware
- Access to parallel file system
- Running environment that supports distributed programs
- Workload management
- System monitoring

System management software

The system management software (see “Management node” in Figure 1-2 on page 11) is a cornerstone of the system software stack. It helps to automate the process of deployment and maintenance of an HPC system. Often, this software is the first piece of software that is deployed when an HPC system is installed. All the other software components are installed at a later stage.

Note: The choice of system management software is an important architectural decision because it can be quite time consuming to switch to another software after selecting and implementing the software.

The following tasks often are performed by using the system management software[3]:

- Discovery of the hardware servers
- Remote system management against the discovered server:
  - Remote power control
  - Remote console support
  - Remote inventory information query
- Provisioning operating system on physical (bare-metal) or virtual machines
- Installation and configuration of software:
  - During operating system installation
  - After the operating system installation
- System management in a parallel manner:
  - Parallel shell (that is, running shell command against nodes in parallel)
  - Parallel copy

These tasks mean that the system management software is crucial for the automation of the following routine tasks:

- Provisioning and deployment of multiple identical compute servers
- Maintenance of multiple identical system images:
  - Applying operating system updates
  - System-wide software settings change
- Replacement of a failing node with a new node

[3] This list of tasks partially originates from the list of xCAT features, but does not cover all of them.

System management software includes the following examples:

- xCAT (Extreme Cloud/Cluster Administration Toolkit

  For more information about shows how to use xCAT as a tool for an HPC system deployment, see Chapter 5, “Node and software deployment” on page 193.

- IBM Platform Cluster Manager

For more information about the IBM Spectrum Computing products family, see the IBM Spectrum Computing website.

Operating system

The operating system of choice in an HPC cluster is typically Linux. Also, typically all servers of an HPC system run the same Linux version. In rare cases, some components of an HPC system are required to run a specific version of Linux.

The HPC solution that is described in this book is based on Red Hat Enterprise Linux (RHEL) Server operating system. For information about how to install RHEL with xCAT, see 5.4.1, “RHEL server” on page 209.

Device drivers

Most of the hardware components of a server commonly include built-in support in an OS and do not require special handling. However, the device drivers that operate the high-performance computing pieces of hardware are not typically included with the OS and must be installed separately.

A modern HPC solution often needs drivers for the following devices:

- Hardware accelerator, such as a GPU

  For example, drivers for the NVIDIA Tesla GPU of the IBM Power System S822LC server are installed as part of NVIDIA CUDA Toolkit (see 5.6.3, “CUDA Toolkit” on page 247).

- High-performance interconnect host channel adapter that supports RDMA technology

  For more information about how to install the Mellanox Open Fabrics Enterprise Distribution (OFED) package that enables the Mellanox InfiniBand adapter, see 5.6.4, “Mellanox OFED” on page 249.

Parallel file system

Unlike compute accelerators and network adapters, a parallel file system is not a piece of hardware that is directly installed in a machine. A parallel file system can be thought of as a software service that is external to the machine (see Figure 1-2 on page 11). Typically, the interaction between machines and a parallel file system is organized in the following client/server manner:

- A parallel file system exports its services by running server software components.
- The machines that need access to a parallel file system run a client software component.

Therefore, parallel file system client software must be installed and configured on all machines that use a parallel file system. A parallel file system client software helps an OS to make a parallel file system available for a user by mounting it to a directory tree. As a result, the interaction with the parallel file system from the user perspective does not differ from the interaction with any other file system that is mounted to a machine.

This book shows how to couple an HPC system with a parallel file system: IBM Spectrum Scale™. IBM Spectrum Scale is a proven, scalable, high-performance data and file management solution. IBM Spectrum Scale is based on the IBM General Parallel File System (GPFS) technology. For more information about the deployment, see 5.6.13, “Spectrum Scale (formerly GPFS)” on page 261.

Distributed execution environment

The need for a distributed execution environment emerges when an application developer wants to use multiple compute nodes within a single application. HPC systems are used for this purpose (running parallel programs). Therefore, multiple frameworks are available that facilitate the development and running of this sort of computer codes.

The following sections describe the view of a distributed execution environment from a perspective of application development. For more information, see “Message passing interface (MPI)” on page 17. This section described how the distributed execution environment removes the burden of distributed application loading and running from application users and application developers.

Typically, a distributed execution environment (see Figure 1-2 on page 11) facilitates the following routine tasks that are related to running parallel applications:

- Runs an executable file on specific multiple compute nodes simultaneously.
- Monitors the runtime status of a parallel program.
- Stops a parallel program and cleans up compute nodes.
- Exports the OS environment variables to compute nodes before running a program.
- Manages standard input, output, and error streams (`stdin`, `stdout`, and `stderr`).
- Controls the binding and affinity of parallel processes and threads to logical processors.
- Selects a type of interconnect to use and tunes its parameters.
- Provides distributed debugging tools.
- Interacts with workload management software (see “Workload management software” on page 15).

Additionally, if a shared file system is not available in an HPC system, some distributed execution environments provide help with the following routine file operations:

- Copy a specified executable file to compute nodes before starting remote processes and delete it upon completing a job.
- Preinstall files to compute nodes where processes are run just before starting those processes.

Distributed execution environments include the following examples:

- IBM Parallel Environment Runtime Edition (for more information, see 3.3, “Using IBM Parallel Environment v2.3” on page 104)
- OpenMPI project

Workload management software

Usually, an HPC system is shared by many users and multiple computing tasks that are running at the same time. A workload management software automates the task of handling system resources and user jobs in such circumstances.

The following case provides an insight into a typical scenario where a workload management software becomes useful:

- An HPC system has a limited number of compute nodes.
- The application user must run parallel applications that require several compute nodes.
- A user has several computing jobs that must be run.
- Many users work with an HPC system at the same time.

Workload management software (see Figure 1-2 on page 11) uses the following two logical components to cope with this scenario:

- *Resource manager* monitors and controls compute nodes. A resource manager is aware of compute nodes that are idle or busy with user applications.
- *Job scheduler* maintains a queue of tasks from application users. Job scheduler uses information from resource manager to schedule the tasks for execution.

The interaction between application user, workload management software, and its components is based on the following scenario:

1. User submits a task to a job of the scheduler. The minimal specification of a task typically includes the following information:
   - Application name (path to an executable file)
   - Number of compute nodes to be used
   - Maximum time that is expected to be taken by a program to run
2. Job scheduler places the task into a job queue.
3. Workload management software schedules computing resources for the task and arranges a time slot for the execution.
4. At some moment in time, the workload management software sends the task for running.
5. When the task completes, it is removed from the job queue, and the user can collect the results.

The job scheduler uses multiple criteria to arrange jobs. Modern workload management software provides flexible options for tuning the configuration of a job queue and a scheduler to adhere to local site policies.

Application users can also complete the following tasks by using the workload management software:

- Request the status of a job queue.
- Inquire about the status of a particular job from a job queue.
- Change the specification of a task that is submitted to a job queue.
- Cancel a task that is submitted to a job queue.

At the core, the workload management software provides the following basic advantages for application users, system administrators, and HPC systems owners:

- Automation of task management
- Better system utilization

This book focuses on the IBM Spectrum LSF workload management software. For more information, see the IBM Spectrum LSF website.

For more information about the IBM Spectrum LSF deployment, see 5.6.14, “IBM Spectrum LSF” on page 266.

1.3.2 Application development software

As its name implies, application development software (see Figure 1-2 on page 11) is used to develop software. However, application development software is needed by the following categories of users:

- Application developers
- Application users
- System administrators

If application software or system software is distributed in source code package form, *application users* and *system administrators* use application development tools to create ready-to-use binary packages. However, the main target audience of application development software is *application developers*.

This chapter provides a brief overview of only the application development software stack.

Compilers

A compiler is a tool that converts source code into a binary executable file. The most popular languages in the area of HPC are C, C++, and Fortran. Most major distributions of Linux provide compilers from these languages. System vendors provide state-of-the-art compilers that can make full use of the underlying hardware. The following C, C++, and Fortran compilers are relevant for HPC systems that are based on IBM POWER® processors:

- GNU Compiler Collection (GCC)
- IBM Advance Toolchain for Linux on Power
- IBM XL compiler products
- NVIDIA compiler for GPU

For more information about these compilers, see 8.7, “IBM XL compilers, GCC, and Advance Toolchain” on page 355.

For more information about show how to deploy compilers, see the following sections:

- 5.6.5, “XL C/C++ runtime libraries” on page 251
- 5.6.6, “XL Fortran runtime libraries” on page 253
- 5.6.7, “Advance Toolchain runtime libraries” on page 254.

For more information about how to use these compilers for application development, see Chapter 2, “Compilation, execution, and application development” on page 23.

Parallel computing application interfaces

To use an HPC system, an application must run in parallel mode. It is the responsibility of an application developer to write the program in such a way as to make it possible to use the HPC system. If a program is not designed to run in an HPC system, the program cannot be easily parallelized.

You can make a program run in parallel by using three different methods, although these methods can be intermixed with each other. For more information, see 2.6, “Development models” on page 54.

OpenMP

OpenMP is the simplest way to enable parallelism in an application. The OpenMP standard defines a set of directives to be embedded into the source code. However, substantial effort is still needed to identify parallelism in an application domain, and design algorithms and data structures to fit the OpenMP parallel programming model.

OpenMP can be used to use the parallel capabilities of a single server. Parallel threads that are created by OpenMP require shared address space. A pool of OpenMP threads that is created by a process cannot span multiple compute nodes.

GCC and IBM compilers enable support of OpenMP through a compiler option.

NVIDIA CUDA

CUDA is a parallel computing platform and programming model by NVIDIA. CUDA is a way to use GPUs by NVIDIA. GPUs are especially efficient in solving data parallel problems. CUDA programs run within a single machine. A compiler that is a part of the CUDA Toolkit is needed to produce binary files that use GPUs.

Message passing interface (MPI)

MPI is the most widespread parallel programming interface to develop applications that span execution across multiple compute nodes.

Application that is written with MPI runs multiple threads on different compute nodes. Processes of an application coordinate their execution by sending messages to each other or by using remote memory access techniques.

Note: In contrast to problems that can be solved with widely accepted map-reduce type programming model, HPC problems often are highly sensitive to the latency of individual operations. Ideally, processes of an HPC application must run in sync with each other and communicate with minimal latency and maximum bandwidth. MPI facilitates the development in this programming model and uses the underlying high-performance interconnect.

For more information about shows how to develop MPI programs with IBM Parallel Environment Runtime Edition, see 2.6.5, “MPI programs with IBM Parallel Environment v2.3” on page 70.

Mathematical libraries

Software libraries of mathematical routines are essential part of an HPC system. A mathematical library is a software package that implements some numerical algorithms. Application developers access the algorithms through a programming interface that is made available by a library. Hardware vendors often supply libraries that are optimized for a particular architecture. Many libraries also implement parallel algorithms.

By using mathematical libraries, an application developer receives the following benefits:

- Saves time on implementing standard mathematical routines
- Uses optimized implementation that is supplied by a hardware vendor
- Uses parallelism that is hidden inside a library

This book focuses on the IBM Engineering and Scientific Subroutine Library (ESSL) offering. For more information, see the following sections:

- For an overview, see 8.9, “IBM Engineering and Scientific Subroutine Library and IBM Parallel ESSL” on page 357.

- For more information about deployment, see 5.6.11, “ESSL” on page 259 and 5.6.12, “PESSL” on page 260.
- For more information about usage, see 2.3, “IBM Engineering and Scientific Subroutine Library” on page 34 and 2.4, “Parallel ESSL” on page 40.

Integrated development environments

Integrated development environment (IDE) provides a convenient GUI for application developers. IDE often features the following tools:

- Code editor with syntax highlighting and code completion
- Building tools
- Remote application launcher
- Debugger
- Code analyzer
- Profiler

For more information about IDE options, see 4.6, “Application development and tuning tools” on page 159.

Debuggers

A code debugger is an application development tool that facilitates the process of eliminating programming errors from a source code. Parallel applications provide more challenges for debugging compared to serial applications.

For more information about the tools that are available for debugging MPI and CUDA programs, see 4.6, “Application development and tuning tools” on page 159.

Performance analysis tools

Profilers (or performance analysis tools) automate the process of finding hotspots in a code. These tools help to evaluate the following metrics:

- Stalls of processor core units
- Effective memory bandwidth
- Cache hits and misses
- Graphical processing unit performance
- Network performance

For more information about performance analysis tools, see 4.6, “Application development and tuning tools” on page 159. For more information about performance optimization with the help of performance analysis tools, see the following publications:

- *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, SG24-8171
- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263

1.3.3 Application software

Running application software is essentially the ultimate purpose of an HPC system’s existence. Application software (see Figure 1-2 on page 11) is a tool that users employ to solve problems from application domains.

However, this book targets system administrators and application developers. Therefore, it does not cover application software extensively. For more information about examples of application software, see Appendix A, “ISV Applications” on page 361.

1.4 HPC system solution

This chapter presented a generic overview of the hardware and the software components of an HPC system. This section revisits the schemes that are shown in Figure 1-1 on page 5 and Figure 1-2 on page 11 and provides the specific names of the IBM products.

For more information about HPC system solution implementation, see Chapter 5, “Node and software deployment” on page 193.

1.4.1 Compute nodes

Use the IBM Power System S822LC (model 8335-GTB) server offering for high-performance computing as compute nodes. Consider the option to augment the server with the following devices:

- Two NVIDIA Tesla P100 GPUs (see 7.2, “NVIDIA Tesla P100” on page 324)
- One 100Gb EDR InfiniBand Adapter (see 7.14, “Mellanox InfiniBand” on page 349)

Processor options and system memory

This IBM Power System S822LC model has two sockets, and all memory slots are populated with memory modules. When choosing the processor option and the amount of system memory, consider the anticipated workloads.

Disk features

A compute node does not need large and fast disks because it stores only the OS, drivers, and minor pieces of system software (see Figure 1-2 on page 11). However, if you plan to extensively use local disk space during computations, consider the option of larger and faster disks.

1.4.2 Management node

A management node does not need GPUs and high memory bandwidth because it does not use many processor cycles. Therefore, even an entry-level server option has enough resources to satisfy the needs of a management node. Consider the IBM Power System S812LC or the IBM Power System S812L offerings as a possible management node.

1.4.3 Login node

A login node is not used as a computing resource. Therefore, the most basic solution can be built that is based on the IBM Power System S812LC or the IBM Power System S812L offerings. However, if the GPU is required in a login node, consider the use of an IBM Power System S822LC or an IBM Power System S824L for this node.

1.4.4 Combining the management and the login node

Consider the following scenario:

- No large workload is expected on the management and login nodes.
- The login node does not need GPU.

In this case, consider the use of only one physical machine (IBM Power System S812LC or IBM Power System S812L). In such a scenario, the management and login node can coexist as PowerKVM guests.

1.4.5 Parallel file system

To implement the parallel file system, consider one of the following options:

- IBM Elastic Storage Server
- IBM Spectrum Scale (built on the former IBM GPFS)

1.4.6 High-performance interconnect switch

The implementation of the high-performance interconnect is built around an InfiniBand switch. You can consider an offering from Mellanox. For more information about a suitable product, see the hardware compatibility matrix that is available at the IBM HPC Clustering with the InfiniBand Switch and IBM POWER8 S822LC Compute Nodes - Service Pack 1.2 page of the IBM developerWorks website.

Part 1

Developers guide

This part incldue three chapters that provide developers with guidance about how to create, compile, run, measure, and enhance the performance of applications to use the capabilities of the latest IBM high performance computing solution.

This part contains the following chapters:

2

Compilation, execution, and application development

This chapter provides information about software, compilers, and tools that can be used for application development and tuning on the IBM Power System S822LC. A few development models also are described.

This chapter includes the following topics:

2.1 Compiler options

Compiler options are one of the main tools that are used to debug and optimize the performance of your code during development. Several compilers, including IBM XL, GNU Compiler Collection (GCC), PGI, and NVIDIA compilers support the latest IBM POWER8 processor features and enhancements.

2.1.1 IBM XL compiler options

XL C/C++ for Linux v13.1.5 and XL Fortran for Linux v15.1.5 support POWER8 processors with new features and enhancements, such as little endian distributions support, compiler optimizations, and built-in functions for POWER8 processors. In addition, many OpenMP 4.5 features were introduced[1], such as the constructs that enable the loading and offloading of applications and data to NVIDIA GPU technology.

By default, these compilers generate code that runs on various IBM Power Systems. Options can be added to exclude older processor chips that are not supported by the target application. The following major XL compiler options control this support:

- `-mcpu`

  Specifies the family of the processor architecture for which the instruction code is generated.

- `-mtune`

  Indicates the processor chip generation of most interest for performance. It tunes instruction selection, scheduling, and other architecture-dependent performance enhancements to run optimally on a specific hardware architecture.

- `-qcache`

  Defines a specific cache or memory geometry.

The `-mcpu=pwr8` suboption produces object code that contains instructions that run on the POWER8 hardware platforms. With the `-mtune=pwr8` suboption, optimizations are tuned for the POWER8 hardware platforms. This configuration can enable better code generation because the compiler uses capabilities that were not available on those older systems.

For all production codes, it is imperative to enable a minimum level of compiler optimization by adding the `-0` option for the XL compilers. Without optimization, the focus of the compiler is on faster compilation and debug ability, and it generates code that performs poorly at run time.

For projects with increased focus on runtime performance, use the more advanced compiler optimization. For numerical or compute-intensive codes, the XL compiler options `-03` or `-qhot -03` enable loop transformations, which improve program performance by restructuring loops to make their execution more efficient by the target system. These options perform aggressive transformations that can sometimes cause minor differences in the precision of floating point computations. If the minor differences are a concern, the original program semantics can be fully recovered with the `-qstrict` option.

For more information about XL C/C++ support for POWER8 processor, see the IBM XL C/C++ for Linux, V13.1.5 (little endian distributions) documentation page at the IBM Knowledge Center website.

[1] OpenMP (Open Multi-Processing) is an application programming interface that facilitates the development of parallel applications for shared memory systems.

For more information about XL Fortran, see the IBM XL Fortran for Linux, V15.1.5 (little endian distributions) documentation page at the IBM Knowledge Center website.

Optimization parameters

The strength of the XL compilers is in their optimization and ability to improve code generation. Optimized code runs with greater speed, uses less machine resources, and increases your productivity.

For XL C/C++ v13.1.5, the available compiler options to maximize application development performance are listed in Table 2-1.

Table 2-1 Optimizations levels and options

Based optimization level	Other options that are implied by level	Other suggested options	Other options with possible benefits
-O0	None	-mcpu	None
-O2	-qmaxmem=8192	-mcpu -mtune	-qmaxmem=-1 -qhot=level=0
-O3	-qnostrict -qmaxmem=-1 -qhot=level=0	-mcpu -mtune	-qpdf
-O4	-qnostrict -qmaxmem=-1 -qhot -qipa -qarch=auto -qtune=auto -qcache=auto	-mcpu -mtune -qcache	-qpdf -qsmp=auto
-O5	All of -O4 -qipa=level=2	-mcpu -mtune -qcache	-qpdf -qsmp=auto

Several options are used to control the optimization and tuning process, so users can improve the performance of their application at run time.

When you compile programs with any of the following sets of options, the compiler automatically attempts to vectorize calls to system math functions. It does so by calling the equivalent vector functions in the Mathematical Acceleration Subsystem (MASS) libraries, with the exceptions of functions `vdnint`, `vdint`, `vcosisin`, `vscosisin`, `vqdrt`, `vsqdrt`, `vrqdrt`, `vsrqdrt`, `vpopcnt4`, and `vpopcnt8`:

- `-qhot -qignerrno -qnostrict`
- `-03 -qhot`
- `-04`
- `-05`

If the compiler cannot vectorize, it automatically tries to call the equivalent MASS scalar functions. For automatic vectorization or scalarization, the compiler uses versions of the MASS functions that are contained in the system library `libxlopt.a`.

In addition to any of the sets of options, if the compiler cannot vectorize when the `-qipa` option is in effect, it tries to inline the MASS scalar functions before it decides to call them.

Not all options benefit all applications. Tradeoffs sometimes occur between an increase in compile time, a reduction in debugging capability, and the improvements that optimization can provide. For more information about the optimization and tuning process and writing optimization-friendly source code, see the options that are listed in Table 2-2 and *Optimization and Programming Guide - XL C/C++ for Linux, V13.1.5, for little endian distributions,* SC27-6560.

Table 2-2 Optimization and tuning options

Option name	Description
`-qhot`	Performs aggressive, high-order loop analysis and transformations during optimization (implies -O2 optimization). Look for the suboptions available for `-qhot` to enable even more loop transformation, cache reuse, and loop parallelization when used with `-qsmp`.
`-qipa`	Enables or customizes a class of optimizations that are known as interprocedural analysis (IPA). These optimizations enable a whole-program analysis at one time rather than a file-by-file basis. The XL compiler receives the power to restructure your application and perform optimizations, such as inlining for all functions and disambiguation of pointer references and calls. Look over `-qipa` suboptions for more features.
`-qmaxmem`	Limits the amount of memory that the compiler allocates while it performs specific, memory-intensive optimizations to the specified number of KBs.
`-qignerrno`	Allows the compiler to perform optimizations as though system calls will not modify `errno`.
`-qpdf1`, `-qpdf2`	Tunes optimizations through profile-directed feedback (PDF), where results from sample program execution that was compiled with `-qpdf1` are used to improve optimization near conditional branches and in frequently run code sections afterward with `-qpdf2`, which creates a profiled optimized executable file.
`-p`, `-pg`, `-qprofile`	The compiler prepares the object code for profiling by attaching monitoring code to the executable file. This code counts the number of times each routine is called and creates a `gmon.out` file if the executable file can run successfully. Afterward, a tool, such as `gprof`, can be used to generate a runtime profile of the program.
`-qinline`	Attempts to inline functions instead of generating calls to those functions for improved performance.
`-qstrict`	Ensures that optimizations that are performed by default at the `-O3` and higher optimization levels, and, optionally at `-O2`, and do not alter the semantics of a program.
`-qsimd`	Controls whether the compiler can automatically use vector instructions for processors that support them.
`-qsmp`	Enables parallelization of program code automatically by the compiler. Options, such as schedulers and chunk sizes, can be enabled to the code by this option.

Option name	Description
`-qoffload`	Enables support for offloading OpenMP target regions to a NVIDIA GPU. Use the OpenMP `#pragma omp target` directive to define a target region. Moreover, to `-qoffload` to take effect, you must specify the `-qsmp` option during compilation.
`-qunroll`	Controls loop unrolling for improved performance.

For more information about the XL C/C++ compiler options and for XL Fortran, see the following IBM Knowledge Center pages:

- Organization and tuning page for version 13.1.5
- Organization and tuning page for version 15.1.5

2.1.2 GCC compiler options

For GCC, a minimum level of compiler optimization is `-O2`, and the suggested level of optimization is `-O3`. The GCC default is a strict mode, but the `-ffast-math` option disables strict mode. The `-O` fast option combines `-O3` with `-ffast-math` in a single option. Other important options include `-fpeel-loops`, `-funroll-loops`, `-ftree-vectorize`, `-fvect-cost-model`, and `-mcmodel=medium`.

Support for the POWER8 processor is now available on GCC-4.8.5 through the `-mcpu=power8` and `-mtune=power8` options. The `-mcpu` options automatically enable or disable the following options:

- `-maltivec -mfprnd -mhard-float -mmfcrf -mmultiple`
- `-mpopcntb -mpopcntd -mpowerpc64`
- `-mpowerpc-gpopt -mpowerpc-gfxopt -msingle-float -mdouble-float`
- `-msimple-fpu -mstring -mmulhw -mdlmzb -mmfpgpr -mvsx`
- `-mcrypto -mdirect-move -mpower8-fusion -mpower8-vector`
- `-mquad-memory -mquad-memory-atomic`

The ABI type to use for intrinsic vectorizing can be set by specifying the `-mveclibabi=mass` option and linking to the MASS libraries, which enables more loops with `-ftree-vectorize`. The MASS libraries support only static archives for linking. Therefore, the following explicit naming and library search order is required for each platform or mode:

- POWER8 32-bit: `-L<MASS-dir>/lib -lmassvp8 -lmass_simdp8 -lmass -lm`
- POWER8 64-bit: `-L<MASS-dir>/lib64 -lmassvp8_64 -lmass_simdp8_64 -lmass_64 -lm`

For more information about GCC support on POWER8, see the GNU Manual website.

Optimization parameters

The most commonly used optimization options are listed in Table 2-3.

Table 2-3 Optimization options for GCC

Option name	Description
-0, -01	With the -0 option, the compiler tries to reduce code size and execution time without performing any optimizations that significant compilation time.
-02	The -02 option turns on all optional optimizations except for loop unrolling, function inlining, and register renaming. It also turns on the -fforce-mem option on all machines and frame pointer elimination on machines where frame pointer elimination does not interfere with debugging.
-03	The -03 option turns on all optimizations that are specified by -02 and turns on the -finline-functions, -frename-registers, -funswitch-loops, -fpredictive-commoning, -fgcse-after-reload, -ftree-vectorize, -fvect-cost-model, -ftree-partial-pre, and -fipa-cp-clone options.
-00	Reduces compilation time and grants debuggers full access to the code. Do not optimize.
-0s	Optimize for size. The -0s option enables all -02 optimizations that do not typically increase code size. It also performs further optimizations that are designed to reduce code size.
-ffast-math	Sets -fno-math-errno, -funsafe-math-optimizations, and -fno-trapping-math, -ffinite-math-only, -fno-rounding-math, -fno-signaling-nans, and -fcx-limited-range. This option causes the preprocessor macro __FAST_MATH__ to be defined. This option must never be turned on by any -0 besides -0fast option because it can result in incorrect output for programs that depend on an exact implementation of IEEE or ISO rules or specifications for math functions.
-funroll-loops	Unroll loops whose number of iterations can be determined at compile time or upon entry to the loop. The -funroll-loops option implies both the -fstrength-reduce, -frerun-cse-after-loop, -fweb, and -frename-registers options.
-fno-inline	Do not expand any functions inline apart from those functions that are marked with the always_inline attribute. This setting is the default setting when not optimizing. Single functions can be exempted from inlining by marking them with the noinline attribute.
-fno-math-errno	Do not set ERRNO after math functions are called that are run with a single instruction; for example, sqrt. A program that relies on IEEE exceptions for math error handling can use this flag for speed while maintaining IEEE arithmetic compatibility.
-finline-functions	Consider all functions for inlining, even if they are not declared inline. The compiler heuristically decides which functions are worth integrating in this way. If all calls to a specific function are integrated and the function is declared static, the function is normally not output as assembler code in its own right. Enabled at level -O3.

2.2 Porting applications to IBM Power Systems

The first challenge that often is encountered when working on Power Systems is the fact that software that is written in C, C++, or Fortran includes some significant differences from applications that were developed for x86 and x86_64 systems. Therefore, considering that many applications exist on these architectures, the work of writing and porting applications are assumed to involve a large amount of time, effort, and investment. If done manually, this task can be burdensome.

For example, consider the snippet that is shown in Example 2-1.

Example 2-1 x86 code that does not work properly on Power Systems

```
1. #include <stdio.h>
2.
3. int main()
4. {
5.       char c;
6.
7.       for( c=0 ; c>=0 ; c++)
8.              printf("%d: %c\n", (int) c, c);
9.
10.      return 0;
11.}
```

The code that is shown in Example 2-1 runs 256 times and displays the expected characters on a x86_64 system, but turns into an infinite loop in a ppc64le system. Considering that characters on ppc are mapped by default to unsigned character at the same time x86 characters are mapped to a signed character, the port as shown in Example 2-2 is necessary to fix the infinite loop challenge.

Example 2-2 The ppc port of Example 2-1

```
--- char_x86.c2016-11-15 11:38:13.153959471 -0500
+++ char_ppc.c2016-11-15 11:38:21.833959907 -0500
@@ -2,7 +2,7 @@

 int main()
 {
-  char c;
+  signed char c;

   for( c=0 ; c>=0 ; c++)
      printf("%d: %c\n",(int)c,c);
```

If you are coming from a 32-bit environment, several problems can occur, mostly because of long pointer data types sizes and the alignment of your variables as listed in Table 2-4.

Table 2-4 Data types

Data type	32-bit mode		64-bit mode	
	Size	Alignment	Size	Alignment
long, signed long, unsigned long	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
pointer	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
size_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries
ptrdiff_t (defined in the header file <cstddef>)	4 bytes	4-byte boundaries	8 bytes	8-byte boundaries

Moreover, architecture that is built in functions can also be a problem. Consider an example of SIMD vectors for the following code that is designed for an x86 system, as shown in Example 2-3.

Example 2-3 Code with x86 built in functions that need porting

```
1. #include <stdlib.h>
2. #include <stdio.h>
3.
4. typedef float v4sf __attribute__ ((mode(V4SF)));
5.
6. int main()
7. {
8.       int i;
9.       v4sf v1, v2, v3;
10.
11.      //Vector 1
12.      v1[0] = 20;
13.      v1[1] = 2;
14.      v1[2] = 9;
15.      v1[3] = 100;
16.
17.      //Vector 2
18.      v2[0] = 2;
19.      v2[1] = 10;
20.      v2[2] = 3;
21.      v2[3] = 2;
22.
23.      // Sum 2 Vectors
24.      v3 = __builtin_ia32_addps(v1, v2);
25.
26.      printf("SIMD addition\nResult\nv3 = < ");
27.      for (i = 0; i < 4; i++)
28.             printf("%.1f ", v3[i]);
29.      printf(">\n");
30.
```

```
31.      // Compare element by element from both vectors and get the higher value
   from each position
32.      v3 = __builtin_ia32_maxps(v1, v2);
33.
34.      printf("SIMD maxps\nResult\nv3 = < ");
35.      for (i = 0; i < 4; i++)
36.             printf("%.1f ", v3[i]);
37.      printf(">\n");
38.
39.      return 0;
40.}
```

This code uses x86 built-in functions, such as `__builtin_ia32_addps` and `__builtin_ia32_maxps`. It is possible to look over both documentations to find the corresponding functions; that is, if they exist between these architectures. However, this strategy can be burdensome.

To mitigate all of these challenges, IBM created an all-in-one solution for developing and porting applications on power servers that are denominated IBM Software Development Kit for Linux on Power, see the IBM Software Development Kit for Linux on Power (SDK) website.

Among other features, this SDK aims to aid developers to code, fix, simulate, port, and run software locally in a x86_64 machine, virtually in a x86_64 simulation, or remotely in a power server. Currently, this SDK integrates the Eclipse integrated development environment (IDE) with IBM XL C/C++, Advance Toolchain and open source tools, such as OProfile, Valgrind, and Autotools. Figure 2-1 shows the detection of such of function in the code that is shown in Example 2-3 on page 30.

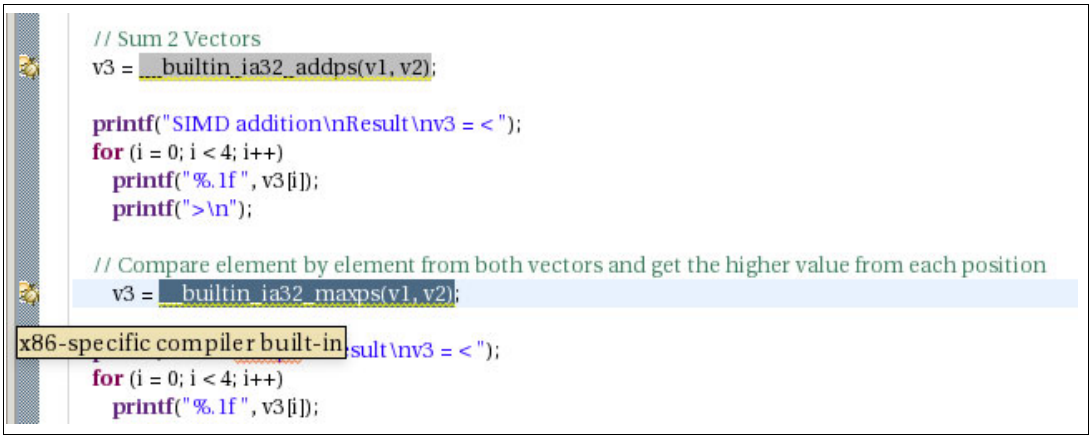


Figure 2-1 Porting problems on Example 2-3 on page 30 detected by ibm-sdk-lop

In addition, this tool integrates the Feedback Directed Program Restructuring (IBM FDPR®) and pthread monitoring tool, which are designed to analyze and use Power Systems servers, including powerful porting and analytic tools, such as Migration Advisor (MA), Source Code Advisor, and CPI Breakdown.

By starting the Migration Advisor Wizard on the code that is shown in Example 2-3 on page 30, two options are available, as shown in Figure 2-2.

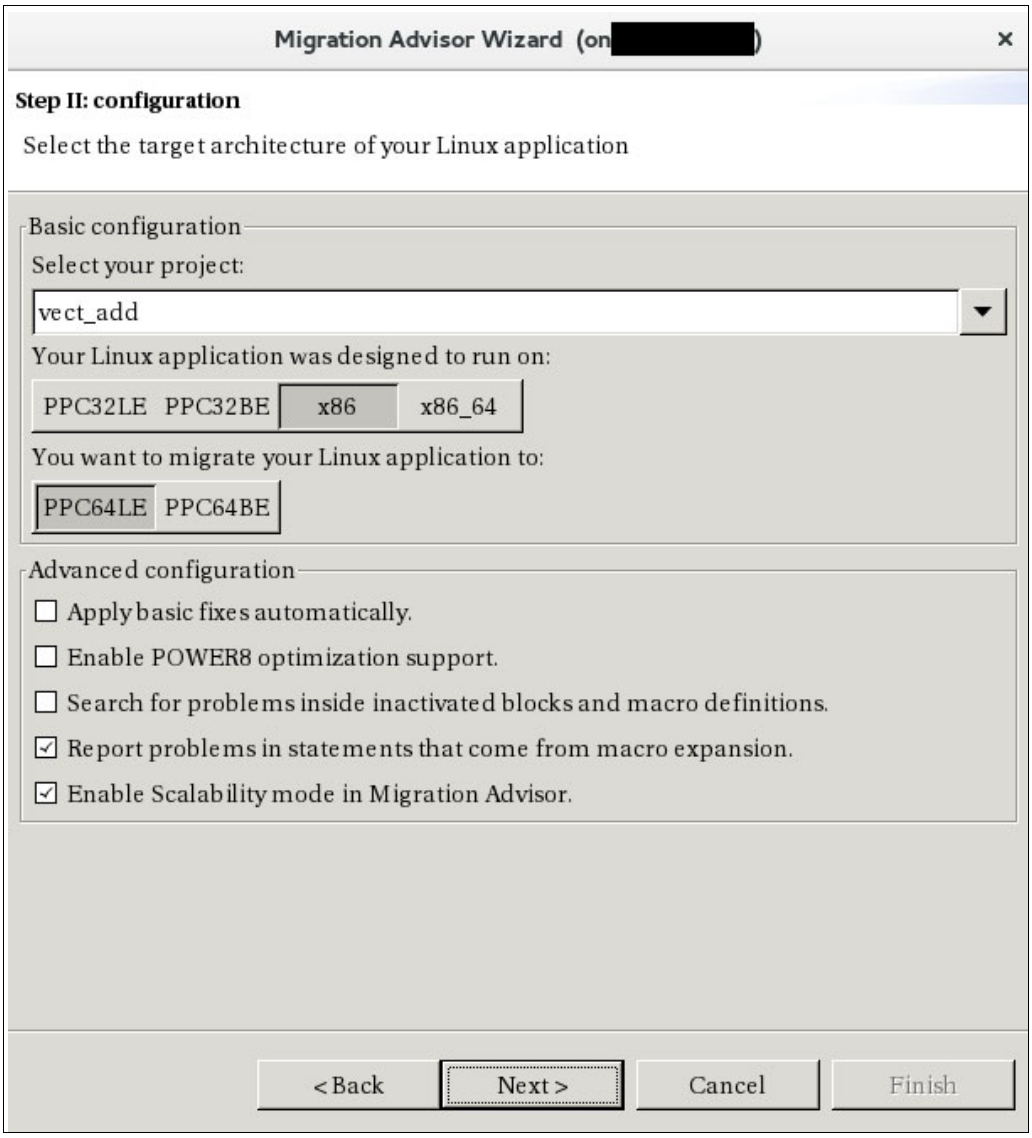


Figure 2-2 Migration Wizard enables porting code from different architectures and endianness

The code that is shown in Example 2-3 on page 30 leads to the port report as shown in Figure 2-3.

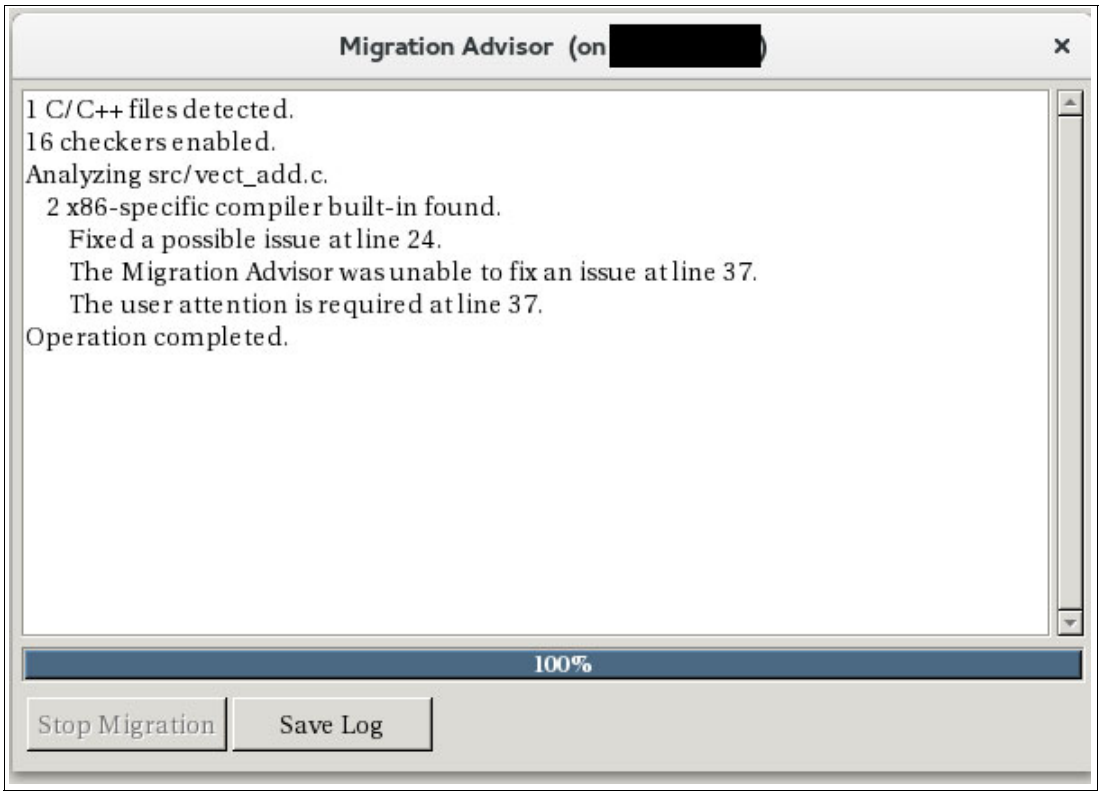


Figure 2-3 Porting log showing changes that were performed on Example 2-3 on page 30

Consider the changes that the advisor performed as shown in Example 2-4.

Example 2-4 Porting changes that performed by the Migration Wizard on Example 2-3 on page 30

```
--- vector_before_migration     2016-12-01 17:04:47.332015615 -0500
+++ vector_after_migration      2016-12-01 17:04:11.783017631 -0500
@@ -3,6 +3,12 @@

 typedef float v4sf __attribute__ ((mode(V4SF)));

+#ifdef __PPC__
+__vector float vec__builtin_ia32_maxps(__vector float a, __vector float b) {
+       //TODO: You need to provide a PPC implementation.
+}
+#endif
+
 int main()
 {
        int i;
@@ -21,7 +27,11 @@ int main()
        v2[3] = 2;

        // Sum 2 Vectors
-       v3 = __builtin_ia32_addps(v1, v2);
+       #ifdef __PPC__
+               v3 = vec_add(v1, v2);
```

```
+       #else
+               v3 = __builtin_ia32_addps(v1, v2);
+       #endif

        printf("SIMD addition\nResult\nv3 = < ");
        for (i = 0; i < 4; i++)
@@ -29,7 +39,11 @@ int main()
        printf(">\n");

        // Compare element by element from both vectors and get the higher value
from each position
+       #ifdef __PPC__
+       v3 = vec__builtin_ia32_maxps(v1, v2);
+       #else
        v3 = __builtin_ia32_maxps(v1, v2);
+       #endif

        printf("SIMD maxps\nResult\nv3 = < ");
        for (i = 0; i < 4; i++)
```

The wizard found a corresponding function for `function __builtin_ia32_addps()`, and informed us that we must implement our own version of `function __builtin_ia32_maxps()` on `vec__builtin_ia32_maxps()` that is now before `main()`. Moreover, all PPC wrappers were created.

This tool can automatically solve most (hundreds) of porting errors from x86 to ppc by clicking a button on the `ibm-sdk-lop` interface. For more information about the SDK features, see the *Developing software using the IBM Software Development Kit for Linux on Power* manual.

2.3 IBM Engineering and Scientific Subroutine Library

The IBM Engineering and Scientific Subroutine Library (ESSL) includes the following runtime libraries:

- ESSL Serial Libraries and ESSL SMP Libraries
- ESSL SMP CUDA Library

The following mathematical subroutines, in nine computational areas, are tuned for performance:

- Linear Algebra Subprograms
- Matrix Operations
- Linear Algebraic Equations
- Eigensystem Analysis
- Fourier Transforms, Convolutions and Correlations, and Related Computations v Sorting and Searching
- Interpolation
- Numerical Quadrature
- Random Number Generation

2.3.1 ESSL Compilation in Fortran, XL C/C++, and GCC/G++

The ESSL subroutines are callable from programs, which are written in Fortran, C, and C++. Table 2-5, Table 2-7 on page 36, and Table 2-9 on page 37 list how compilation on Linux depends on the type of ESSL library (serial, SMP, or SMP CUDA), environment (32-bit integer, 64-bit pointer or 64-bit integer, or 64-bit pointer) and compiler (XLF, XL C/C++, or gcc/g++) is used.

Table 2-5 Fortran compilation commands for ESSL

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	xlf_r -O -qnosave program.f -lessl
	64-bit integer, 64-bit pointer	xlf_r -O -qnosave program.f -lessl6464
SMP	32-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmp xlf_r -O -qnosave program.f -lesslsmp -lxlsmp
	64-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmp6464 xlf_r -O -qnosave program.f -lesslsmp6464 -lxlsmp
SMP CUDA	32-bit integer, 64-bit pointer	xlf_r -O -qnosave -qsmp program.f -lesslsmpcuda -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 xlf_r -O -qnosave program.f -lesslsmpcuda -lxlsmp -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64

To use the FFTW Wrapper libraries, the header file `fftw3.f` that contains the constant definitions must be included. To use these definitions, complete one of the following tasks:

- Add the following line to your Fortran application: `include "fftw3.f"`
- Add the `fftw3.f` header file in your application

You also can compile and link with the FFTW Wrapper libraries by using the commands that are listed in Table 2-6.

Table 2-6 Fortran compilation guidelines assume FFTW wrappers are in /usr/local/include

Type of ESSL library	Environment	Compilation command

Serial	32-bit integer, 64-bit pointer	`xlf_r -O -qnosave program.f`
		`-lessl -lfftw3_essl`
		`-I/usr/local/include`
		`-L/usr/local/lib64`

SMP	32-bit integer, 64-bit pointer	`xlf_r -O -qnosave program.f`
		`-lesslsmp -lfftw3_essl`
		`-I/usr/local/include`
		`-L/usr/local/lib64`

Table 2-7 lists the compilation guidelines for ESSL using XL C/C++.

Table 2-7 XL C/C++ compilation commands (cc_r for XLC, xlC_r for XLC++) for ESSL

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	`cc_r (xlC_r) -O program.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer, 64-bit pointer	`cc_r (xlC_r) -O -D_ESV6464 program.c -lessl6464 -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
SMP	32-bit integer, 64-bit pointer	`cc_r (xlC_r) -O program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
	64-bit integer, 64-bit pointer	`cc_r (xlC_r) -O -D_ESV6464 program.c -lesslsmp6464 -lxlf90_r -lxlsmp -lxlfmath -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`
SMP CUDA	32-bit integer, 64-bit pointer	`cc_r (xlC_r) -O program.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib`

To compile and link the FFTW Wrapper libraries, the commands that are listed in Table 2-8 must be used. Also, use the header file `fftw3_essl.h` instead of `fftw3.h`.

Table 2-8 IBM XL C compilation guidelines to use the fftw wrapper libraries

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	`cc_r (xlC_r) -O program.c -lessl -lxlf90_r -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64`

Type of ESSL library	Environment	Compilation command
SMP	32-bit integer, 64-bit pointer	cc_r (xlC_r) -O program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -lfftw3_essl -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib -I/usr/local/include -L/usr/local/lib64

Table 2-9 lists the compilation commands for gcc/g++.

Table 2-9 C/C++ compilation commands (gcc for C, g++ for C++) for ESSL[2]

Type of ESSL library	Environment	Compilation command
Serial	32-bit integer, 64-bit pointer	gcc (g++) program.c -lessl -lxlf90_r -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib
	64-bit integer, 64-bit pointer	gcc (g++) -D_ESV6464 program.c -lessl6464 -lxlf90_r -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib
SMP	32-bit integer, 64-bit pointer	gcc (g++) program.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib
	64-bit integer, 64-bit pointer	gcc (g++) -D_ESV6464 program.c -lesslsmp6464 -lxlf90_r -lxlsmp -lxlfmath -lm -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib
SMP CUDA	32-bit integer, 64-bit pointer	gcc (g++) program.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lm -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -L/opt/ibm/xlsmp/<xlsmp_version_release>/lib -L/opt/ibm/xlf/<xlf_version_release>/lib -R/opt/ibm/lib

To compile and link the FFTW Wrapper libraries, the commands that are listed in Table 2-8 on page 36 must be used. Also, use the header file `fftw3_essl.h` instead of `fftw3.h`.

[2] The ESSL SMP libraries require XL OpenMP runtime. The gcc OpenMP runtime is not compatible with XL OpenMP run time.

Table 2-10 lists the IBM XL C compilation guidelines to use the FFTW wrapper libraries, assuming that the FFTW wrappers files are installed in `/usr/local/include`.

Table 2-10 IBM XL C compilation guidelines to use the FFTW wrapper libraries

Type of ESSL library	Environment	Compilation command

Serial	32-bit integer, 64-bit pointer	`gcc (g++) program.c`
		`-lessl -lxlf90_r -lxlfmath -lfftw3_essl -lm`
		`-L/opt/ibm/xlsmp/<xlsmp_version_release>/lib`
		`-L/opt/ibm/xlf/<xlf_version_release>/lib`
		`-R/opt/ibm/lib`
		`-I/usr/local/include`
		`-L/usr/local/lib64`

SMP	32-bit integer, 64-bit pointer	`gcc (g++) program.c`
		`-lesslsmp -lxlf90_r -lxlsmp -lxlfmath`
		`-lfftw3_essl -lm`
		`-L/opt/ibm/xlsmp/<xlsmp_version_release>/lib`
		`-L/opt/ibm/xlf/<xlf_version_release>/lib`
		`-R/opt/ibm/lib`
		`-I/usr/local/include`
		`-L/usr/local/lib64`

2.3.2 ESSL example

To present the power of our ESSL API, Example 2-5 shows a C sample source code that uses ESSL to perform the well-known `dgemm` example with matrixes of dimension [20000, 20000], while measuring the execution time and performance of this calculation.

Example 2-5 ESSL C dgemm_sample.c source code for a [20,000x20,000] calculation of dgemm

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. #include <essl.h> //ESSL header for C/C++
5.
6. //Function to calculate time in milliseconds
7. long timevaldiff(struct timeval *starttime, struct timeval *finishtime)
8. {
9.   long msec;
10.  msec=(finishtime->tv_sec-starttime->tv_sec)*1000;
11.  msec+=(finishtime->tv_usec-starttime->tv_usec)/1000;
12.  return msec;
13.}
14.
15.int main()
16.{
17.  struct timeval start, end;
18.  double diff;
19.  long n, m, k;
20.  double *a, *b, *c;
21.  double rmax, rmin;
22.  double seed1, seed2, seed3;
23.  double flop;
24.
25.  //Seeds for matrix generation
```

```
26.  seed1 = 5.0;
27.  seed2 = 7.0;
28.  seed3 = 9.0;
29.
30.  //Maximum and minimum value elements of matrixes
31.  rmax = 0.5;
32.  rmin = -0.5;
33.
34.  //Size of matrixes
35.  n = 20000; m = n; k =n;
36.
37.  //Number of additions and multiplications
38.  flop = (double)(m*n*(2*(k-1)));
39.
40.  //Memory allocation
41.  a = (double*)malloc(n*k*sizeof(double));
42.  b = (double*)malloc(k*m*sizeof(double));
43.  c = (double*)malloc(n*m*sizeof(double));
44.
45.  //Matrix generation
46.  durand(&seed1,n*k,a); //DURAND are included to ESSL, not to CBLAS
47.  cblas_dscal(n*k,rmax-rmin,a,1);
48.  cblas_daxpy(n*k,1.0,&rmin,0,a,1);
49.
50.  durand(&seed2,k*m,b);
51.  cblas_dscal(k*m,rmax-rmin,b,1);
52.  cblas_daxpy(k*m,1.0,&rmin,0,b,1);
53.
54.  durand(&seed3,n*m,c);
55.  cblas_dscal(n*m,rmax-rmin,c,1);
56.  cblas_daxpy(n*m,1.0,&rmin,0,c,1);
57.
58.  //Matrix multiplication (DGEMM)
59.  gettimeofday(&start,NULL);
60.  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
61.                             m,n,k,1.0,a,n,b,k,1.0,c,n);
62.  gettimeofday(&end,NULL);
63.
64.  //Print results
65.  printf("%.3lf seconds, ",(double)timevaldiff(&start,&end)/1000);
66.  printf("%.3lf MFlops\n",flop/(double)timevaldiff(&start,&end)/1000.0);
67.
68.  //Memory deallocation
69.  free(a);
70.  free(b);
71.  free(c);
72.
73.  return 0;
74.}
```

The code uses ESSL routines that are called by CBLAS interfaces, which enable more options to specify matrix order (column-major or row-major), for example. The calculation that is performed by `dgemm` is done by using the following formula:

$$C = \alpha[A] \cdot [B] + \beta[C]$$

where, *alpha* and *beta* are real scalar values; and *A*, *B*, and *C* are matrixes of conforming shape.

The algorithm is shown in Example 2-5 on page 38.

Example 2-6 shows how to compile and run the algorithm that is shown in Example 2-5 on page 38 by using the serial version of ESSL and the XLC compiler instructions that are listed in Table 2-7 on page 36.

Example 2-6 Compilation and execution of dgemm_sample.c for serial ESSL

```
$ cc_r -O3 dgemm_sample.c -lessl -lxlf90_r -lxlfmath -L/opt/ibm/xlsmp/4.1.5/lib
-L/opt/ibm/xlf/15.1.5/lib -R/opt/ibm/lib -o dgemm_serial

$ ./dgemm_serial
1070.551 seconds, 14944.950 MFlops
```

For more information about how to use and tune this example to gain performance, see 3.2.1, “ESSL execution in multiple CPUs and GPUs” on page 94.

For more information about the new ESSL features, see the Engineering and Scientific Subroutine Library page of the IBM Knowledge Center website.

For more information about the ESSL SMP CUDA library, see the Using the ESSL SMP CUDA Library page of the IBM Knowledge Center website.

2.4 Parallel ESSL

Parallel ESSL v5.3.0 is a highly optimized mathematical subroutine library for clusters of POWER8 processor nodes. Parallel ESSL supports the single program, multiple data (SPMD) programming model that uses the Message Passing Interface (MPI) library. It also assumes that your program is using the SPMD programming model. This configuration means that all parallel tasks are identical and work on different sets of data.

Parallel ESSL supports only 32-bit integer and 64-bit pointer environment libraries, which must be used with IBM Parallel Environment (PE) Runtime Edition MPICH library.

Parallel ESSL subroutines cover following computational areas:

- Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- Level 3 PBLAS
- Linear Algebraic Equations
- Eigensystem Analysis and Singular Value Analysis
- Fourier Transforms
- Random Number Generation

Parallel ESSL uses calls of the ESSL subroutines for computational purposes.

For communication, Basic Linear Algebra Communications Subprograms (BLACS) is included, which is based on MPI.

2.4.1 Program development

During the development process of your program, the BLACS subroutines must be used. To include Parallel ESSL calls into the code of the program, complete the following steps:

1. Start the process grid by using the BLACS subroutines (`BLACS_GET` call and `BLACS_GRIDINIT` or `BLACS_GRIDMAP` after it).
2. Distribute data across process grid. Try to use different block sizes to improve performance of the program. For example, some Parallel ESSL subroutines start to use GPU for large sizes of blocks (about 2000 by 2000).
3. Call the Parallel ESSL subroutine on all processes of the BLACS process grid.
4. Aggregate results of Parallel ESSL runs from all processes.
5. Call the BLACS subroutines to clean the process grid and exit, such as `BLACS_GRIDEXIT`, `BLACS_EXIT`

Example 2-7 shows a sample Fortran dgemm implementation that uses the Parallel ESSL version of PDGEMM subroutine for a 20000 by 20000 matrix size. PDGEMM works by using the following formula:

$$C = \alpha\left[A\right] \cdot \left[B\right] + \beta\left[C\right]$$

where, *alpha* and *beta* are real scalar values*; and* *A*, *B*, and *C* are matrixes of conforming shape.

The algorithm is shown in Example 2-7.

Example 2-7 PESSL Fortran example source code pdgemm_sample.f

```
1.       program pdgemm_sample
2.       implicit none
3.       real*8, allocatable, dimension(:) :: a,b,c
4.       integer, dimension(9) :: desca,descb,descc
5.       integer m,n,k
6.       integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
7.       integer acol, bcol, ccol
8.       real*8, parameter :: alpha = 2.d0
9.       real*8, parameter :: beta  = 3.d0
10.      integer, parameter :: ia = 1, ib = 1, ic = 1
11.      integer, parameter :: ja = 1, jb = 1, jc = 1
12.
13.      integer, parameter :: nb = 200
14.! Bigger size for CUDA runs
15.!     integer, parameter :: nb = 3000
16.
17.! Initialization of process grid
18.      call blacs_pinfo(iam,np)
19.      if (np.ne.20) then
20.        print *, 'Test expects 20 nodes'
21.        stop 1
22.      else
23.        nr = 5
```

```
24.        nc = 4
25.      endif
26.      call blacs_get(0,0,icontxt)
27.      call blacs_gridinit(icontxt,'r',nr,nc)
28.      call blacs_gridinfo(icontxt,nr,nc,mr,mc)
29.
30.! Size of matrixes
31.      m = 20000
32.      n = 20000
33.      k = 20000
34.
35.! Fill parameters for PDGEMM call
36.      desca(1) = 1
37.      desca(2) = icontxt
38.      desca(3) = m
39.      desca(4) = k
40.      desca(5:6) = nb
41.      desca(7:8) = 0
42.      desca(9) = numroc(m,nb,mr,0,nr)
43.      acol = numroc(k,nb,mc,0,nc)
44.
45.      descb(1) = 1
46.      descb(2) = icontxt
47.      descb(3) = k
48.      descb(4) = n
49.      descb(5:6) = nb
50.      descb(7:8) = 0
51.      descb(9) = numroc(k,nb,mr,0,nr)
52.      bcol = numroc(n,nb,mc,0,nc)
53.
54.      descc(1) = 1
55.      descc(2) = icontxt
56.      descc(3) = m
57.      descc(4) = n
58.      descc(5:6) = nb
59.      descc(7:8) = 0
60.      descc(9) = numroc(m,nb,mr,0,nr)
61.      ccol = numroc(n,nb,mc,0,nc)
62.
63.      allocate(a(desca(9)*acol))
64.      allocate(b(descb(9)*bcol))
65.      allocate(c(descc(9)*ccol))
66.
67.! PDGEMM call
68.      a = 1.d0
69.      b = 2.d0
70.      c = 3.d0
71.      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
72.     &            beta,c,ic,jc,descc)
73.
74.! Deallocation of arrays and exit from BLACS
75.      deallocate(a)
76.      deallocate(b)
77.      deallocate(c)
78.      call blacs_gridexit(icontxt)
```

```
79.       call blacs_exit(0)
80.       end
```

For more information about the use of BLACS with the Parallel ESSL library, see the *BLACS Quick Reference Guide*.

For more information about the concept of development using the Parallel ESSL library, see this Concepts page of the IBM Knowledge Center website.

2.4.2 Using GPUs with Parallel ESSL

GPUs can be used by the local MPI tasks in the following ways within the Parallel ESSL programs:

- GPUs are not shared

  This setting means that each MPI task on a node uses unique GPUs. For this case, the local rank of MPI tasks can be used.

  Example 2-8 shows how to work with local rank by using the `MP_COMM_WORLD_LOCAL_RANK` variable. It is created from Example 2-7 on page 41 with another section that gets the local rank of the MPI task and assigns this task to a respective GPU by rank. Also, change the size of the process grid to 4 by 2 to fit your cluster, which has two nodes with 4 GPUs on each node, and uses a block size of 3000 by 3000.

  The algorithm is shown in Example 2-8.

Example 2-8 PESSL Fortran example source code for non-shared GPUs

```
1.     program pdgemm_sample_local_rank
2.       implicit none
3.       real*8, allocatable, dimension(:) :: a,b,c
4.       integer, dimension(9) :: desca,descb,descc
5.       integer m,n,k
6.       integer ids(1)
7.       integer ngpus
8.       integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
9.       integer acol, bcol, ccol
10.      real*8, parameter :: alpha = 2.d0
11.      real*8, parameter :: beta  = 3.d0
12.      integer, parameter :: ia = 1, ib = 1, ic = 1
13.      integer, parameter :: ja = 1, jb = 1, jc = 1
14.
15.      integer, parameter :: nb = 3000
16.
17.      character*8 rank
18.      integer lrank,istat
19.
20.! Initialization of process grid
21.      call blacs_pinfo(iam,np)
22.      if (np.ne.8) then
23.        print *, 'Test expects 8 nodes'
24.        stop 1
25.      else
26.        nr = 4
27.        nc = 2
28.      endif
```

```
      call blacs_get(0,0,icontxt)
      call blacs_gridinit(icontxt,'r',nr,nc)
      call blacs_gridinfo(icontxt,nr,nc,mr,mc)

! Get local rank and assign respective GPU
      call getenv('MP_COMM_WORLD_LOCAL_RANK',value=rank)
      read(rank,*,iostat=istat) lrank
      ngpus = 1
      ids(1) = lrank
      call setgpus(1,ids)

! Size of matrixes
      m = 20000
      n = 20000
      k = 20000

! Fill parameters for PDGEMM call
      desca(1) = 1
      desca(2) = icontxt
      desca(3) = m
      desca(4) = k
      desca(5:6) = nb
      desca(7:8) = 0
      desca(9) = numroc(m,nb,mr,0,nr)
      acol = numroc(k,nb,mc,0,nc)

      descb(1) = 1
      descb(2) = icontxt
      descb(3) = k
      descb(4) = n
      descb(5:6) = nb
      descb(7:8) = 0
      descb(9) = numroc(k,nb,mr,0,nr)
      bcol = numroc(n,nb,mc,0,nc)

      descc(1) = 1
      descc(2) = icontxt
      descc(3) = m
      descc(4) = n
      descc(5:6) = nb
      descc(7:8) = 0
      descc(9) = numroc(m,nb,mr,0,nr)
      ccol = numroc(n,nb,mc,0,nc)

      allocate(a(desca(9)*acol))
      allocate(b(descb(9)*bcol))
      allocate(c(descc(9)*ccol))

! PDGEMM call
      a = 1.d0
      b = 2.d0
      c = 3.d0
      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
     &            beta,c,ic,jc,descc)

```

```
84.! Deallocation of arrays and exit from BLACS
85.       deallocate(a)
86.       deallocate(b)
87.       deallocate(c)
88.       call blacs_gridexit(icontxt)
89.       call blacs_exit(0)
90.       end
```

- GPUs are shared

  This setting is used when the number of MPI tasks per node oversubscribe the GPUs. Parallel ESSL recommends the use of NVIDIA MPS, as described in 4.5.2, “CUDA Multi-Process Service” on page 156. The process allows you to use multiple MPI tasks concurrently by using GPUs.

  Note: It is possible that Parallel ESSL is unable to allocate memory on the GPU. In this case, you can reduce the number of MPI tasks per node.

  To inform Parallel ESSL which GPUs to use for MPI tasks, use the SETGPUS subroutine. Example 2-9 shows the updated version of Example 2-15 on page 61, where Parallel ESSL uses only one GPU for each MPI task, and the GPU is assigned in round-robin order. The algorithm is shown in Example 2-9.

Example 2-9 Call of SETGPUS subroutine for 1 GPU usage

```
1.     program pdgemm_sample
2.        implicit none
3.        real*8, allocatable, dimension(:) :: a,b,c
4.        integer, dimension(9) :: desca,descb,descc
5.        integer m,n,k
6.        integer ids(1)
7.        integer ngpus
8.        integer np,nr,nc,mr,mc,icontxt,numroc,iam,nnodes
9.        integer acol, bcol, ccol
10.       real*8, parameter :: alpha = 2.d0
11.       real*8, parameter :: beta  = 3.d0
12.       integer, parameter :: ia = 1, ib = 1, ic = 1
13.       integer, parameter :: ja = 1, jb = 1, jc = 1
14.
15.       integer, parameter :: nb = 3000
16.
17.! Initialization of process grid
18.       call blacs_pinfo(iam,np)
19.       if (np.ne.20) then
20.         print *, 'Test expects 20 nodes'
21.         stop 1
22.       else
23.         nr = 5
24.         nc = 4
25.       endif
26.       ngpus = 1
27.       ids(1) = mod(iam,4)
28.       call setgpus(ngpus,ids)
29.       call blacs_get(0,0,icontxt)
30.       call blacs_gridinit(icontxt,'r',nr,nc)
31.       call blacs_gridinfo(icontxt,nr,nc,mr,mc)
```

```
32.
33.! Size of matrixes
34.      m = 20000
35.      n = 20000
36.      k = 20000
37.
38.! Fill parameters for PDGEMM call
39.      desca(1) = 1
40.      desca(2) = icontxt
41.      desca(3) = m
42.      desca(4) = k
43.      desca(5:6) = nb
44.      desca(7:8) = 0
45.      desca(9) = numroc(m,nb,mr,0,nr)
46.      acol = numroc(k,nb,mc,0,nc)
47.
48.      descb(1) = 1
49.      descb(2) = icontxt
50.      descb(3) = k
51.      descb(4) = n
52.      descb(5:6) = nb
53.      descb(7:8) = 0
54.      descb(9) = numroc(k,nb,mr,0,nr)
55.      bcol = numroc(n,nb,mc,0,nc)
56.
57.      descc(1) = 1
58.      descc(2) = icontxt
59.      descc(3) = m
60.      descc(4) = n
61.      descc(5:6) = nb
62.      descc(7:8) = 0
63.      descc(9) = numroc(m,nb,mr,0,nr)
64.      ccol = numroc(n,nb,mc,0,nc)
65.
66.      allocate(a(desca(9)*acol))
67.      allocate(b(descb(9)*bcol))
68.      allocate(c(descc(9)*ccol))
69.
70.! PDGEMM call
71.      a = 1.d0
72.      b = 2.d0
73.      c = 3.d0
74.      call pdgemm('N','N',m,n,k,alpha,a,ia,ja,desca,b,ib,jb,descb,
75.     &            beta,c,ic,jc,descc)
76.
77.! Deallocation of arrays and exit from BLACS
78.      deallocate(a)
79.      deallocate(b)
80.      deallocate(c)
81.      call blacs_gridexit(icontxt)
82.      call blacs_exit(0)
83.      end
```

Example 2-8 on page 43 explicitly states to use two GPUs per MPI task. To change to this configuration, change the following lines:

```
ngpus = 1
ids(1) = mod(iam,4)
```

to the following lines:

```
ngpus = 2
ids(1) = mod(iam,2)*2
ids(2) = mod(iam,2)*2 + 1
```

2.4.3 Compilation

The Parallel ESSL subroutines can be called from 64-bit-environment application programs, which are written in Fortran, C, and C++. Compilation commands of source code for different type of Parallel ESSL library (SMP and SMP CUDA) using MPICH libraries are described in Table 2-11, Table 2-12, Table 2-13 on page 48, Table 2-14 on page 48 and Table 2-15 on page 49.

You do not need to modify your Fortran compilation procedures when Parallel ESSL for Linux is used. When linking and running your program, you must modify your job procedures to set up the necessary libraries. If you are accessing Parallel ESSL for Linux from a Fortran program, you can compile and link by using the commands that are listed in Table 2-11.

Table 2-11 Fortran compilation commands for PESSL using Spectrum MPI libraries

Type of PESSL library	Compilation command
64-bit SMP	`mpifort -O program.f -lesslsmp -lpesslsmp -lblacssmp -lxlsmp`
64-bit SMP CUDA	`mpifort -O xyz.f -lesslsmpcuda -lpesslsmp -lblacssmp -lxlsmp -lcublas -lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`

The Parallel ESSL for Linux header files `pessl.h` and `Cblacs.h`, which are used for C and C++ programs, are installed in the `/usr/include` directory. When linking and running your program, you must modify your job procedures to set up the necessary libraries. If you are accessing Parallel ESSL for Linux from a C program, you can compile and link by using the commands that are listed in Table 2-12.

Table 2-12 C program compile and link commands for use with Spectrum MPI

Type of PESSL library	Compilation command
64-bit SMP	`mpicc -O program.c -lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp -lxlfmath -lmpi_mpifh -L/opt/ibm/xlsmp/xlsmp_version.release/lib -L/opt/ibm/xlf/xlf_version.release/lib -R/opt/ibm/lib`

Type of PESSL library	Compilation command

64-bit SMP CUDA	`mpicc -O program.c`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp`
	`-lxlfmath -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To use `gcc` as your C compiler, you can compile and link by using the commands that are listed in Table 2-13.

Table 2-13 C program gcc compile and link commands for use with Spectrum MPI

Type of PESSL library	Compilation command

64-bit SMP	`export OMPI_CC=gcc`
	
	`mpicc -O xyz.c -lesslsmp -lpesslsmp -lblacssmp -lxlf90_r`
	`-lxl -lxlsmp -lxlfmath -lm -lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`export OMPI_CC=gcc`
	
	`mpicc -O program.c`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To develop C++ programs with PEESL, use the commands that are listed in Table 2-14.

Table 2-14 C++ program compile and link commands for use with Spectrum MPI

Type of PESSL library	Compilation command

64-bit SMP	`mpiCC -O program.C`
	`-lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp -lxlfmath`
	`-lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`mpiCC -O program.C`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxlsmp`
	`-lxlfmath -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

To use g++ as your C++ compiler, you can compile and link by using the commands that are listed in Table 2-15.

Table 2-15 C++ program g++ compile and link commands for use with Spectrum MPI

Type of PESSL library	Compilation command

64-bit SMP	`export OMPI_CXX=g++`
	
	`mpiCC -O program.C`
	`-lesslsmp -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

64-bit SMP CUDA	`export OMPI_CXX=g++`
	
	`mpiCC -O program.C`
	`-lesslsmpcuda -lpesslsmp -lblacssmp -lxlf90_r -lxl -lxlsmp`
	`-lxlfmath -lm -lmpi_mpifh -lcublas -lcudart`
	`-L/usr/local/cuda/lib64`
	`-R/usr/local/cuda/lib64`
	`-L/opt/ibm/xlsmp/xlsmp_version.release/lib`
	`-L/opt/ibm/xlf/xlf_version.release/lib`
	`-R/opt/ibm/lib`

For more information about all of the features of our new Parallel ESSL library, see the Parallel Engineering and Scientific Subroutine Library V5.3 Documentation page of the IBM Knowledge Center website.

2.5 Using POWER8 vectorization

The single-instruction, multiple-data (SIMD) instructions are building blocks that are used to use parallelism at CPU level. The Power architecture implements SIMD through the VMX and VSX technologies, which are specified in the Power Instruction Set Architecture (Power ISA) version 2.07 for POWER8 generation processor.

Techniques to use data parallelism through SIMD instructions are often called *vectorization*. They can be used by the compiler in the form of auto-vectorization transformations or made available to applications as application programming interface (API).

GNU GCC and IBM XL compilers provide vector APIs that are based on the AltiVec specification for C, C++, and Fortran. Their API is composed of built-in functions (also known as intrinsics), defined vector types, and extensions to the programming language semantics. Each compiler vector implementation is explained in the following sections.

For more information about the auto-vectorization features of the GCC and XL compilers, see 2.1, “Compiler options” on page 24.

2.5.1 AltiVec operations with GNU GCC

GNU GCC provides a modified API that enables the use of AltiVec operations for the PowerPC family of processors. This interface is made available by including `<altivec.h>` in the source code and using `-maltivec` and `-mabi=altivec` compiling functions. This feature is also made available if the code is compiled with `-mvsx`, because this option enables `-maltivec` with more features that use VSX instructions.

For more information about the AltiVec API specification, see the “PowerPC AltiVec Built-in Functions” section of *Using the GNU Compiler Collection (GCC)* manual.

The following features are implemented:

- Add the keywords `__vector`, `vector`, `__pixel`, `pixel`, `__bool`, and `bool`. The `vector`, `pixel`, and `bool` keywords are implemented as context-sensitive, predefined macros that are recognized only when used in C-type declaration statements. In C++ applications, they can be undefined for compatibility.
- Unlike the AltiVec specification, the GNU/GCC implementation does not allow a `typedef` name as a type identifier. You must use the actual `__vector` keyword; for example, `typedef signed short int16; __vector int16 myvector`.
- Vector data types are aligned on a 16-byte boundary.
- Aggregates (structures and arrays) and unions that contain vector types must be aligned on 16-byte boundaries.
- Load or store to unaligned memory must be carried out explicitly by one of the `vec_ld`, `vec_ldl`, `vec_st`, or `vec_stl` operations. However, the load of an array of components does not need to be aligned, but it must be accessed with attention to its alignment, which is often carried out with a combination of `vec_lvsr`, `vec_lvsl`, and `vec_perm` operations.
- The use of `sizeof()` for vector data types (or pointers) returns 16, for 16 bytes.
- Assignment operation (`a = b`) is allowed only if both sides have the same vector types.
- Address operation `&p` is valid if `a` is `p` vector type.
- The usual pointer arithmetic can be performed on vector type pointer `p`, in particular:
  - `p+1` is a pointer to the next vector after `p`.
  - Pointer dereference (`*p`) implies either a 128-bit vector load from or store to the address that is obtained by clearing the low-order bits of `p`.

C arithmetic and logical operators (+, -, *, /, unary minus, ^, |, &, ~, and %), shifting operators (<< and >>), and comparison operators (==, !=, <, <=, >, and >=) can be used on these types. The compiler generates the correct SIMD instructions for the hardware.

Table 2-16 shows vector data type extensions as implemented by GCC. Vector types are signed by default, unless an otherwise `unsigned` keyword is specified. The only exception is `vector char`, which is unsigned by default. The hardware does not have instructions for supporting `vector long long` and `vector bool long long` types, but they can be used for float-point/integer conversions.

Table 2-16 Vector types as implemented by GCC

Vector types	Description
vector char	Vector of 16 8-bit char
vector bool	Vector of 16 8-bit unsigned char
vector short	Vector of eight 16-bit short

Vector types	Description
vector bool short	Vector of eight 16-bit unsigned short
vector pixel	Vector of eight 16-bit unsigned short
vector int	Vector of four 32-bit integer
vector bool int	Vector of four 32-bit integer
vector float	Vector of four 32-bit float
vector double	Vector of two 64-bit double. Requires compile with `-mvsx`
vector long	Vector of two 64-bit signed integer. It is implemented in 64-bit mode only. Requires compile with `-mvsx`
vector long long	Vector of two 64-bit signed integer
vector bool long	Vector of two 64-bit signed integer

In addition to vector operations, GCC has a built-in function for cryptographic instructions that operate in vector types. For more information about the implementation and a comprehensive list of built-in functions, see the *PowerPC AltiVec section* in GNU GCC.

2.5.2 AltiVec operations with IBM XL

The IBM XL compiler family provides an implementation of AltiVec APIs through feature extensions for C and C++. Fortran extensions for vector support are also available.

XL C/C++

To use vector extensions, the application must be compiled with `-mcpu=pwr8`, and `-qaltivec` must be in effect.

The XL C/C++ implementation defines the `vector` (or alternatively, `__vector`) keywords that are used in the declaration of vector types.

Similar to GCC implementation, XL C/C++ allows unary, binary, and relational operators to be applied to vector types. It implements all data types that are listed in Table 2-16 on page 50.

The indirection operator, asterisk (`*`), is extended to handle pointer to vector types. Pointer arithmetic is also defined so that a pointer to the following vector can be expressed as `v+ 1`.

Vector types can be cast to other vector types (but not allowed to a scalar). The casting *does not* represent a conversion; therefore, it is subject to changes in element value. Casting between vector and scalar pointers is also allowed if memory is maintained on 16-byte alignment.

For more information about XL C/C++ 13.1.5 vector support, vector initialization, and the `vec_step` operator, see the *Extensions for vector processing support* manual.

For more information about built-in functions for vector operations is available at XL C and C++ 1.3.1.5 manual at the Vector built-in functions page of the IBM Knowledge Center website.

XL Fortran

To use vector extensions, the application must comply with `-qarch=pwr8`.

The XL Fortran language extension defines the `VECTOR` keyword, which is used to declare 16-byte vector entities that can hold `PIXEL`, `UNSIGNED`, `INTEGER`, and `REAL` type elements. `PIXEL` (2 bytes) and `UNSIGNED` (unsigned integer) types also are extensions to the language. They must be used within vectors only.

Vectors are automatically aligned to 16 bytes, but exceptions apply. For more information about vector types on XL Fortran 15.1.5, see the Vector (IBM extension) for Version 15.1.5 page of the IBM Knowledge Center website.

For the list of vector intrinsic procedures available with XL Fortran 15.1.3, see the Vector instrinsic procedures (IBM extension) for Version 15.1.5 page of the IBM Knowledge Center website.

Example 2-10 uses the Fortran XL vector library to perform the following calculation:

$$C = \alpha[A] + [B]$$

where, *alpha* and *beta* are real scalar values; and *A*, *B*, and *C* are matrixes of conforming shapes.

Lines 11 - 14 in Example 2-10 show declaration of vectors with 16 elements of 8 bytes of real types. Called methods `vec_xld2` (load), `vec_permi` (permuting), and `vec_madd` (fused) are multiply add SIMD operations that are applied to the vector types.

Example 2-10 Fortran program that demonstrates use of XL compiler vectors

```
 1      SUBROUTINE VSX_TEST
 2      implicit none
 3
 4      real*8, allocatable :: A(:), B(:), C(:), CT(:)
 5      real*8 alpha
 6      integer*8 max_size, ierr
 7      integer*8 i, j, it
 8      integer*8 ia, ialign
 9      integer   n, nalign
10
11      vector(real*8) va1, va2, va3
12      vector(real*8) vb1, vb2
13      vector(real*8) vc1, vc2
14      vector(real*8) valpha
15
16      max_size = 2000
17      alpha = 2.0d0
18
19      ierr = 0
20      allocate(A(max_size),stat=ierr)
21      allocate(B(max_size),stat=ierr)
22      allocate(C(max_size),stat=ierr)
23      allocate(CT(max_size),stat=ierr)
24      if (ierr .ne. 0) then
25        write(*,*) 'Allocation failed'
26        stop 1
27      endif
```

```
28
29      do i = 1, max_size
30        a(i) = 1.0d0*i
31        b(i) = -1.0d0*i
32        ct(i) = alpha*a(i) + b(i)
33      enddo
34
35      ia     = LOC(A)
36      ialign = IAND(ia, 15_8)
37      nalign = MOD(RSHIFT(16_8-ialign,3_8),7_8) + 2
38
39 !     Compute Head
40      j = 1
41      do i = 1, nalign
42        C(j) = B(j) + alpha * A(j)
43        j = j + 1
44      enddo
45
46      n = max_size - nalign - 4
47      it = rshift(n, 2)
48
49      va1 = vec_xld2( -8, A(j))
50      va2 = vec_xld2(  8, A(j))
51      va3 = vec_xld2( 24, A(j))
52
53      va1 = vec_permi( va1, va2, 2)
54      va2 = vec_permi( va2, va3, 2)
55
56      vb1 = vec_xld2(  0, B(j))
57      vb2 = vec_xld2( 16, B(j))
58
59      do i = 1, it-1
60        vc1 = vec_madd( valpha, va1, vb1)
61        vc2 = vec_madd( valpha, va2, vb2)
62
63        va1 = va3
64        va2 = vec_xld2( 40, A(j))
65        va3 = vec_xld2( 56, A(j))
66
67        va1 = vec_permi( va1, va2, 2)
68        va2 = vec_permi( va2, va3, 2)
69
70        call vec_xstd2( va1,  0, C(j))
71        call vec_xstd2( va2, 16, C(j))
72
73        vb1 = vec_xld2( 32, B(j))
74        vb1 = vec_xld2( 48, B(j))
75
76        j = j + 4
77      enddo
78
79      vc1 = vec_madd( valpha, va1, vb1)
80      vc2 = vec_madd( valpha, va2, vb2)
81
82      call vec_xstd2( va1,  0, C(j))
```

```
83      call vec_xstd2( va2, 16, C(j))
84
85 !     Compute Tail
86      do i = j, max_size
87        C(i) = B(i) + alpha * A(i)
88      enddo
89
90      do i = 1, 10
91        write(*,*) C(i), CT(i)
92      enddo
93
94      END SUBROUTINE VSX_TEST
```

2.6 Development models

The high-performance computing (HPC) solution that is proposed in this book contains a software stack that allows for the development of C, C++, and Fortran applications by using different parallel programming models. In this context, applications can be implemented by using *pure models* as MPI, OpenMP, CUDA, OpenACC, PAMI, or OpenSHMEM, or by using some combinations of these models (also known as *hybrid models*).

This section describes aspects of the IBM PE, compilers (GNU and IBM XL families), libraries, and toolkits that developers can use to implement applications on pure or hybrid parallel programming models. It does not describe how those applications can be run with IBM PE (for more information, see 3.3, “Using IBM Parallel Environment v2.3” on page 104).

2.6.1 OpenMP programs with the IBM Parallel Environment

The OpenMP applies a shared memory parallel programming model of development. It is a multi-platform directive-based API that is available to many languages, including C, C++, and Fortran.

The development and execution of OpenMP applications are fully supported by the IBM PE Runtime. OpenMP parallelism uses directives that use what is known as *shared memory parallelism*, considering it defines various types of parallel regions. The parallel regions can include iterative and non-iterative segments of code.

The use of the `#pragma omp` directives ordinarily can occasionally be distinguished into the following general categories:

- Defines parallel regions in which work is done by threads in parallel (`#pragma omp parallel`). Most of the OpenMP directives statically or dynamically bind to an enclosing parallel region.
- Defines how work is distributed or shared across the threads in a parallel region (`#pragma omp sections`, `#pragma omp for`, `#pragma omp single`, and `#pragma omp task`).
- Controls synchronization among threads (`#pragma omp atomic`, `#pragma omp master`, `#pragma omp barrier`, `#pragma omp critical`, `#pragma omp flush`, and `#pragma omp ordered`).
- Defines the scope of data visibility across parallel regions within the same thread (`#pragma omp threadprivate`).
- Controls synchronization (`#pragma omp taskwait` and `#pragma omp barrier`).

- Controls data or computation that is on another computing device.

Also, you can specify visibility context for selected data variables by using some OpenMP clauses. Scope attribute clauses are listed in Table 2-17.

Table 2-17 Openmp variable scope on IBM Parallel Environment

Data scope attribute clause	Description
private	The private clause declares the variables in the list to be private to each thread in a team.
firstprivate	The firstprivate clause declares the variables in the list to be private to each thread in a team with the value that variable had before the parallel section.
lastprivate	Similar to the firstprivate, the lastprivate updates the variable in the from outer scope of the parallel region with the last value it had in the parallel region.
shared	The shared clause declares the variables in the list to be shared among all the threads in a team. All threads within a team access the same storage area for shared variables.
reduction	The reduction clause performs a reduction on the scalar variables that appear in the list, with a specified operation.
default	The default clause allows the user to affect the data-sharing attribute of the variables that appeared in the parallel construct.

To demonstrate some of the clauses that are listed in Table 2-17, Example 2-11 shows the following simple parallel algorithm that can be used to calculate the dot product of two vectors:

$$\left(A \cdot B = \sum_{i=1}^{N} a_i b_i \right), A = (a_1, a_2, \ldots, a_N); B = (b_1, b_2, \ldots, b_N)$$

where A and B are vectors of conforming shape. The algorithm is shown in Example 2-11.

Example 2-11 ESSL C dgemm_sample.c source code for a [20,000x20,000] calculation of dgemm

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <stdint.h>
4. #include <time.h>
5. #include <sys/time.h>
6. #include <omp.h>
7.
8. void print_v(double*,int);
9.
10.int main(int argc, char* argv[])
11.{
12.        if(argc!=4) { printf("No verbose || size of vectors || or number of
   threads provided\nAborting...\n"); return -1; }
13.
14.        int i, verbose, N, n_threads;
```

```
15.        double *a, *b, sum;
16.
17.        sum = 0.0;
18.        verbose = atoi(argv[1]);
19.        N = atoi(argv[2]);
20.        n_threads = atoi(argv[3]);
21.        a = (double*) malloc(sizeof(double)*N);
22.        b = (double*) malloc(sizeof(double)*N);
23.
24.        for(i=0;i<N;i++)
25.                a[i]=b[i]=(double)(i+1);
26.
27.        #pragma omp parallel for default(none) firstprivate(N) private(i)
   shared(a,b) reduction(+:sum) num_threads(n_threads)
28.        for(i=0; i<N; i++)
29.                sum += a[i] * b[i];
30.
31.if(verbose) { printf("[A] = "); print_v(a,N); printf("[B] = "); print_v(b,N); }
32.
33.        printf("[A]*[B] = %.2f\n", sum);
34.
35.        return 0;
36.}
37.
38.void print_v(double *v, int N)
39.{
40.        int i; printf("[");
41.        for(i=0;i<N;i++)
42.        {
43.                if( i!=(N-1) )
44.                        printf("%.2f ",v[i]);
45.                else
46.                        printf("%.2f]\n",v[i]);
47.        }
48.}
```

Line 27 from the source code that is shown in Example 2-11 on page 55 is of our interest in this section. The `default(none)` is used to make the compiler remind us that it must know the scope of each variable in the parallel section.

The `firstprivate(N)` is used because we want each thread to not only have its own `N` value, but the previous value of `N`. However, the `private(i)` is there because we want the loop variable to be thread independent and we do not need its previous value. Also, the shared variables are the data to be accessed by each thread independently because of the private `i` index. The `reduction(sum)` is there because we want to add all independent products in the sum variable after the loop. The `num_thread(n_threads)` is there to define the number of threads to run our `#pragma omp` parallel section.

Example 2-12 shows how to compile the source code.

Example 2-12 Compiling an OpenMP program with XL

```
$ /opt/ibm/xlC/13.1.5/bin/cc_r -O3 -Wall -o dot_xlc dot_xlc.c -qsmp=omp -mcpu=pwr8
-mtune=pwr8
```

From a compiler perspective, XL C/C++ 13.1.5 and Fortran 15.1.5 provide full support to OpenMP API version 3.1, and partial support to versions 4.0 and 4.5. Similarly, the GNU GCC compiler that is provided by the Linux distribution is based on OpenMP API version 3.1.

For more information about XL 13.1.5 and OpenMP, see *Optimization and Programming Guide for Little Endian Distributions*.

For more information about OpenMP examples and the directives and their applications, see *OpenMP Application Programming Interface: Examples*.

For more information about offloading code to GPU, see the Offloading Support in GCC page of the GCC Wiki.

2.6.2 CUDA C programs with the NVIDIA CUDA Toolkit

The development of parallel programs that use the General Purpose GPU (GPGPU) model is provided in the IBM Power System S822LC with support of the NIVIDIA CUDA Toolkit 8.0 for Linux on POWER8 Little Endian.

This section describes relevant aspects to the CUDA development in the proposed solution. This section also describes characteristics of the NVIDIA P100 GPU, integration between compilers, and availability of libraries. For more information about the NVIDIA CUDA development model and resources, see the NVIDIA CUDA Zone website.

Understanding the NVIDIA P100 GPU CUDA capabilities

The NVIDIA Tesla P100 is a dual-GPU product with two attached GPUs, each implementing the Pascal P100/SXM2 CUDA compute architecture. As such, the host operating system recognizes four GPUs in an IBM S822LC system that is fully populated with two Tesla P100 cards.

The Pascal P100 architecture delivers CUDA compute capability version 6.0.

Table 2-18 lists the available resources (per GPU) and the limitations for the CUDA C++ applications that are found with the deviceQuery script that is available with the sample codes from NVIDIA.

Table 2-18 CUDA available resources per GPU

GPU resources	Value
Total of global memory	16 GB
Total amount of constant memory	64 KB
Shared memory per block	48 KB
Stream Multiprocessors (SMs)	56
Maximum warps per SM	64
Threads per Warps	32
Maximum threads per SM	2048
Maximum thread blocks per SM	32
Maximum threads per block	1024
Maximum grid size	(x, y, z) = (2147483647, 65535, 65535)

GPU resources	Value
Max dimension size of a thread block (x,y,z)	(x,y,z) = (1024, 1024, 64)
Maximum Texture Dimension Size (x,y,z)	1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
Maximum layered texture Size and number of layers	1D=(32768), 2048 layers, 2D=(32768, 32768), 2048 layers

The GP100 GPU that is included in the Tesla P100 has the following innovative features:

- Extreme performance: Powering HPC, deep learning, and many more GPU Computing areas
- NVLink: NVIDIA's new high-speed, high-bandwidth interconnect for maximum application scalability
- HBM2: Fastest, high-capacity, efficient stacked GPU memory architecture
- Unified memory and compute preemption: Significantly improved programming model
- 16nm FinFET: Enables more features, higher performance, and improved power efficiency

Note: GPUDirect is not supported on the IBM Power Systems servers at the time this writing, although the IBM Parallel Environment provides an CUDA-Aware API that supports GPU to GPU buffer transfers. For more information, see 2.6.6, "Hybrid MPI and CUDA programs with IBM Parallel Environment" on page 75.

NVIDIA nvcc compiler

The NVIDIA `nvcc` compiler driver is responsible for generating the final executable file, which is a combination of host (CPU) and device (GPU) codes.

IBM XL and GNU GCC compilers can be used to generate the host code and their flags. As described in 2.1, "Compiler options" on page 24, this process produces an optimized code to run in the IBM POWER8 processor. By default, `nvcc` uses the GNU GCC, unless the **`-ccbin`** flag is passed to set another back-end compiler.

Note: The `nvcc` compiler uses the GNU GCC C++ compiler by default to generate the host code. If wanted, the IBM XL C++ compiler instead uses the `-ccbin xlC` option.

The NVIDIA Tesla P100 GPUs are built on Pascal GP100 architecture, which provides CUDA compute capability v6.0. Use the `-gencode` compiler option to set the virtual architecture and binary compatibility; for example, `-gencode arch=compute_60,code=sm_60` generates code that is compatible with the Pascal GP100 architecture (virtual architecture 6.0).

Example 2-13 shows a CUDA program that prints to standard output the index values for 2D thread blocks. The executable file is built with `nvcc` that uses xlC as the host compiler (`-ccbin xlC` option). Parameters to xlC are passed with `-Xcompiler` option.

Example 2-13 CUDA C program and nvcc compilation

```
$ cat -n hello.cu
1. #include<cuda_runtime.h>
2. #include<stdio.h>
3.
4.   __global__ void helloKernel() {
5.       int tidx = threadIdx.x;
```

```
6.       int tidy = threadIdx.y;
7.       int blockidx = blockIdx.x;
8.       int blockidy = blockIdx.y;
9.
10.      printf("I am CUDA thread (%d, %d) in block (%d, %d)\n",
11.        tidx, tidy, blockidx, blockidy);
12.  };
13.
14.int main(int argc, char* argv[]) {
15.  dim3 block(16,16);
16.  dim3 grid(2,2);
17.  helloKernel<<<grid, block>>>();
18.  cudaDeviceSynchronize();
19.  return 0;
20.}
```

```
$ nvcc -ccbin xlC -Xcompiler="-qarch=pwr8 -qtune=pwr8 -qhot -O3" -gencode
arch=compute_60,code=sm_60 hello.cu -o helloCUDA
$ ./helloCUDA
I am CUDA thread (0, 0) in block (1, 0)
I am CUDA thread (1, 0) in block (1, 0)
I am CUDA thread (2, 0) in block (1, 0)
I am CUDA thread (3, 0) in block (1, 0)
I am CUDA thread (4, 0) in block (1, 0)
I am CUDA thread (5, 0) in block (1, 0)
<... Output omitted ...>
```

The `nvcc` also supports cross-compilation of CUDA C and C++ code to PowerPC 64-bit Little-Endian (ppc64le).

Note: The support for cross-compilation for POWER8 Little-Endian was introduced in CUDA Toolkit 7.5.

For more information about the nvcc compilation process and options, see the NVIDIA CUDA Compiler NVCC page of the CUDA Toolkit Documentation website.

CUDA libraries

Accompanying the CUDA Toolkit for Linux are the following mathematical utility and scientific libraries that use the GPU to improve performance and scalability:

- `cuBLAS`: Basic Linear Algebra Subroutines
- `cuFFT`: Fast Fourier Transforms
- `cuRAND`: Random Number Generation
- `cuSOLVER`: Dense and Sparse Direct Linear Solvers and Eigen Solvers
- `cuSPARSE`: Sparse matrix routines
- `Thrust`: Parallel Algorithm and data structures

Example 2-14 shows the use of the following cuBLAS API to calculate the dot product of two vectors.

$$\left(A \cdot B = \sum_{i=1}^{N} a_i b_i \right), A = (a_1, a_2, \ldots, a_N); B = (b_1, b_2, \ldots, b_N)$$

where A and B are vectors of conforming shape.

This algorithm uses IBM XLC++ (xlC) to generate the host code and compile with `-lcublas` flag that links dynamically with the `cuBLAS` library.

Example 2-14 Sample code for CUDA C using cuBLAS library

```
1. #include<cublas_v2.h>
2. #include<cuda_runtime.h>
3. #include<cuda_runtime_api.h>
4. #include<stdlib.h>
5. #include<stdio.h>
6.
7. #define N 10000
8.
9. int main(int argc, char* argv[]) {
10.         int const VEC_SIZE = N*sizeof(double);
11.         double* h_vec_A = (double*) malloc(VEC_SIZE);
12.         double* h_vec_B = (double*) malloc(VEC_SIZE);
13.
14.         double *d_vec_A, *d_vec_B, result;
15.         cublasStatus_t status;
16.         cublasHandle_t handler;
17.         cudaError_t error;
18.         // Initialize with random numbers between 0-1
19.         int i;
20.         for(i=0; i<N; i++) {
21.                 h_vec_A[i] = (double) (rand() % 100000)/100000.0;
22.                 h_vec_B[i] = (double) (rand() % 100000)/100000.0;
23.         }
24.cudaMalloc((void **)&d_vec_A, VEC_SIZE);
25.         cudaMalloc((void **)&d_vec_B, VEC_SIZE);
26.         cudaMemcpy(d_vec_A, h_vec_A, VEC_SIZE ,cudaMemcpyHostToDevice);
27.         cudaMemcpy(d_vec_B, h_vec_B, VEC_SIZE ,cudaMemcpyHostToDevice);
28.
29.         // Initialize cuBLAS
30.         status = cublasCreate(&handler);
31.         // Calculate DOT product
32.         status = cublasDdot(handler, N , d_vec_A, 1, d_vec_B, 1, &result);
33.         if(status != CUBLAS_STATUS_SUCCESS) {
34.                 printf("Program failed to calculate DOT product\n");
35.                 return EXIT_FAILURE;
36.         }
37.         printf("The DOT product is: %G\n", result);
38.
39.         // Tear down cuBLAS
40.         status = cublasDestroy(handler);
41.         return EXIT_SUCCESS;
42.}
```

The source code as shown in Example 2-14 on page 60 is built and run as shown in Example 2-15.

Example 2-15 Building and running source code an cuBLAS sample code

```
$ make
Building file: ../main.c
Invoking: NVCC Compiler
/usr/local/cuda-8.0/bin/nvcc -I/usr/local/cuda-8.0/include/ -G -g -O0 -ccbin xlC
-Xcompiler -qtune=pwr8 -Xcompiler -qhot -Xcompiler -O3 -Xcompiler -qarch=pwr8
-gencode arch=compute_60,code=sm_60 -m64 -odir "." -M -o "main.d" "../main.c"

/usr/local/cuda-8.0/bin/nvcc -I/usr/local/cuda-8.0/include/ -G -g -O0 -ccbin xlC
-Xcompiler -qtune=pwr8 -Xcompiler -qhot -Xcompiler -O3 -Xcompiler -qarch=pwr8
--compile -m64  -x c -o  "main.o" "../main.c"
Finished building: ../main.c

Building target: cudaBLAS
Invoking: NVCC Linker
/usr/local/cuda-8.0/bin/nvcc --cudart static -ccbin xlC
--relocatable-device-code=false -gencode arch=compute_60,code=compute_60 -gencode
arch=compute_60,code=sm_60 -m64 -link -o  "cudaBLAS"  ./main.o   -lcublas
Finished building target: cudaBLAS

$ LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-8.0/lib64/
$ ./cudaBLAS
The DOT product is: 2500.22
```

2.6.3 OpenACC

As an option that is provided by the OpenPower foundation, IBM worked with NVIDIA to enable the use of PGI compilers in POWER8 systems. As with the IBM XL, this compiler also can use most of the features that NVLINK technology provides to offload code to the Pascal GPUs.

The offload through the PGI compiler is performed by the OpenACC model, which not only was developed similar to OpenMP, but it was developed to ease the programming of heterogeneous CPU/GPU hybrid software for programmers. By using simple clauses, such as `#pragma acc kernels{}`, the programmer can indicate loops where parallelism might be found, while the compiler creates CUDA kernels under the hood for the programmer.

In this section, we consider the integral of the following curve that leads to the value of pi, if performed in the closed interval [0,1] as shown in Table 2-7 on page 36:

$$\int_0^1 \frac{4}{1+x^2} dx = \pi \approx 3,1416$$

The simple addition of the `#pragma acc kernels{}` in line 29 of Example 2-16 resulted in a significant performance increase.

Example 2-16 Parallel code for integration using openacc

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <stdint.h>
4. #include <time.h>
5. #include <sys/time.h>
6. #include <omp.h>
7.
8. int32_t timer(struct timeval *, struct timeval *, struct timeval *);
9.
10.int32_t main(int32_t argc, char* argv[])
11.{
12.        if(argc!=4)
13.        {
14.                printf("No number of steps OR number of blocks OR number of
   threads were provided\nAborting...\n");
15.                return -1;
16.        }
17.
18.        uint32_t i, N, n_blocks, n_threads;
19.        double x, pi, step, sum;
20.        struct  timeval start_princ, finish_princ, diff_princ;
21.
22.        sum = 0.0;
23.        N = atol(argv[1]);
24.        n_blocks = atol(argv[2]);
25.        n_threads = atol(argv[3]);
26.        step = 1.0 / ((double) N);
27.
28.        gettimeofday(&start_princ,NULL);
29.                #pragma acc kernels
30.                for(i=0; i<N; i++)
31.                {
32.                        x = (i + 0.5) * step;
33.                        sum += 4.0 / ( 1.0 + x * x);
34.                }
35.                pi = step * sum * n_blocks * n_threads;
36.        gettimeofday(&finish_princ,NULL);
37.
38.        timer(&diff_princ, &finish_princ, &start_princ);
39.        printf("PI: %.15f\nExecution Time   = %ld.%06ld s\n", pi,
   diff_princ.tv_sec, diff_princ.tv_usec);
40.        fprintf(stderr,"%ld.%06ld\n", diff_princ.tv_sec, diff_princ.tv_usec);
41.
42.        return 0;
43.}
44.
45.int32_t timer(struct timeval *result, struct timeval *t2, struct timeval *t1)
46.{
47.        int32_t diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec +
   1000000 * t1->tv_sec);
48.        result->tv_sec = diff / 1000000;
```

```
49.        result->tv_usec = diff % 1000000;
50.        return (diff<0);
51.}
```

Example 2-16 on page 62 is an excellent starting ground to parallelize software through OpenACC. However, to achieve better performance, the programmer must fine-tune the `#pragma acc` calls. Therefore, consider the effect of sharing the integral loop among multiple blocks and threads as shown in Example 2-17.

Example 2-17 OpenACC parallelization in n_blocks with n_threads each

```
--- acc_pi_simple.c     2016-11-30 14:22:07.842632526 -0500
+++ acc_pi.c    2016-11-30 14:33:19.972673136 -0500
@@ -26,7 +26,7 @@ int32_t main(int32_t argc, char* argv[])
        step = 1.0 / ((double) N);

         gettimeofday(&start_princ,NULL);
-             #pragma acc kernels
+            #pragma acc parallel loop reduction(+:sum) device_type(nvidia)
vector_length(n_threads) gang worker num_workers(n_blocks)
              for(i=0; i<N; i++)
              {
                      x = (i + 0.5) * step;
```

In Example 2-17, we are dividing the for loop between blocks of several threads. Also, we must add a reduction clause to add all individual computation between the blocks at the end of the execution.

The compilation command for our configuration that uses a Tesla P100 is shown in Example 2-18. The `-acc` tells the compiler to process the source recognizing `#pragma acc` directives. At the same time, the `-Minfo` tells the compiler to share information about the optimization during the compilation process. Also, all stands for `accel,inline,ipa,loop,lre,mp,opt,par,unified,vect` and the intensity asks for the computing loop information.

Example 2-18 OpenACC compilation example

```
$ pgcc -acc -Minfo=all,intensity -ta=tesla:cc60 -o acc_pi acc_pi.c -lm
main:
     29, Accelerator kernel generated
         Generating Tesla code
         29, Generating reduction(+:sum)
        30, #pragma acc loop gang, vector(n_threads), worker(n_blocks) blockIdx.x
threadIdx.x threadIdx.y */
```

In addition, the `-ta` flag chooses the target accelerator, which can be set to the host CPU through the option multicore, and therefore run in our 192 cores. However, in our example, it is set to our Tesla P100 GPU that has compute capacity 60. Much information is available by using the **`pgaccelinfo`** command. To find the correct `cc gpu` flag, run the command that is shown in Example 2-19.

Example 2-19 Finding more information about the NVIDIA board

```
0$ pgaccelinfo

CUDA Driver Version:           8000
```

```
NVRM version:                  NVIDIA UNIX ppc64le Kernel Module  361.103  Tue Oct
25 12:57:47 PDT 2016

Device Number:                 0
Device Name:                   Tesla P100-SXM2-16GB
Device Revision Number:        6.0
Global Memory Size:            17071669248
Number of Multiprocessors:     56
Concurrent Copy and Execution: Yes
Total Constant Memory:         65536
Total Shared Memory per Block: 49152
Registers per Block:           65536
Warp Size:                     32
Maximum Threads per Block:     1024
Maximum Block Dimensions:      1024, 1024, 64
Maximum Grid Dimensions:       2147483647 x 65535 x 65535
...
...
... Output Omited
```

The command that is shown in Example 2-20 also can be used.

Example 2-20 Finding the compute capacity of your board

```
0$ pgaccelinfo | grep -m 1 "PGI Compiler Option"
PGI Compiler Option:           -ta=tesla:cc60
```

For more information about running OpenACC parallelization and the performance that is achieved in Example 2-17 on page 63, see 3.2.2, “OpenACC execution and scalability” on page 101.

For more information about how to use and install `openacc`, see the following resources:

- Resources page of the OpenACC website
- OpenACC Courses page of the NVIDIA Accelerated Computing website.

2.6.4 IBM XL C/C++ and Fortran offloading

Another much awaited feature on the IBM Power System S822LC is the use of OpenMP directives to perform GPU offloading through IBM XL C/C++ programs. The combination of the POWER processors with the NVIDIA GPUs provide a robust platform for heterogeneous, high-performance computing that can run several scalable technical computing workloads efficiently. The computational capability is built on top of massively parallel and multithreaded cores within the NVIDIA GPUs and the IBM POWER processors.

For more information about the power of this feature, see *Optimization and Programming Guide for Little Endian Distributions*.

IBM XL Fortran and IBM XL C/C++ V15.1.5 partially support the OpenMP Application Program Interface Version 4.5 specification. Compute-intensive parts of an application and associated data can be offloaded to the NVIDIA GPUs by using the supported OpenMP preprocessor directives device constructs to control parallel processing.

Note: The pragma calls take effect only when parallelization is enabled with the `-qsmp` compiler option. Also, the compilation also must include the `-qoffload` to offload.

For example, you can use the `omp` target directive to define a target region, which is a block of computation that operates within a distinct data environment and is intended to be offloaded into a parallel computation device during execution. Table 2-19 lists some of the supported OpenMP 4.5 pragma clauses.

Table 2-19 Supported OpenMP 4.5 pragma clauses

XL Fortran	XL C/C++	Description
TARGET DATA	omp target data	The omp target data directive maps variables to a device data environment, and defines the lexical scope of the data environment that is created. The omp target data directive can reduce data copies to and from the floading device when multiple target regions are using the same data.
TARGET ENTER DATA	omp target enter data	The omp target enter data directive maps variables to a device data environment. The omp target enter data directive can reduce data copies to and from the offloading device when multiple target regions are using the same data, and when the lexical scope requirement of the omp target dataconstruct is not appropriate for the application.
TARGET EXIT DATA	omp target exit data	The omp target exit data directive unmaps variables from a device data environment. The omp target exit data directive can limit the amount of device memory when you use the omp target enter data construct to map items to the device data environment.
TARGET	omp target	The omp target directive instructs the compiler to generate a target task; that is, to map variables to a device data environment and to run the enclosed block of code on that device. Use the omp target directive to define a target region, which is a block of computation that operates within a distinct data environment and is intended to be offloaded onto a parallel computation device during execution.
TARGET UPDATE	omp target update	The omp target update directive makes the list items in the device data environment consistent with the original list items by copying data between the host and the device. The direction of data copying is specified by motion-type.

XL Fortran	XL C/C++	Description (Fort.)
DECLARE TARGET	omp declare target	The omp declare target directive specifies that variables and functions are mapped to a device so that these variables and functions can be accessed or run on the device.
TEAMS	omp teams	The omp teams directive creates a collection of thread teams. The master thread of each team runs the teams.
DISTRIBUTE	omp distribute	The omp distribute parallel for directive runs a loop by using multiple teams where each team typically consists of several threads. The loop iterations are distributed across the teams in chunks in round robin fashion.
DISTRIBUTE PARALLEL DO	omp distribute parallel for	The omp distribute parallel for directive runs a loop by using multiple teams where each team typically consists of several threads. The loop iterations are distributed across the teams in chunks in round-robin fashion

More information about all supported XL OpenMP clauses, see the *Compiler Reference for Little Endian Distributors* document.

To test the scalability of this feature, the following well-known dgemm example is used:

$$C = \alpha[A] \cdot [B] + \beta[C]$$

where, *alpha* and *beta* are real scalar values, and *A*, *B*, and *C* are matrixes of conforming shape.

Example 2-21 presents a column major order implementation of the dgemm algorithm. This code was created to present the compilation and execution of a real example of an algorithm by using XL C offloading on NVIDIA GPUs. In addition, it aimed to present a comparison between a hand implementation against the advantages of using the vast API of PESSL, such as the implementations that are described in 2.4.2, “Using GPUs with Parallel ESSL” on page 43. Finally, this implementation was designed to ease the execution of a scalability curve, which were created for a broader perspective of the performance gain that is achieved by offloading the code to the GPU.

Example 2-21 shows how to compile the source code.

Example 2-21 How to compile a program that uses XL offloading

```
$ /opt/ibm/xlC/13.1.5/bin/cc_r -O3 -Wall -o dgemm_xlc dgemm_xlc.c -qsmp=omp
-qoffload -mcpu=pwr8 -mtune=pwr8
```

Depending on the computation logic, some algorithms can perform better when column major order is used than line major order, and vice versa. This issue usually occurs because of the increase of the cache hit ratio, but one of these configurations can mask a false sharing situation.

Although Example 2-22 dynamically allocates the matrixes and calculates the indexes of the matrix throughout all the code, the ESSLs in Example 2-5 on page 38 performed this function on line 60 by selecting only one parameter at the dgemm function. Therefore, it is important to study and use the highly optimized ESSL API as much as possible.

Example 2-22 also shows the syntax of copying dynamically allocated vectors to the GPU (normal vector uses the name of the variable). The data copy is performed on `#pragma omp target` call on line 61 of Example 2-22, where we use the `map()` function to copy the A, B, and C into the GPU, and the tmp out of the GPU at the end of the calculation. This process avoids multiple copies at the end of each iteration of the outer for loop.

We then call the `#pragma omp teams` that are nested with the `thead_limit` to run our code in warps, as a CUDA code can. Finally, we use the `#pragma omp distribute` to share the code according to our blocks of a specific number of threads.

Example 2-22 Column major order implementation of dgemm to test scalability of XL C offloading

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <time.h>
5. #include <sys/time.h>
6. #include <omp.h>
7.
8. #define interval 0.5
9. #define verbose 0
10.
11.double  float_rand();
12.void    timer(struct timeval *, struct timeval *, struct timeval *);
13.void    print_mat(double *mat, long N);
14.
15.int main(int argc, char* argv[])
16.{
17.        if(argc!=4)
18.        {
19.                printf("No matrix dimension OR number of blocks OR number of
   threads were provided\nAborting...\n");
20.                return -1;
21.        }
22.
23.        struct timeval diff, start, end;
24.        long i, j, k, N, n_blocks, n_threads;
25.        double *a, *b, *c, *tmp, alpha, beta, sum;
26.        double flop, exec_time;
27.
28.        srand((unsigned int)time(NULL));
29.
30.        N = atol(argv[1]);
31.        n_blocks = atol(argv[2]);
32.        n_threads = atol(argv[3]);
33.
```

```
34.        alpha = ((double)1.3);
35.        beta  = ((double)2.4);
36.        flop  = (double)(N*N*(2*(N-1)));
37.
38.        a = (double*) malloc(N * N * sizeof(double));
39.        b = (double*) malloc(N * N * sizeof(double));
40.        c = (double*) malloc(N * N * sizeof(double));
41.        tmp = (double*) malloc(N * N * sizeof(double));
42.
43.    for(i=0;i<N;i++)
44.        {
45.                for(j=0;j<N;j++)
46.                {
47.                        a[ (i) * N + (j) ] = float_rand();
48.                        b[ (i) * N + (j) ] = float_rand();
49.                        c[ (i) * N + (j) ] = float_rand();
50.                }
51.        }
52.
53.        if(verbose)
54.        {
55.                printf("A\n");   print_mat(a,N);
56.                printf("\nB\n"); print_mat(b,N);
57.                printf("\nC\n"); print_mat(c,N);
58.        }
59.
60.        gettimeofday(&start,NULL);
61.                #pragma omp target map(to: a[0:N*N], b[0:N*N], c[0:N*N])
   map(from: tmp[0:N*N])
62.                #pragma omp teams num_teams(n_blocks) thread_limit(n_threads)
63.                #pragma omp distribute parallel for private(sum)
64.                for(i=0;i<N;i++)
65.                {
66.                        for(j=0;j<N;j++)
67.                        {
68.                                sum = 0.0;
69.                                for(k=0;k<N;k++)
70.                                {
71.                                     sum += a[ (i)*N + (k) ] * b[ (k)*N + (j) ];
72.                                }
73.                               tmp[ (i)*N + (j) ] = alpha * sum + beta * c[ (i)*N
   + (j) ];
74.                        }
75.                }
76.        gettimeofday(&end,NULL);
77.
78.        memcpy(c, tmp, N * N * sizeof(double));
79.
80.        if(verbose)
81.        {
82.                printf("\nC = alpha * A * B + beta * C\n");
83.                print_mat(c,N);
84.        }
85.
86.        timer(&diff, &end, &start);
```

```
87.        exec_time = diff.tv_sec + 0.000001 * diff.tv_usec;
88.        printf("\nExecution time = %lf s, %lf MFlops\n\n", exec_time, (flop /
   exec_time));
89.        fprintf(stderr, "%.6lf,%.6lf\n", exec_time, (flop / exec_time));
90.
91.        free(a); free(b); free(c);
92.
93.        return 0;
94.}
95.
96.double float_rand()
97.{
98.        double a = ((double)-1.0) * interval;
99.        double b = interval;
100.        return b + ((double)rand() / ( RAND_MAX / (a-b) ) ) ;
101.}
102.
103.void timer(struct timeval *result, struct timeval *t2, struct timeval *t1)
104.{
105.        long diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec +
   1000000 * t1->tv_sec);
106.        result->tv_sec = diff / 1000000;
107.        result->tv_usec = diff % 1000000;
108.}
109.
110.void print_mat(double *mat, long N)
111.{
112.        long i, j;
113.
114.        for(i=0;i<N;i++)
115.        {
116.                for(j=0;j<N;j++)
117.                {
118.                        ( mat[ (i)*N + (j) ] >= 0.0) ? printf(" %f ",mat[ (i)*N
   + (j) ]) : printf("%f ",mat[ (i)*N + (j) ]);
119.                }
120.                printf("\n");
121.        }
122.}
```

For more information about running and the performance that is achieved in Example 2-22 on page 67, see 3.2.3, “XL Offload execution and scalability” on page 101.

For more information about all the features of XL offloading, see the Product documentation for XL C/C++ for Linux, V13.1.5, for little endian distributions page of the IBM Support website.

For more information about OpenMP examples, see the OpenMP 4.5 documentation.

2.6.5 MPI programs with IBM Parallel Environment v2.3

Note: IBM PE is being deprecated in favor IBM Spectrum MPI, which is a lighter and high-performance implementation of the MPI (Message Passing Interface) Standard.

For more information, see 2.6.9, “MPI programs with IBM Spectrum MPI” on page 81.

The following sections remain in this book for completeness and migration purposes. These sections will be removed in future releases of this publication:

- 2.6.5, “MPI programs with IBM Parallel Environment v2.3” on page 70
- 2.6.6, “Hybrid MPI and CUDA programs with IBM Parallel Environment” on page 75
- 2.6.7, “OpenSHMEM programs with the IBM Parallel Environment” on page 79
- 2.6.8, “Parallel Active Messaging Interface programs” on page 80

The MPI development and runtime environment that is provided by the IBM PE version 2.3 includes the following general characteristics:

- Provides the implementation of MPI version 3.0 standard, based on the open source MPICH project.
- The MPI library uses PAMI protocol as a common transport layer.
- Supports MPI application in C, C++, and Fortran.
- Supports 64-bit applications only.
- Supports GNU and IBM XL compilers.
- MPI operations can be carried out on main or user-space threads.
- The I/O component (also known as *MPI-IO*) is an implementation of ROMIO that is provided by MPICH 3.1.2.
- Provides a CUDA-aware MPI implementation.
- Uses a shared memory mechanism for message transport between tasks on the same compute node. In contrast, the User Space (US) communication subsystem, which provides direct access to a high-performance communication network by way of an InfiniBand adapter, is used for internode tasks.
- Allows message stripping, failover, and recovery on multiple or single (with some limitations) network configurations.
- Allows for dynamic process management.

This section introduces some MPI implementation aspects of IBM PE Runtime and general guidance on how to build parallel applications. For more information, see *Parallel Environment Runtime Edition for Linux: MPI Programming Guide*.

MPI API

The MPI implementation of PE is based on MPICH. For information about the MPI API, see the MPICH website.

Provided compilers

The compilers provide a set of compilation scripts that are used to build parallel applications that support GNU and IBM XL family compilers for C, C++, and Fortran.

C, C++, and Fortran applications are built, by using **mpcc**, **mpCC**, and **mpfort** compilation scripts, respectively, that are linked with the threaded version of MPI and **poe** libraries by default. They also apply some instrumentation on binary file so that **poe** is indirectly started to manage the parallel execution.

The **mpicc**, **mpicxx**, **mpif77**, and **mpif90** compilation scripts for C, C++, Fortran77, and Fortran90, respectively, are designed to build MPICH-based parallel applications. A program that is compiled with those scripts can be run through **poe**.

Example 2-23 shows the **mpicc** command compiling of an MPI C application by using the GNU GCC compiler.

Example 2-23 IBM PE Runtime mpicc command to compile an MPI C program

```
$ export PATH=/opt/ibmhpc/pecurrent/base/bin:$PATH
$ mpicc -compiler gnu -O3 -mcpu=power8 -mtune=power8 -o myApp main.c
```

Use the **-show** option to display the command that is run to compile the application. In Example 2-23, the **mpicc -show** command produces the following output:

```
$ mpicc -show -compiler gnu -O3 -mcpu=power8 -mtune=power8 -o myApp main.c
/usr/bin/gcc -Xlinker --no-as-needed  -O3  -mcpu=power8  -mtune=power8  -o  myApp
main.c  -m64 -D__64BIT__ -Xlinker --allow-shlib-undefined -Xlinker
--enable-new-dtags -Xlinker -rpath -Xlinker /opt/ibmhpc/pecurrent/mpich/gnu/lib64
-I/opt/ibmhpc/pecurrent/mpich/gnu/include64 -I/opt/ibmhpc/pecurrent/base/include
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64 -lmpi
```

All of these compilation scripts use the XL compilers, unless the MP_COMPILER variable or **-compiler** option is set, which instructs them to use another compiler. You can use `gnu` or `xl` option values to evoke GNU or XL compilers. For third-party compilers, use the fully qualified path (for example, `MP_COMPILER=/opt/at9.0/bin/gcc`).

Note: The compilation scripts that are provided by the latest version of the IBM PE Runtime are installed in the `/opt/ibmhpc/pecurrent/base/bin` directory.

Details of MPI-IO implementation

The ROMIO implementation of IBM PE Runtime is configured to use the IBM Spectrum Scale file system, which delivers high-performance I/O operations. Some environment variables are also introduced to allow users to control the behavior of some operations, such as collective aggregations.

Note: Although the configuration also supports NFS and POSIX compliance file systems, some limitations cant apply.

The file system detection mechanism uses system calls, unless the parallel file system is set with the `ROMIO_FSTYPE_FORCE` environment variable. Many changes in the default configuration of MPI-IO by passing hints to ROMIO are allowed, which sets them in the `ROMIO_HINTS` environment variable.

Local rank property

The Parallel Environment provided MPI implementation includes a mechanism to determine the rank of a task among others tasks that are running in the same machine, which is also known as the *task local rank*.

The read-only MP_COMM_WORLD_LOCAL_RANK variable can be used to obtain the local rank. Each task reads its local attributed rank and is made available by the runtime environment.

Example 2-24 shows an MPI C program that reads the MP_COMM_WORLD_LOCAL_RANK and prints its value to standard output.

Example 2-24 Simple MPI C program that prints task local rank

```
1. #include<mpi.h>
2. #include<stdio.h>
3. #include<stdlib.h>
4. #include <unistd.h>
5.
6. int main(int argc, char* argv[])
7. {
8.       int world_rank, world_size, local_rank;
9.       char hostname[255];
10.
11.      MPI_Init(&argc, &argv);
12.
13.      gethostname(hostname, 255);
14.      MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
15.      MPI_Comm_size(MPI_COMM_WORLD, &world_size);
16.      local_rank = atoi(getenv("MP_COMM_WORLD_LOCAL_RANK"));
17.      printf("Task %d: running on node %s and local rank %d\n", world_rank,
   hostname, local_rank);
18.
19.      MPI_Finalize();
20.      return EXIT_SUCCESS;
21.}
```

The output of the program in Example 2-24 is shown in Example 2-25.

Example 2-25 Show output of simple MPI C program that prints the local rank

```
$ mpcc main.c
$ MP_RESD=poe MP_PROCS=5 ./a.out
Task 0: running on node xcat-mn.xcat-cluster and local rank 0
Task 1: running on node xcat-mn.xcat-cluster and local rank 1
Task 4: running on node xcat-mn.xcat-cluster and local rank 4
Task 2: running on node xcat-mn.xcat-cluster and local rank 2
Task 3: running on node xcat-mn.xcat-cluster and local rank 3
```

Switch MPI configurations with environment modules

The process of compiling and running a parallel application with PE requires setting several environment variables. Some predefined profiles that use environment modules[3] to change the system's variables are available to ease this task. As of PE 2.3, the following development profiles for MPI applications are provided:

- `perf`: Compile the MPI application with XL and set to run in development mode with minimal error checking.
- `debug`: Compile the MPI application with XL and set to run in development mode with the debug versions of the libraries.

[3] Learn about Linux environment modules at the project website: http://modules.sourceforge.net

- `trace`: Compile the MPI application with XL and set to run in development mode with the trace libraries.

The environment module command (module) can be installed in the system. For RHEL 7.3, it can be installed with the following commands:

```
# yum install environment-modules
```

After the module command is installed in the system, you can list all available modules, add the modules that are provided by PE, and load modules as shown in Example 2-26.

Example 2-26 Show the use of modules to set environment to build and run MPI applications

```
$ module avail

------------------------------------------------------------------------
/usr/share/Modules/modulefiles
------------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own
$ module use -a /opt/ibmhpc/pecurrent/base/module
$ echo $MODULEPATH
/usr/share/Modules/modulefiles:/etc/modulefiles:/opt/ibmhpc/pecurrent/base/module
$ module avail

------------------------------------------------------------------------
/usr/share/Modules/modulefiles
------------------------------------------------------------------------
dot         module-git  module-info modules     null        use.own

-------------------------------------------------------------------------
/opt/ibmhpc/pecurrent/base/module
-------------------------------------------------------------------------
pe2300.xl.debug pe2300.xl.perf  pe2300.xl.trace
$ module whatis pe2300.xl.debug
pe2300.xl.debug      : Adds PE environment variables for xl compiler and debug
develop mode to user environment.

$ module whatis pe2300.xl.perf
pe2300.xl.perf       : Adds PE environment variables for xl compiler and
performance develop mode to user environment.

$ module whatis pe2300.xl.trace
pe2300.xl.trace      : Adds PE environment variables for xl compiler and trace
develop mode to user environment.

$ env | grep MP
$ module load pe2300.xl.perf
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=min
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=min
MP_CONFIG=2300
```

```
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
$ module load pe2300.xl.debug
  Adds these PE settings into your environment:

  MP_COMPILER=xl
  MP_EUIDEVELOP=debug
  MP_MPILIB=mpich
  MP_MSG_API=mpi
  MP_CONFIG=2300
$ env | grep MP
MP_EUIDEVELOP=debug
MP_CONFIG=2300
MP_MSG_API=mpi
MP_MPILIB=mpich
MP_COMPILER=xl
```

Simulating different SMT modes

The example cluster contains two nodes (S822LC), each of which has two sockets. One NUMA node corresponds to one socket and has 10 physical POWER8 cores inside it. The systems are configured with SMT-8, which means that a physical core can split a job between eight logical CPUs (threads). The structure of the S822LC is shown in Figure 2-4.

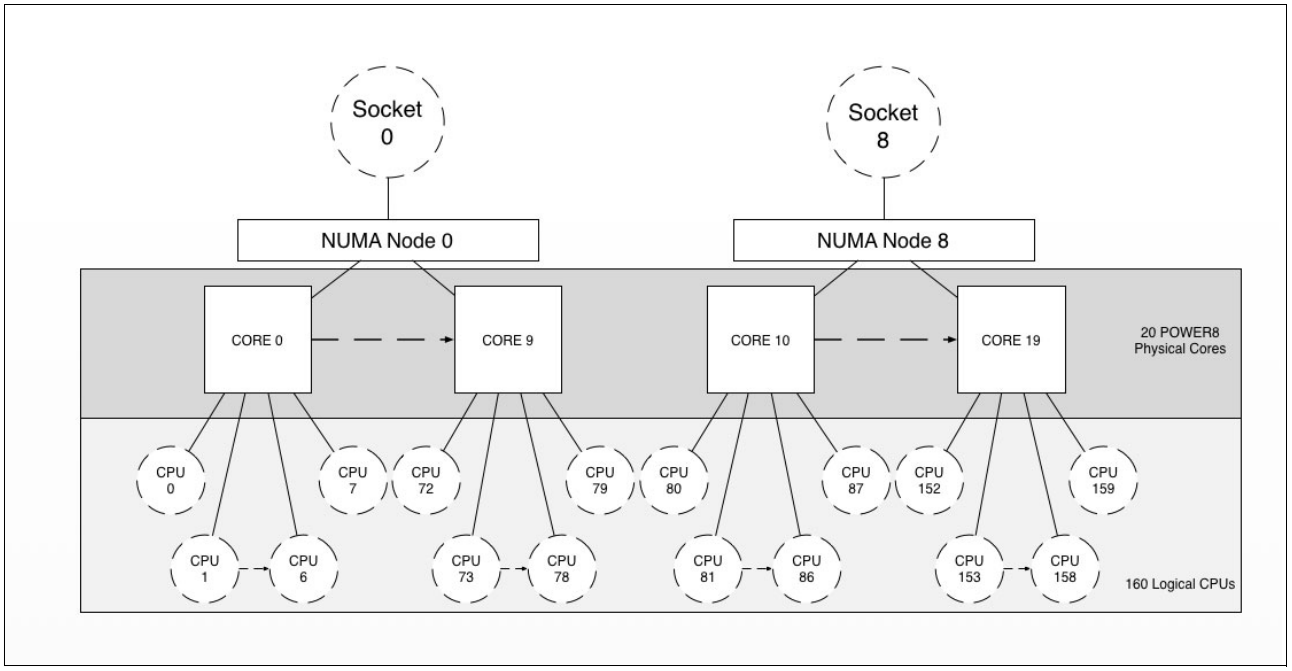


Figure 2-4 Structure of node in cluster

PE MPI provides run options or environment variables, which helps to simulate runs with different SMT modes. The following cases describe two ways when a job uses only 1 logical CPU from each POWER8 core (SMT-1 mode), and fully uses all 160 logical CPUs by using Example 2-15 on page 61 with Parallel ESSL calls:

- To configure the job to use only one logical CPU per POWER8 core, the following run command can be used:

  ```
  MP_TASK_AFFINITY=core:1 MP_RESD=poe MP_PROCS=40 ./test_pdgemm
  ```

  `MP_TASK_AFFINITY=core:1` says to PE MPI that each MPI task can use only one POWER8 core. Overall, the full cluster has 40 POWER8 cores (each node contains 20 cores); therefore, a maximum of 40 MPI (`MP_PROCS=40`) tasks can be used for such run.

  Another possible solution to simulate SMT-1 mode is to take 20 MPI tasks with two POWER8 cores for each of them by using the following run command:

  ```
  MP_TASK_AFFINITY=core:2 MP_RESD=poe MP_PROCS=20 ./test_pdgemm
  ```

- The following run command shows how to use the cluster and all 160 logical CPUs per node:

  ```
  MP_TASK_AFFINITY=cpu:16 MP_RESD=poe MP_PROCS=20 ./test_pdgemm
  ```

  `MP_TASK_AFFINITY=core:1` means for PE MPI that each MPI task can use only 16 logical CPUs. Each node in the cluster contains 160 logical CPUs, so 10 MPI tasks per node and 20 for the overall cluster can be used in this run.

  If you want to change the CPU affinity variable and still use all logical CPUs, the number of MPI tasks must be changed respectively. The following commands describe the alternative calls:

  ```
  MP_TASK_AFFINITY=cpu:160 MP_RESD=poe MP_PROCS=2 ./test_pdgemm
  MP_TASK_AFFINITY=cpu:8 MP_RESD=poe MP_PROCS=40 ./test_pdgemm
  MP_TASK_AFFINITY=cpu:2 MP_RESD=poe MP_PROCS=160 ./test_pdgemm
  ```

2.6.6 Hybrid MPI and CUDA programs with IBM Parallel Environment

The IBM PE compilers and runtime environment support building and running hybrid of MPI and CUDA programs.

Building the program

The common case is to organize sources on separate files for MPI and CUDA codes, and use different compilers to build the objects and then link them by using the MPI compiler. Example 2-27 shows this procedure.

Example 2-27 Show how hybrid MPI and CUDA programs can be built

```
$ mpCC -o helloMPICuda_mpi.o -c helloMPICuda.cpp
$ nvcc -ccbin g++ -m64 -gencode arch=compute_37,code=sm_37 -o helloMPICuda.o -c
helloMPICuda.cu
$ mpCC -o helloMPICuda helloMPICuda_mpi.o helloMPICuda.o
-L/usr/local/cuda-7.5/lib64 -lcudart
```

The MPI source is built by using the **`mpCC`** script, whereas the **nvcc** compiler is used for the CUDA code. Finally, the object files are linked into the executable file, along with the CUDA runtime library. Implicitly, `mpCC` links the executable file to `libmpi` (MPI), `libpami` (PAMI), and `libpoe` (POE).

Source files that contain MPI and CUDA code mixed (often called *spaghetti programming style*) can be compiled, although it is not recognized as a good programming practice. In this case, you can compile it invoking the **nvcc** (CUDA C compiler) and setting the MPI library and the headers as shown in the following example:

```
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
```

CUDA-aware MPI support

The CUDA-aware MPI feature of PE allows direct access of tasks to the GPU memory's buffers on operations of message passing, which can improve the application performance significantly.

Note: The CUDA-aware MPI API was introduced in IBM Parallel Environment version 2.3.

The CUDA development model can be generalized with following steps:

1. Allocate data on the host (CPU) memory.
2. Allocate data on the device (GPU) memory.
3. Move the data from host to device memory.
4. Perform on that data some computation (kernel) on device.
5. Move processed data from device back to the host memory.

In a context of send/receive communication and without the CUDA-aware MPI capability, the tasks cannot access the GPU memory. Therefore, Step 5 is required because the data must be on the host memory before it is sent. However, with CUDA-aware MPI, the task accesses the portion of memory that is allocated in Step 2, which means that data cannot be staged into host memory (Step 5 is optional).

Note: By default, CUDA-aware MPI is disabled on IBM PE run time. That behavior can be changed by exporting the MP_CUDA_AWARE environment variable with `yes` (enable) or `no` (disable).

The code that is shown in Example 2-28 shows how the CUDA-aware feature can be used within a hybrid of CUDA and MPI programs. It is a simple MPI program that is meant to run two jobs where task 0 initialize an array, increment its values by one using the GPU computation, then sends the result to task 1. In line 56, the call to the MPI_Send function uses the device buffer (allocated on line 40) directly.

Example 2-28 Simple CUDA-aware MPI program

```
1 #include<cuda_runtime.h>
2 #include<mpi.h>
3 #include<stdio.h>
4 #include<stdlib.h>
5 #include<assert.h>
6
7  __global__ void vecIncKernel(int* vec, int size) {
8     int tid = blockDim.x * blockIdx.x + threadIdx.x;
9     if(tid < size) {
10        vec[tid] += 1;
11     }
12  }
13
```

```
14 #define ARRAY_ELEM 1024
15 #define ARRAY_ELEM_INIT 555
16
17 int main(int argc, char* argv[]) {
18   int taskid, numtasks, tag=0;
19   MPI_Status status;
20   int array_size = sizeof(int) * ARRAY_ELEM;
21   int threadsPerBlock = 32;
22   int blockSize = ceil(ARRAY_ELEM/threadsPerBlock);
23
24   MPI_Init(&argc, &argv);
25   MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
26   MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
27
28   if(numtasks != 2) {
29     printf("This program must run only 2 tasks\n");
30   }
31   /*
32    * Task 0: initialize an array, increment its values by 1,
33    *  and send it to Task 1.
34    */
35   if(taskid == 0) {
36       int *vSend = (int*) malloc(array_size);
37       for(int i=0; i < ARRAY_ELEM; i++)
38         vSend[i]=ARRAY_ELEM_INIT;
39       int *vDev;
40       if(cudaMalloc((void **)&vDev, array_size) == cudaSuccess) {
41         cudaMemcpy(vDev,vSend, array_size,cudaMemcpyHostToDevice);
42         vecIncKernel<<< blockSize, threadsPerBlock>>>(vDev, ARRAY_ELEM);
43         cudaDeviceSynchronize();
44       } else {
45         printf("Failed to allocate memory on GPU device\n");
46         MPI_Abort(MPI_COMM_WORLD, MPI_ERR_OTHER);
47         exit(0);
48       }
49       if(strcmp(getenv("MP_CUDA_AWARE"),"yes") != 0) {
50           printf("Cuda-aware MPI is disabled, MPI_Send will fail.\n");
51       }
52       /*
53        * CUDA-AWARE MPI Send: using the buffer allocated in GPU device.
54        *  Do not need to transfer data back to host memory.
55        */
56       MPI_Send(vDev, ARRAY_ELEM, MPI_INT, 1, tag, MPI_COMM_WORLD);
57   } else {
58       /*
59        * Task 1: receive array from Task 0 and verify its values
60        *  are incremented by 1.
61        */
62       int *vRecv = (int*) malloc(array_size);
63       MPI_Recv(vRecv, ARRAY_ELEM, MPI_INT, 0, tag, MPI_COMM_WORLD, &status);
64       int expected = ARRAY_ELEM_INIT+1;
65       for(int i=0; i < ARRAY_ELEM_INIT; i++) {
66         assert(vRecv[i]==expected);
67       }
68   }
```

```
69
70   MPI_Finalize();
71   return 0;
72 }
```

The program that is shown in Example 2-28 on page 76 was compiled and run twice, as shown in Example 2-29. The first run enables the CUDA-aware MPI (see the line starting with `MP_CUDA_AWARE=yes`). Then, it runs with the feature disabled (the line starting with `MP_CUDA_AWARE=no`), which forces it to exit with a segmentation fault. In this situation, a remediation can be implemented that consists of copying the buffer back to the host memory and using it in the MPI message pass call.

Example 2-29 Compiling and running the simple CUDA-aware MPI program

```
$
LD_LIBRARY_PATH=/opt/ibmhpc/pecurrent/mpich/gnu/lib64/:/opt/ibmhpc/pecurrent/base/
gnu/lib64/:$LD_LIBRARY_PATH
$ nvcc -I/opt/ibmhpc/pecurrent/mpich/gnu/include64/
-L/opt/ibmhpc/pecurrent/mpich/gnu/lib64/ -L/opt/ibmhpc/pecurrent/base/gnu/lib64/
-lmpi -lpami main.cu
$ MP_CUDA_AWARE=yes MP_RESD=poe MP_PROCS=2 poe ./a.out
$ MP_CUDA_AWARE=no MP_RESD=poe MP_PROCS=2 poe ./a.out
  5 Cuda-aware MPI is disabled, MPI_Send will fail.
  6 ERROR: 0031-250  task 0: Segmentation fault
```

Note: At the time of this writing, the NVIDIA GPUDirect technology is not supported on IBM Power Systems. However, the CUDA-aware MPI API of IBM PE implement transparent means to send/receive data to and from the GPU buffers.

As of PE 2.3, the CUDA-aware MPI is implemented on the following features:

- All two-sided (point-to-point) communication (blocking and non-blocking and intra- and inter-communicator)
- All blocking intra-communicator collectives

However, the following features are not supported on the context of CUDA-aware MPI:

- One-sided communications
- One-sided synchronization calls
- Fabric Collective Accelerator (FCA)
- Non-blocking collective API calls
- Inter-communicator collective API
- The collective selection with the **`pami_tune`** command

Using MPI local rank to balance the use of GPU devices

A policy to grant shared access and load balance to GPU device among local tasks (running on same compute node) can be implemented by using the local rank feature that is available in the PE MPI. For example, one algorithm can limit the number of tasks that use GPUs to avoid oversubscription.

For more information about the PE local rank, see 2.6.5, “MPI programs with IBM Parallel Environment v2.3” on page 70.

Starting concurrent CUDA kernels from multiple tasks

There are some considerations about shared use of GPUs with multiple MPI tasks. For more information, see 4.5.2, “CUDA Multi-Process Service” on page 156.

2.6.7 OpenSHMEM programs with the IBM Parallel Environment

The OpenSHMEM[4] provides a specification API and reference implementation for the Partitioned Global Address Space (PGAS) parallel programming model, which abstracts the concept of global shared memory on processes that are distributed across different address spaces. Its implementation involves the use of a communication library that often uses Remote Direct Memory Access (RDMA) techniques.

PE provides an OpenSHMEM runtime library (`libshmem.so`), compiler scripts, and tools that enable developing and running OpenSHEM programs. The PAMI library is used for communication among processing elements of the PGAS program, and it can use the RDMA capabilities of the InfiniBand interconnect to improve performance.

As of PE version 2.3, support is available only for parallel programs that are written in C and C++. Its implementation is based on the OpenSHEM API specification version 1.2[5] with some minor deviations. For more information about supported and unsupported routines, see the OpenSHMEM API support page of the IBM Knowledge Center website.

The OpenSHMEM tools are installed in the `/opt/ibmhpc/pecurrent/base/bin` directory. The compile scripts for C and C++ are, **`oshcc`** and **`oshCC`**, respectively. The **`oshrun`** script runs programs, although **`poe`** also can be used.

Example 2-30 shows the OpenSHMEM program. The API can be used only after a shared memory section is initialized (line 8). A symmetric memory area (the basket variable that is declared on line 6) is written by all processing elements (call to `shmem_int_put` on line 14) on PE zero. Then, all of the elements read the variable from PE zero (call to `shmem_int_get` on line 15) to print its value.

Example 2-30 Simple OpenSHMEM program

```
 1 #include<shmem.h>
 2 #include<stdio.h>
 3 #include<stdlib.h>
 4
 5   int main() {
 6     static int basket; // Global shared (symetric) variable
 7     int cents_p, cents_g; // Processing element's local variable
 8     shmem_init(); // Initialize SHMEM
 9     int my_pe = shmem_my_pe(); // Processing element's number
10     //printf("Hello, I am PE %d\n", my_pe);
11     cents_p = rand()%10;
12     shmem_barrier_all();
13     if(my_pe != 0)
14       shmem_int_put(&basket, &cents_p, 1, 0);
15     shmem_int_get(&cents_g, &basket, 1, 0);
16     printf("Hello, I am PE %d. I put %d cents but I get %d\n", my_pe, cents_p,
cents_g);
17     shmem_finalize(); // Finalize SHMEM
18
```

[4] Learn more about OpenSHMEM at http://openshmem.org

[5] The OpenSHEM specification version 1.2 document is available at this website: http://openshmem.org/site/sites/default/site_files/openshmem-specification-1.2.pdf

```
19     return 0;
20   }
```

The source code that is shown in Example 2-30 on page 79 can be compiled by using the **oshcc** script, as shown in Example 2-31. It is a common practice that the name of an OpenSHMEM executable file includes the suffix *.x*.

Example 2-31 Show how to compile an OpenSHMEM program

```
$ oshcc -O3 hellosh.c -o hellosh.x
$ ls
hellosh.c  hellosh.x
```

By default, **oshcc** (or **oshCC**) compiles the application with **xlc** (or **xlC**), unless it is not installed into the system or the MP_COMPILER environment variable is set to use **gcc** (or **g++**).

The program is started by using the **oshrun** script. Alternatively, it can evoke **poe** directly. For more information about running OpenSHMEM programs, see 3.3.3, “Running OpenSHMEM programs” on page 111.

2.6.8 Parallel Active Messaging Interface programs

Parallel Active Messaging Interface (PAMI) is a low-level protocol that is the foundation of communications on MPI and OpenSHMEM implementations of the IBM PE run time. It provides an API for programs to access PAMI capabilities, such as collective communications and RDMA that use InfiniBand Host Channel Adapters (HCAs).

Many sample codes that can be used to demonstrate PAMI subroutines are included with the IBM PE run time and are available in the `/opt/ibmhpc/pecurrent/ppe.samples/pami` folder. Example 2-32 shows how to build the PAMI samples and run a program (`alltoall.exe`) that uses the all to all communication functions.

Example 2-32 Show how to build and run PAMI sample codes

```
$ cp -r /opt/ibmhpc/pecurrent/ppe.samples/pami .
$ cd pami/
$ make
$ cd pami_samples
$ vi host.list
$ MP_RESD=poe MP_PROCS=4 ./coll/alltoall.exe
# Context: 0
# Alltoall Bandwidth Test(size:4) 0x1000e400, protocol: I1:Alltoall:P2P:P2P
# Size(bytes)  iterations    bytes/sec      usec
# -----------      -----------    -----------    ---------
            1             100       413223.1         2.42
            2             100       840336.1         2.38
            4             100      1659751.0         2.41
            8             100      3347280.3         2.39
           16             100      6722689.1         2.38
           32             100     13502109.7         2.37
           64             100     26778242.7         2.39
          128             100     53112033.2         2.41
          256             100     81012658.2         3.16
          512             100    171812080.5         2.98
         1024             100    320000000.0         3.20
         2048             100    552021563.3         3.71
         4096             100    869639065.8         4.71
```

For more information, see *Parallel Environment Runtime Edition for Linux: PAMI Programming Guide*.

2.6.9 MPI programs with IBM Spectrum MPI

IBM Spectrum MPI v10-1.0.2 is a high-performance implementation of the Message Passing Interface (MPI) Standard. It is widely used in the high-performance computing (HPC) industry for developing scalable, parallel applications in IBM Power Systems. IBM Spectrum MPI supports a broad range of industry-standard platforms, interconnects, and operating systems, which helps ensure that parallel applications can run almost anywhere.

The new Spectrum MPIA includes the following features:

- Portability: A parallel software developer can create one program in a small cluster, and scale it through the interconnects in a large cluster. This feature reduces development time and effort.
- Network Optimization: Spectrum MPI supports various networks and interconnects.
- Collective Optimization: Through the `libcollectives` library, spectrum MPI can enable GPU buffers and enhance performance and scalability, considering it provides advanced logic to determine the fastest algorithm for any given collective operation.

Spectrum MPI is not ABI compatible. For example, it does not run with OpenMPI, Platform MPI, or even IBM PE Runtime Edition. Multithread I/O also is not supported.

The IBM Spectrum MPI collectives component (`libcollectives`) does not support intercommunicators. For intercommunicator collective support, IBM Spectrum MPI relies on Open MPI collective components.

More information about the usage, installation, and limitations of Spectrum MPI v10-1.0.2 see the following resources:

- *IBM Spectrum MPI Version 10 Release 1.0.2 User's Guide*
- *IBM Spectrum MPI Version 10 Release 1.0.2 Installation* documentation

2.6.10 Migrating from IBM PE Runtime Edition to IBM Spectrum MPI

Table 2-20 lists instructions for how to port code from IBM PE to IBM Spectrum MPI.

Table 2-20 IBM PE Runtime Edition tasks and IBM Spectrum MPI equivalent

Task	IBM PE Runtime Edition method	IBM Spectrum MPI method
Running programs	poe program [args] [options]	mpirun [options] program [args]
Compiling programs	The following compiler commands: * mpfort, mpic77, mpif90 * mpcc, mpicc * mpCC, mpic++, mpicxx Or the following environment variable settings: MP_COMPILER = xl | gcc | nvcc	The following compiler commands: * mpfort * mpicc * mpiCC, mpic++, mpicxx Or the following environment variable settings: OMPI_CC = xl | gcc OMPI_FC = xlf | gfortran OMPI_CXX = xlC | g++
Determining rank before MPI_Init	The MP_CHILD environment variable	The OMPI_COMM_WORLD_RANK environment variable
Specifying the local rank	The MPI_COMM_WORLD_LOCAL_RANK environment variable	The OMPI_COMM_WORLD_LOCAL_RANK environment variable
Setting affinity	The environment variables: * MP_TASK_AFFINITY = cpu * MP_TASK_AFFINITY = core * MP_TASK_AFFINITY = mcm * MP_TASK_AFFINITY = cpu:n * MP_TASK_AFFINITY = core:n * MP_TASK_AFFINITY = 1	mpirun options: -aff width:hwthread -aff width:core -aff width:numa --map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to hwthread --map-by ppr:$MP_TASKS_PER_NODE:node:pe=N --bind-to core -aff none
Setting CUDA-aware	The MP_CUDA_AWARE environment variable	The mpirun -gpu option
Setting FCA	The MP_COLLECTIVE_OFFLOAD environment variable	The mpirun -FCA and -fca options

Task	IBM PE Runtime Edition method	IBM Spectrum MPI method
Setting RDMA	* MP_USE_BULK_XFER * The MP_BULK_MIN_MSG_SIZE environment variable	RDMA default, when MSG_SIZE is greater than 64
controlling level of debug messages	The MP_INFOLEVEL environment variable	The mpirun -d option
Setting STDIO	The environment variables: * MP_STDINMODE * MP_STOUTMODE * MP_LABELIO	The mpirun -stdio * options
Specifying the number of tasks	The MP_PROCS environment variable	The mpirun -np * option
Specifying a host list file	The MP_HOSTFILE environment variable	The mpirun -hostfile * option

For more information about migration, see *IBM Spectrum MPI Version 10 Release 1.0.2 User's Guide*.

2.6.11 Using Spectrum MPI

Spectrum MPI is an implementation of the OpenMPI, which makes the source code structure similar to what you find on OpenMPI. For example, this section describes the six most-used procedures of Spectrum MPI, as listed in Table 2-21.

Table 2-21 IBM Spectrum most used procedures

Spectrum MPI procedure	Functionality
MPI_Init()	Start MPI functionality in a program
MPI_Comm_rank()	ID of current MPI process in execution
MPI_Comm_size()	Total number of MPI processes that were spawned
MPI_Send()	Send data to another MPI Process
MPI_Recv()	Receive data from another MPI Process
MPI_Finalize()	End MPI functionality in a program

To demonstrate these procedures, Example 2-33 on page 84 shows the implementation of a simple parallel algorithm that calculates the well-known integration of a curve following the trapezoidal rule, which is mathematically described as the discretization of an integral, as shown in the following example:

$$\int_a^b f(x)\,dx \approx g(x)$$

$$h = \frac{b-a}{N}$$

$$g(x) \approx \frac{h}{2} \sum_{k=0}^{N-1} (f(x_k) + f(x_{k+1}))$$

$$g(x) \approx \frac{h}{2}[f(x_0) + 2f(x_1) + \ldots + 2f(x_{N-1}) + f(x_N)] \approx \frac{h}{2}(f(x_0) + f(x_N)) + h \sum_{k=1}^{N-1} f(x_k)$$

$$x_k = a + hk,\ k = 0, 1, \ldots, N$$

The trapezoidal rule creates infinitesimal trapezoids under a curve, and through their area summation, the integral value of the curve in a close interval can be estimated. The bigger the number of trapezoids, the more accurate the result of the estimation will be.

Why the interest in this section? Because of the mathematical properties of addition and association, we can split this integral even in `p` processes (for simplicity, an even share between the number of processes and trapezoids). Consider the Spectrum MPI code that is shown in Example 2-33.

Example 2-33 Spectrum MPI Code for a trapezoidal integration

```
1. #include <stdio.h>
2. #include <mpi.h>
3.
4. float g(float x)
5. {
6.         return x*x;
7. }
8.
9. float integral(float a, float b, int n, float h)
10.{
11.        int i;
12.
13.        float x;
14.        float local_sum;
15.
16.        x = a;
17.        local_sum = ( (g(a) + g(b)) / 2.0 );
18.
19.        for (i = 1; i <= n-1; i++)
```

```
20.        {
21.                x += h;
22.                local_sum += g(x);
23.        }
24.
25.        return local_sum * h;
26.}
27.
28.int main(int argc, char *argv[])
29.{
30.        int     src;            // Process id of local integral calculations
31.        int     dst = 0;        // Process id of father that sums local calcs
32.        int     tag = 0;        // MPI tag value for messages. Not used here.
33.        int     np;             // Number of processes
34.        int     p_rank;         // Process rank
35.
36.        float   a = -3.0;       // Left interval limit
37.        float   b =  3.0;       // Right interval limit
38.        int     n = 10000;      // Number of trapezoids
39.
40.        float   local_a;        // Local Left interval limit
41.        float   local_b;        // Local Right interval limit
42.        int     local_n;        // Local Number of trapezoids
43.
44.        float   h;              // Trapezoid length of base
45.        float   local_sum;      // Local Integral per process
46.        float   total_sum;      // Total local_sum of whole interval
47.
48.        MPI_Status status;
49.        MPI_Init(&argc, &argv);
50.        MPI_Comm_rank(MPI_COMM_WORLD, &p_rank);
51.        MPI_Comm_size(MPI_COMM_WORLD, &np);
52.
53.        h = (b-a) / n;          // Base length of trapezoids in all interval
54.        local_n = n / np;       // Chunk size of trapezoids for each process
55.
56.        // Division of full interval using the rank of each process
57.        local_a = a + p_rank * local_n * h;
58.        local_b = local_a + local_n * h;
59.        local_sum = integral(local_a, local_b, local_n, h);
60.
61.        if (p_rank == 0)
62.        {
63.                total_sum = local_sum;
64.                for(src = 1; src < np; src++)
65.                {
66.             MPI_Recv(&local_sum, 1, MPI_FLOAT, src, tag, MPI_COMM_WORLD,
   &status);
67.                        printf("MPI_Recv {%d <- %d} = %f\n", p_rank, src,
   local_sum);
68.                        total_sum = total_sum + local_sum;
69.                }
70.        }
71.        else
72.        {
```

```
73.                 printf("MPI_Send {%d -> %d} = %f\n", p_rank, dst, local_sum);
74.                 MPI_Send(&local_sum, 1, MPI_FLOAT, dst, tag, MPI_COMM_WORLD);
75.         }
76.
77.         if(p_rank == 0)
78.                 printf("\nEstimate of local_sum of x^2 = %f\nInterval
   [%.3f,%.3f]\nUsing %d trapezoids\n\n", total_sum, a, b, n);
79.
80.         MPI_Finalize();
81.         return 0;
82.}
```

On line 4 in Example 2-33 on page 84, notice the function of interest is `g(x) = x^2`. From lines 36 - 37, notice the interval that is being calculated in the closed interval of `[-3;+3]`. Also, in the main function, observe that the `MPI_Init()` function is used to start the MPI program. Immediately after, the `MPI_Comm_rank()` and `MPI_Comm_size()` are called to fetch the current ID of each process that is spawned by `MPC_Init` and the total number of processes, respectively.

At running time, Spectrum MPI fetched the number of processes from the environmental variable (or the cli) and created processes for multiple instances of this program. Each program featured its unique rank ID, which is the target of interest in this code.

The `integral()` function sequentially calculates the trapezoid integral in a closed local interval section, following the formulas that were described in 2.6.11, “Using Spectrum MPI” on page 83. However, on lines 57 - 59 in Example 2-33 on page 84, notice that the local variables that were created by combining the limits from our interval with the number of trapezoids and the rank of each MPI process created separate local intervals of our main interval. Therefore, we can calculate the trapezoid integral for each section independently by each MPI process.

What is still necessary is to join the results of each local integral? On line 61 in Example 2-33 on page 84, we check whether we are running the main process (which have `p_rank` equal to `0`). If we are running the main process, we iterate between all MPI processes instances through the function `MPI_Recv` to fetch their partial calculation and add that to `total_sum`. However, if we are not running the process, we are in a child process; therefore, we use `MPI_Send` to give the partial calculations to the main process.

Next, we describe what happened by using `MPI_Send()` and `MPI_Recv()`. Consider the following points;

- `MPI_Send` (`buf`, `count`, `type`, `dest`, `tag`, and `comm`) is a blocking MPI operation that uses the following arguments:
  - `buf` is the address of the buffer that is being sent
  - `count` is the number of elements in the send buffer
  - `type` is the `MPI_Datatype` of the elements in the buffer
  - `dest` is the node rank ID of the destination process
  - `tag` is the tag that the message can receive
  - `comm` is the communicator of this message
- `MPI_Recv` (`buf`, `count`, `type`, `src`, `tag`, `comm`, and `status`) is a blocking MPI operation that uses the following arguments:
  - `buf` is the address of the buffer where the process receives data
  - `count` is the number of elements in the receiving buffer
  - `type` is the `MPI_Datatype` of the elements in the buffer
  - `src` is the node rank id of the process from where data is coming

– `tag` is the tag that the message can receive
– `comm` is the communicator of this message
– `status` is the object status

How to use the `MPI_Datatype` is listed in Table 2-22.

Table 2-22 Using MPI_Datatype

MPI_Datatype	C Type equivalent
MPI_C_BOOL	_Bool
MPI_CHAR	char (text)
MPI_UNSIGNED_CHAR	unsigned char (integer)
MPI_SIGNED_CHAR	signed char (integer)
MPI_INT	signed int
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_C_DOUBLE_COMPLEX	double _Complex

Finally, we analyze the compilation and running of Example 2-33 on page 84 next. Observe that we use **`mpicc`** to compile, and then **`mpirun`** to run the parallel program, as shown in Example 2-34. On the `mpirun` call, we provide the number of processes that we want to run the code. The `-pami_noib` is used because we are using only one compute node to spawn our processes; therefore, no InfiniBand is connected between other node.

Note: Although we iterate in what looks like a sequential order, our use output has no order.

Example 2-34 Compiling with Spectrum MPI

```
$ /opt/ibm/spectrum_mpi/bin/mpicc -O2 trap.c -o trap.mpi

$ /opt/ibm/spectrum_mpi/bin/mpirun -np 5 -pami_noib ./trap.mpi

MPI_Send {3 -> 0} = 1.871934
MPI_Send {1 -> 0} = 1.872049
MPI_Recv {0 <- 1} = 1.872049
MPI_Recv {0 <- 2} = 0.144001
MPI_Recv {0 <- 3} = 1.871934
MPI_Recv {0 <- 4} = 7.056407

Estimate of local_sum of x^2 = 17.999882
Interval [-3.000,3.000]
Using 10000 trapezoids

MPI_Send {2 -> 0} = 0.144001
MPI_Send {4 -> 0} = 7.056407
```

If you perform an analytic integration of `g(x) = x^2 in [-3;3]`, you find the value of 18.

3

Running parallel software, performance enhancement, and scalability testing

This chapter describes some techniques that are used to gain performance out of CPUs and GPUs in POWER8 systems.

We also provide the results of running some source codes that were presented in Chapter 2, “Compilation, execution, and application development” on page 23. These codes are run to demonstrate a few tips to the reader about how to enhance, test, and scale parallel programs in POWER8 systems.

This chapter includes the following topics:

3.1 Controlling the running of multithreaded applications

Gaining performance by computing applications that run on a computing node typically is achieved by using multiple threads, cores, or GPUs. The runtime environment has several options to support the runtime fine-tuning of multithreaded programs.

First, this chapter describes how to control the OpenMP workload by setting certain environment variables in XL C. Then, the chapter shows how to control OpenMP by using the integration of IBM ESSL.

The chapter also shows the tools that can be used to retrieve or set affinity of a process at run time. Finally, the chapter shows how to control the nonuniform memory access (NUMA) policy for processes and shared memory1. GPUs are shown to explain the tools that can be used to retrieve or set affinity of a process at run time. It also shows how to control the nonuniform memory access (NUMA) policy for processes and shared memory[1].

3.1.1 Running OpenMP applications

A user can control the running of OpenMP applications by setting environment variables. This section describes only a subset of the environment variables that are especially important for technical computing workloads.

Note: The environment variables that are prefixed with `OMP_` are defined by the OpenMP standard. Other environment variables that are described in this section are specific to a particular compiler.

Distribution of a workload

The `OMP_SCHEDULE` environment variable controls the distribution of a workload among threads. If the workload items have uniform computational complexity, the static distribution fits well in most cases. If an application does not specify the scheduling policy internally, a user can set it to `static` at run time by exporting the environment variable, as shown in the following example:

```
export OMP_SCHEDULE="static"
```

However, the application can control the scheduling policy from the source code as shown in the following example:

```
#pragma omp parallel for schedule(kind [,chunk size])
```

Table 3-1 shows the options for the OpenMP scheduler of a parallel section.

Table 3-1 Scheduling options for an OpenMP parallel section

Type of scheduling	Description
static	Divide the loop as evenly as possible in chunks of work to the threads. By default, the chunk size is calculated by the loop_count divided by the number_of_threads.
dynamic	By using the internal work queue, this scheduling provides blocks of different chunk sizes to each thread at cost of extra overhead involved. By default, the chunk size is equal to 1.

[1] This section is based on the content that originally appeared in Chapter 7 of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263.

Type of scheduling	Description
guided	Works similar to the dynamic scheduling; however, this scheduling orders the chunks of work from the largest to the smallest, which can handle imbalances more properly. By default, the chunk size is calculated by the loop_count divided by the number_of_threads
auto	The compiler decides between static, dynamic. and guided.
runtime	Use the OMP_SHEDULE environment variable. The advantage of this scheduler is that rewriting or recompiling the source code is unnecessary.

Specifying the number of OpenMP threads

OpenMP applications are often written so that they can create an arbitrary number of threads. In this case, the user sets the `OMP_NUM_THREADS` environment variable to specify the number of threads to create. For example, to run the application with `20` threads, use this following setting:

```
export OMP_NUM_THREADS=20
```

Again, the application can control the scheduling policy from the source code as shown in the following example:

```
#pragma omp parallel ... num_threads(4)
```

Showing OpenMP data

The OpenMP 4.5 standard introduced the `OMP_DISPLAY_ENV` environment variable to show the OpenMP version and list the internal control variables (ICVs). The OpenMP run time prints data to the `stderr` output stream. If the user sets the value to `TRUE`, the OpenMP version number and initial values of ICVs are printed.

The `VERBOSE` value instructs the OpenMP run time to augment the output with the values of vendor-specific variables. If the value of this environment variable is set to `FALSE` or undefined, no information is printed. This variable is useful when you must be certain that the runtime environment is configured as expected at the moment that the program loads.

Placement of OpenMP threads

The POWER8 microprocessor can handle multiple hardware threads simultaneously. With the increasing number of logical processors, the operating system kernel scheduler has more possibilities for automatic load balancing. For technical computing workloads, the fixed position of threads within the server is typically preferred.

The IDs of the logical processors are zero-based. Each logical processor has the same index, regardless of the simultaneous multithreading (SMT) mode. Logical processors are counted starting from the first core of the first socket. The first logical processor of the second core features the index 8. Start numbering the logical processors of the second socket only after you finish the numbering of the logical processors of the first socket.

IBM XL compilers

For the OpenMP application that is compiled with IBM XL compilers, you must use the `XLSMPOPTS` environment variable to control thread placement. This environment variable includes many suboptions, and only a few of these options control thread binding. You can use the combination of `startproc` and `stride` suboptions, or the `procs` suboption. Consider the following points:

- The `startproc` suboption is used to specify the starting logical processor number for binding the first thread of an application. The `stride` suboption specifies the increment for the subsequent threads. For example, the following value of `XLSMPOPTS` instructs the OpenMP runtime environment to bind OpenMP threads to logical processors 80, 84, 88, and so on, up to the last available processor:

  ```
  export XLSMPOPTS=startproc=80:stride=4
  ```

- A user can also explicitly specify a list of logical processors to use for thread binding with the `procs` suboption. For example, to use only even-numbered logical processors of a processor's second core, specify the following value of `XLSMPOPTS`:

  ```
  export XLSMPOPTS=procs=8,10,12,14
  ```

Note: The `startproc`, `stride`, and `procs` suboptions were deprecated in favor of the `OMP_PLACES` environment variable. IBM intends to remove these suboptions in upcoming releases of the IBM XL compiler runtime environment.

For more information about the `XLSMPOPTS` environment variable, see the XLSMPOPTS section of the online manuals at the following websites:

- XL C/C++ for Linux
- XL Fortran for Linux

GCC compilers

For the OpenMP application that is compiled with GNU Compiler Collection (GCC) compilers, use the `GOMP_CPU_AFFINITY` environment variable. Assign a list of the logical processors that you want to use to the `GOMP_CPU_AFFINITY` environment variable. The syntax and semantics are the same as with the `procs` suboption of the IBM XL compilers `XLSMPOPTS` environment variable. For more information about the `GOMP_CPU_AFFINITY` environment variable, see the corresponding section of the GCC manual.

Support for thread binding in the recent versions of the OpenMP standard

The OpenMP 3.1 revision introduced the `OMP_PROC_BIND` environment variable. The Open MP 4.0 revision introduced the `OMP_PLACES` environment variable. These variables control thread binding and affinity in a similar manner to `XLSMPOPTS` and `GOMP_CPU_AFFINITY`, although their syntax slightly differs.

Performance affect

For more information about the thread binding affect on the performance, see 4.1, "Effects of basic performance tuning techniques" on page 122. An easy-to-use code also is available to generate your own binding map. For more information, see 4.3, "Sample code for the construction of thread affinity strings" on page 145.

3.1.2 Setting and retrieving process affinity at run time

By using the Linux **taskset** command, you can manipulate the affinity of any multithreaded program, even if you do not have access to the source code. You can use the **taskset** command to start a new application with a certain affinity by specifying a mask or a list of logical processors. The Linux scheduler restricts the application threads to a certain set of logical processors only.

You can also use the **taskset** command when an application creates many threads and you want to set the affinity for highly loaded threads only. In this circumstance, identify the process identifiers (PIDs) of highly loaded running threads and perform binding only on those threads. You can discover these threads by examining the output of the **top -H** command.

Knowing the PID, you can also use the **taskset** command to retrieve the affinity of the corresponding entity (a thread or a process).

3.1.3 Controlling NUMA policy for processes and shared memory

By using the **numactl** command, you can specify a set of nodes and logical processors on which you want your application. In the current context, you can assume that this tool defines a *node* as a group of logical processors that are associated with a particular memory controller. For POWER8, such a node is a whole processor.

To discover the indexes of nodes and estimate the memory access penalty, run the **numactl** command with the **-H** argument. Example 3-1 shows the corresponding output for a 20-core IBM Power System S822LC (Model 8335-GTB) server that is running Red Hat Enterprise Linux Server release 7.3 little endian.

Example 3-1 The numactl -H command (truncated) output in a 20-core IBM Power System S822LC

```
$ numactl -H
available: 2 nodes (0,8)
< ... output omitted ... >
node distances:
node   0   8
  0:  10  40
  8:  40  10
```

You can pass these indexes of the nodes to **numactl** as an argument for the **-N** option to bind the process to specific *nodes*. To bind the process to specific *logical processors*, use the **-C** option. In the latter case, the indexes follow the same conventions as the OpenMP environment variables and a **taskset** command.

The memory placement policy significantly affects the performance of technical computing applications. You can enforce a certain policy by the **numactl** command. The **-l** option instructs the operating system to always allocate memory pages on the current node. Use the **-m** option to specify a list of nodes that the operating system can use for memory allocation. You need to use the **-i** option to ask the operating system for a round-robin allocation policy on specified nodes.

3.2 Performance enhancements and scalability tests

This section provides details about performance enhancements and scalability tests.

3.2.1 ESSL execution in multiple CPUs and GPUs

One of the many features of ESSL is the ability to compile the same source to code to run automatically in different CPUs without any parallel programming that is performed by the developer. This feature is demonstrated in Example 3-2, which describes how to compile and execute Example 2-5 on page 38 from Chapter 2 by using the SMP version of ESSL and the XLC compiler as described in 3.1.1, “Running OpenMP applications” on page 90.

Example 3-2 Compilation and execution of dgemm_sample.c for SMP ESSL

```
$ export XLSMPOPTS=parthds=20

$ cc_r -O3 dgemm_sample.c -lesslsmp -lxlf90_r -lxlsmp -lxlfmath
-L/opt/ibm/xlsmp/4.1.5/lib -L/opt/ibm/xlf/15.1.5/lib -R/opt/ibm/lib -o dgemm_smp

$ ./dgemm_smp
99.766 seconds, 161383.202 MFlops
```

To set the number of SMP threads, the environmental flag `XLSMPOPTS` is used. Therefore, 20 CPUs are running at maximum throughput when this feature is used, as shown in Figure 3-1.

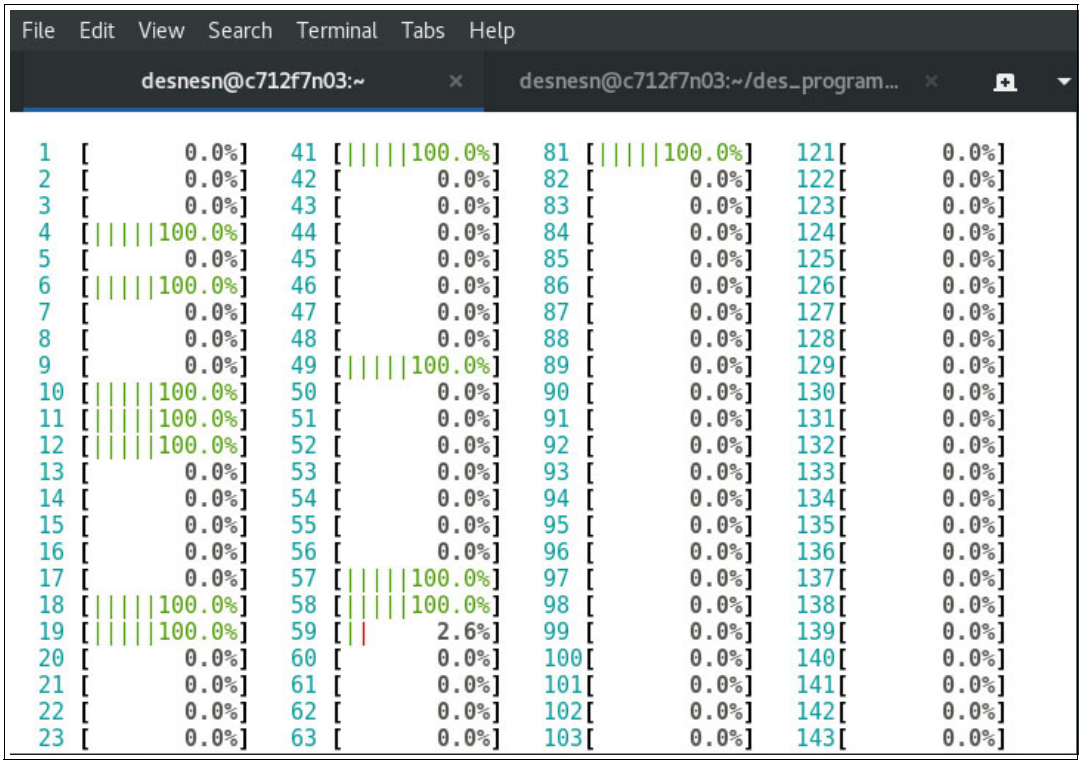


Figure 3-1 Print of htop during execution of a dgemm_smp calculation of order [20,000x20,000]

This result presents a gain of approximately 11x for SMP execution when compared to its serial runs.

Performance can be enhanced further by using the SMP Compute Unified Device Architecture (CUDA) version of ESSL, which enables the use of four GPUs to run the code, as shown on Example 3-3 and in Figure 3-2.

Example 3-3 Compilation and execution of dgemm_sample.c for SMP GPU ESSL

```
$ export XLSMPOPTS=parthds=20

$ cc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/4.1.5/lib -L/opt/ibm/xlf/15.1.5/lib -R/opt/ibm/lib -o dgemm_cuda

$ ./dgemm_cuda
6.933 seconds, 2461471.563 MFlops
```

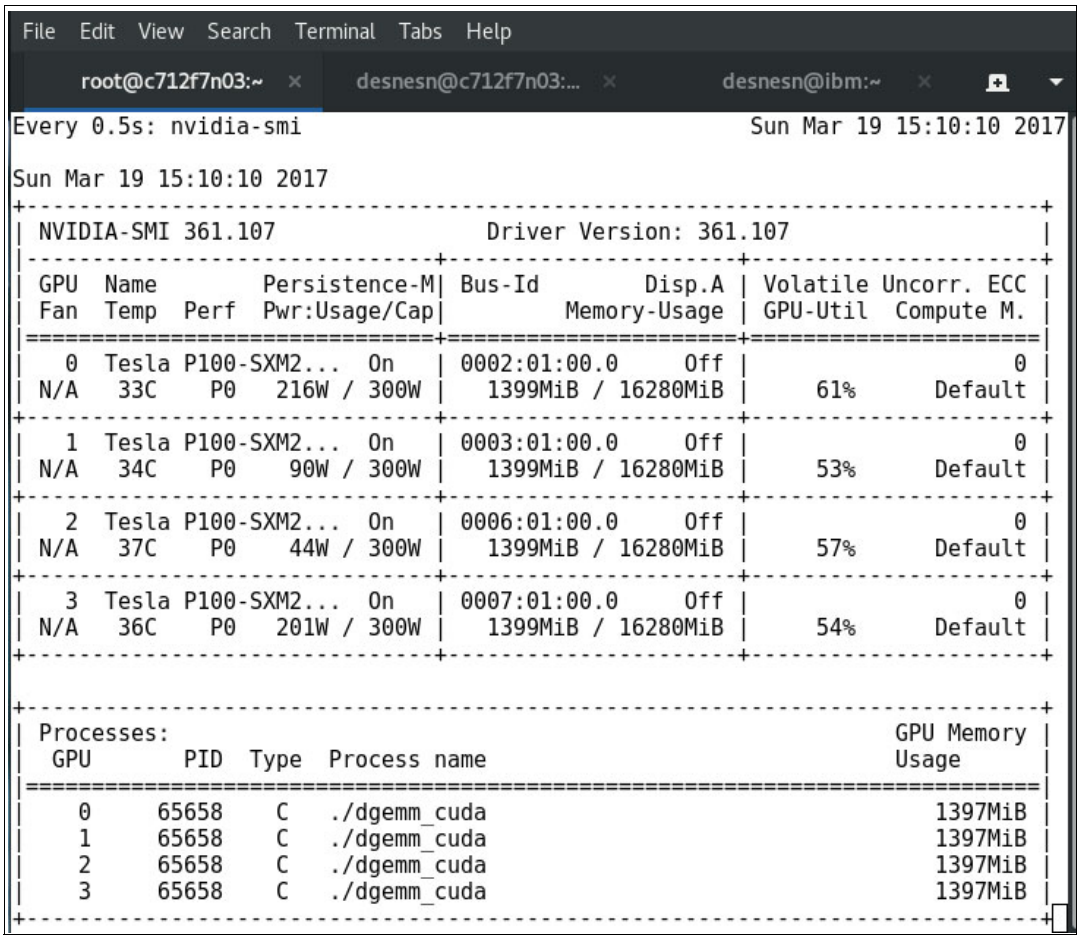


Figure 3-2 A print of nvidia-smi during the execution of dgemm_cuda

A 77x speedup gain can be observed when GPUs are used for this calculation. In addition, the CUDA execution reaches almost 2.5 teraflops, as shown in Figure 3-3.

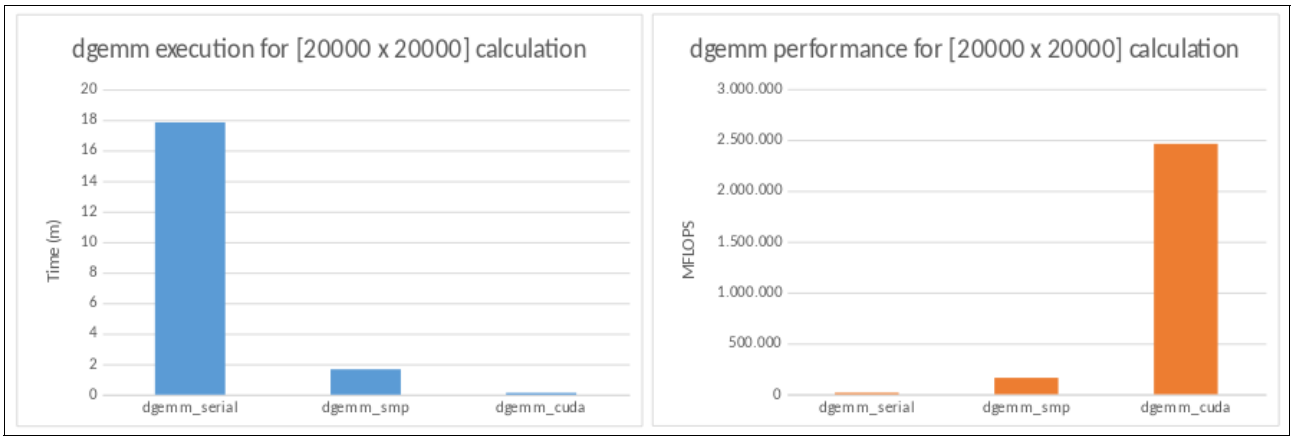


Figure 3-3 Execution and performance of dgemm

Another strategy that was used on older GPUs was the adjustment of the Power Cap Limit, which can improve the performance of parallel software on GPU cards. However, Pascal GPUs include the Power Cap at maximum capacity (see Example 3-4) when compared against Figure 3-3. Therefore, this strategy is obsolete for performance improvement.

Example 3-4 Compilation and execution of dgemm_sample.c for SMP ESSL

```
0$ nvidia-smi -q | grep '^GPU\|Power Limit'
GPU 0002:01:00.0
        Power Limit                 : 300.00 W
        Default Power Limit         : 300.00 W
        Enforced Power Limit        : 300.00 W
        Min Power Limit             : 150.00 W
        Max Power Limit             : 300.00 W
GPU 0003:01:00.0
        Power Limit                 : 300.00 W
        Default Power Limit         : 300.00 W
        Enforced Power Limit        : 300.00 W
        Min Power Limit             : 150.00 W
        Max Power Limit             : 300.00 W
GPU 0006:01:00.0
        Power Limit                 : 300.00 W
        Default Power Limit         : 300.00 W
        Enforced Power Limit        : 300.00 W
        Min Power Limit             : 150.00 W
        Max Power Limit             : 300.00 W
GPU 0007:01:00.0
        Power Limit                 : 300.00 W
        Default Power Limit         : 300.00 W
        Enforced Power Limit        : 300.00 W
        Min Power Limit             : 150.00 W
        Max Power Limit             : 300.00 W
```

For more information about the new ESSL features, see the Engineering and Scientific Subroutine Library page of the IBM Knowledge Center.

Running on different SMT modes

Example 3-2 on page 94 shows how to request several CPUs to the operating system to run a job. You can also use the environment variable `XLSMPOPTS` to control the distribution.

The sample system has 24 physical POWER8 cores and 192 logical CPUs. Each 8 CPUs depend on one physical core. By default, the system is set to SMT-8 mode, which means that all 8 CPUs per core can be used. In the first run that is shown in Example 3-5, the example uses CPUs 0 - 19. In the second run, only even numbers of CPUs are used, which simulates SMT-4 mode. The third and fourth runs work like SMT-2 and SMT-1 modes, respectively.

Example 3-5 Different CPU binding for run with 20 SMP threads

```
$ export
XLSMPOPTS=parthds=20:PROCS=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
$ ./dgemm_smp
810.352 seconds, 19743.519 MFlops

$ export
XLSMPOPTS=parthds=20:PROCS=0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38
$ ./dgemm_smp
312.992 seconds, 51116.961 MFlops

$ export
XLSMPOPTS=parthds=20:PROCS=0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68,72,7
6
$ ./dgemm_smp
231.852 seconds, 69006.090 MFlops

$ export
XLSMPOPTS=parthds=20:PROCS=0,8,16,24,32,40,48,56,64,72,80,88,96,104,112,120,128,13
6,144,152
$ ./dgemm_smp
54.732 seconds, 292318.936 MFlops
```

Figure 3-4 shows the result of `dgemm_smp` with the first export configuration.

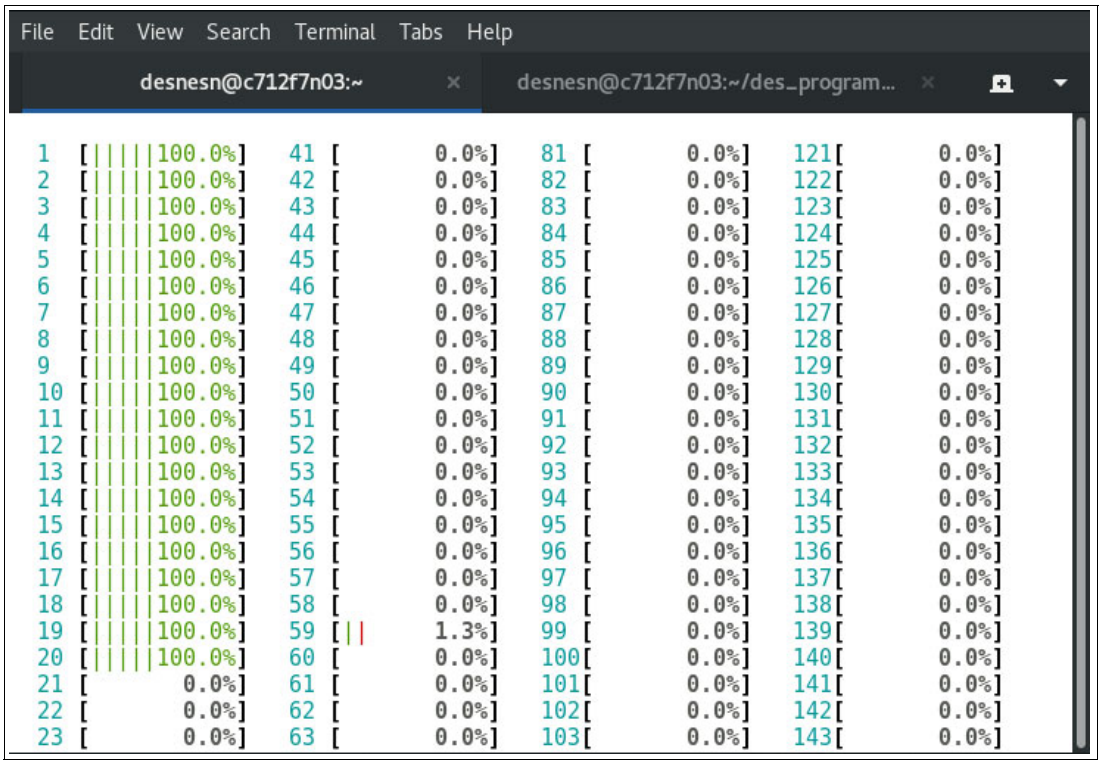


Figure 3-4 Result of dgemm_smp with the first export configuration

For the fourth run with SMT-1 mode, running one thread on each POWER8 core helps to double performance that is compared with runs without CPU binding.

ESSL SMP CUDA library options

The ESSL SMP CUDA library has the following options that are controlled by the user:

- Control which GPUs ESSL uses

  By default, ESSL uses all available devices, but you can change it by using the environment variable `CUDA_VISIBLE_DEVICES` or the `SETGPUS` subroutine. GPUs have numeration from 0 in the operating system. For example, if you want to use only second and third GPUs in your run, set the environment variable as shown in the following example:

  ```
  $ export CUDA_VISIBLE_DEVICES=1,2
  ```

  You also can place the call into the code, as shown in Example 3-6.

Example 3-6 Binding ESSL code to GPUs

```
int ids[2] = {1, 2}; //GPUs IDs
int ngpus = 2; //Number of GPUs
...
setgpus(ngpus, ids);
/*your ESSL SMP CUDA call*/
```

You can also specify a different order of devices. It can be useful in cases when you want to reduce latency between specific CPUs and GPUs.

- Disable or enable hybrid mode

  By default, ESSL runs in hybrid mode. Therefore, ESSL routines use POWER8 CPUs and NVIDIA GPUs. To disable this mode and start using only GPUs, you must specify the following environment variable:

  ```
  export ESSL_CUDA_HYBRID=no
  ```

  To renable it, unset this variable or set it to `yes`.

- Pin host memory buffers

  The following options are provided by ESSL:

  - Not pin host memory buffers (default).
  - Allow ESSL to pin buffers alone. To use this option, set the following environment variable:

    ```
    export ESSL_CUDA_PIN=yes
    ```

  - Provide information to ESSL that you will pin the buffers before the ESSL routines calls:

    ```
    export ESSL_CUDA_PIN=pinned
    ```

Example 3-7 shows runs of source code from Example 2-5 on page 38 with different ESSL SMP CUDA library options. The examples use the adjusted Power Cap Limit.

Example 3-7 The dgemm_sample.c runs with different SMP CUDA options

```
$ export XLSMPOPTS=parthds=20
$ cc_r -O3 dgemm_sample.c -lesslsmpcuda -lxlf90_r -lxlsmp -lxlfmath -lcublas
-lcudart -L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64
-L/opt/ibm/xlsmp/4.1.5/lib -L/opt/ibm/xlf/15.1.5/lib -R/opt/ibm/lib -o dgemm_cuda

$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 4 GPUs hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 4 GPUs non-hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=yes
$ export CUDA_VISIBLE_DEVICES=0,1,2
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=yes
$ export CUDA_VISIBLE_DEVICES=0,1
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=yes
$ export CUDA_VISIBLE_DEVICES=1,2
```

```
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=yes
$ export CUDA_VISIBLE_DEVICES=0
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 1 GPU (1st) hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=yes
$ export CUDA_VISIBLE_DEVICES=3
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 1 GPU (4th) hybrid mode: $VAR"

$ export ESSL_CUDA_HYBRID=no
$ VAR=`./dgemm_cuda`
$ echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode: $VAR"
```

The results of the runs are shown in Example 3-8.

Example 3-8 Result of different ESSL SMP CUDA runs

```
SMP CUDA run with 4 GPUs hybrid mode: 6.965 seconds, 2297085.427 MFlops
SMP CUDA run with 4 GPUs non-hybrid mode: 5.436 seconds, 2943193.525 MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode: 6.118 seconds, 2615102.975
MFlops
SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode: 20.126 seconds,
794951.804 MFlops
SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode: 7.141 seconds, 2240470.522 MFlops
SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode: 6.544 seconds, 2444865.526
MFlops
SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode: 7.375 seconds, 2169383.051 MFlops
SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode: 8.270 seconds, 1934607.013
MFlops
SMP CUDA run with 1 GPU (1st) hybrid mode: 10.700 seconds, 1495252.336 MFlops
SMP CUDA run with 1 GPU (1st) non-hybrid mode: 15.782 seconds, 1013762.514 MFlops
SMP CUDA run with 1 GPU (4th) hybrid mode: 10.469 seconds, 1528245.296 MFlops
SMP CUDA run with 1 GPU (4th) non-hybrid mode: 11.282 seconds, 1418117.355 MFlops
```

In this set of examples, hybrid mode is usually better than non-hybrid. However, this configuration can be useful for cases with significant problems or large number of ESSL runs, where improvement and a slight gain of performance is an excellent advantage.

For more information about the ESSL SMP CUDA library, see the Using the ESSL SMP CUDA Library page of the IBM Knowledge Center website.

3.2.2 OpenACC execution and scalability

This section describes testing the performance of the OpenACC model in our system. Example 2-17 on page 63 is run for an average of three times to create a speedup[2] per block analysis that is shown in Figure 3-5.

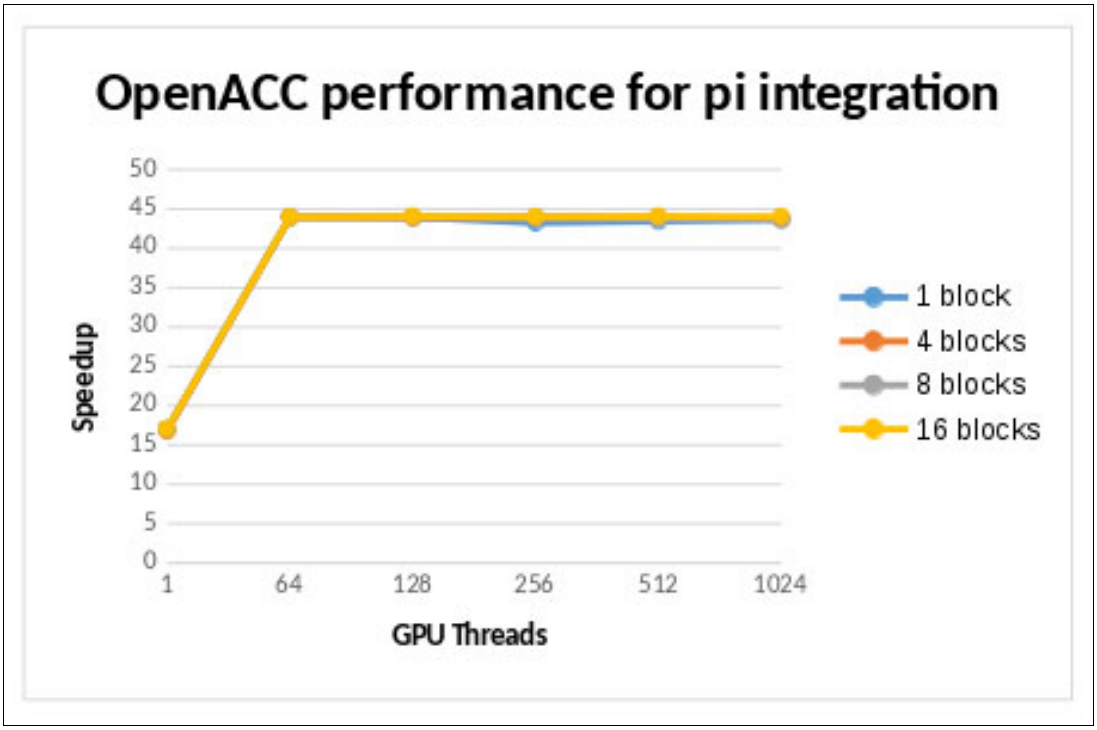


Figure 3-5 Scalability curve from the pi integration in OpenACC in Example 2-17 on page 63

As shown in the graph in Figure 3-5, an instant gain with all block configurations can be observed, which saturates in approximately a 47 speedup with the pi parallelization example by using 1,000,000,000,000 steps. This result likely indicates that the parallel part of this piece of code was reduced to its minimum, which created the speedup threshold that is stated on Ahmdah'ls law[3].

Considering that parallelization occurs only in a for loop where the measurement is performed, it is a fair conclusion that a threshold of maximum performance is achieved.

3.2.3 XL Offload execution and scalability

Figure 3-6 on page 102 shows the scalability curve that is obtained by a different setup of blocks and threads in a P100 GPU. The more that the code is distributed, the more that the speedup increases showing scalability.

[2] The improvement speed of the execution of a fixed task is achieved by increasing computing elements to process that task. It is the ratio between the sequential time of executing a task divided by the time that is taken to execute that task in parallel with a number of processing elements.

[3] Ahmdah's Law is a known calculation in parallel computing of the theoretical speedup that can be achieved when computing a fixed task by using multiple resources. It states that the speedup is limited by the serial part of code; therefore, the programmer seeks to grow the parallel part of code as much as possible.

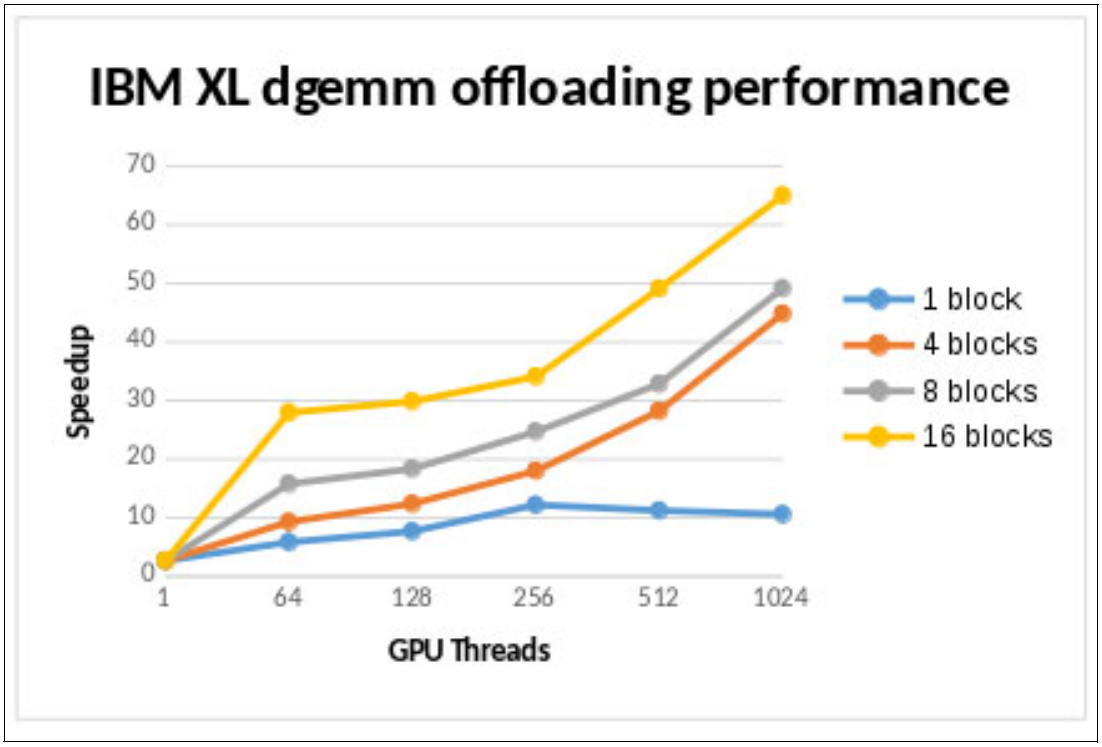


Figure 3-6 Scalability curve from the dgemm average execution

However, the ESSL code that is shown in Example 3-3 on page 95 still out performs this parallelization with its 77x speedup. The example also uses matrixes of [20.000 x 20.000] dimension, outperforming this algorithm in size.

The ESSL API not only performed its calculations in the most optimized way possible, but it used all four GPUs to run that code. To choose which GPU to offload to, the `#pragma omp target device()` call must be used, which splits and shares the parts of a vector to multiple GPUs in our system, as shown in Example 3-9.

Example 3-9 XL C code that process a dynamically allocated vector in multiple GPUs

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <omp.h>
4.
5. #define length 16000000
6. #define verbose 0
7.
8. int main()
9. {
10.         int i, *x, *y, ll, ul, dev, n_dev, chunk;
11.
12.         x = (int*) malloc(sizeof(int)*length);
13.         y = (int*) malloc(sizeof(int)*length);
14.         n_dev = omp_get_num_devices();
15.         chunk = length / n_dev;
16.
17.         printf("n_dev: %d\nchunk: %d\n", n_dev, chunk);
18.
19.         for(i=0;i<length;i++)
```

```
20.        {
21.                x[i]=(i+1);
22.                if(verbose) printf("%d ",x[i]);
23.        }
24.        if(verbose) printf("\n");
25.
26.        for (dev = 0; dev < n_dev; dev++)
27.        {
28.                ll = dev * chunk;
29.                ul = (dev+1) * chunk;
30.                #pragma omp target device(dev) firstprivate(ll, ul) map(to:
   x[ll:chunk]) map(from: y[ll:chunk])
31.                {
32.                        #pragma omp parallel for
33.                        for (int i = ll; i < ul; i++)
34.                        {
35.                                y[i] = 2 * x[i];
36.                        }
37.                }
38.        }
39.
40.        if(verbose)
41.        {
42.                for (i=0; i<length; i++)
43.                {
44.                        printf("%d ",y[i]);
45.                }
46.                printf("\n");
47.        }
48.
49.        return 0;
50.}
```

However, this setup is sharing and running the data asynchronously. Each GPU is being called sequentially in the loop order to execute its data. To have a full heterogeneous system, share and distribute the data synchronously, which can be performed in XL C by offloading to multiple GPUs by multiple CPUs. Completing this configuration is a matter of performing the changes that are shown in Example 3-10.

Example 3-10 XL C code processing a dynamically allocated vector in multiple GPUs synchronously

```
1. --- multiple_gpu.c      2017-01-05 14:56:37.245221037 -0500
2. +++ sync_multiple_gpu.c 2017-01-05 14:58:24.455228579 -0500
3. @@ -23,6 +23,9 @@ int main()
4.         }
5.         if(verbose) printf("\n");
6.
7. +       omp_set_num_threads(n_dev);
8. +
9. +       #pragma omp parallel for private(ll, ul)
10.        for (dev = 0; dev < n_dev; dev++)
11.        {
12.                ll = dev * chunk;
```

For more information about the features of XL offloading, see the product documentation for XL C/C++ for Linux, V13.1.5, for little endian distributions.

For more information about OpenMP, see the OpenMP Application Programming Interface Examples documentation.

3.3 Using IBM Parallel Environment v2.3

This section describes the execution of a parallel application through the IBM Parallel Environment (PE) Runtime.

Note: IBM PE is being deprecated in favor IBM Spectrum MPI, which is a lighter and high-performance implementation of the Message Passing Interface (MPI) Standard.

For more information about some differences, porting, and other related topics, see 2.6.9, “MPI programs with IBM Spectrum MPI” on page 81.

3.3.1 Running applications

The IBM PE provides an environment to manage the execution of parallel applications, and the Parallel Operating Environment (POE) is started with a call to the **poe** command line.

The execution of an application with POE spreads processes across the cluster nodes. Consider the following points:

- Parallel tasks are created on compute nodes.
- One instance of the partition manager daemon (PMD) per compute node, which features tasks of the running application.
- The `poe` process is on the submitting node (home node) machine where it was started.

The PMD controls communication between the home **poe** process and the created tasks in a specific compute node. The PMD is also used to pass the standard input, output, and error streams of the home **poe** to the tasks.

However, if the PMD process exits abnormally, such as with `kill` signals or the **bkill** command of the IBM Spectrum LSF, the shared memory that is used for intra-node message passing cannot get cleaned up properly. In this case, use the **ipcrm** command to reclaim that memory.

The environment works in two modes: interactive and batch. It also allows single program, multiple data (SPMD) and multiple program, multiple data (MPMD) programs.

Many environment variables are in place to control the behavior of **poe**. Some of the variables are described next.

The number of tasks in the parallel application is specified with `MP_PROCS` variable that is equivalent to the `-procs` option of the **poe** command. The application is set with 10 tasks in the following example:

```
$ MP_PROCS=10 poe ./myApp
```

The **poe** command manages the execution of applications that are implemented with different models and eventually mixed. To set the messaging API, set the `MP_MSG_API` variable (equivalent to the `-msg_api` option). The accepted values are `MPI`, `PAMI`, and `shmem`. It does not need to be set for MPI programs because it is the default. In following command, **poe** is called to execute a 10 task OpenSHMEM program:

```
$ MP_PROCS=10 MP_MSG_API=shmem poe ./myApp
```

Compute nodes allocation

The partition manager can connect to an external resources manager that determines the allocation of the compute nodes. For example, the configuration that is described in 8.11, “IBM Spectrum LSF (formerly IBM Platform LSF)” on page 359 can be configured to integrate seamlessly with IBM PE so that hosts are selected by the Spectrum LSF **bsub** command for batch jobs submission.

By default, the allocation uses any resources manager that is configured. However, it can be disabled by setting `MP_RESD` (or `-resd` option). Also, native partition manager nodes allocation mechanism are enabled with `MP_RESD=poe` and require a hosts list file.

Example 3-11 shows what a host list file looks like when resource manager is not used with **poe**. Notice `MP_RESD=poe` is exported to enable the internal host allocation mechanism.

Example 3-11 Run parallel application without resource manager through IBM PE

```
$ cat host.list
! Host list - One host per line
! Tasks 0 and 2 run on p8r1n1 host
! Tasks 1 and 3 run on p8r2n2 host
p8r1n1
p8r2n2
p8r1n1
p8r2n2
$ MP_PROCS=4 MP_RESD=poe poe ./myApp
```

The format and content of the hosts list file changes whether a resource manager is used. For more information, see the IBM PE documentation.

If it needs to point out the host file location, use the `MP_HOSTFILE` variable (same as `-hostfile` option of **poe**). If it is not set, **poe** looks for a file that is named `host.list` in the local directory.

When a resource manager is in place, the system administrators often configure pools of hosts. Therefore, consider the use of the `MP_RMPOOL` variable (or `-rmpool` option) to determine which of the pools of machines were configured (if any) by the administrators to use.

Other variables are available to configure the resource manager behavior. The `MP_NODES` (`-nodes` option) set the number of physical nodes and `MP_TASKS_PER_NODE` (`-tasks_per_node` option) variables set the number of tasks per nodes.

Considerations about network configuration

The IBM PE provides variables to control and configure the use of network adapters by the parallel applications. Some variables might be implicitly set, depending on the combination of other settings or by the use of a resource manager.

To specify the network adapters to use for message passing, set the `MP_EUIDEVICE` variable (or `-euidevice` option). It accepts the value `sn_all` (one or more windows are on each network) or `sn_single` (all windows are on a single network). The `sn_all` value is frequently used to enable protocol stripping, failover, and recovery.

Network adapters can be shared or dedicated. That behavior is defined by using the `MP_ADAPTER_USE` variable (or `-adapter_use` option). It accepts the `shared` and `dedicated` values.

Considerations about Remote Direct Memory Access

The IBM PE implements message passing by using Remote Direct Memory Access (RDMA) through the InfiniBand interconnect. In such a mechanism, memory pages are automatically pinned and buffer transferences are handled directly by the InfiniBand adapter without the host CPU involvement.

RDMA on messaging passing is disabled by default. Export `MP_USE_BULK_XFER=yes` to enable bulk data transfer mechanism. Also, use the `MP_BULK_MIN_MSG_SIZE` variable to set the minimum message length for bulk transfer.

Considerations about affinity

Several levels of affinity are available for parallel applications through **poe**. These levels are also controlled by environment variables (or **poe** options). Because resource managers usually employ their own affinity mechanisms, those variables can be overwritten or ignored.

The primary variable to control placement of message passing interface (MPI) tasks is `MP_TASK_AFFINITY` (or `-task_affinity` option), when a resource manager is not used. You can bind tasks at the physical processor (`core` value), logical CPU (`cpu` value), and multi-chip module (`mcm` value) levels.

For example, the following command allocates one core per task:

```
$ poe -task_affinity core -procs 2 ./myApp
```

More examples of `MP_TASK_AFFINITY` being used to control task affinity are described in 2.6.5, “MPI programs with IBM Parallel Environment v2.3” on page 70.

The `MP_CPU_BIND_LIST` (or `-cpu_bind_list` option) and `MP_BIND_MODE` (or `-bind_mode spread` option) environment variables can be used with `MP_TASK_AFFINITY` to further control the placement of tasks. `MP_CPU_BIND_LIST` specifies a list of processor units for establishing task affinity. In the following command, affinity is restricted to only the second core of each socket:

```
$ poe -task_affinity core -cpu_bind_list 0/16,8/1040 -procs 2
```

When a resource manager is used, the `MP_PE_AFFINITY` variable can be set to `yes` so that **poe** assumes control over affinity. However, if IBM Spectrum Load Sharing Facility (LSF) is used and it set the affinity, **poe** honors the allocated CPUs. If `MP_PE_AFFINITY=yes` is enabled in Spectrum LSF batch jobs, it enables the InfiniBand adapter affinity.

To assist with defining affinity, the IBM PE run time provides the **cpuset_query** command that displays information about current assignments of a running program. It also provides the topology of any specific compute node. As shown in Example 3-12 on page 107, **cpuset_query -t** displays the topology of an IBM S822LC system that is running on SMT-8 mode with two sockets with 10 cores each with eight hardware threads.

Example 3-12 The cpuset_query command to show the node topology

```
$ cpuset_query -t
MCM(Socket): 0
         CORE: 8
                  CPU: 0
                  CPU: 1
                  CPU: 2
                  CPU: 3
                  CPU: 4
                  CPU: 5
                  CPU: 6
                  CPU: 7
         CORE: 16
                  CPU: 8
                  CPU: 9
                  CPU: 10
                  CPU: 11
                  CPU: 12
                  CPU: 13
                  CPU: 14
                  CPU: 15
<... Output Omitted ...>
MCM(Socket): 8
         CORE: 1032
                  CPU: 80
                  CPU: 81
                  CPU: 82
                  CPU: 83
                  CPU: 84
                  CPU: 85
                  CPU: 86
                  CPU: 87
         CORE: 1040
                  CPU: 88
                  CPU: 89
                  CPU: 90
                  CPU: 91
                  CPU: 92
                  CPU: 93
                  CPU: 94
                  CPU: 95
         CORE: 1048
<... Output Omitted ...>
```

The affinity can be checked by running the **cpuset_query** command through **poe**, as shown in Example 3-13. The command shows a two tasks program that is started with affinity at a core level. Each task is allocated (see CPUs with value 1) with a full core that has eight hardware threads on SMT-8 mode node.

Example 3-13 cpuset_query command to show task affinity

```
MCM/QUAD(0) contains:
cpu0, cpu1, cpu2, cpu3, cpu4,
cpu5, cpu6, cpu7, cpu8, cpu9,
cpu10, cpu11, cpu12, cpu13, cpu14,
cpu15, cpu16, cpu17, cpu18, cpu19,
cpu20, cpu21, cpu22, cpu23, cpu24,
cpu25, cpu26, cpu27, cpu28, cpu29,
cpu30, cpu31, cpu32, cpu33, cpu34,
cpu35, cpu36, cpu37, cpu38, cpu39,
cpu40, cpu41, cpu42, cpu43, cpu44,
cpu45, cpu46, cpu47, cpu48, cpu49,
cpu50, cpu51, cpu52, cpu53, cpu54,
cpu55, cpu56, cpu57, cpu58, cpu59,
cpu60, cpu61, cpu62, cpu63, cpu64,
cpu65, cpu66, cpu67, cpu68, cpu69,
cpu70, cpu71, cpu72, cpu73, cpu74,
cpu75, cpu76, cpu77, cpu78, cpu79,
[Total cpus for MCM/QUAD(0)=80]
MCM/QUAD(8) contains:
cpu80, cpu81, cpu82, cpu83, cpu84,
cpu85, cpu86, cpu87, cpu88, cpu89,
cpu90, cpu91, cpu92, cpu93, cpu94,
cpu95, cpu96, cpu97, cpu98, cpu99,
cpu100, cpu101, cpu102, cpu103, cpu104,
cpu105, cpu106, cpu107, cpu108, cpu109,
cpu110, cpu111, cpu112, cpu113, cpu114,
cpu115, cpu116, cpu117, cpu118, cpu119,
cpu120, cpu121, cpu122, cpu123, cpu124,
cpu125, cpu126, cpu127, cpu128, cpu129,
cpu130, cpu131, cpu132, cpu133, cpu134,
cpu135, cpu136, cpu137, cpu138, cpu139,
cpu140, cpu141, cpu142, cpu143, cpu144,
cpu145, cpu146, cpu147, cpu148, cpu149,
cpu150, cpu151, cpu152, cpu153, cpu154,
cpu155, cpu156, cpu157, cpu158, cpu159,
[Total cpus for MCM/QUAD(8)=80]

Total number of MCMs/QUADs found  = 2
Total number of COREs found       = 20
Total number of CPUs found        = 160
cpuset for process 95014 (1 = in the set, 0 = not included)
cpu0  = 1
cpu1  = 1
cpu2  = 1
cpu3  = 1
cpu4  = 1
cpu5  = 1
cpu6  = 1
cpu7  = 1
```

```
cpu8  = 0
cpu9  = 0
cpu10 = 0
cpu11 = 0
cpu12 = 0
cpu13 = 0
cpu14 = 0
cpu15 = 0
<... Output Omitted ...>
MCM/QUAD(0) contains:
cpu0, cpu1, cpu2, cpu3, cpu4,
cpu5, cpu6, cpu7, cpu8, cpu9,
cpu10, cpu11, cpu12, cpu13, cpu14,
cpu15, cpu16, cpu17, cpu18, cpu19,
cpu20, cpu21, cpu22, cpu23, cpu24,
cpu25, cpu26, cpu27, cpu28, cpu29,
cpu30, cpu31, cpu32, cpu33, cpu34,
cpu35, cpu36, cpu37, cpu38, cpu39,
cpu40, cpu41, cpu42, cpu43, cpu44,
cpu45, cpu46, cpu47, cpu48, cpu49,
cpu50, cpu51, cpu52, cpu53, cpu54,
cpu55, cpu56, cpu57, cpu58, cpu59,
cpu60, cpu61, cpu62, cpu63, cpu64,
cpu65, cpu66, cpu67, cpu68, cpu69,
cpu70, cpu71, cpu72, cpu73, cpu74,
cpu75, cpu76, cpu77, cpu78, cpu79,
[Total cpus for MCM/QUAD(0)=80]
MCM/QUAD(8) contains:
cpu80, cpu81, cpu82, cpu83, cpu84,
cpu85, cpu86, cpu87, cpu88, cpu89,
cpu90, cpu91, cpu92, cpu93, cpu94,
cpu95, cpu96, cpu97, cpu98, cpu99,
cpu100, cpu101, cpu102, cpu103, cpu104,
cpu105, cpu106, cpu107, cpu108, cpu109,
cpu110, cpu111, cpu112, cpu113, cpu114,
cpu115, cpu116, cpu117, cpu118, cpu119,
cpu120, cpu121, cpu122, cpu123, cpu124,
cpu125, cpu126, cpu127, cpu128, cpu129,
cpu130, cpu131, cpu132, cpu133, cpu134,
cpu135, cpu136, cpu137, cpu138, cpu139,
cpu140, cpu141, cpu142, cpu143, cpu144,
cpu145, cpu146, cpu147, cpu148, cpu149,
cpu150, cpu151, cpu152, cpu153, cpu154,
cpu155, cpu156, cpu157, cpu158, cpu159,
[Total cpus for MCM/QUAD(8)=80]

Total number of MCMs/QUADs found  = 2
Total number of COREs found       = 20
Total number of CPUs found        = 160
cpuset for process 95015 (1 = in the set, 0 = not included)
cpu0  = 0
cpu1  = 0
cpu2  = 0
cpu3  = 0
cpu4  = 0
```

```
cpu5  = 0
cpu6  = 0
cpu7  = 0
cpu8  = 1
cpu9  = 1
cpu10 = 1
cpu11 = 1
cpu12 = 1
cpu13 = 1
cpu14 = 1
cpu15 = 1
cpu16 = 0
cpu17 = 0
cpu18 = 0
cpu19 = 0
cpu20 = 0
cpu21 = 0
cpu22 = 0
cpu23 = 0
<... Output Omitted ...>
```

Considerations about CUDA-aware MPI

The IBM PE run time implements a CUDA-aware MPI mechanism, but it is disabled by default. Use the `MP_CUDA_AWARE=yes` variable to enable it. For more information, see 2.6.6, “Hybrid MPI and CUDA programs with IBM Parallel Environment” on page 75.

3.3.2 Managing application

IBM PE Runtime provides a set of commands to manage stand-alone **poe** jobs. This section introduces the most common commands.

Canceling an application

Because the **poe** command handles the signal of all tasks in the partition, sending an interrupt (SIGINT) or terminate (SIGTERM) signal triggers it to all remote processes. If **poe** runs with 100413 process ID, you can end the program by using the following command:

```
$ kill -s SIGINT 100413
```

However, if some remote processes are orphan, use the **poekill** *program_name* to end all remaining tasks. In fact, **poekill** can send any signal to all the remote processes.

Suspend and resume a job

To cancel a **poe** process, suspend and resume applications by way of signals. Use a **poekill** or **kill** command to send a `SIGTSTP` to suspend the **poe** process (which triggers the signal to all tasks).

The application can be resumed by sending a `SIGCONT` to continue **poe**. Use a **poekill**, **kill**, **fg**, or **bg** command to deliver the signal.

3.3.3 Running OpenSHMEM programs

The following environment variables can be set to run an OpenSHMEM program with the IBM PE run time through a **poe** command:

- `MP_MSG_API=shmem`

  Instructs **poe** to use `openSHMEM` message passing API.

- `MP_USE_BULK_XFER=yes`

  Enables the use of RDMA in Parallel Active Messaging Interface (PAMI).

`MP_PROCS=<num>` can be used to set the number of processing elements.

NAS Parallel Benchmarks with OpenSHMEM

The NAS Parallel Benchmarks[4] (NPBs) are a well-known suite of parallel applications that are often used to evaluate high-performance computers. The benchmark provides programs, kernels, and problem solvers that simulate aspects, such as computation, data movement, and I/O of real scientific applications. You can select the size of the workload that each benchmark processes among a list of classes (A, B, C, and so on).

A version of NPB that is rewritten by using `OpenSHMEM` in C and Fortran was released by the openshmem.org group. NPB3.2-SHMEM is the implementation of NPB 3.2 and provides some benchmarks in Fortran and only one in C, as shown in Example 3-14.

Example 3-14 The openshmen-npbs implementation

```
$ git clone https://github.com/openshmem-org/openshmem-npbs
$ cd openshmem-npbs/C
$ cd config
$ cp suite.def.template suite.def
$ cp make.def.template make.def
# Set CC in make.def
$ cd ../
$ make is NPROCS=2 CLASS=A
make suite
$ cd bin
$ ls
host.list  is.A.2
[developer@redbook01 bin]$ MP_RESD=poe oshrun -np 2 ./is.A.2
```

This section uses the Integer Sort kernel implementation of NPB3.2-SHMEM to demonstrate the use of OpenSHMEM with IBM PE, and the affect of some configurations on the performance of the application.

The Integer Sort benchmark was compiled by using the **oshcc** compiler script of the IBM PE and the IBM XL C compiler.

The workload class C of the Integer Sort benchmark was run as shown in Example 3-14.

[4] For more information about NAS Parallel Benchmarks, see http://www.nas.nasa.gov/publications/npb.html.

3.4 Using the IBM Spectrum LSF

The IBM Spectrum LSF is a load and a resources manager that allows shared access to cluster hosts while maximizing occupancy and the efficient use of resources.

Spectrum LSF provides a queue-based and policy-driven scheduling system for a user's batch jobs that employs mechanisms to optimize resource selection and allocation based on the requirements of the application.

All development models that are described in Chapter 2, "Compilation, execution, and application development" on page 23 are fully supported by Spectrum LSF. The preferred way to run applications in a production cluster is by using the Spectrum LSF job submission mechanism. Also, you can manage any job. This section describes how to submit and manage jobs by using Spectrum LSF commands.

3.4.1 Submit jobs

This section describes the job submission process. For more information about the tools that are used to monitor jobs and queues, see Chapter 6, "Cluster monitoring and health checking" on page 289.

To submit a job to Spectrum LSF, use the **bsub** command. Spectrum LSF allows you to submit by using command-line options, interactive command-line mode, or a control file. The tool provides the following options that allows fine-grained job management:

- Control input and output parameters
- Define limits
- Specify submission properties
- Notify users
- Control scheduling and dispatch
- Specify resources and requirements

The simplest way to submit a job is to use the command-line options, as shown in Example 3-15.

Example 3-15 Spectrum LSF bsub command to submit a job by using command-line options

```
$ bsub -o %J.out -e %J.err -J 'omp_affinity' -q short './affinity'
Job <212> is submitted to queue <short>.
```

Example 3-15 shows some basic options of the **bsub** command. The `-o`, `-i`, and `-e` flags specify standard output, input, and error files, respectively. As shown in Example 3-15, `-J` sets the job name, but it can also be used to submit multiple jobs (also know as an *array of jobs*). and `-q` sets the queue of which it can be a part. If the `-q` flag is not specified, the default queue is used (usually the `normal` queue). The last option that is shown in Example 3-15 is the application to be run.

The use of the shell script is convenient when you must submit jobs regularly or that require many parameters. The file content that is shown in Example 3-16 on page 113 is a regular shell script that embodies special comments (lines starts with `#BSUB`) to control the behavior of the **bsub** command, and runs the **noop** application by using the `/bin/sh` interpreter.

Example 3-16 Shell script to run Spectrum LSF batch job

```
#!/bin/sh

#BSUB -o %J.out -e %J.err
#BSUB -J serial

./noop
```

The **bsub myscript** or **bsub < myscript** commands can be issued to submit a script to Spectrum LSF. In the first case, `myscript` is not spooled, which means that changes on the script takes effect if the job is still executing. On the other side, **bsub < myscrypt** (see Example 3-17) spools the script.

Example 3-17 Spectrum LSF bsub command to submit a script job

```
$ bsub < noop_lsf.sh
Job <220> is submitted to default queue <normal>.
```

Considerations for OpenMP programs

You can use environment variables to control the execution of OpenMP applications as described in 3.1.1, “Running OpenMP applications” on page 90. By default, the **bsub** command propagates all variables on the submitting host to the environment on the target machine.

Example 3-18 shows some OpenMP control variables (`OMP_DISPLAY_ENV`, `OMP_NUM_THREADS`, and `OMP_SCHEDULE`), which are exported on the environment before a job is scheduled to run on the `p8r2n2` host. The content of `230.err` file, where errors are logged, shows that those variables are propagated on the remote host.

Example 3-18 Exporting OpenMP variables to Spectrum LSF bsub command

```
$ export OMP_DISPLAY_ENV=true
$ export OMP_NUM_THREADS=20
$ export OMP_SCHEDULE="static"
$ bsub -m "p8r2n2" -o %J.out -e %J.err ./affinity
Job <230> is submitted to default queue <normal>.
$ cat 230.err

OPENMP DISPLAY ENVIRONMENT BEGIN
  _OPENMP = '201307'
  OMP_DYNAMIC = 'FALSE'
  OMP_NESTED = 'FALSE'
  OMP_NUM_THREADS = '20'
  OMP_SCHEDULE = 'STATIC'
  OMP_PROC_BIND = 'FALSE'
  OMP_PLACES = ''
  OMP_STACKSIZE = '70368222510890'
  OMP_WAIT_POLICY = 'PASSIVE'
  OMP_THREAD_LIMIT = '4294967295'
  OMP_MAX_ACTIVE_LEVELS = '2147483647'
  OMP_CANCELLATION = 'FALSE'
  OMP_DEFAULT_DEVICE = '0'
OPENMP DISPLAY ENVIRONMENT END
```

If you do not want to export OpenMP environment variables, the **-env** option can be used to control how **bsub** propagates them. For example, the same results that are shown in Example 3-18 can be achieved by using the following command, but without exporting any variables:

```
$ bsub -m "p8r2n2" -o %J.out -e %J.err -env "all, OMP_DISPLAY_ENV=true,
OMP_NUM_THREADS=20,  OMP_SCHEDULE='static'" ./affinity
```

Example 3-19 shows how the environment variables can be set in a job script to control the OpenMP behavior.

Example 3-19 Exporting OpenMP variables to Spectrum LSF job script

```
#!/bin/bash

#BSUB -J "openMP example"
#BSUB -o job_%J.out -e job_%J.err
#BSUB -q short
#BSUB -m p8r2n2

export OMP_NUM_THREADS=20
export OMP_SCHEDULE=static
export OMP_DISPLAY_ENV=true

./affinity
```

Considerations for MPI programs

Use the **-n** option of the **bsub** command to allocate the number of tasks (or job slots) for the parallel application. Depending on the configuration of Spectrum LSF, job slots can be set in terms of CPUs in the cluster. For example, the following command submits an MPI job with six tasks:

```
$ bsub -n 6 -o %J.out -e %J.err poe ./helloMPI
```

You can select a set of hosts for a parallel job by using the following **bsub** command options:

- Use the `-m` option to select hosts or groups of hosts.
- Resources-based selection with requirements expressions is done by using the `-R` option.
- Indicate a host file by using the `-hostfile` option. Do not use with `-m` or `-R` options.

The following examples show the use of `-m` and `-R`, respectively, to allocate hosts. In the following example, run two tasks of an MPI application on hosts `p8r1n1` and `p8r2n2`:

```
$ bsub -n 2 -m "p8r1n1! p8r2n2" -o %J.out -e %J.err poe ./myAPP
```

In the following example, the "`!`" symbol indicates that **poe** is first executed on `p8r1n1`:

```
$ bsub -n 4 -R "select[ncores==20] same[cpuf]" -o %J.out -e %J.err poe ./myApp
```

Run four tasks of an MPI application on hosts with 20 CPU cores (`select[ncores==20]`) if they have same CPU factor (`same[cpuf]`).

The job locality can be specified with the `span` string in a resources requirement expression (`-R` option). One of the following formats can be used to specify the locality:

- `span[hosts=1]`: Set to run all tasks on same host
- `span[ptile=`*`n`*`]`: Where *n* is an integer that sets the number of tasks per host
- `span[block=`*`size`*`]`: Where *size* is an integer that sets the block size

Job task affinity is enabled by using the `affinity` string in the resources requirement expression (`-R` option). The affinity applies to CPU or memory, and is defined in terms of processor units that are assigned per task (`core`, `numa`, `socket`, and `task`). The following example features a 10 task MPI application, allocated five per host, and each with four designated cores with binding by threads:

```
$ bsub -n 10 -R "select[ncores >= 20] span[ptile=5]
affinity[core(4):cpubind=thread]"  -o %J.out -e %J.err
```

Further processor unit specification makes the affinity expression powerful. For more information about affinity expressions in Spectrum LSF, see the Affinity string page of the IBM Knowledge Center website.

Spectrum LSF integrates well with the IBM PE Runtime Edition and supports the execution of parallel applications through **poe**. Some configurations in Spectrum LSF are required. For more information, see 5.6.14, “IBM Spectrum LSF” on page 266.

Example 3-20 includes a Spectrum LSF job script to execute an MPI program with **poe**. (Notice that the IBM PE environment variables are going to take effect.)

Example 3-20 Spectrum LSF job script to submit a simple IBM PE application

```
#!/bin/bash

#BSUB -J "MPILocalRank"    # Set job name
#BSUB -o lsf_job-%J.out    # Set output file
#BSUB -e lsf_job-%J.err    # Set error file
#BSUB -q short             # Set queue
#BSUB -n 5                 # Set number of tasks

export MP_INFOLEVEL=1
export LANG=en_US
export MP_RESD=POE
poe ./a.out
```

Spectrum LSF provides a native network-aware scheduling of the IBM PE parallel application through the `-network` option of the **bsub** command. That option encloses the attributes that are listed in Table 3-2.

Table 3-2 Spectrum LSF bsub network attributes for IBM PE

Attribute	Description	Values
`type`	Manage network windows reservation	► `sn_single` (reserves windows from one network for each task) ► `sn_all` (reserve windows from all networks for each task)
`protocol`	Set the messaging API in use	► `mpi` ► `shmem` ► `pami`
`mode`	The network type	► `US` ► `IP`

Attribute	Description	Values
usage	The network adapter usage among processes	► shared ► dedicated
instance	Number of instances for reservation window	Positive integer number

The following command submits a two tasks (-n 2) MPI (protocol=mpi) job. It shares the network adapters (usage=shared) and reserves windows on all of them:

```
$ bsub -n 2 -R "span[ptile=1]" -network "protocol=mpi:type=sn_all:
instances=2:usage=shared" poe ./myApp
```

Considerations for CUDA programs

You can use graphics processing unit (GPU) resources mapping when requirements expressions are used to allocate hosts and express usage reservation. If Spectrum LSF is configured (see 5.6.14, “IBM Spectrum LSF” on page 266), the following resource fields are available:

- ngpus: Total number of GPUs
- ngpus_shared: Number of GPUs in share mode
- ngpus_excl_t: Number of GPUs in exclusive thread mode
- ngpus_excl_p: Number of GPUs in exclusive process mode

The ngpus resource field can be used in requirement expressions (-R option) for demanding the number of GPUs that are needed to execute a CUDA application. The following command submits a job to any host with one or more GPU:

```
$ bsub -R "select [ngpus > 0]" ./cudaApp
```

In terms of usage, it can reserve GPUs by setting *ngpus_shared* (number of shared), *ngpus_excl_t* (number of GPUs on exclusive thread mode), *ngpus_excl_p* (number of GPUs on exclusive process mode) resources. Use the rusage with -R option of the **bsub** command to reserve GPU resources.

In Example 3-21, the job script sets one GPU in shared mode to be used by a cudaCUDA application.

Example 3-21 Script to set one GPU in shared mode to be used by a cudaCUDA application

```
#!/bin/bash

#BSUB -J "HelloCUDA"
#BSUB -o helloCUDA_%J.out -e helloCUDA_%J.err
#BSUB -R "select [ngpus > 0] rusage [ngpus_shared=1]"

./helloCUDA
```

Regarding exclusive use of GPUs by parallel applications, set the value of ngpus_excl_t or ngpus_excl_p to change run mode properly. The following example executes a parallel two tasks (-n 2) application in one host (span[hosts=1]), where each host reserves two GPUs on exclusive process mode (rusage[ngpus_excl_p=2]):

```
$ bsub -n 2 -R “select[ngpus > 0] rusage[ngpus_excl_p=2] span[hosts=1]” poe
./mpi-GPU_app
```

Considerations for OpenSHMEM

The `-network` option of the **bsub** command can set the parallel communication protocol of the application. For an OpenSHMEM application, it can be set by using the following command:

```
$ bsub -n 10 -network "protocol=shmem" poe ./shmemApp
```

3.4.2 Manage jobs

Spectrum LSF provides a set of commands to manage batch jobs. This section introduces the most common commands.

Modifying a job

The batch job can assume several statuses throughout its lifecycle. The following statuses are most common:

- PEND: Waiting to be scheduled status
- RUN: Running status
- PSUSP, USUSP, or SSUSP: Suspended status
- DONE or EXIT: Terminated status

Submission parameters of a job can be modified in pending or running status. To change it, use the **bmod** command.

Most submission parameters can be changed by using the **bmod** command. The command can include an option to cancel the option, reset the option to its default value, or override the option.

To override a submission parameter, use the same option as in **bsub**. In the following example, `-o "%J_noop.out"` changes the output file of the job with identifier 209:

```
$ bmod -o "%J_noop.out" 209
Parameters of job <209> are being changed
```

To cancel a submission parameter, append *n* to the option. For example, **-Mn** removes the memory limits.

For more information about the **bmod** command, see the bmod page of the IBM Knowledge Center website.

Canceling a job

To cancel one or more jobs, use the **bkill** command. This command by default sends the `SIGINT`, `SIGTERM`, and `SIGKILL` signals in sequence on Linux. The time interval can be configured in the `lsb.params` configuration file. In reality, **bkill -s <*signal*>** sends the <*signal*> signal to the job.

The user can cancel their own jobs only. The root and Spectrum LSF administrators can end any job.

In the following example, the job with identifier 217 is ended:

```
$ bkill 217
Job <217> is being terminated
```

Suspend and resume a job

To suspend one or more unfinished jobs, use the **bstop** command. In Linux, it sends the `SIGSTOP` signal to serial jobs and the `SIGTSTP` signal to parallel jobs). As an alternative, the **bkill -s SIGSTOP** or the **bkill -s SIGTSTP** commands send the stop signal to a job.

In Example 3-22, the job with identifier 220 did not finish (status RUN) when it is stopped (bstop 220). As a result, bjobs 220 shows it is now in a suspended status (USUSP).

Example 3-22 Spectrum LSF bstop command to suspend a job

```
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers RUN   normal     p8r1n1      p8r2n2      serial     Apr 17 16:06
$ bstop 220
Job <220> is being stopped
$ bjobs 220
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
220     wainers USUSP normal     p8r1n1      p8r2n2      serial     Apr 17 16:06
```

To resume a job, use the **bresume** command. In Linux, it sends a SIGCONT signal to the suspended job. To stop or end a job, **bkill -s** can be used to send the continue signal **bkill -s CONT**. The following command shows how to resume the job that was stopped in Example 3-22:

```
$ bresume 220
Job <220> is being resumed
```

3.5 Running tasks with IBM Spectrum MPI

In 2.6.11, “Using Spectrum MPI” on page 83, the use of Spectrum MPI to run jobs in a compute node through the **mpirun** command is described. This section shows several other features of running Spectrum MPI jobs through **mpirun**.

Portable Hardware Locality (hwloc)

A useful API to know the physical and logical architecture of your POWER8 system is the hwloc (Portable Hardware Locality). This API aids to identify and display information about NUMA memory nodes, sockets, shared caches, cores, simultaneous multithreading, system attributes, and the locality of I/O.

For example, through hwloc you can use the **--report-bindings** on the **mpirun** command with any Spectrum MPI program to fetch a hardware description, as shown in Example 3-23.

Example 3-23 Usage of --report-bindings in the mpirun command

```
$ [desnesn@c712f7n03 examples]$ /opt/ibm/spectrum_mpi/bin/mpirun -np 1
--report-bindings -pami_noib ./trap.mpi
[c712f7n03:66785] MCW rank 0 bound to socket 0[core 0[hwt 0-7]], socket 0[core
1[hwt 0-7]], socket 0[core 2[hwt 0-7]], socket 0[core 3[hwt 0-7]], socket 0[core
4[hwt 0-7]], socket 0[core 5[hwt 0-7]], socket 0[core 6[hwt 0-7]], socket 0[core
7[hwt 0-7]], socket 0[core 8[hwt 0-7]], socket 0[core 9[hwt 0-7]]:
[BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/
BBBBBBBB][......../......../......../......../......../......../......../
......../........]

Estimate of local_sum of x^2 = 18.000229
Interval [-3.000,3.000]
Using 10000 trapezoids
```

The last line that is presented, for example, `[BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB/BBBBBBBB][......../......../......../......../......../......../......../......../......../......../........]` means that two sockets are available, each with 10 cores, and each core with 8 hyper-threads. This configuration is compliant with the run of **`nproc`**, which presents a value of 160. Also, the sets of the letter B means that the MPI processes are bound to the first socket.

GPU support

Another valuable feature of the IBM Spectrum MPI is the support of running GPU-accelerated applications over CUDA-aware MPI by using the CUDA Toolkit version 8.0.

Note: By default, GPU support is turned off. To turn it on, use the **`-gpu`** flag on the **`mpirun`** command.

Sharing work with different hosts in your network

To distribute instances of a Spectrum MPI program through **`mpirun`**, use the following command:

```
$ mpirun -host h1,h2 prog1
```

This command runs one instance of prog1 on host h1 and host h2.

However, **`mpirun`** can be used to run jobs in a Single Program, Multiple Data (SPMD) manner. For example, if you want host h2 to receive three instances of prog1, **`mpirun`** is run as shown in the following command:

```
$ mpirun -host h1, h2, h2, h2 prog1
```

SPMD can also be performed by using a host file, as shown in Example 3-24.

Example 3-24 Performing SPMD by using a host file

```
$ cat hosts
c712f7n02.somedomain.ibm.com
c712f7n03.somedomain.ibm.com
$ mpirun -np 2 --host file hosts prog2
```

Also, **`mpirun`** can be used to run jobs in a Multiple Program, Multiple Data (MPMD) manner, which can be performed by using a host file as shown in the following command:

```
$ mpirun -np 2 prog3 : -np 3 prog4
```

This command includes two instances of prog3 and three instances of prog4.

For more information about how to use **`mpirun`**, use the **`-h`** (help) option with the command, or see the IBM Spectrum MPI Version 10 Release 1.0.2 User's Guide.

4

Measuring and tuning applications

This chapter describes how to measure and tune applications.

This chapter includes the following topics:

4.1 Effects of basic performance tuning techniques

This section evaluates the effects of basic performance tuning techniques. It uses the NAS Parallel Benchmarks (NPB)[1] suite of applications as an example.

The NPB suite was originally used for complex performance evaluation of supercomputers. The developers of the NPB programs distilled the typical computational physics workloads and put the most widespread numerical kernels into their product.

This section uses the OpenMP types of NPB benchmarks to achieve the following goals:

- Show the performance variation for the different SMT modes
- Provide guidance for the choice of compilation parameters
- Demonstrate the importance of binding threads to logical processors

The benchmarking used a 20-core IBM Power System S822LC (model 8335-GTB) based on the POWER8 processor. The processor base frequency was 2.86 GHz. The Linux scaling governors were set to `performance` so that the effective frequency increased to 3.93 GHz.

Each of eight memory riser cards of the system was populated with four 8 GB RAM modules for a total of 256 GB. The server was running Red Hat Enterprise Linux operating system version 7.3 (little-endian). The operating system was installed in a non-virtualized mode. The Linux kernel version was 3.10.0-514. Version v15.1.5 of IBM XL Fortran compiler was used to compile the sources.

Note: The performance numbers that are shown in this section must not be treated as the official results. They are provided to demonstrate the possible effect of various performance tuning techniques on application performance. The results that are obtained in different hardware and software environments or with other compilation parameters can vary widely from the numbers that are shown here.

The size of the problems in the NPB benchmarking suite is predefined by the developers. The example uses the benchmarks of class C. The source code of the NPB suite was not changed. The code generation was controlled by setting compilation parameters in a `make.def` file, as shown in Example 4-1.

Example 4-1 NPB: A sample make.def file for the -O3 parameter set

```
F77 = xlf_r -qsmp=noauto:omp -qnosave
FLINK = $(F77)
FFLAGS = -O3 -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
FLINKFLAGS = $(FFLAGS)
CC = xlc_r -qsmp=noauto:omp
CLINK = $(CC)
C_LIB = -lm
CFLAGS = $(FFLAGS)
CLINKFLAGS = $(CFLAGS)
UCC = xlc_r
BINDIR = ../O3
RAND = randi8
WTIME = wtime.c
MACHINE = -DIBM
```

[1] The NPB suite was developed by NASA Advanced Supercomputing (NAS) Division. For more information, see "NAS Parallel Benchmarks" at this website: http://www.nas.nasa.gov/publications/npb.html

It is not feasible to cover all compilation parameters, so the tests varied only the level of optimization. The following parameters were common for all builds:

```
-qsmp=noauto:omp -qnosave -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
```

The example considers four sets of compilation parameters. In addition to the previous compilation parameters, the remaining parameters are listed in Table 4-1. The column Option set name lists shortcuts that were used to reference each set of compilation parameters that are presented in the Compilation parameters columns.

Table 4-1 NPB: Compilation parameters used to build the NPB suite executable files

Option set name	Compilation parameters	
	Varying	Common
-O2	-02	-qsmp=noauto:omp -qnosave -qmaxmem=-1 -qarch=auto -qtune=auto:balanced
-O3	-03	
-O4	-04	
-O5	-05	

All of the runs were performed with the following environment variables:

```
export OMP_DYNAMIC="FALSE"
export OMP_SCHEDULE="static"
```

To establish the affinity of threads, the example used a simple utility program. The source code for this program is described in 4.3, “Sample code for the construction of thread affinity strings” on page 145.

Note: In several cases, performance results that are presented deviate significantly from the general behavior within the same plot. The plot bars with deviations can be ignored because the tests can experience unusual conditions (operating system jitters, parasitic external workload, and so on).

4.1.1 Performance effect of a Rational choice of an SMT mode

The POWER8 core can run instructions from up to eight application threads simultaneously. This capability is known as SMT. The POWER8 architecture supports the following four multithreading levels:[2]

- ST (single-thread)[3]
- SMT2 (two-way multithreading)
- SMT4 (four-way multithreading)
- SMT8 (eight-way multithreading)

The SMT mode, which can be used to obtain the optimal performance, depends on the characteristics of the application. Compilation parameters and mapping between application threads and logical processors can also affect the timing.

[2] B. Sinharoy et al, “IBM POWER8 processor core microarchitecture,” IBM J. Res. & Dev., vol. 59, no. 1, Paper 2, pp. 2:1–2:21, Jan./Feb. 2015, http://dx.doi.org/10.1147/JRD.2014.2376112.

[3] Single-thread mode is referred sometimes as SMT1.

Reason behind a conscious choice of an SMT mode

Execution units of a core are shared by all logical processors of a core (two logical processors in SMT2 mode, four logical processors in SMT4 mode, and eight logical processors in SMT8 mode). Execution units are available with multiple instances (for example, load/store units, fixed-point units) and single instances (for example, branch execution unit).

In ideal conditions, application threads do not compete for execution units. This configuration results in each of eight logical processors of a core that is running in SMT8 mode being almost as fast as a core that is running in ST mode.

Depending on the application threads instruction flow, some execution units become fully saturated with instructions that come from different threads. As a result, the progress of the depended instructions is postponed. This postponement limits the overall performance of the eight logical processors of a core that is running in SMT8 to the performance of a core that is running in ST mode.

It is also possible for even a single thread to fully saturate resources of a whole core. Therefore, adding threads to a core can result in performance degradation. For example, see the performance of `mg.C` benchmark that is shown in Figure 4-7 on page 128.

Note: Generally, the performance of each logical processor of a core that is running in SMT2, SMT4, or SMT8 mode is not equivalent to the performance of a core that is running in ST mode. The performance of each logical processor of a core is influenced by all other logical processors in a core. The influence comes from the application threads instruction flow.

Performance effect of SMT mode on NPB benchmarks

The bar charts that are shown in Figure 4-1 on page 125, Figure 4-2 on page 125, Figure 4-3 on page 126, Figure 4-4 on page 127, Figure 4-5 on page 127, Figure 4-6 on page 128, Figure 4-7 on page 128, Figure 4-8 on page 129, and Figure 4-9 on page 129 show the performance benefits that come from the IBM Rational® choice of SMT mode for applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`). The performance of the applications was measured for each combination of the following options:

- Four sets of compiler parameters, as listed in Table 4-1 on page 123.
- Five core layouts:
  - 1, 2, 5, and 10 cores from one socket
  - A total of 20 cores from both sockets

The plots are organized according to the following scheme:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular set of compiler options.
- The horizontal axis lists core layouts. For example, the third pair (sockets: 1, cores: 5) designates the benchmarking run where application threads were bound to five cores within one socket.
- The vertical axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we choose an SMT mode that is less favorable for the particular application.

The results show that the choice of SMT mode affects performance substantially.

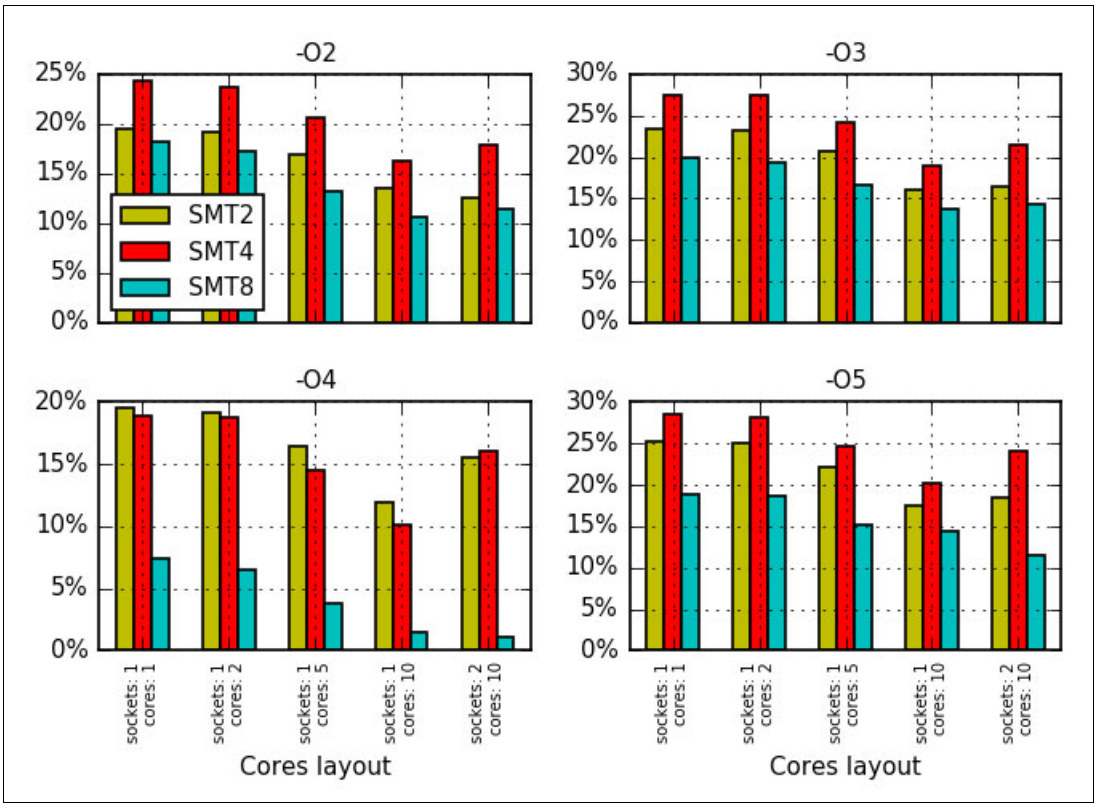


Figure 4-1 Performance benefits from the Rational choice of SMT mode for the bt.C benchmark

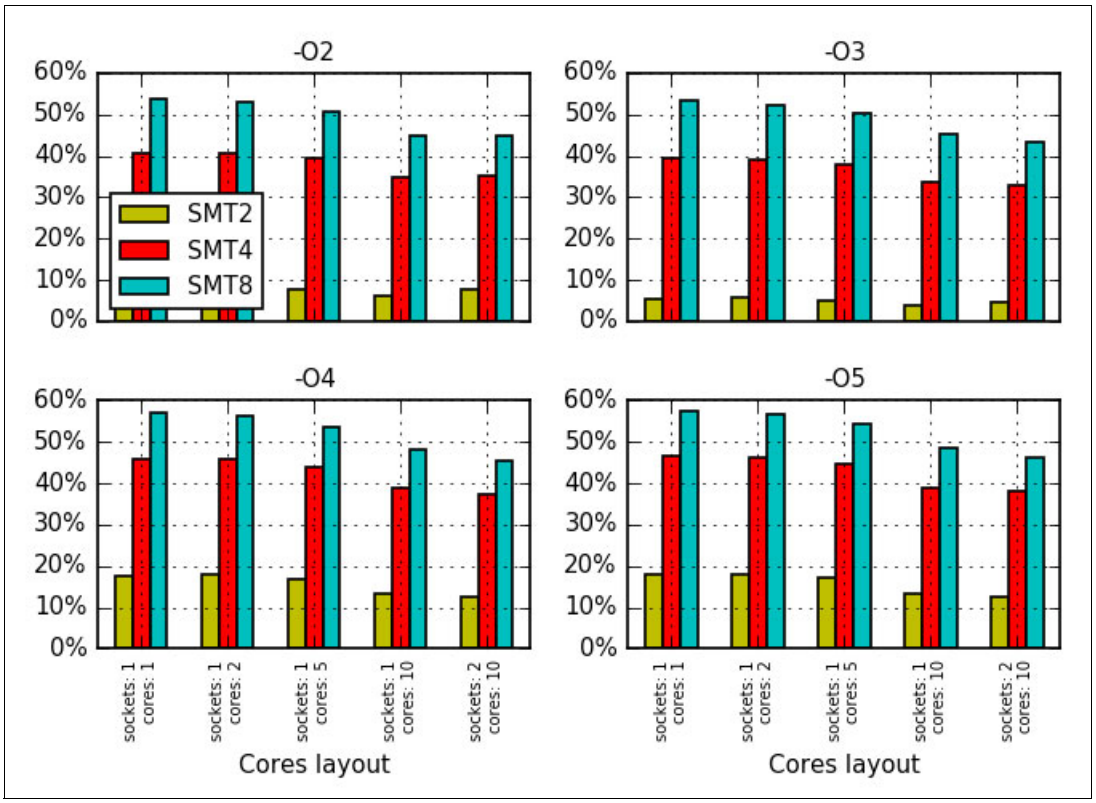


Figure 4-2 Performance benefits from the Rational choice of SMT mode for the cg.C benchmark

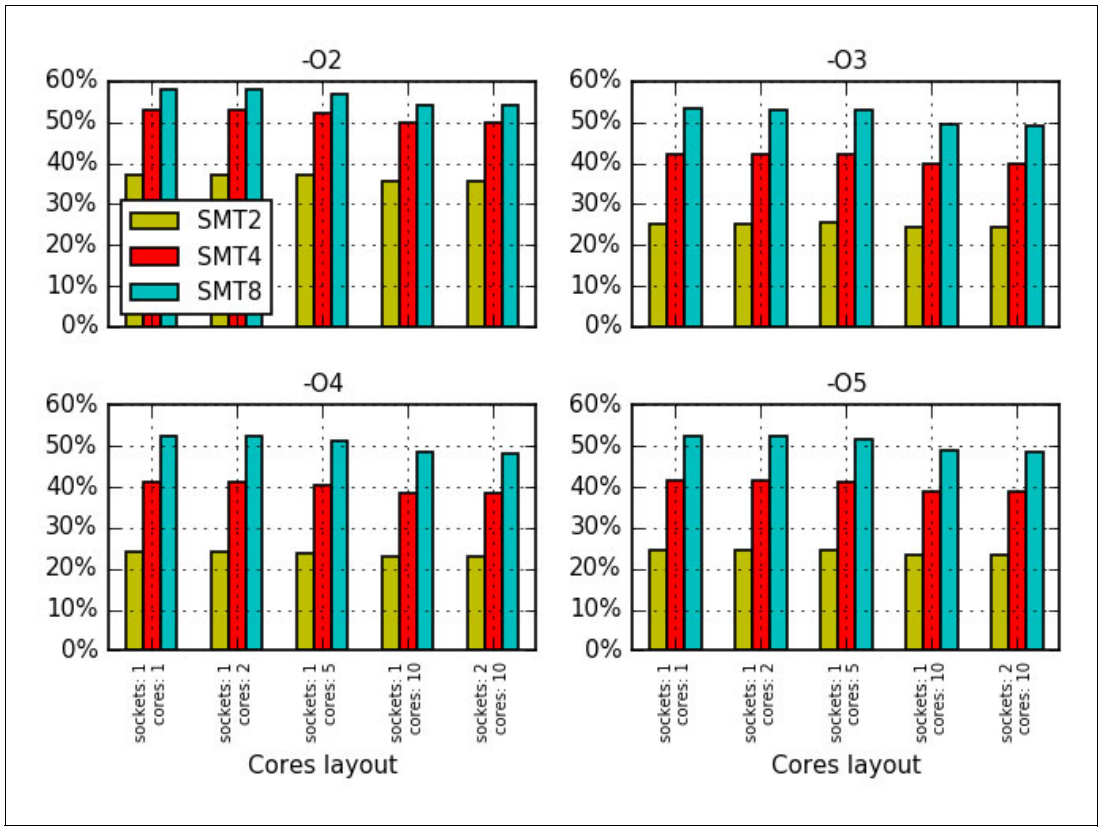


Figure 4-3 Performance benefits from the Rational choice of SMT mode for the ep.C benchmark

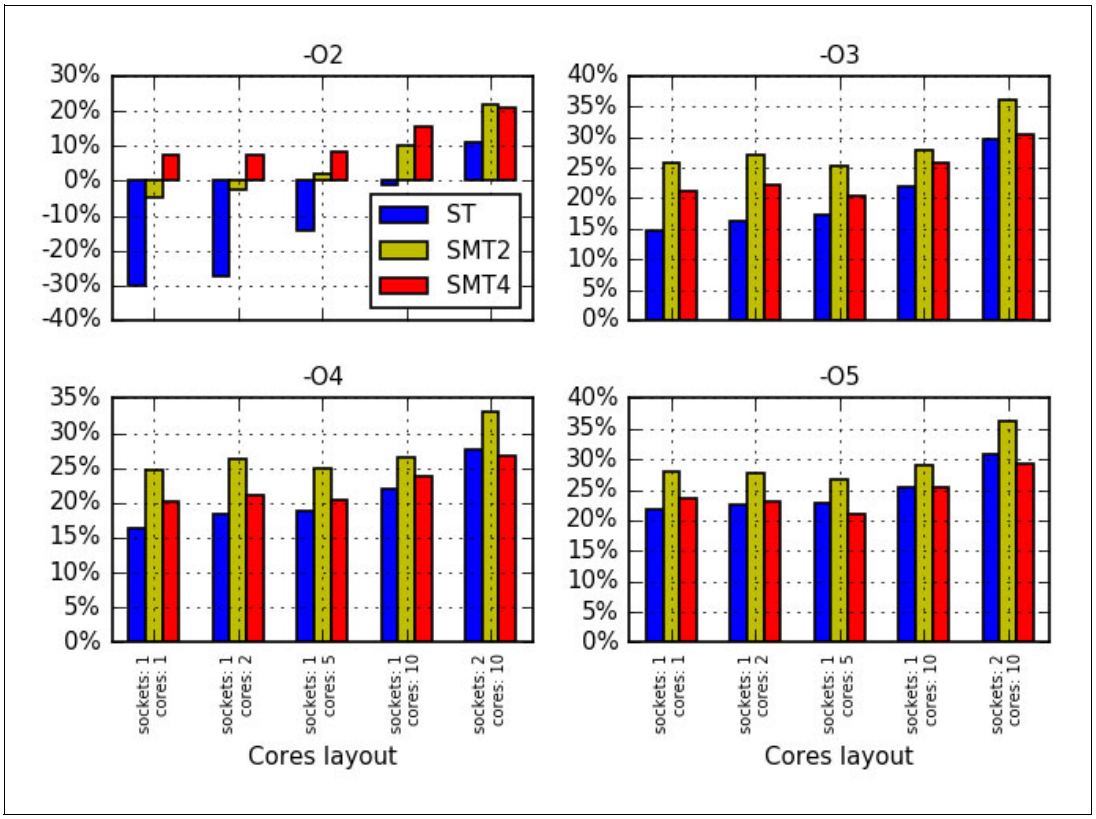


Figure 4-4 Performance benefits from the Rational choice of SMT mode for the ft.C benchmark

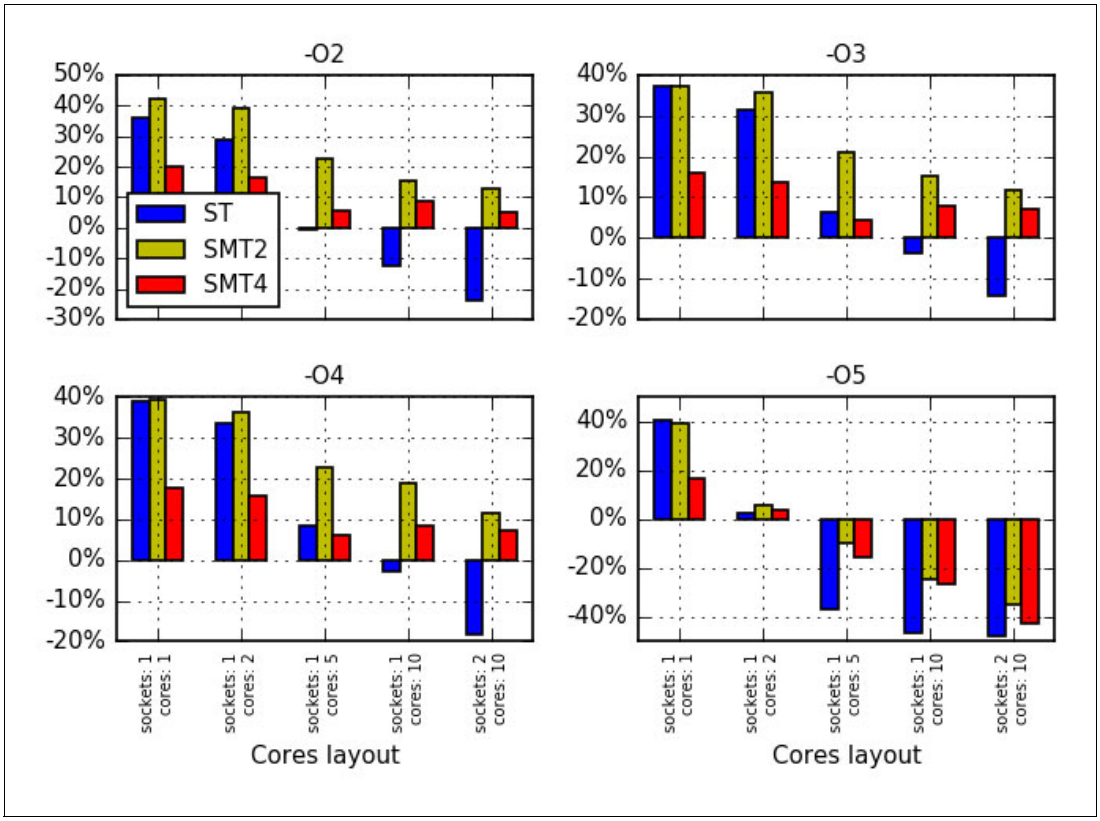


Figure 4-5 Performance benefits from the Rational choice of SMT mode for the is.C benchmark

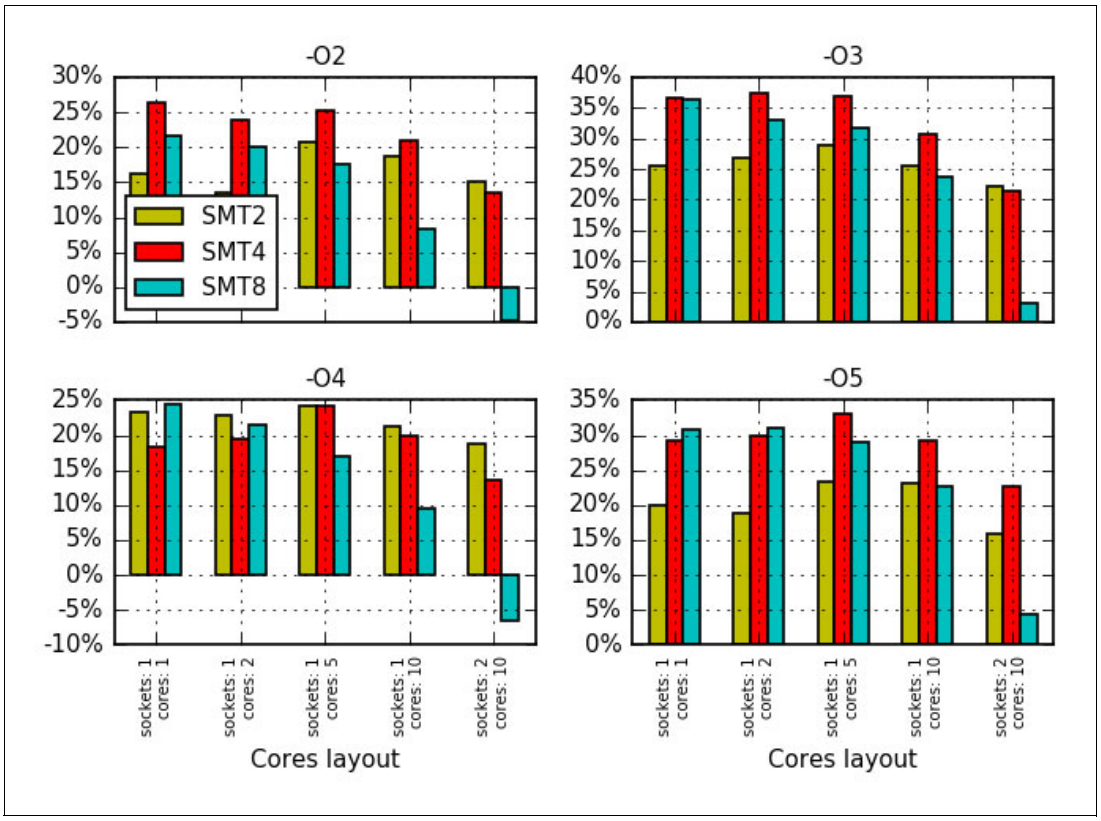


Figure 4-6 Performance benefits from the Rational choice of SMT mode for the lu.C benchmark

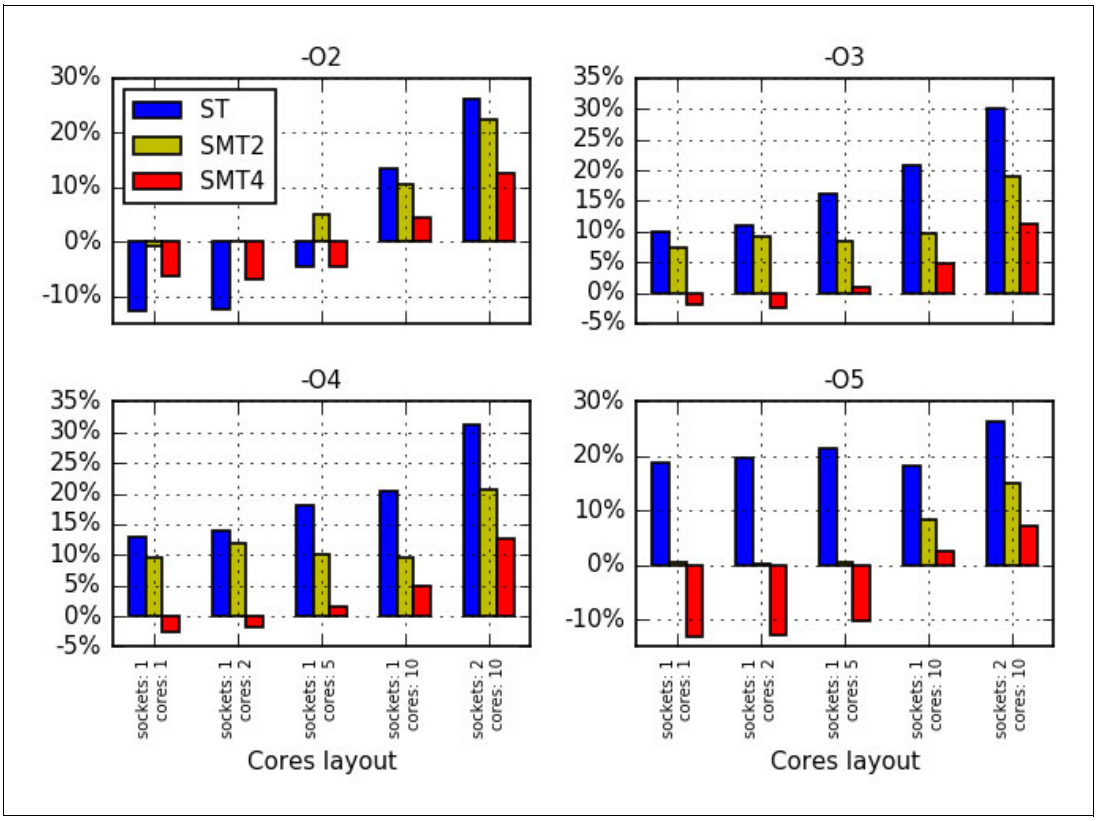


Figure 4-7 Performance benefits from the Rational choice of SMT mode for the mg.C benchmark

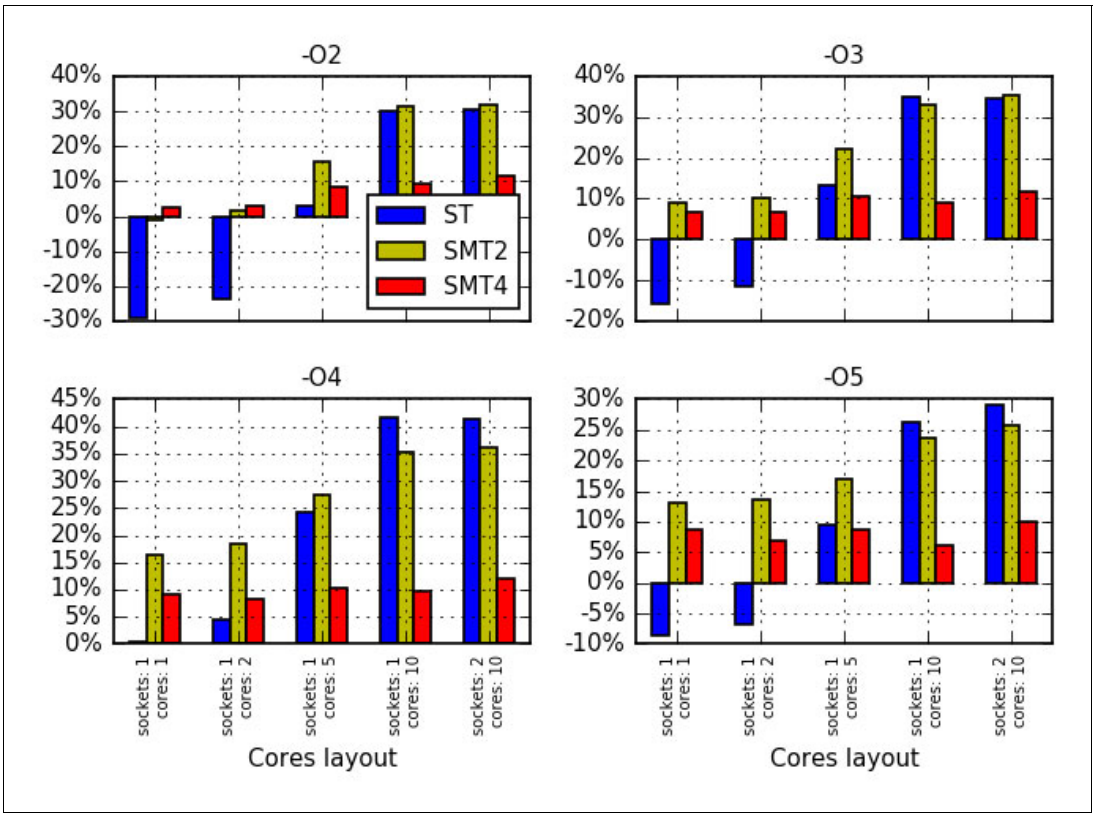


Figure 4-8 Performance benefits from the Rational choice of SMT mode for the sp.C benchmark

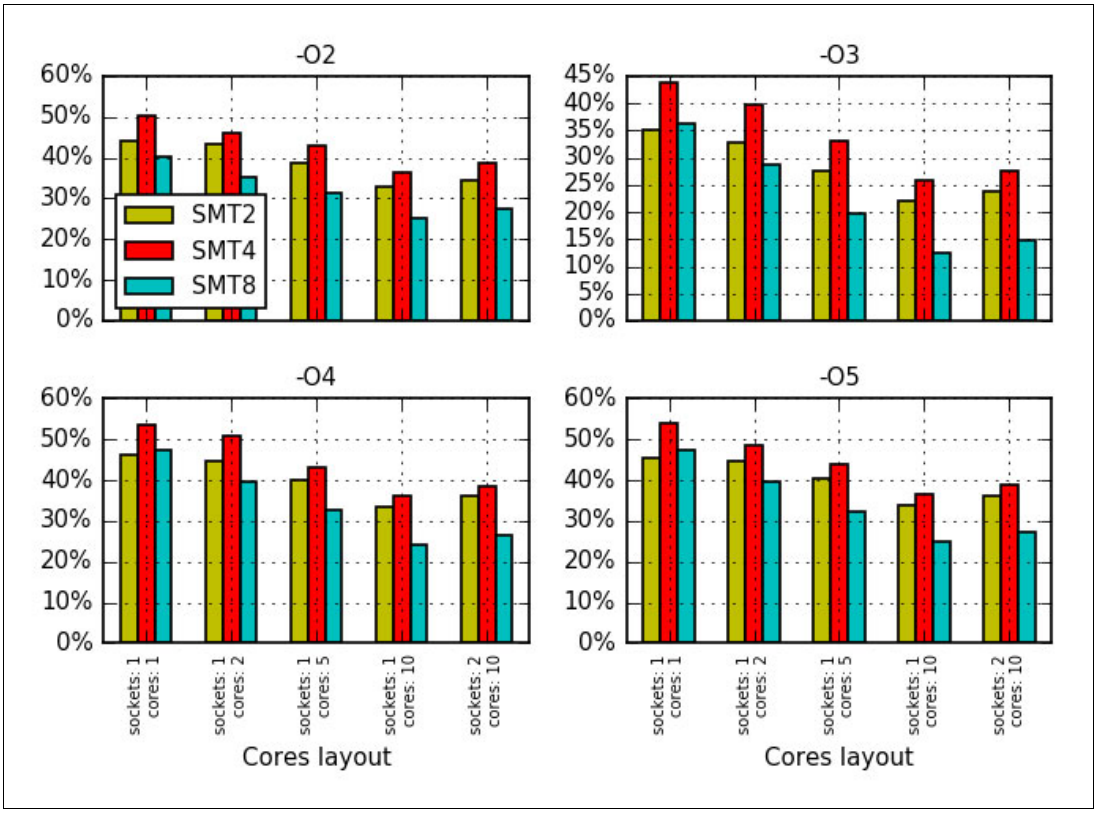


Figure 4-9 Performance benefits from the Rational choice of SMT mode for the ua.C benchmark

Choice of SMT mode for computing nodes

Many NPB applications benefit from SMT8 mode. Therefore, from the general system management perspective, it can be unwise to restrict users to lower SMT values. The administrator considers the following recommendations:

- Run the system in SMT8 mode
- Use a processor core as a unit of hardware resource allocation[4]

By using the physical processor as a unit of hardware resource allocation, you ensure that no other applications use the idle logical processors of a physical core that is assigned to the user. Before the users run a productive workload, they must execute several benchmarks for an application of their choice. The benchmarks are necessary to determine a favorable number of software threads to run at each core. After the value is identified, that number must be taken into account when users arrange productive workloads.

If possible, recompile the application with the `-qtune=pwr8:smtX` option of the IBM XL compiler (where *X* is 1, 2, 4, or 8, depending on the favorable SMT mode), and repeat the benchmark.

Reason for MPI applications

The same logic holds for applications that use MPI. For programs that are based on OpenMP, seek a favorable number of threads to execute on each core. Similarly, for an application that is created with MPI, find a favorable number of MPI processes to execute on each core.

4.1.2 Effect of optimization options on performance

Various compiler options affect the performance characteristics of produced binary code. However, not all of the optimization options are equally suited for all types of workloads. Compilation parameters that result in good performance for one type of application might not perform equally well for other types of computations. Choose a favorable set of compiler options that is based on the timing of a particular application.

The bar charts that are shown in Figure 4-10 on page 131, Figure 4-11 on page 132, Figure 4-12 on page 132, Figure 4-13 on page 133, Figure 4-14 on page 133, Figure 4-15 on page 134, Figure 4-16 on page 134, Figure 4-17 on page 135, and Figure 4-18 on page 135 compare the performance benefits that come from the rational choice of compiler options for the same applications from the NPB suite (`bt.C`, `cg.C`, `ep.C`, `ft.C`, `is.C`, `lu.C`, `mg.C`, `sp.C`, and `ua.C`) that are described in 4.1.1, “Performance effect of a Rational choice of an SMT mode” on page 123.

Again, the four sets of compiler options considered are listed in Table 4-1 on page 123. The application threads are bound to 1, 2, 5, or 10 cores of one socket or 20 cores of two sockets.

The organization of the plots is similar to the following scheme that was described in 4.1.1, “Performance effect of a Rational choice of an SMT mode” on page 123:

- Each figure presents results for a particular benchmark from the NPB suite.
- Each subplot is devoted to a particular SMT mode.
- The horizontal axis lists the number of cores that is used.

[4] This recommendation is targeted to applications that are limited by the computing power of a processor only. It does not account for interaction with the memory subsystem and other devices. At some supercomputing sites and user environments, it can be reasonable to use a socket or even a server as a unit of hardware resource allocation.

- The vertical axis shows the performance gain as measured in percentage relative to a baseline. As a baseline, we chose a compiler option set that is less favorable for the particular application.

The results show that the Rational choice of a compiler option set substantially affects the performance for all of the considered NPB applications.

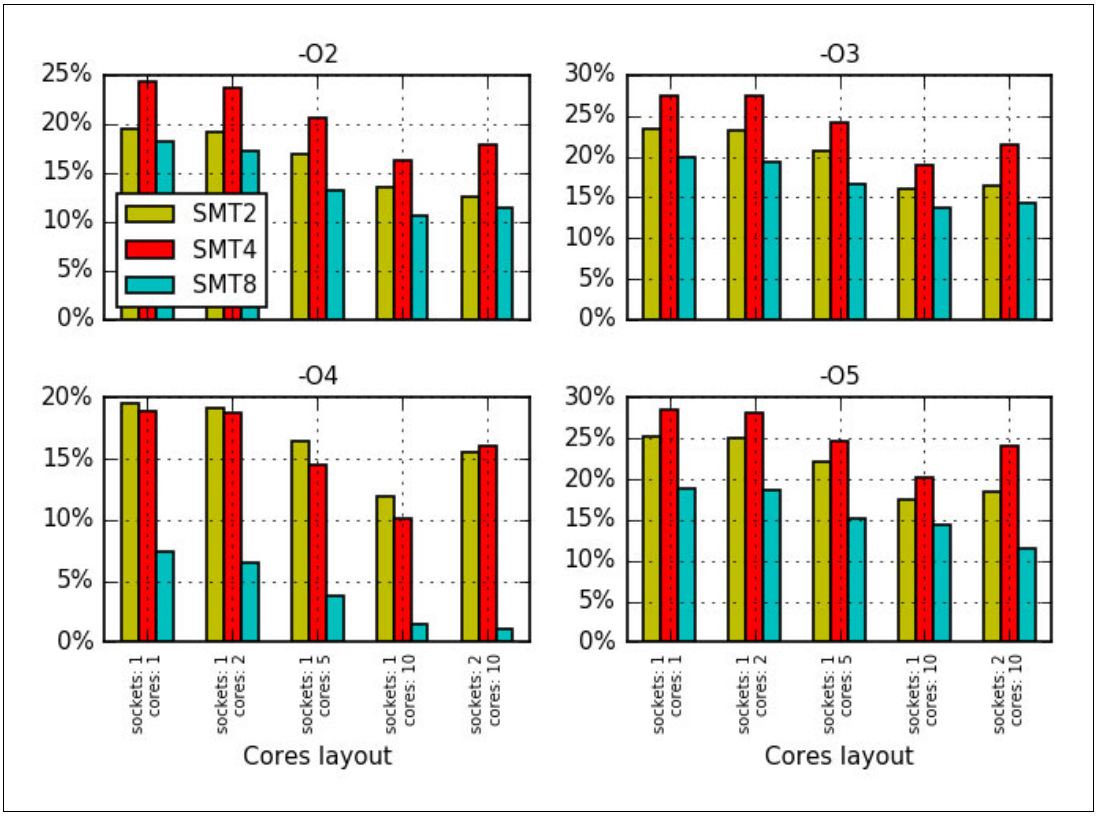


Figure 4-10 Performance gain from the Rational choice of compiler options for the bt.C test

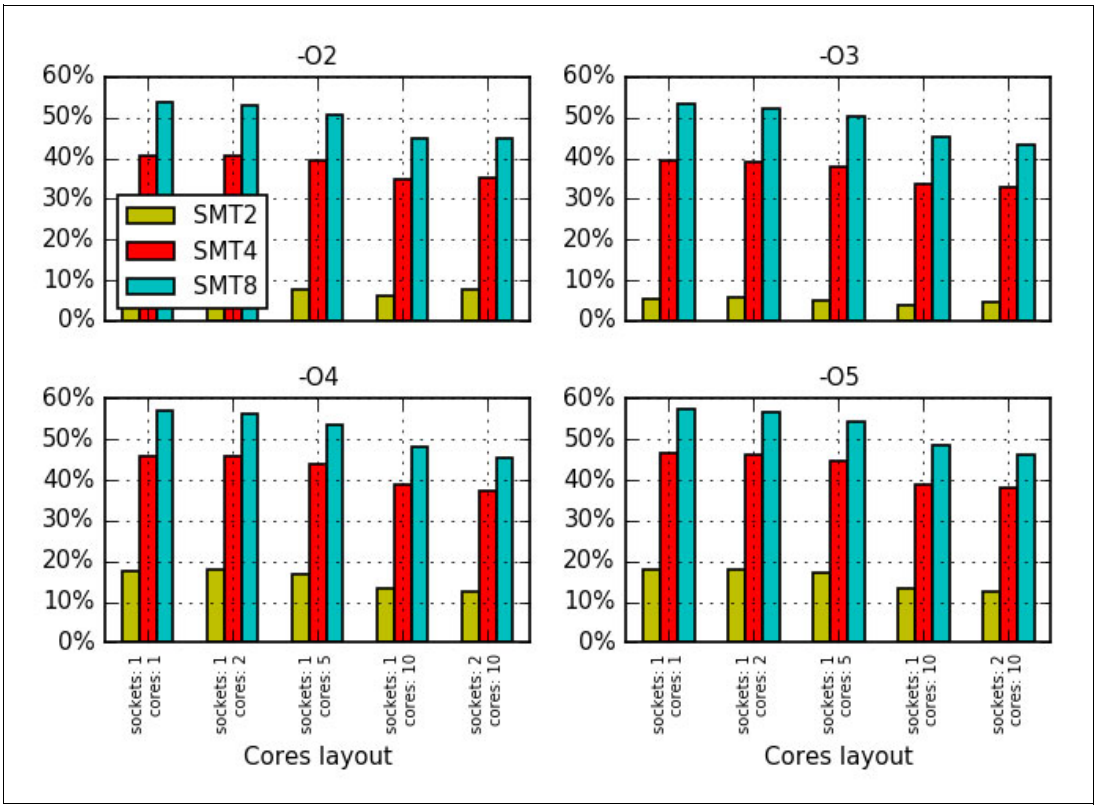


Figure 4-11 Performance gain from the Rational choice of compiler options for the cg.C test

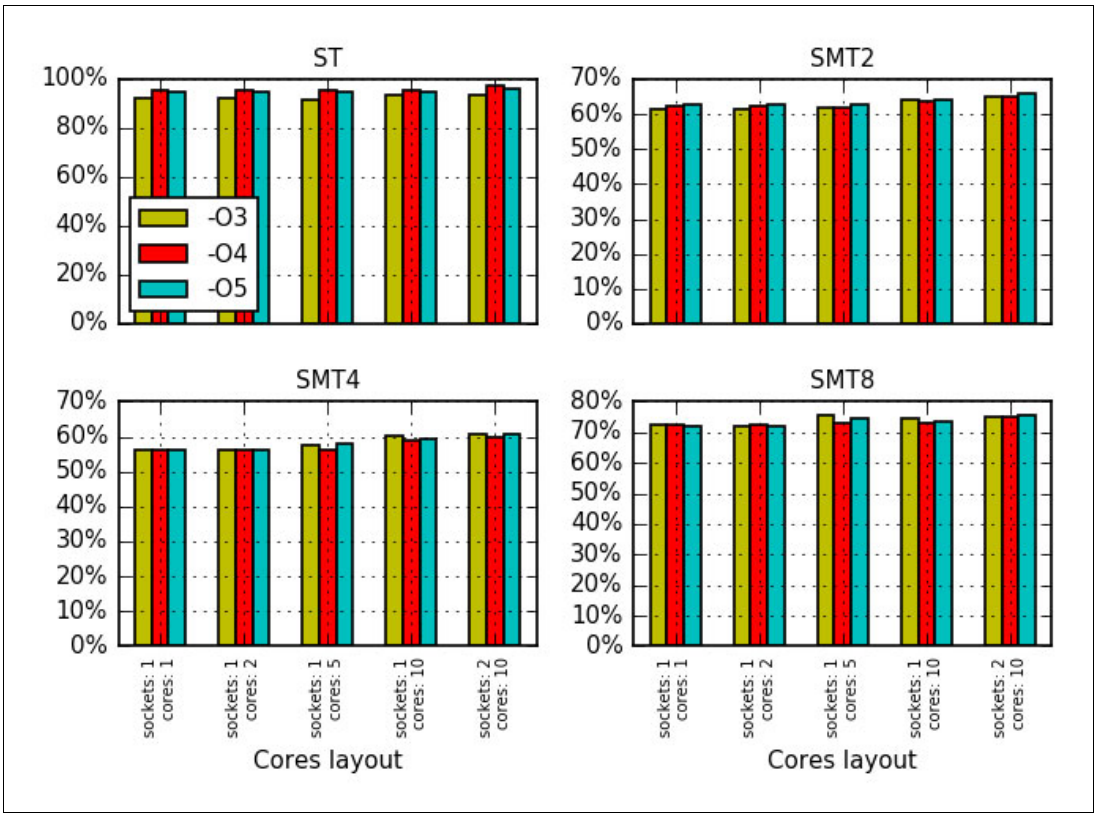


Figure 4-12 Performance gain from the Rational choice of compiler options for the ep.C test

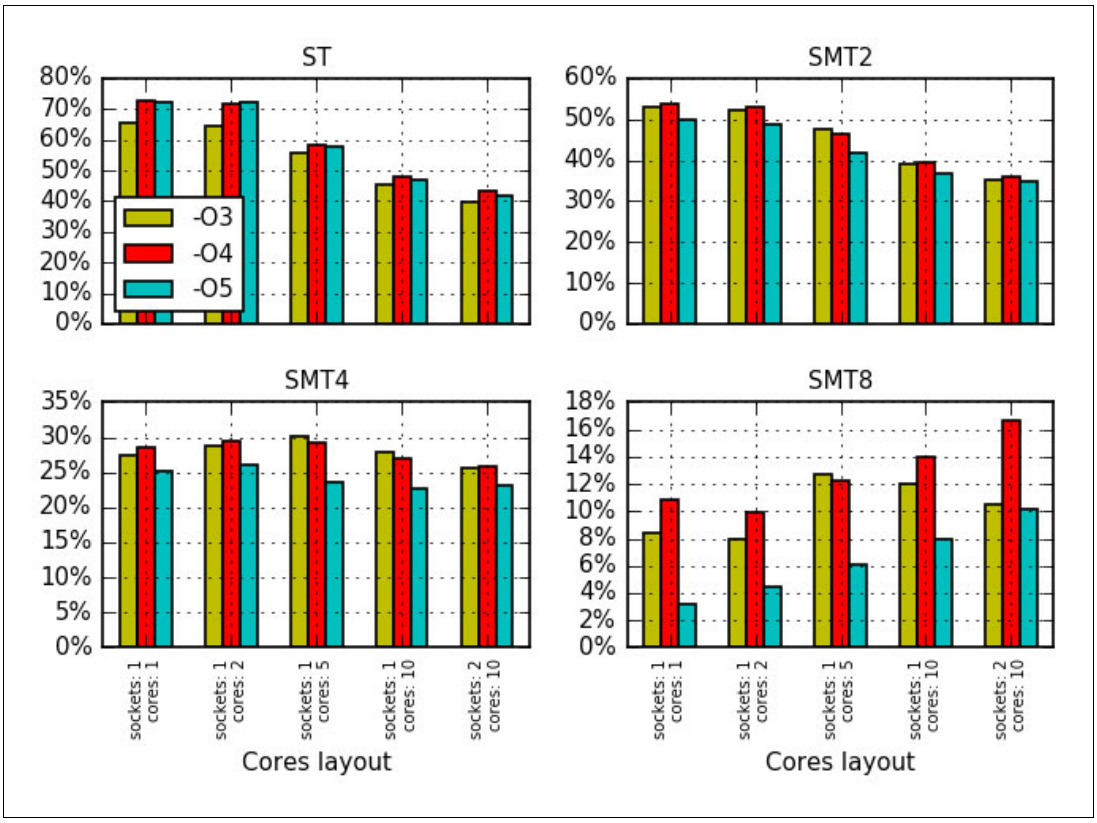


Figure 4-13 Performance gain from the Rational choice of compiler options for the ft.C test

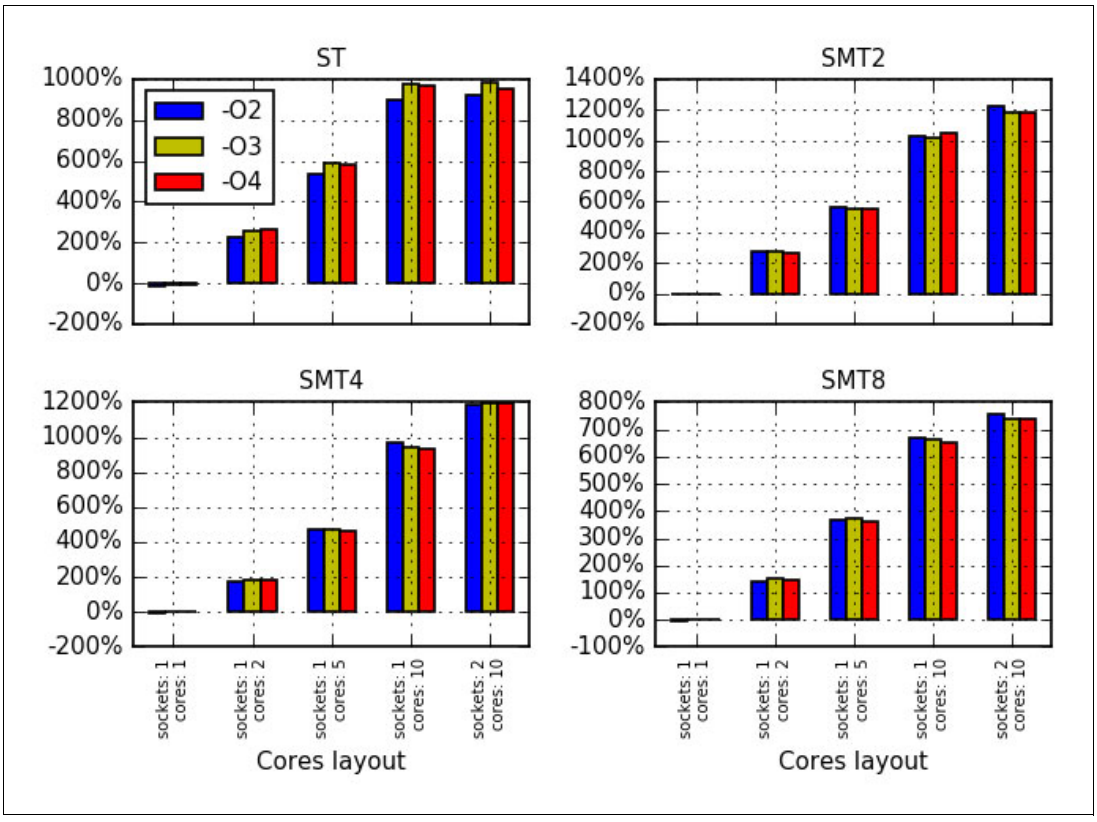


Figure 4-14 Performance gain from the Rational choice of compiler options for the is.C test

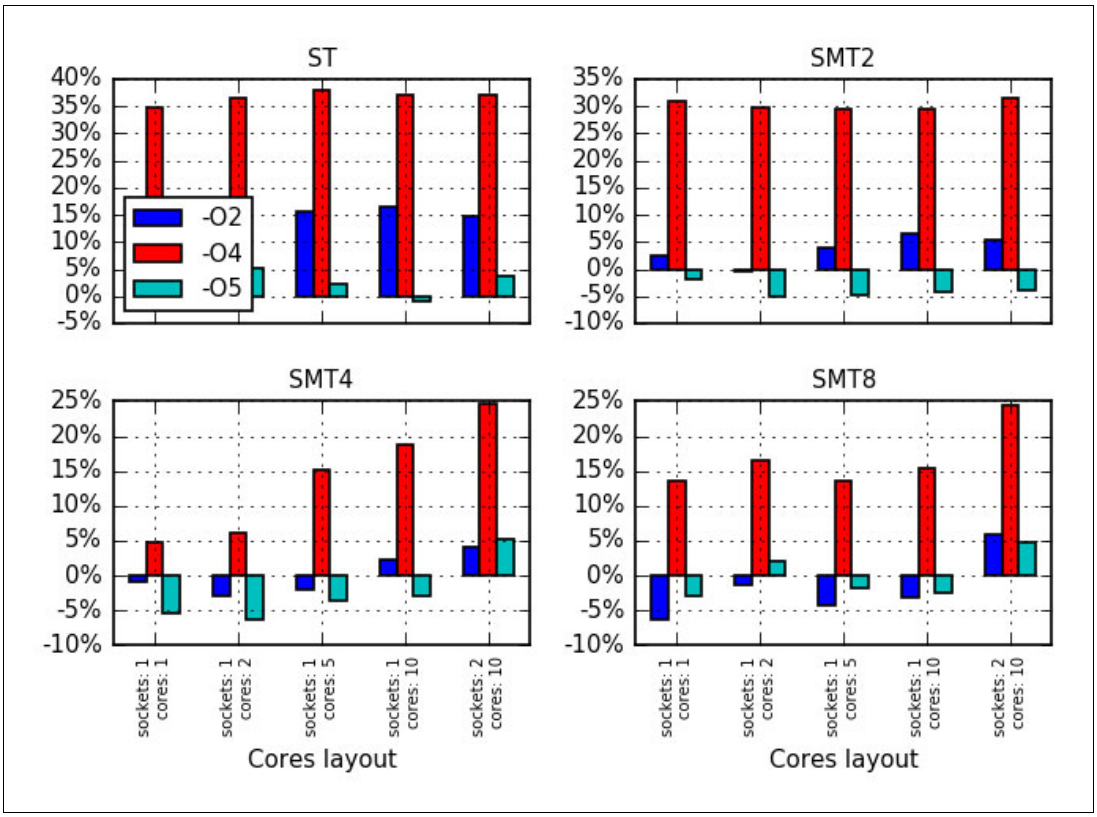


Figure 4-15 Performance gain from the Rational choice of compiler options for the lu.C test

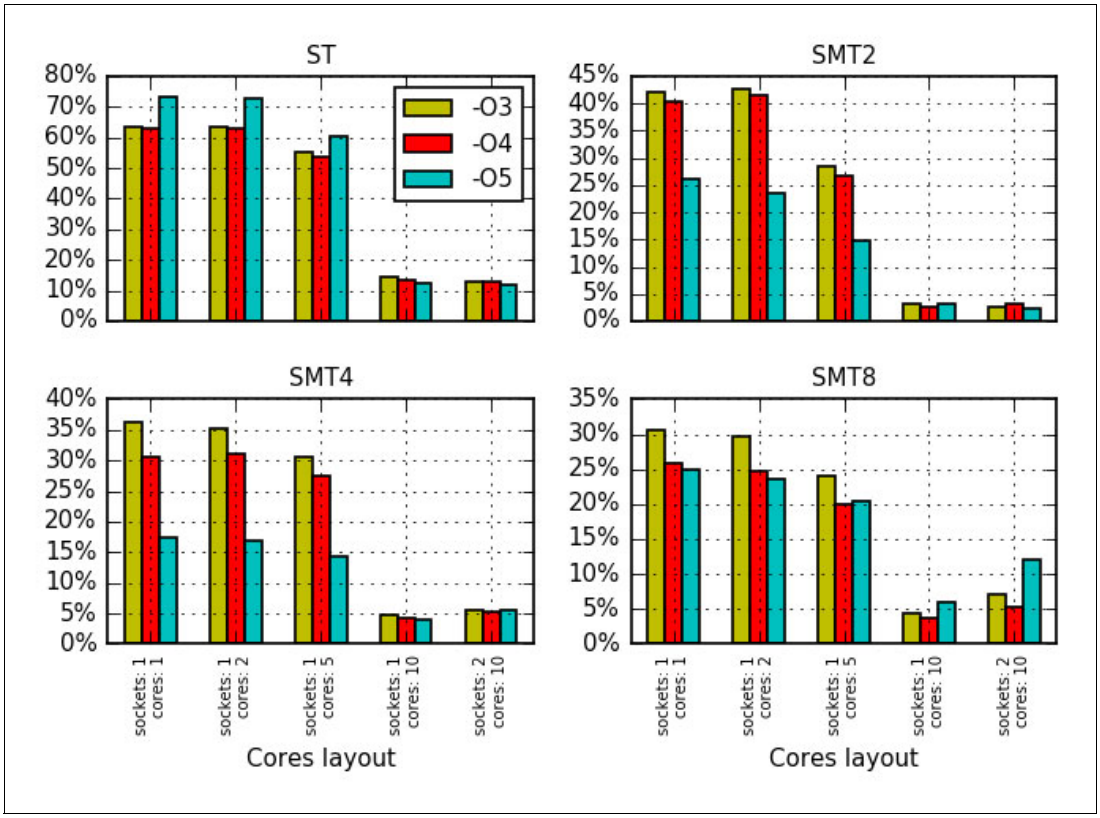


Figure 4-16 Performance gain from the Rational choice of compiler options for the mg.C test

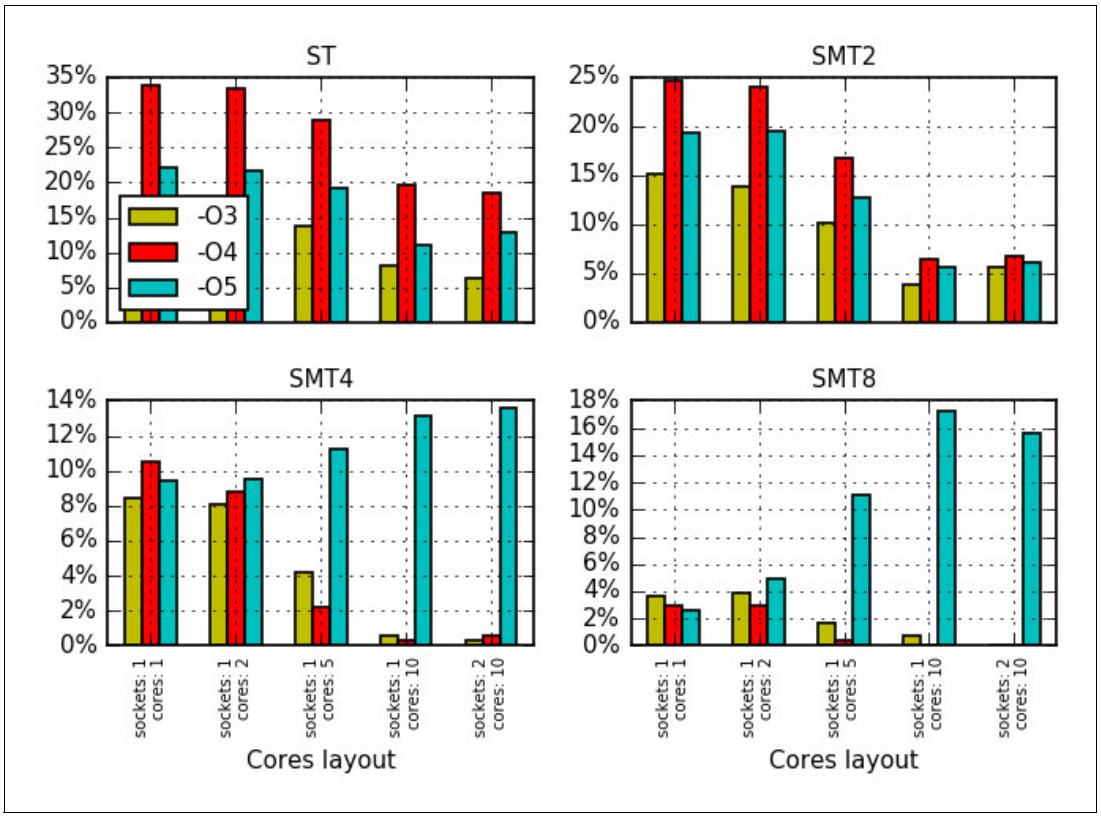


Figure 4-17 Performance gain from the Rational choice of compiler options for the sp.C test

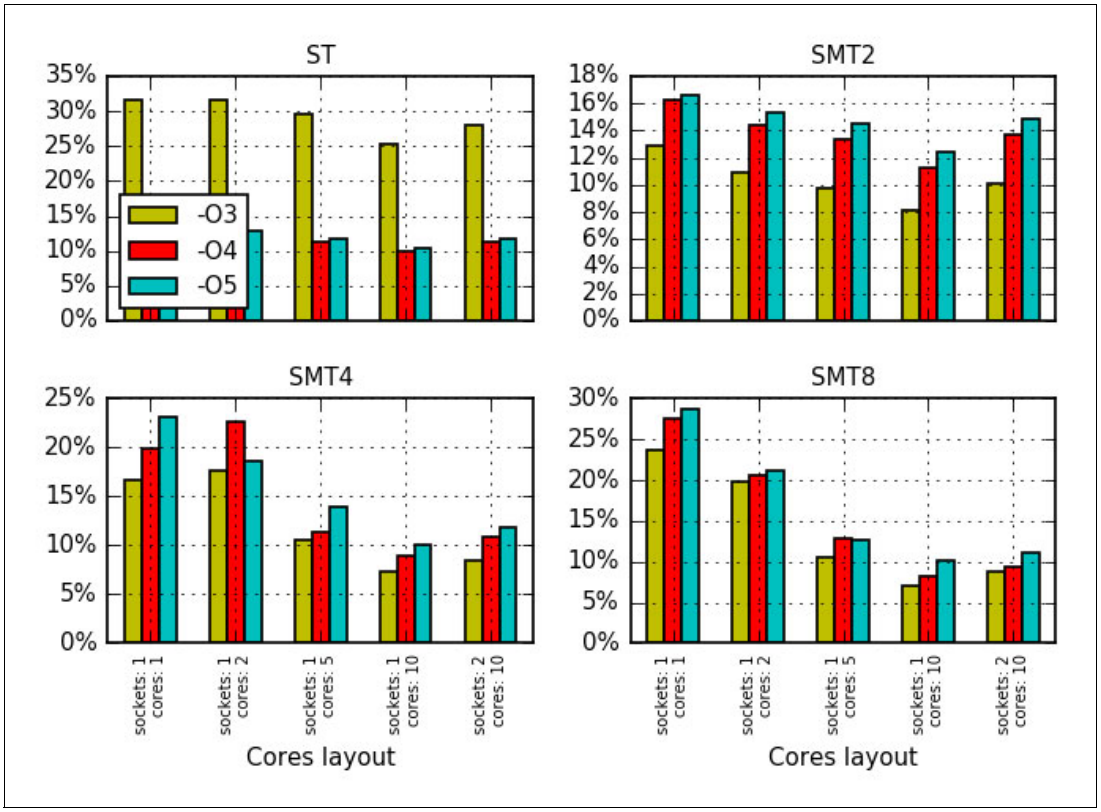


Figure 4-18 Performance gain from the Rational choice of compiler options for the ua.C test

4.1.3 Favorable modes and options for applications from the NPB suite

Table 4-2 lists SMT modes and compiler optimization options that are favorable for most of the runs that are described in 4.1.1, “Performance effect of a Rational choice of an SMT mode” on page 123 and 4.1.2, “Effect of optimization options on performance” on page 130. The row headers (ST, SMT2, SMT4, and SMT8) designate SMT modes. The column headers (-O2, -O3, -O4, and -O5) refer to sets of compiler options that are listed in Table 4-1 on page 123.

Table 4-2 Favorable modes and options for applications from the NPB suite

	-O2	**-O3**	**-O4**	**-O5**
ST	—	—	sp.C	mg.C
SMT2	is.C	—	ft.C, lu.C	—
SMT4	—	—	—	bt.C, ua.C
SMT8	—	ep.C	—	cg.C

Favorable SMT modes can vary from ST to SMT8 and favorable compilation options can vary from -O2 to -O5.

It is difficult to know before experimenting which SMT mode and which set of compilation options will be favorable for a user application. Therefore, establish benchmarks before conducting production runs. For more information, see 4.2, “General methodology of performance benchmarking” on page 137.

4.1.4 Importance of binding threads to logical processors

The operating system can migrate application threads between logical processors if a user does not explicitly specify thread affinity. As described in 3.1, “Controlling the running of multithreaded applications” on page 90, a user can specify the binding of application threads to a specific group of logical processors. One option for binding threads is to use system calls in a source code. The other option is to set environment variables that control threads affinity.

For technical computing workloads, you want to ensure that application threads are bound to logical processors. Thread affinity often helps to take advantage of the POWER architecture memory hierarchy and reduce the overhead that is related to the migration of threads by an operating system.

To demonstrate the importance of this technique, the `mg.C` test from the NPB suite was chosen as an example. The performance of the `mg.C` application peaks at SMT1 (see Table 4-2). For this benchmark, you can expect a penalty if an operating system puts more than one thread on each core.

Figure 4-19 on page 137 shows the performance improvement that was obtained by binding application threads to logical processors. The baseline corresponds to runs without affinity. The bars show the relative gain that was obtained after the assignment of software threads to hardware threads. As SMT mode increases, the operating system has more freedom in scheduling threads. As the result, the effect of thread binding becomes more pronounced for higher values of SMT mode.

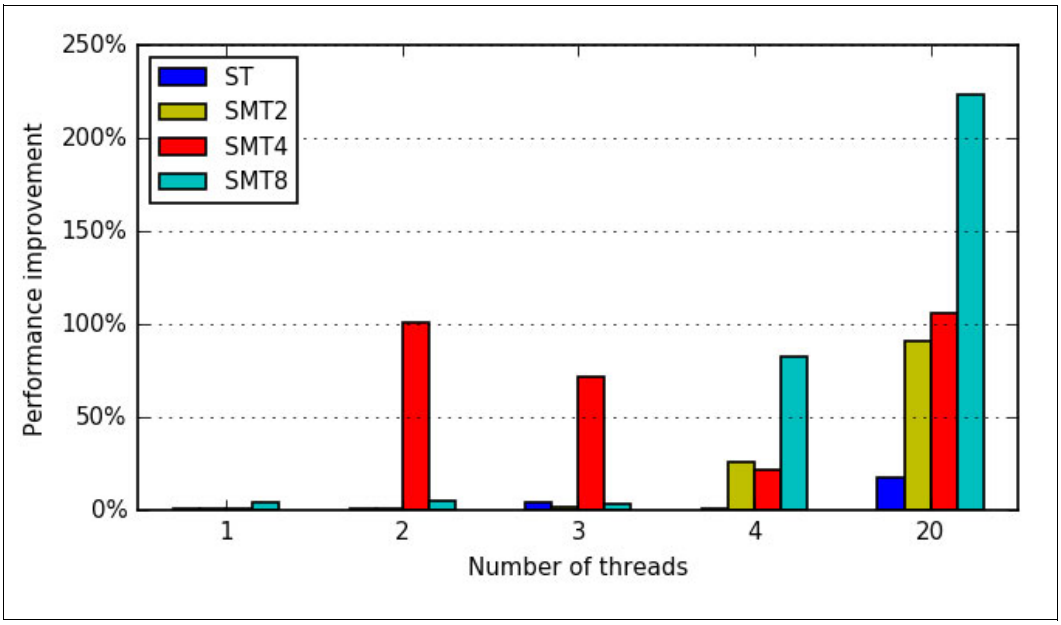


Figure 4-19 Performance improvement for an application when thread binding is used

4.2 General methodology of performance benchmarking

This section describes how to evaluate the performance of an application on a system with massively multithreaded processors. It also provides some hints about how to take advantage of the performance of an application without access to the source code.

This example assumes that the application is thread-parallel and does not use Message Passing Interface (MPI)[5] or general-purpose computing on graphics processing units (GPGPU)[6]. However, the generalization of the methodology to a broader set of applications is relatively straightforward.

For the simplicity of table structure throughout this section, the examples make the following assumptions about the configuration of a computing server:

- The server is a two-socket system with fully populated memory slots.
- The server has 10 cores per socket (20 cores in total).

The computing server reflects the architecture of the IBM Power Systems S822LC (8335-GTB) system.

This section also describes performance benchmarking and summarizes the methodology in a step-by-step instruction form. For more information, see 4.2.11, “Summary” on page 145.

4.2.1 Defining the purpose of performance benchmarking

Before starting a performance benchmarking, it is important to clarify the purpose of this activity. A clearly defined purpose of the benchmarking is helpful in creating a plan of the benchmarking and defining success criteria.

[5] MPI is a popular message-passing application programmer interface that is used for parallel programming.
[6] GPGPU is the use of GPUs for the solution of compute intensive data parallel problems.

The purpose of performance benchmarking looks different from each of the following two points of view:

- Performance benchmarking that is carried out by an *application developer*.
- Performance benchmarking that is done by an *application user* or a *system administrator*.

Benchmarking by application developers

From the perspective of an application developer, the performance benchmarking is part of a software development process. Application developer uses performance benchmarking in pursuing the following goals:

- Comparison of a code's performance with the performance model.
- Identification of bottlenecks in a code that limit its performance (this process is typically done by using profiling).
- Comparison of the code's scalability with the scalability model.
- Identification of parts of the code that prevent scaling.
- Tuning the code to specific computer hardware.

A software developer carries out a performance benchmarking of an application to understand how to improve application performance by changing an algorithm. Performance benchmarking as it is viewed from an application developer perspective has complex methodologies. These methodologies are not covered in this publication.

Benchmarking by application users and system administrators

From the perspective of an *application user* or a *system administrator,* performance benchmarking is a necessary step before using an application for production runs on a computing system that is available for them. Application users and system administrators make the following assumptions:

- An application will be used on a computing system many times in future.
- It makes sense to invest time into performance benchmarking because the time will be made up by faster completion of production runs in future.
- Modification of an application source code is out of their scope.

  This statement implies that application users and system administrators are limited to the following choices to tune the performance:

  - Different compilers
  - Compiler optimization options
  - SMT mode
  - Number of computing cores
  - Runtime system parameters and environment variables
  - Operating system parameters

Essentially, applications users and system administrators are interested in how to make an application solve problems as fast as possible without changing the source code.

System administrators also need performance benchmarking results to help determine hardware allocation resources when configuring a computing system. For more information, see "Choice of SMT mode for computing nodes" on page 130.

This book considers performance benchmarking from the perspective of an application user or a system administrator.

4.2.2 Benchmarking plans

Complete the following steps for your benchmarking project:

1. A kick-off planning session with the experts.
2. Workshop with the experts.
3. Benchmarking.
4. Session to discuss the intermediate results with the experts.
5. Benchmarking.
6. Preparation of the benchmarking report.
7. Session to discuss the final results with the experts.

You also must prepare the following lists:

- Applications to be benchmarked
- Persons who are responsible for specific applications

Planning and discussion sessions are ideally face-to-face meetings between the benchmarking team and POWER architecture professionals. A workshop with the POWER architecture experts is a form of knowledge transfer educational activity with hands-on sessions.

The specific technical steps that you perform when benchmarking individual applications are described next.

For an example of the technical part of a plan, see 4.2.11, “Summary” on page 145.

4.2.3 Defining the performance metric and constraints

The *performance metric* provides a measurable indicator of application performance. Essentially, a performance metric is a number that can be obtained in a fully automated manner. The most commonly used performance metrics include the following examples:

- Time of application execution
- Number of operations performed by an application in a unit of time

Some applications impose constraints in addition to the performance metric. Typically, the violation of a constraint means that running the application in such conditions is unacceptable. Constraints include the following examples:

- Latency of individual operations (for example, execution of a specific ratio of operations takes no longer than a specific time threshold).
- Quality metric of the results produced by the application does not fall below a specified threshold (for example, a video surveillance system drops no more than a specified number of video frames in a unit of time).

4.2.4 Defining the success criteria

Satisfying a success criteria is a formal reason to finalize performance benchmarking. Usually, success criteria is based on the performance results. Typically, *success criteria* is a numeric threshold that provides a definition of an acceptable performance (see 4.2.3, “Defining the performance metric and constraints” on page 139).

The performance benchmarking can result in the following outcomes:

- The success criteria are satisfied. This result means that the process of performance tuning can be finalized.

- The application does not scale as expected (see “Probing the scalability” on page 142). This result indicates that you must reconsider the success criteria.
- The success criteria are not satisfied. Use the following techniques to solve the problem:
  - Discuss the performance tuning options that you tried over and the results you obtained with the experts. For this purpose, keep a verbose benchmarking log. For more information, see “Keeping the log of benchmarking” on page 140.
  - Engage the experts to perform deep performance analysis.
  - Seek help of software developers to modify the source code.

Performance of a logical processor versus performance of a core

It is important to understand that in the most cases, there is no sense in defining success criteria based on the performance of a logical processor of a core taken in isolation. As described in 4.1.1, “Performance effect of a Rational choice of an SMT mode” on page 123, the POWER8 core can run instructions from multiple application threads simultaneously. Therefore, the whole core is a minimal entity of reasoning about performance. For more information, see “Choice of SMT mode for computing nodes” on page 130.

Aggregated performance statistics for poorly scalable applications

Similarly, some applications are not designed to scale up to a whole computing server. For example, an application can violate constraints under a heavy load (see 4.2.3, “Defining the performance metric and constraints” on page 139). In such situation, it makes sense to run several instances of an application on a computing node and collect aggregated performance statistics. This technique allows you to evaluate performance of a whole server that is running a particular workload.

4.2.5 Correctness and determinacy

Before you start performance benchmarking, check that the application works correctly and produces deterministic results. If the application produces undeterministic results by design (for example, the application implements the Monte-Carlo method or other stochastic approach), you have at least two options:

- Modify the application by making its output deterministic (for example, fix the seed of a random number generator).
- Develop a reliable approach for measuring performance of an undeterministic application (this process can require many more runs than for a deterministic application).

During each stage of the performance tuning, verify that the application still works correctly and produces deterministic results (for example, by using regression testing).

4.2.6 Keeping the log of benchmarking

Preserve the history and output of all commands that are used in the preparation of the benchmark and during the benchmark. This log includes the following files:

- Makefiles
- Files with the output of the **`make`** command
- Benchmarking scripts
- Files with the output of benchmarking scripts
- Files with the output of individual benchmarking runs

A benchmarking script is a shell script that includes the following commands:

- A series of commands that capture the information about the execution environment
- A sequence of commands that run applications under various conditions

Example 4-2 shows several commands that you might want to include in the beginning of a benchmarking script to capture the information about the execution environment.

Example 4-2 Example of the beginning of a benchmarking script

```
#!/bin/bash -x
uname -a
tail /proc/cpuinfo
numactl -H
ppc64_cpu --smt
ppc64_cpu --cores-present
ppc64_cpu --cores-on
gcc --version
xlf -qversion
env
export OMP_DISPLAY_ENV="VERBOSE"
```

Typically, a benchmarking script combines multiple benchmarking runs in a form of loops over a number of sockets, cores, and SMT modes. By embedding the commands that you use to execute an application into a script, you keep the list of commands along with their outputs.

Running a benchmarking script

If you do not use a job scheduler to submit a benchmarking script, run a benchmarking script inside a session that is created by using the **screen** command. Doing so gives protection against network connection issues and helps to keep applications running, even during a network failure.

One option to run a benchmarking script is to execute the following command:

```
./benchmarking_script_name.sh 2>&1 | tee specific_benchmarking_scenario.log
```

This command combines the standard output and the standard error streams, whereas the **tee** command writes the combined stream to a workstation and a specified file.

Choosing names for log files

It is helpful to write the output of individual benchmarking runs to separate files with well-defined descriptive names. For example, we found the following format useful for the purposes of the benchmarking runs that are described in 4.1, “Effects of basic performance tuning techniques” on page 122:

```
$APP_NAME.t$NUM_SMT_THREADS.c$NUM_CORES.s$NUM_SOCKETS.log
```

This format facilitated the postprocessing of the benchmarking logs by using the **sed** command.

4.2.7 Probing the scalability

Probing the scalability is a necessary step of the performance benchmarking for the following reasons:

- It is the way to evaluate the behavior of an application when the number of computing resources increases.
- It helps to determine the favorable number of processor cores to use for production runs.

To be concise, the ultimate purpose of probing the scalability is to check whether an application is scalable.

Note: In the performance benchmarking within a one-node system, the scalability is probed only in a *strong* sense, also known as a *speed-up* metric. This metric is obtained by fixing the size of a problem and varying the number of processor cores. For a multi-node system, the complimentary metric is a scalability in a *weak* sense. It is obtained by measuring performance when the amount of computing work per node is fixed.

Before performing runs, the user answers the following questions:

- Does the application has some specific limits on the number of logical processors it can use?

  (For example, some applications are designed to run with the number of logical processors that is a power of two only: 1, 2, 4, 8, and so on).
- Is the scalability model of the application available?

  The model can be pure analytical (for example, the application vendor can claim a linear scalability) or the model can be based on the results of previous runs.
- What is the scalability criteria for the application? That is, what is the numeric threshold that defines the “yes” or “no” answer to the following formal question: “Given $N > N_{base}$ logical processors, are you ready to agree that the application is scalable on N logical processors in respect to N_{base} logical processors?”

If available, the scalability model helps you to choose the scalability criteria.

The next step is to compile the application with a basic optimization level. For example, you can use the following options:

- `-O3 -qstrict` (with IBM XL Compilers)
- `-O3` (with GCC[7] and IBM Advance Toolchain)

For more information about IBM XL Compilers, GNU Compiler Collection (GCC), and IBM Advance Toolchain, see 8.7, “IBM XL compilers, GCC, and Advance Toolchain” on page 355 and 2.1, “Compiler options” on page 24.

For scalability probing, choose the size of a problem to be large enough to fit the memory of a server. In contrast, solving a problem of a small size often is a waste of computing cores. If you must process multiple small problems, run multiple one-core tasks simultaneously on one node.

When you decide on a problem size, run the application in ST mode with different number of logical processors. Complete the Performance, Meets the scalability model? (yes/no), and Meets the scalability criteria? (yes/no) rows, as shown in Table 4-3 on page 143.

[7] GNU Compiler Collection (GCC) contains various compilers, including C, C++, and Fortran compilers.

Table 4-3 Scalability probing measurements table

Number of cores	1	2	3	…	10	12	14	16	18	20
Number of sockets	1	1	1	…	1	2	2	2	2	2
Number of cores per socket	1	2	3	…	10	6	7	8	9	10
Performance										
Meets the scalability model? (yes/no)	—									
Meets the scalability criteria? (yes/no)	—									

Note: Remember to bind application threads to logical processors. Failing to do so typically ends up in worse results (see 4.1.4, “Importance of binding threads to logical processors” on page 136).

When probing the scalability, vary the number of cores and run each core in ST mode. Therefore, the total number of threads used can be equal to the number of cores.

Depending on the results of the benchmarking, you end up with one of the following outcomes:

- The application is scalable up to 20 cores.
- The application is scalable within the socket (up to 10 cores).
- The application is not scalable within the socket.

If the scalability of the application meets your expectations, proceed to the next step and evaluate the performance by using the maximum number of cores that you determined in this section.

If the application does not meet your scalability expectations, you first clarify the reasons of that behavior and repeat probing the scalability until satisfied. Only after that process can you proceed to the next step.

4.2.8 Evaluation of performance on a favorable number of cores

The performance of an application heavily depends on the choice of SMT mode and compilation modes. For more information, see “” on page 131. It is difficult to know beforehand which set of compiler options and number of hardware threads per core can be a favorable choice for an application. Therefore, you must run the application with several compilation options or SMT modes and select the most appropriate combination.

In 4.2.7, “Probing the scalability” on page 142, the maximum reasonable number of cores to be used by an application was determined. The next step is to use that number of cores to run the application with different sets of compiler options and SMT modes.

You must try each of four SMT modes for several sets of compiler options. If you are limited on time, try fewer sets of compiler options, but go through all SMT modes. Enter the results of your performance measurements in Table 4-4 on page 144 (the table is similar to the Table 4-2 on page 136). Your actual version of a table can include other columns and different headings of the columns, depending on the sets of compiler options you choose.

Table 4-4 Performance measurements results table

	-O2	-O3	-O4	-O5
ST				
SMT2				
SMT4				
SMT8				

Optionally, you can repeat the step that is described in 4.2.7, “Probing the scalability” on page 142 with the compiler options that give the highest performance.

4.2.9 Evaluation of scalability

The core of the IBM POWER8 chip is a multithreaded processor. Therefore, measuring the performance of a single application thread is not recommended. More meaning in application performance is available when a whole core is used. The application performance depends on the SMT mode of a core as described in “Reason behind a conscious choice of an SMT mode” on page 124.

Before the scalability of an application is evaluated, select a favorable SMT mode and compilation options, as described in 4.2.8, “Evaluation of performance on a favorable number of cores” on page 143. Then, use that SMT mode and vary the number of cores to get the scalability characteristics. Enter the results of the measurements in the Performance row of Table 4-5 (the table is similar to Table 4-3 on page 143). Compute values for the Speed-up row based on the performance results. The number of columns in your version of the table depends on the maximum number of cores you obtained when you probed the scalability. For more information, see 4.2.7, “Probing the scalability” on page 142.

Table 4-5 Scalability evaluation results table

Number of cores	1	2	3	…	10	12	14	16	18	20
Number of sockets	1	1	1	…	1	2	2	2	2	2
Number of cores per socket	1	2	3	…	10	6	7	8	9	10
Performance										
Speed-up	—									

Note: When we probed the scalability, we were running the cores in ST mode. In the scalability evaluation step, we ran the cores in an SMT mode that we determined in the performance evaluation step. The total number of threads in each run of the scalability evaluation is equal to the number of used cores multiplied by the number of threads per core in a chosen SMT mode.

4.2.10 Conclusions

As a result of the benchmark, you have the following information for each application:

- The level of the application scalability in ST mode.
- A favorable SMT mode and compiler options that delivered the highest performance.
- Scalability characteristics in the favorable SMT mode.

4.2.11 Summary

The process that is used to facilitate your benchmarking includes the following steps:

1. Define the purpose of the benchmarking (see “Defining the purpose of performance benchmarking” on page 137) and choose which of the following options you will use for performance tuning:
   - Choice between different compilers
   - Choice of compiler optimization options
   - Choice of an SMT mode
   - Choice of the number of computing cores
   - Choice of runtime system parameters and environment variables
   - Choice of operating system parameters
2. Create the benchmarking plan (see “Benchmarking plans” on page 139).
3. Define the performance metric (see “Defining the performance metric and constraints” on page 139).
4. Define the success criteria (see “Defining the success criteria” on page 139).
5. Verify that the application works correctly (see “Correctness and determinacy” on page 140).
6. Probe the scalability (see “Probing the scalability” on page 142) to determine the limits of the application scalability:
   a. Complete the questionnaire on a scalability model and scalability criteria (see page 142).
   b. Complete a results table (see Table 4-3 on page 143).
7. Obtain favorable SMT mode and compilation options (see “Evaluation of performance on a favorable number of cores” on page 143) by completing Table 4-4 on page 144.
8. Evaluate the scalability (see “Evaluation of scalability” on page 144) by completing Table 4-5 on page 144.

4.3 Sample code for the construction of thread affinity strings

Example 4-3 provides the source code `t_map.c` for the program `t_map`. You can use this small utility to construct a text string that describes the mapping of OpenMP threads of an application to logical processors. Text strings of this kind are intended to be assigned to OpenMP thread affinity environment variables.

Example 4-3 Source code t_map.c (in C programming language) for the program t_map

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. #define MAX_TPC 8 // 8 is for the POWER8 processor (max SMT mode is SMT8)
5. #define MAX_CPS 10 // 8 is for a 16-core system, 10 is for a 20-core system
6. #define MAX_SPS 2
7. #define MAX_THR (MAX_TPC * MAX_CPS * MAX_SPS)
8.
9. void Print_Map(int sps, int cps, int tpc, int base) {
10.  const int maps[MAX_TPC][MAX_TPC] = {
11.    { 0                       },
12.    { 0,          4           },
```

```
13.    { 0,    2,    4          },
14.    { 0,    2,    4,    6    },
15.    { 0, 1, 2,    4,    6    },
16.    { 0, 1, 2,    4, 5, 6    },
17.    { 0, 1, 2, 3, 4, 5, 6    },
18.    { 0, 1, 2, 3, 4, 5, 6, 7 }
19.  };
20.
21.  const int sep = ',';
22.
23.  int thread, core, socket;
24.
25.  int tot = sps * cps * tpc;
26.  int cur = 0;
27.
28.  for (socket = 0; socket < sps; ++socket) {
29.    for (core = 0; core < cps; ++core) {
30.      for (thread = 0; thread < tpc; ++thread) {
31.        int shift = socket * MAX_CPS * MAX_TPC +
32.                                  core * MAX_TPC;
33.        shift += base;
34.        ++cur;
35.        int c = (cur != tot) ? sep : '\n';
36.        printf("%d%c", shift + maps[tpc-1][thread], c);
37.      }
38.    }
39.  }
40.}
41.
42.void Print_Usage(char **argv) {
43.    fprintf(stderr, "Usage: %s "
44.        "threads_per_core=[1-%d] "
45.        "cores_per_socket=[1-%d] "
46.        "sockets_per_system=[1-%d] "
47.        "base_thread=[0-%d]\n",
48.        argv[0], MAX_TPC, MAX_CPS, MAX_SPS, MAX_THR-1);
49.}
50.
51.int main(int argc, char **argv) {
52.  const int num_args = 4;
53.
54.  if (argc != num_args+1) {
55.    fprintf(stderr, "Invalid number of arguments (%d). Expecting %d "
56.        "arguments.\n", argc-1, num_args);
57.    Print_Usage(argv);
58.    exit(EXIT_FAILURE);
59.  }
60.
61.  int tpc = atoi(argv[1]);
62.  int cps = atoi(argv[2]);
63.  int sps = atoi(argv[3]);
64.  int base = atoi(argv[4]);
65.
66.  if (tpc < 1 || tpc > MAX_TPC ||
67.      cps < 1 || cps > MAX_CPS ||
```

```
68.      sps < 1 || sps > MAX_SPS) {
69.    fprintf(stderr, "Invalid value(s) specified in the command line\n");
70.    Print_Usage(argv);
71.    exit(EXIT_FAILURE);
72.  }
73.
74.  int tot = sps * cps * tpc;
75.
76.  if (base < 0 || base+tot-1 >= MAX_THR) {
77.    fprintf(stderr, "Invalid value specified for the base thread (%d). "
78.        "Expected [0, %d]\n", base, MAX_THR-tot);
79.    Print_Usage(argv);
80.    exit(EXIT_FAILURE);
81.  }
82.
83.  Print_Map(sps, cps, tpc, base);
84.
85.  return EXIT_SUCCESS;
86.}
```

Note: Example 4-3 on page 145 lists a revised version of the code that originally appeared in Appendix D of *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263. The code was adjusted to better fit the architecture of the IBM Power System S822LC servers.

To use the tool, you must compile the code first by using the C compiler of your choice. For example, run the following command with **gcc** compiler:

```
$ gcc -o t_map t_map.c
```

If you run the tool without any arguments, you are provided a brief hint about the usage, as shown in the following example:

```
$ ./t_map
Invalid number of arguments (0). Expecting 4 arguments.
Usage: ./t_map threads_per_core=[1-8] cores_per_socket=[1-10]
sockets_per_system=[1-2] base_thread=[0-159]
```

The utility needs you to specify the following amounts and locations of resources that you want to use:

- Number of threads per core
- Number of cores per socket
- Number of sockets
- The initial logical processor number (counting from zero)

Example 4-4 shows how to generate a thread mapping string for an OpenMP application that uses the following resources of a 20-core IBM Power Systems S822LC server:

- Twenty OpenMP threads in total
- Two threads on each core
- Ten cores on each socket
- Only the second socket

Example 4-4 Sample command for the generation of a thread mapping string

```
$ ./t_map 2 10 1 80
80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
```

The runtime system of an OpenMP application obtains the thread mapping string from an environment variable. You must use different OpenMP thread affinity environment variables, depending on the compiler you use for your OpenMP application. Table 4-6 lists references for OpenMP thread affinity environment variables.

Table 4-6 OpenMP thread affinity environment variables

Compiler family	Some compilers from the compiler family	OpenMP thread affinity environment variable

GNU Compiler Collection (GCC)	`gcc`
	`g++`
	`gfortran`
		`GOMP_CPU_AFFINITY`

IBM XL compilers	`xlc_r`
	`xlc++_r`
	`xlf2008_r`
		`XLSMPOPTS`, suboption `procs`

The information in Table 4-6 also applies to the OpenMP applications that are built with the derivatives of GCC and IBM XL compilers (for example, MPI wrappers).

Example 4-5 shows how to assign values to OpenMP thread environment variables to implement the scenario in Example 4-4.

Example 4-5 Assigning values to the OpenMP thread affinity environment variables

```
$ export XLSMPOPTS=procs="`t_map 2 10 1 80`"
$ echo $XLSMPOPTS
procs=80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
$ export GOMP_CPU_AFFINITY="`t_map 2 10 1 80`"
$ echo $GOMP_CPU_AFFINITY
80,84,88,92,96,100,104,108,112,116,120,124,128,132,136,140,144,148,152,156
```

Note: Example 4-5 implies that the `t_map` program is in your `PATH`. You need to specify the full path to the tool if the operating system does not find it in your `PATH`.

Example 4-5 shows the value of the environment variables with the **`echo`** command to demonstrate the result of the assignment. This command does not affect the affinity.

4.4 ESSL performance results

The ESSL library includes the implementation of the famous DGEMM routine, which is used in a large spectrum of libraries, benchmarks, and other ESSL routines. Therefore, its performance is significant.

DGEMM implements the following formula:

$$C = \alpha\left[A\right] \cdot \left[B\right] + \beta\left[C\right]$$

where, *alpha* and *beta* are real scalar values; and *A*, *B*, and *C* are matrixes of conforming shape.

Example 4-6 features a sample Fortran program with multiple calls of DGEMM for different sizes of input matrixes.

Example 4-6 ESSL Fortran example source code dgemm_sample.f

```
1.       program dgemm_sample
2.       implicit none
3.
4.       real*8 diff
5.       integer n, m, k
6.       integer maxn
7.       integer step
8.       integer i
9.       real*8,allocatable :: a(:,:), b(:,:), c(:,:)
10.      real*8,allocatable :: at(:,:), bt(:,:), ct(:,:)
11.      real*8 rmin
12.      real*8 seed1, seed2, seed3
13.      real*8 dtime, mflop
14.      real*8 flop
15.      integer tdummy, tnull, tstart, tend, trate, tmax
16.
17.      maxn = 20000
18.      step = 1000
19.
20.      seed1 = 5.0d0
21.      seed2 = 7.0d0
22.      seed3 = 9.0d0
23.      rmin = -0.5d0
24.
25.      call system_clock(tdummy,trate,tmax)
26.      call system_clock(tstart,trate,tmax)
27.      call system_clock(tend,trate,tmax)
28.      tnull = tend - tstart
29.
30.      allocate( at(maxn, maxn) )
31.      allocate( bt(maxn, maxn) )
32.      allocate( ct(maxn, maxn) )
33.      allocate( a(maxn, maxn) )
34.      allocate( b(maxn, maxn) )
35.      allocate( c(maxn, maxn) )
36.
37.      call durand(seed1, maxn*maxn, at)
```

```
38.       call dscal(maxn*maxn, 1.0d0, at, 1)
39.       call daxpy(maxn*maxn, 1.0d0, rmin, 0, at, 1)
40.
41.       call durand(seed2, maxn*maxn, bt)
42.       call dscal(maxn*maxn, 1.0d0, bt, 1)
43.       call daxpy(maxn*maxn, 1.0d0, rmin, 0, bt, 1)
44.
45.       call durand(seed3, maxn*maxn, ct)
46.       call dscal(maxn*maxn, 1.0d0, ct, 1)
47.       call daxpy(maxn*maxn, 1.0d0, rmin, 0, ct, 1)
48.
49.       do i = 1, maxn/step
50.         n = i*step
51.         m = n
52.         k = n
53.
54.         flop = dfloat(n)*dfloat(m)*(2.0d0*(dfloat(k)-1.0d0))
55.
56.         call dcopy(n*k, at, 1, a, 1)
57.         call dcopy(k*m, bt, 1, b, 1)
58.         call dcopy(n*m, ct, 1, c, 1)
59.
60.         call system_clock(tstart,trate,tmax)
61.         call dgemm('N','N',m,n,k,1.0d0,a,n,b,k,1.0d0,c,n);
62.         call system_clock(tend,trate,tmax)
63.
64.         dtime = dfloat(tend-tstart)/dfloat(trate)
65.         mflop = flop/dtime/1000000.0d0
66.
67.         write(*,1000) n, dtime, mflop
68.
69.       enddo
70.
71. 1000 format(I6,1X,F10.4,1X,F14.2)
72.
73.       end program dgemm_sample
```

The use of the commands that are shown Example 4-7 compile run this program by using different types of ESSL libraries (serial, SMP, and SMP CUDA). For SMP runs, it uses 20 SMP threads with each thread bound to a different POWER8 physical core.

Example 4-7 Compilation and execution of dgemm_sample.f

```
echo "Serial run"
xlf_r -O3 -qnosave dgemm_sample.f -lessl -o dgemm_fserial
./dgemm_fserial

echo "SMP run"
export
XLSMPOPTS=parthds=20:spins=0:yields=0:PROCS=0,8,16,24,32,40,48,56,64,72,80,88,96,1
04,112,120,128,136,144,152
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmp -o dgemm_fsmp
./dgemm_fsmp

echo "SMP CUDA run with 4 GPUs hybrid mode"
```

```
xlf_r -O3 -qnosave -qsmp dgemm_sample.f -lesslsmpcuda -lcublas -lcudart
-L/usr/local/cuda/lib64 -R/usr/local/cuda/lib64 -o dgemm_fcuda
./dgemm_fcuda

echo "SMP CUDA run with 4 GPUs non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1,2
./dgemm_fcuda

echo "SMP CUDA run with 3 GPUs (1st, 2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (1st, 2nd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0,1
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (1st, 2nd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (2nd, 3rd) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=1,2
./dgemm_fcuda

echo "SMP CUDA run with 2 GPUs (2nd, 3rd) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (1st) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=0
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (1st) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (4th) hybrid mode"
export ESSL_CUDA_HYBRID=yes
export CUDA_VISIBLE_DEVICES=3
./dgemm_fcuda

echo "SMP CUDA run with 1 GPU (4th) non-hybrid mode"
export ESSL_CUDA_HYBRID=no
./dgemm_fcuda
```

Figure 4-20 shows the dependence of performance in MFlops to size of matrixes for different calls from Example 4-6 on page 149.

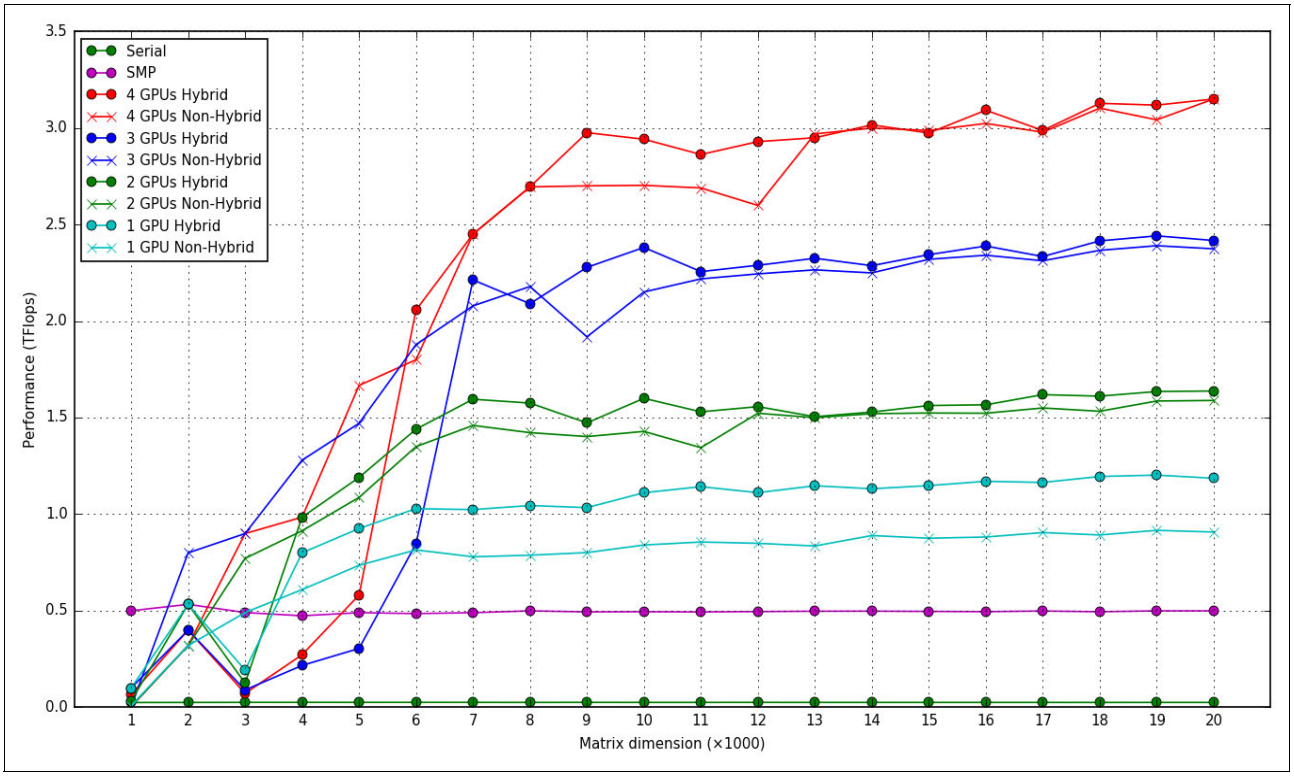


Figure 4-20 DGEMM performance results for different type of ESSL library

The chart shows that the GPU gives advantage in performance starting from the 3000 - 5000 problem size. A smaller size is not enough to run the computation in the NVIDIA (by using GPU) card, and it is better to run the computation run in the CPU by using the ESSL SMP library.

Another conclusion from these performance results is to use hybrid calls of the ESSL SMP CUDA library for large problem sizes, especially for runs with one GPU where improvement of performance is about 20%. You can look closer at performance of one GPU case in Figure 4-21, and can compare runs with different GPUs in the system.

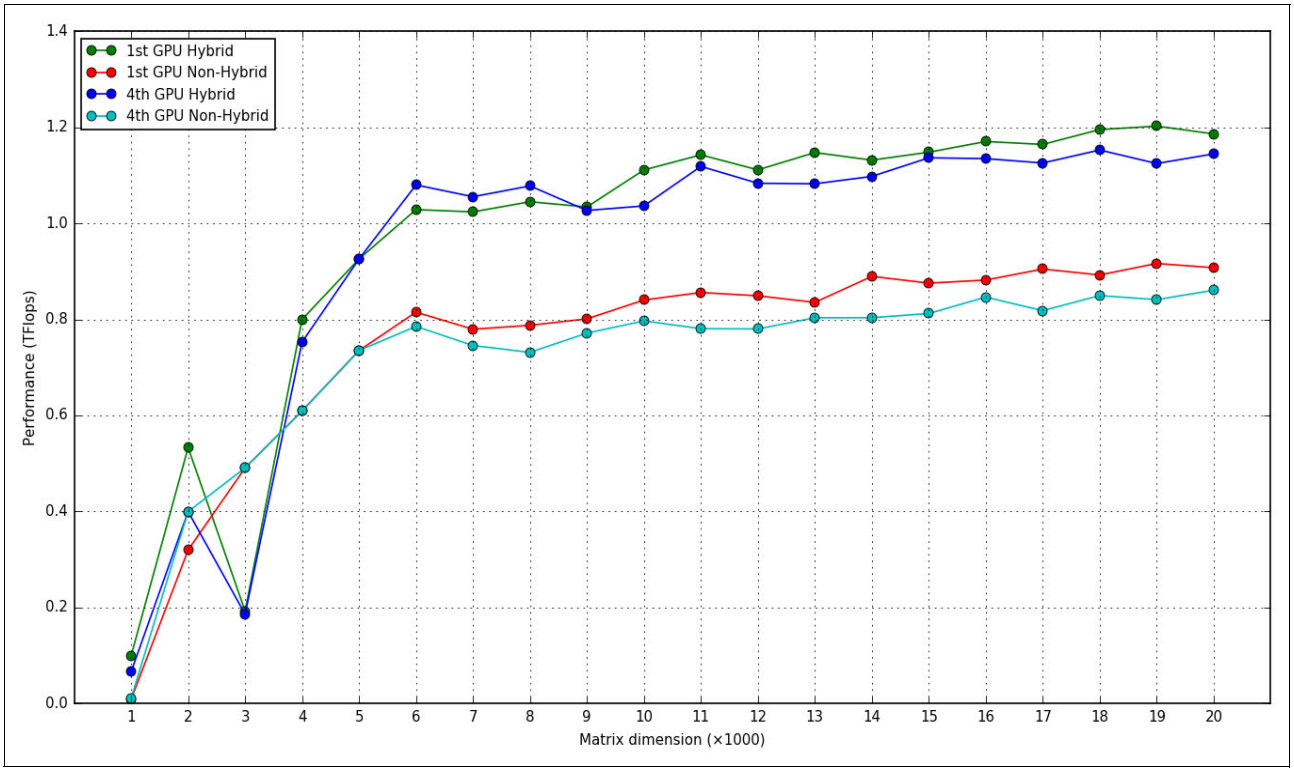


Figure 4-21 ESSL SMP CUDA runs with one GPU

The first GPU has slightly better results than the fourth GPU. It is possible because these GPUs connected to different NUMA nodes and connection delays can occur. Run your program on different GPUs to find the environment with the best performance results.

Figure 4-22 shows the performance chart for different combinations of two GPUs calls.

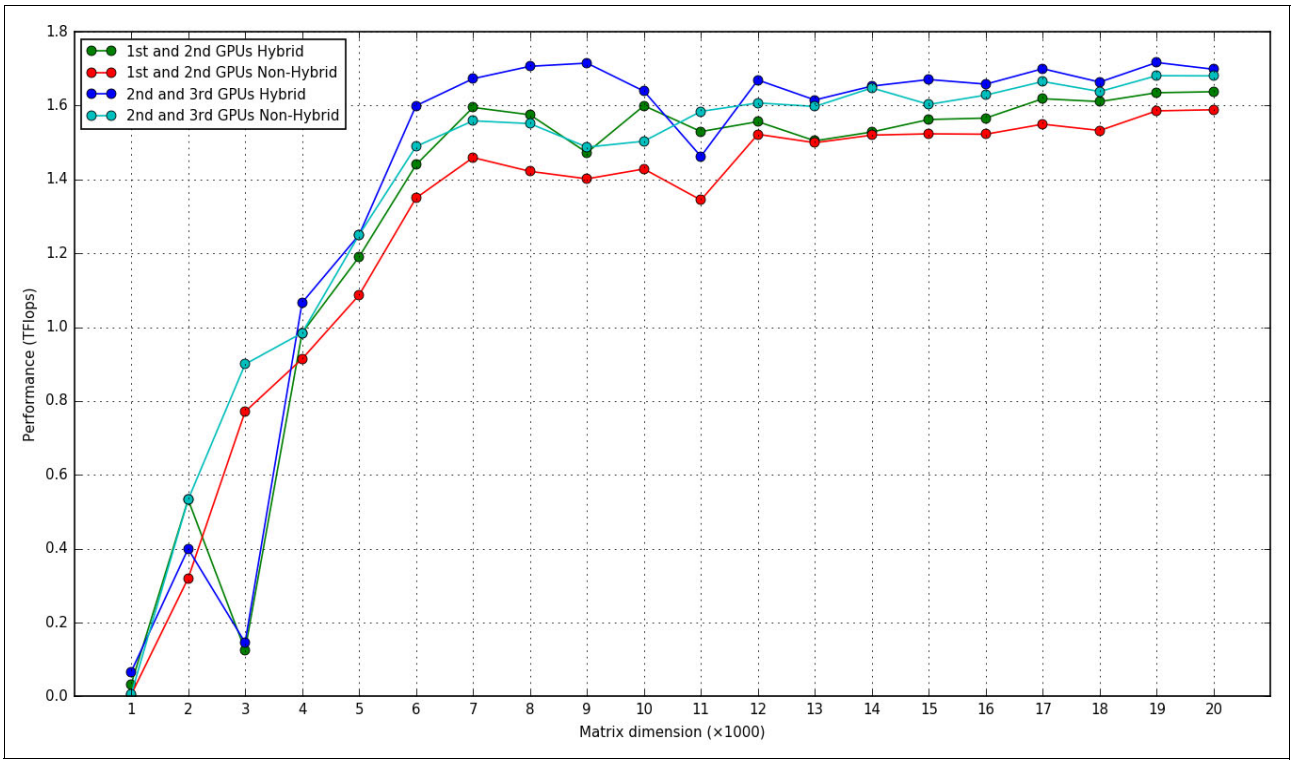


Figure 4-22 ESSL SMP CUDA runs with two GPUs

Note: Performance results can have drops because other jobs are running in the system at the same time.

4.5 GPU tuning

This section explains how to overcome performance degradation because of the Power Cap Limit that is featured in the NVIDIA GPU. Also, it shows how to manage shared access to GPUs by multi-process applications through Multi-Process Service (MPS).

4.5.1 Power Cap Limit

Attention: This section describes Tesla K80 GPU features. These features are not used on Pascal P100. However, this section remains as part of this book for the reader that still owns a K80 GPU.

While running your applications, the Power Cap Limit can exceed the Software Power Cap Limit of the NVIDIA GPU cards. If this problem occurs, performance is degraded because the frequency of the GPU clock is reduced because the GPU is using too much power.

To adjust the Power Cap Limit and check how performance looks after this adjustment, you need to make following checks:

1. Check the current, default, and maximum power limits by using the following command:

   ```
   nvidia-smi -q | grep 'Power Limit'
   ```

   Example 4-8 contains output for the example NVIDIA Tesla K80 after execution of this command.

   Example 4-8 NVIDIA Tesla K80 Power limits by default

   ```
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   Power Limit                 : 149.00 W
   Default Power Limit         : 149.00 W
   Enforced Power Limit        : 149.00 W
   Min Power Limit             : 100.00 W
   Max Power Limit             : 175.00 W
   ```

2. Set persistence to the following settings:

   ```
   nvidia-smi -pm 1
   ```

3. Increase the Power Cap limit to, for example, to 175 watts:

   ```
   nvidia-smi -pl 175
   ```

You can check the limits again after these three steps are completed by using the command from the first step, as shown in Example 4-9.

Example 4-9 NVIDIA Tesla K80 Power limits after changes

```
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
```

```
Max Power Limit             : 175.00 W
Power Limit                 : 175.00 W
Default Power Limit         : 149.00 W
Enforced Power Limit        : 175.00 W
Min Power Limit             : 100.00 W
Max Power Limit             : 175.00 W
```

Example 2-8 on page 43 (default) and Example 2-9 on page 45 (adjusted Power Cap Limit) show runs of sample ESSL SMP CUDA program. This technique gives performance improvements of approximately 2.5 times for this case.

Note: You must set the Power Cap Limit and persistence after each system restart.

4.5.2 CUDA Multi-Process Service

MPS is client/server runtime implementation of the CUDA API that helps multiple CUDA processes to share GPUs. MPS takes advantage of parallelism between MPI tasks to improve GPU utilization.

MPS contains the following components:

- Control Daemon Process

  Coordinates connections between servers and clients, and starts and stops server.
- Client Runtime

  Any CUDA application can use the MPS client run time from the CUDA driver library.
- Server Process

  This connection is the shared clients' shared connection to the GPU and provides concurrency between clients.

MPS is recommended for use with jobs that do not generate enough work for GPUs. If you have multiple runs at the same time, MPS helps to balance the workload of the GPU and increases its utilization.

Also, MPI programs that have different MPI-tasks but share GPU can see performance improvements by using MPS to control access to the GPU between tasks.

Note: The MPS Server supports up to 16 client CUDA contexts. These contexts can be distributed over up to 16 processes.

A recommendation is to use MPS with the EXCLUSIVE_PROCESS mode to be sure that only the MPS server runs on GPUs. Other compute modes, as described in 6.4.3, "Compute modes" on page 308, are not recommended or unsupported.

The CUDA program works by using MPS if the MPS control daemon runs in the system. The program at start attempts to connect to the MPS control daemon, which creates an MPS server or reuses a server if it has the same user ID as the user who started the job. Therefore, each user has its own MPS server.

The MPS server creates the shared GPU context for the different jobs in the system, manages its clients, and calls to GPU on behalf of them. An overview of this process is shown in Figure 4-23.

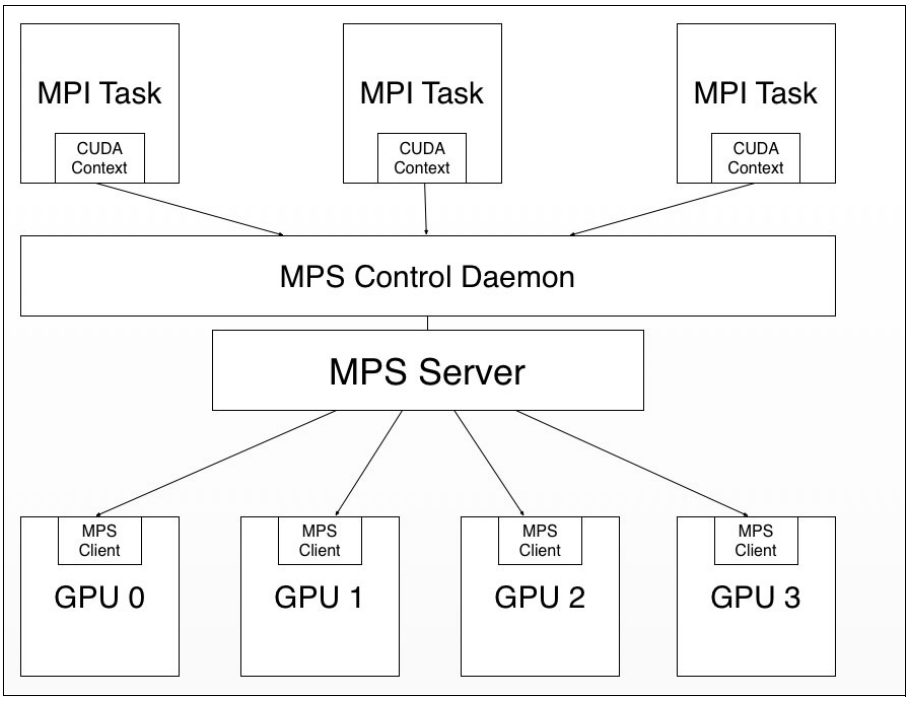


Figure 4-23 NVIDIA Multi-Process Service

To run MPS, you need to run the following commands, as root:

1. (This step is optional.) Set environment variable `CUDA_VISIBLE_DEVICES` to inform the system which GPUs will be used. To use all GPUs start from the second step, issue the following command:

   ```
   export CUDA_VISIBLE_DEVICES=0,1 #Use only first and second GPUs
   ```

2. Change the compute mode for all GPUs or to specific GPUs, which are chosen in the first step:

   ```
   nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
   nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
   ```

3. Start the daemon:

   ```
   nvidia-cuda-mps-control -d
   ```

To stop the MPS daemon, run the following command as root:

```
echo quit | nvidia-cuda-mps-control
```

Example 4-10 shows the output after the execution starts and stops steps in the example system.

Example 4-10 Start and stop commands of the MPS for 4 GPUs

```
$ nvidia-smi -i 0 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0000:03:00.0.
All done.
$ nvidia-smi -i 1 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0000:04:00.0.
All done.
$ nvidia-smi -i 2 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0002:03:00.0.
All done.
$ nvidia-smi -i 3 -c EXCLUSIVE_PROCESS
Set compute mode to EXCLUSIVE_PROCESS for GPU 0002:04:00.0.
All done.
$ nvidia-cuda-mps-control -d
$ echo quit | nvidia-cuda-mps-control
```

If the system has running jobs, running the **nvidia-smi** command provides output similar to the output that is shown in Example 4-11.

Example 4-11 nvidia-smi output for system with MPS

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0     23715    C   nvidia-cuda-mps-server                         89MiB |
|    1     23715    C   nvidia-cuda-mps-server                         89MiB |
|    2     23715    C   nvidia-cuda-mps-server                         89MiB |
|    3     23715    C   nvidia-cuda-mps-server                         89MiB |
+-----------------------------------------------------------------------------+
```

For more information about the CUDA MPS, see the *Multi-Process Service* documentation from NVIDIA.

4.6 Application development and tuning tools

Many tools for parallel program development and tuning are available for the cluster environment that is described in this book. Because different development models are allowed, more than one tool is often needed to develop the application.

This section describes some tools that were developed by IBM, and other tools that are useful but are maintained by third-party companies or open source communities.

Table 4-7 lists the tools and development models.

Table 4-7 Tools for development and tuning of parallel applications

Development model	Provider	Tool
MPI	Parallel Performance Toolkit	Call graph analysis
		I/O Profiling (MIO)
		MPI Profiling
		Hardware Performance Monitor (HPM)
	RogueWave Software	TotalView
	Allinea	DDT
Hybrid of MPI and OpenMP	Parallel Performance Toolkit	OpenMP profiling
CUDA	CUDA Toolkit	CUDA-MEMCHECK
		CUDA Debugger
		nvprof
		Nsight Eclipse Edition
Hybrid of MPI and CUDA	Parallel Performance Toolkit	GPU Performance Monitor (GPU)
MPI, OpenMP, PAMI, OpenSHMEM	Eclipse Parallel Tools Platform (PTP)	Integrated development environment (IDE)

4.6.1 Parallel Performance Toolkit

The IBM Parallel Performance Toolkit v2.3 provides a set of tools (see Table 4-7) and libraries to help during the development of applications that are written in C, C++, or Fortran by using pure MPI or hybrid with OpenMP and CUDA.

Note: The IBM Parallel Environment Developer Edition (PE DE) is branded to Parallel Performance Toolkit since version 2.3. Although it is renamed, most of components and usage documentation that is available in other IBM Redbooks publications and the IBM Knowledge Center can be up-to-date with this new version.

This toolkit encompassed the following main components:

- Parallel Performance Toolkit: Provides back-end runtime and development libraries and command-line tools.
- hpctView: Provides a GUI front end to the Parallel Performance Toolkit. Also, hpctView plug-ins to Eclipse for Parallel Application Developers (PTP) are available.

The fluxogram (see Figure 4-24) sees the analysis cycle of a parallel application by using the toolkit. As general rule, it evolves in the following phases:

- Instrument: An application executable file is designed to use a specific tool of the HPC Toolkit.
- Profile: Run the application to record execution data.
- Visualize: Visualize the resulting profile by using command-line or GUI (hpctView) tool.

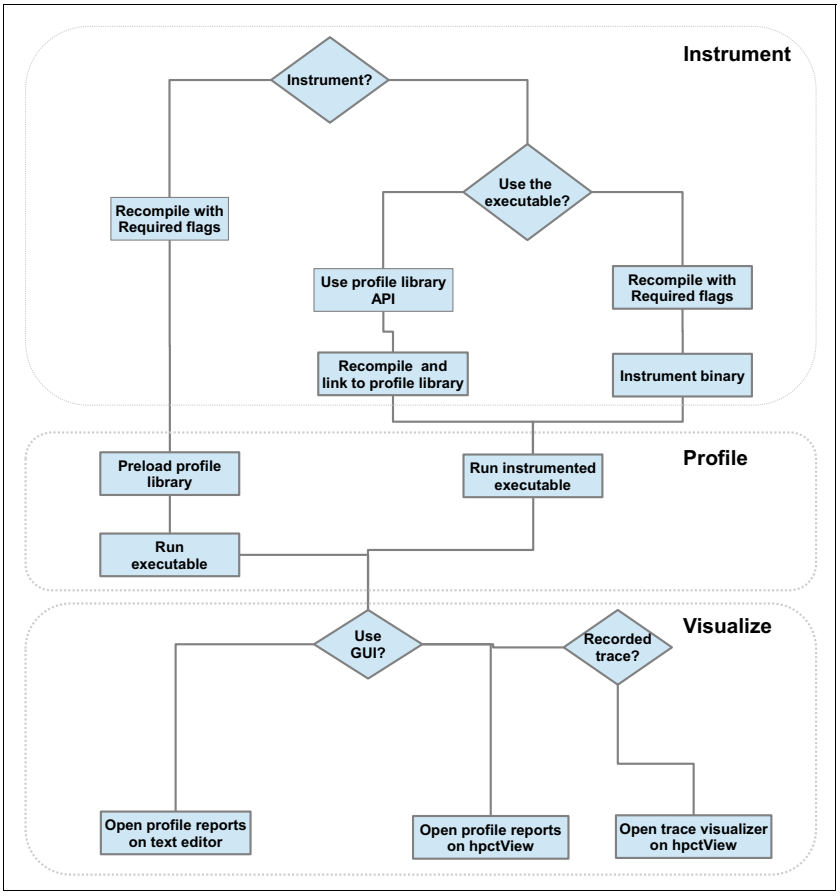


Figure 4-24 Fluxogram shows how to use IBM Parallel Performance Toolkit

Instrumentation is an optional step for some tools. If you need a fine-grained collection of runtime information, use any of the following instrument mechanisms that are provided:

- Binary instrumentation: Uses the **`hpctInst`** command or hpcView (GUI) to place probes on the executable file. It does not require you to rebuild the application.
- Profiling library API: Places calls to library methods to control profiling at the source level. It does require you to recompile the application and link it to the profiling library.

The profile phase is to run an instrumented executable file with a relevant work load so that useful data is collected. For situations where the binary is not instrumented, it requires you to preinstall at run time the corresponding profiling tool library and use of environment variables to control the behavior of the library. The library then takes over some method calls to produce the profile data.

To visualize the data that is produced by the profiling tools, you can use hpctView for the graphical visualization or Linux commands, such as **`cat`**, to inspect the report files.

The use of hpctView to carry on the instrument-profile-visualize analysis cycle is a convenient method because it permits fine-grained instrumentation at file, function, MPI call, or region levels. It also provides other means to profile the parallel application, and easy visualization of profile data and graphical tracing.

The Parallel Performance Toolkit tools are briefly described in the next sections, which show the many ways to profile and trace a parallel application. For more information about those topics, see the guide to getting started at the IBM Knowledge Center and click **Select a product** → **Parallel Performance Toolkit** → **Parallel Performance Toolkit 2.3.0**.

Application profiling and tracing tool

Often, the first task in performance tuning an application involves some sort of call graph analysis. To that purpose, the toolkit includes the application profiling and tracing tool, which is used to visualize GNU profile (gprof) files in large scale.

To use the tool, you must compile the application with the **`-pg`** compiler flag, then run the application to produce the profile data. It is important to use a workload that is as close as possible to the production scenario so that good profile data can be generated. As an alternative, the application can be profiled by using Oprofile and converting the collected data by using the opgprof tool. However, this approach does not scale well for SPMD programs.

The following example shows how to prepare a Fortran 77 application to generate gprof data:

```
$ mpif77 -g -c compute_pi.f
$ mpi77 -g -pg -o compute_pi compute_pi.o
```

After the application is run, one `gmon.out` profile file is built per task.

Because MPI applications generates several performance files (one per rank), the hpctView provides an visualization mode specifically to combine the various file and ease data navigation.

Complete steps to load the `gmon.out` files into hpctView:

1. Import the profiled application executable file by clicking **File** → **Open Executable**.
2. On the menu bar, select **File** → **Load Call Graph (gmon.out)**.
3. In the Select gmon.out Files window, browse the remote file system to select the `gmon.out` files that you want to load. Click **OK**.

The hpctView callgraph visualizer opens, as shown in Figure 4-25.

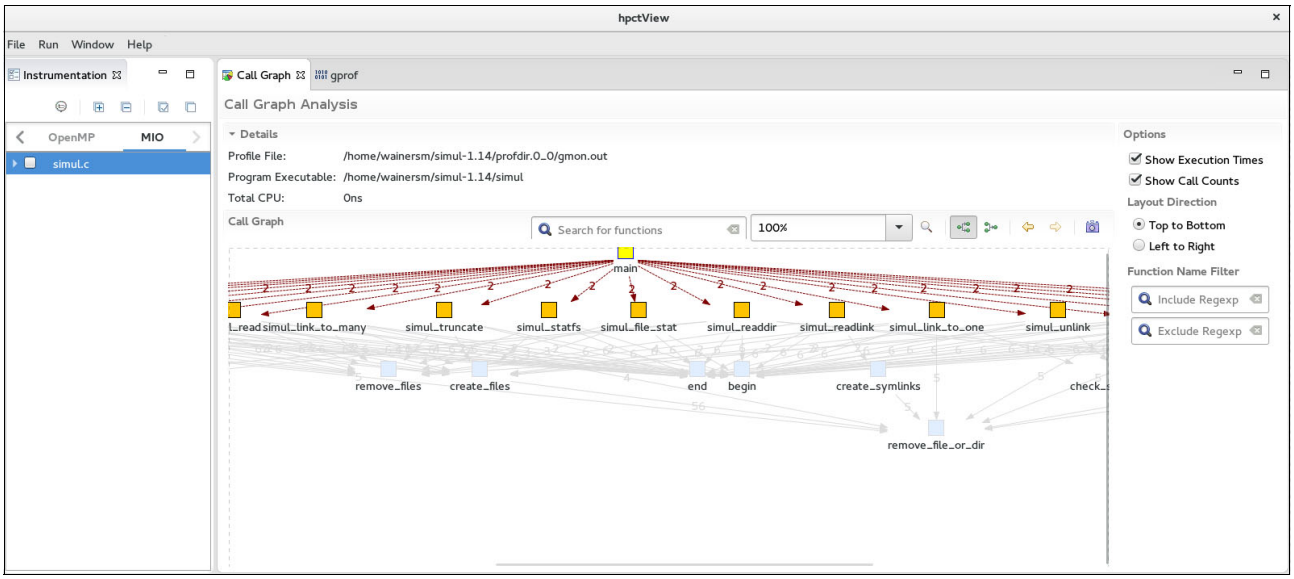


Figure 4-25 The hpctView callgraph visualizer with example gprof data loaded on callgraph tool

Information per called method number of calls, average amount of time spent in each call, and the total percentage of time, are presented in the gprof tab, as shown in Figure 4-26.

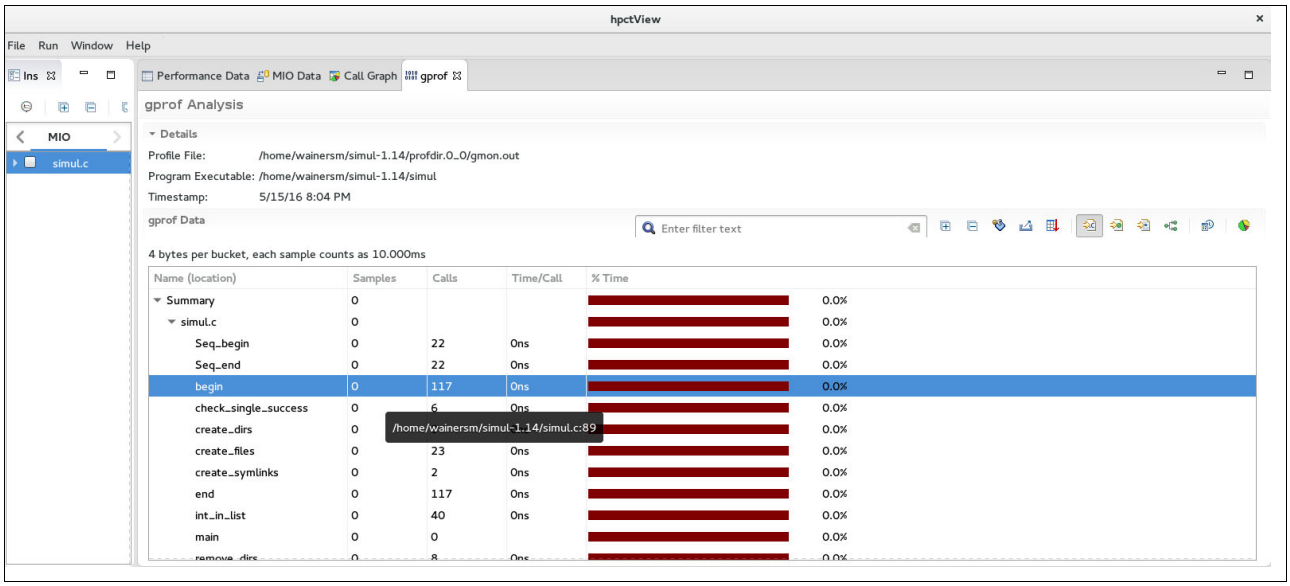


Figure 4-26 The hpctView callgraph visualizerwith an example of gprof loaded

MPI profiling and tracing

Performance analysis of MPI applications usually involves understanding many aspects of messages that are exchanged between the tasks, such as communication patterns, synchronization, and the data that is moved. To assist with application analysis is the Parallel Performance Toolkit, which provides tools for profiling, tracing, and visualization of the produced data.

The following profiles and reporting information (per tasks) are available:

- Number of times that MPI routines are run.
- Total wall time that is spent in each MPI routine.
- Average transferred bytes on message passing routines.
- Highlight the tasks with minimal, median, and maximum communication time. This information is provided only in the report of task 0.

You can use the profile and trace data to perform many types of analysis, such as communication patterns, hot spots, and data movements.

Example 4-12 shows the textual report for task 0 of an MPI program. The last section of the report (Communication summary for all tasks) shows the minimum (task 7), median (task 4), and maximum (tasks 5) communication time that was spent on tasks.

Example 4-12 HPC Toolkit - MPI profiling report for task 0

```
$ cat hpct_0_0.mpi.txt
-----------------------------------------------------------------
MPI Routine                     #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        1          0.0          0.000
MPI_Comm_rank                        1          0.0          0.000
MPI_Bcast                            1          4.0          0.000
MPI_Barrier                          1          0.0          0.000
MPI_Allreduce                        4         26.0          0.000
-----------------------------------------------------------------
total communication time = 0.000 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                       #calls   avg. bytes      time(sec)
                                     1          4.0          0.000

MPI_Allreduce                   #calls   avg. bytes      time(sec)
                                     3          8.0          0.000
                                     1         80.0          0.000

-----------------------------------------------------------------

Communication summary for all tasks:

  minimum communication time = 0.000 sec for task 7
  median  communication time = 0.001 sec for task 4
  maximum communication time = 0.002 sec for task 5
```

Example 4-13 on page 164 lists the files that are generated by the tool. Notice that although the parallel job that is used in this example has eight tasks, only the reports of tasks 0, 4, 5, and 7 are available. The MPI profiling tool by default generates reports for the four most significant tasks according to the communication time criteria: Task 0 (an aggregate of all tasks) and tasks with minimum, median, and maximum communication time.

Example 4-13 Listing the files generated by the tool

```
$ ls
hpct_0_0.mpi.mpt  hpct_0_0.mpi.txt  hpct_0_0.mpi.viz  hpct_0_4.mpi.txt
hpct_0_4.mpi.viz  hpct_0_5.mpi.txt  hpct_0_5.mpi.viz  hpct_0_7.mpi.txt
hpct_0_7.mpi.viz
```

Still taking as example the same parallel job, the task with maximum communication time is 5, as shown in Example 4-14.

Example 4-14 Contents of the profiling tasks

```
$ cat hpct_0_5.mpi.txt
-----------------------------------------------------------------
MPI Routine                     #calls   avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        1          0.0          0.000
MPI_Comm_rank                        1          0.0          0.000
MPI_Bcast                            1          4.0          0.000
MPI_Barrier                          1          0.0          0.000
MPI_Allreduce                        4         26.0          0.001
-----------------------------------------------------------------
total communication time = 0.002 seconds.
total elapsed time       = 2.725 seconds.

-----------------------------------------------------------------
Message size distributions:

MPI_Bcast                       #calls   avg. bytes      time(sec)
                                     1          4.0          0.000

MPI_Allreduce                   #calls   avg. bytes      time(sec)
                                     3          8.0          0.001
                                     1         80.0          0.000
```

Along with the profile data, the tool records MPI routines calls over time that can be used within hpctView trace visualizer. This information is useful to analyze the communication patterns within the parallel program.

The easiest way to profile your MPI application is to load the trace library before the execution by exporting the `LD_PRELOAD` environment variable. However, it can produce too much data from large programs.

The use of the hpctView instrumentation-run-visualize analysis cycle from within hpctView is also a convenient way to profile the application as it permits fine-grained instrumentation at levels of files, functions, MPI routines, or code regions.

The examples that are shown in this section were generated by using the script that is shown in Example 4-15 on page 165. To use the MPI profile preload library, the parallel program can be compiled by using the **`-g -Wl,--hash-style=sysv -emit-stub-syms`** flags.

Example 4-15 Script to profile MPI with HPC Toolkit

```
01 #!/bin/bash
02 #
03 # Use it as: $ MP_PROCS=<num> poe ./hpct_mpiprofile.sh <app> <args>
04 #
05
06 . /opt/ibmhpc/ppedev.hpct/env_sh
07
08 #
09 # HPCT MPI Trace control variables
10 #
11
12 ## Uncomment to set maximum limit of events traced.
13 ## Default is 30000.
14 #MAX_TRACE_EVENTS=
15
16 ## Uncomment to generate traces for all ranks.
17 ## Default are 4 tasks: task 0 and tasks with maximum, minimum and median
communication time.
18 #OUTPUT_ALL_RANKS=yes
19
20 ## Uncomment to enable tracing of all tasks.
21 ## Default are tasks from 0 to 255 ranks.
22 #TRACE_ALL_TASKS=yes
23
24 ## Uncomment to set maximum limit of rank traced.
25 ## Default are 0-255 ranks or all if TRACE_ALL_TASKS is set.
26 #MAX_TRACE_RANK=
27
28 ## Uncomment to set desired trace back level. Zero is level where MPI function
is called.
29 ## Default is the immediate MPI function's caller.
30 #TRACEBACK_LEVEL=
31
32 # Preload MPI Profile library
33 LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libmpitrace.so
34
35 $@
```

MPI I/O profiling

Characterization and analysis of I/O activities is an important topic of performance improvement, especially regarding parallel applications that demand critical usage of storage. The MPI I/O (MIO) tool consists of a profiler and trace recorder for collecting information about I/O system calls that are carried out by the parallel program to access and manipulate files.

The tool records information about I/O events that are triggered when the parallel application accesses files. General statistics per file are calculated out of the events that are probed; for example, the number of times each operation occurred and total time spent. Also, depending on the event, it collects the following information:

- Total of bytes requested
- Total of bytes delivered
- Minimum requested size in bytes
- Maximum requested size in bytes

- Rate in bytes
- Suspend wait count
- Suspend wait time
- Forward seeks average
- Backward seeks average

In particular, for events as read and write, the tool can trace operations. For example, you can access start offset, number of bytes, end offset.

To use MIO, you must compile the application by using the **`-g -Wl,--hash-style=sysv -Wl,--emit-stub-syms`** flags to prepare the binary for instrumentation.

You also can use `hpctView` to easily cycle instrumentation, execution, and visualization of information that is implemented by MIO tool. The tool abstracts the many environment variables that can be used to control type and amount of information that is gathered.

Figure 4-27 shows the hpctView visualization mode for data that is generated by using the MIO tool. More information about each I/O system call is provided on a per-task basis.

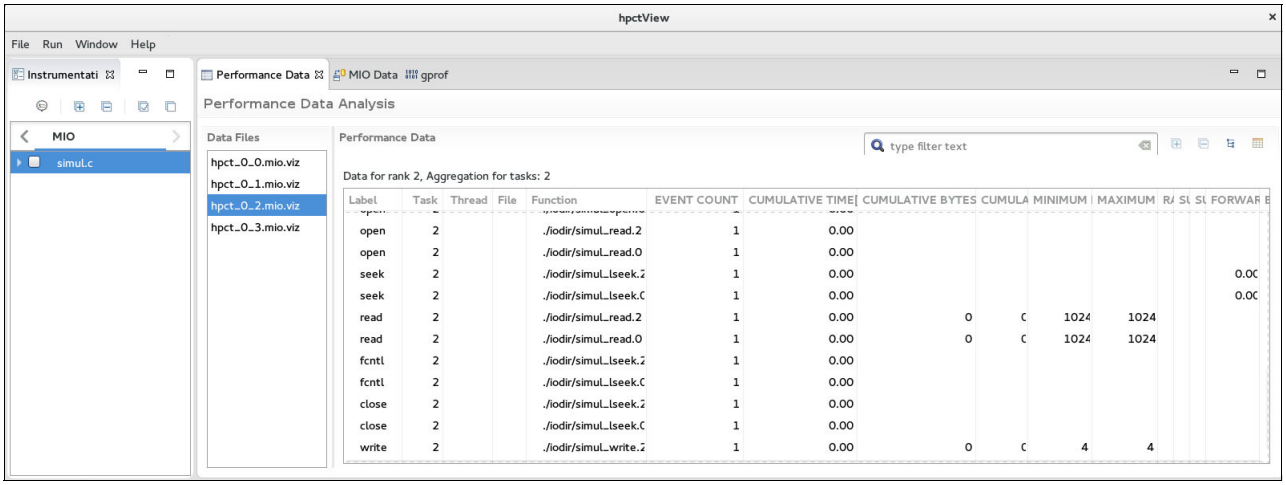


Figure 4-27 hpctView: MPI I/O tool profiling visualization

Another view of hpctView displays the I/O trace data, as shown in Figure 4-28.

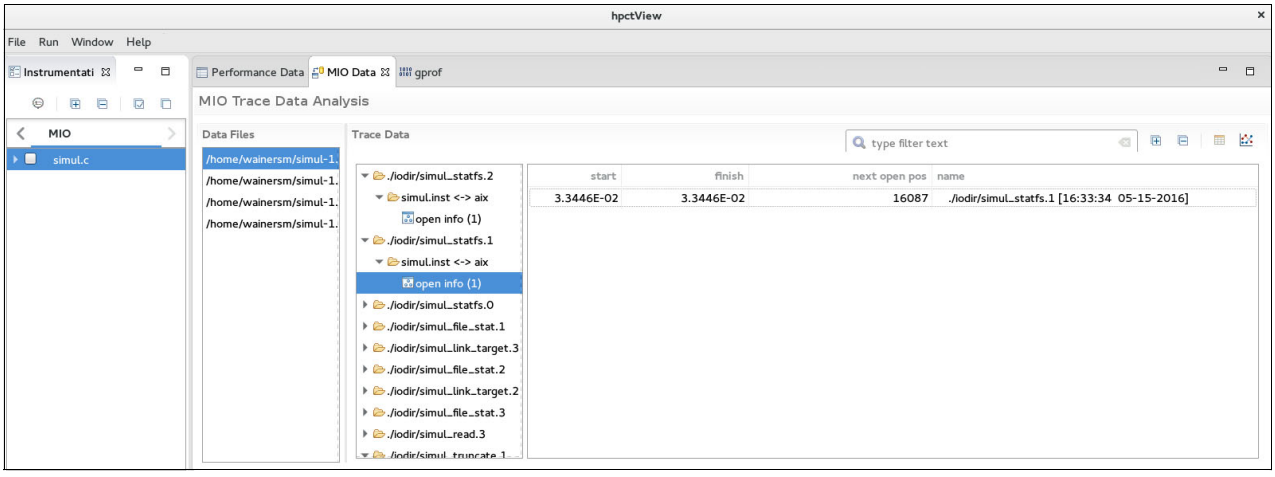


Figure 4-28 hpctView: MPI I/O tool trace visualization

Hardware Performance Monitor

The Hardware Performance Monitor (HPM) tool is an analog tool that provides easy SPMD programs profiling on Linux on Power Systems. By using HPM, you can profile MPI programs regarding any of the hardware events available or obtain any of the predefined metrics that are most commonly used in performance analysis.

The Processor Monitor Unit (PMU) of POWER8 is a part of the processor that is dedicated to recording hardware events. It has six Performance Monitor Counter (PMC) registers: Registers 0 - 3 can be programmed to count any of the more than one thousand events that are available, register 4 can count run instructions completed, and register 5 can count run cycles.

These events are important because they can reveal performance issues, such as pipeline bubbles, inefficient use of caches, and the high ratio of branches misprediction from the perspective of the processor. Metrics for performance measurements can also be calculated from hardware events, such as instructions per cycle, million of instructions per second (MIPS), and memory bandwidth.

For single programs, tools, such as Oprofile and Perf, provide access to system-wide or application profiling of those hardware events on POWER8. Parallel SPMD programs are often difficult to profile with these traditional tools because they are designed to deal with a single process (whether multi-threaded or not).

For more information about Oprofile, see the following resources:

- Oprofile tool website
- Perf tool Wiki webpage

For more information about predefined metrics, see the Derived metrics defined for POWER8 architecture page of the IBM Knowledge Center.

Figure 4-29 shows an HPM report that is opened by using hpctView.

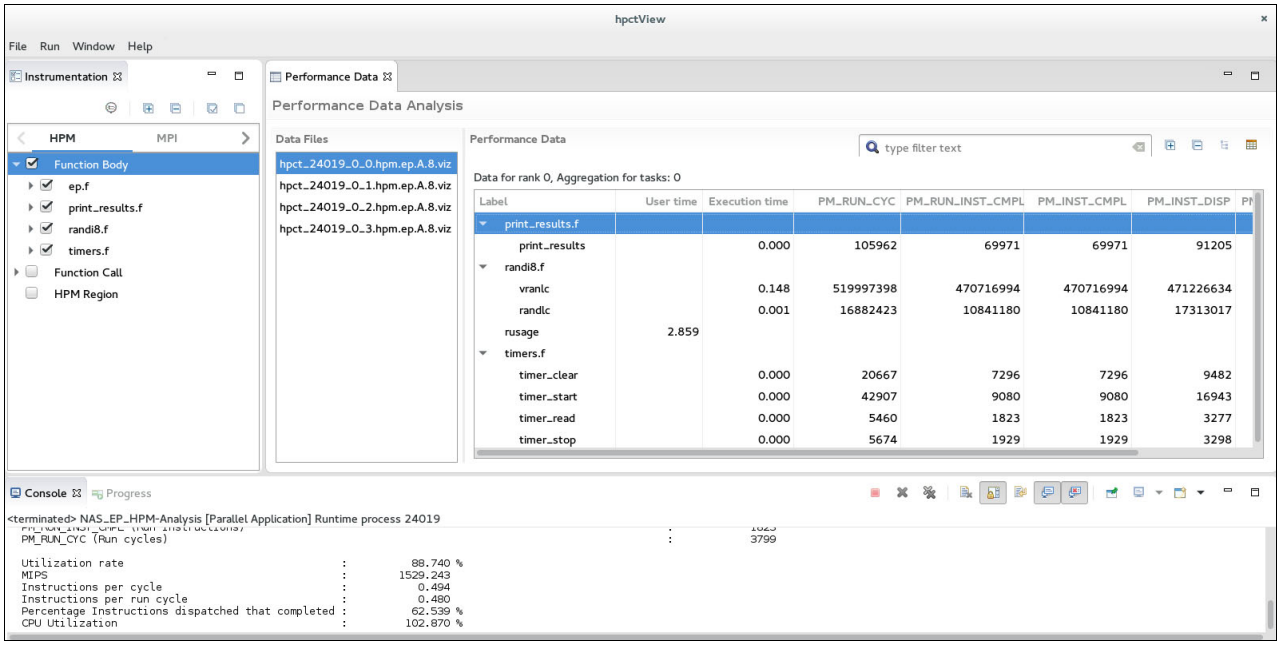


Figure 4-29 hpctView: HPC tool visualization

GPU Performance Monitoring

The GPU Performance Monitoring (GPM) tool is designed to profile and trace hardware events that are triggered in the GPU on hybrid MPI with CUDA C programs. Similar to HPM, GPM profiles raw hardware events and provides a set of predefined metrics that are calculated out of those events.

The HPM tool can profile raw hardware events and obtain metrics from a predefined list. The **gpmlist** command is the interface that is used to fetch the list of events and supported metrics. Example 4-16 shows a sample of the list of events that are available for the NVIDIA P100 GPU.

Example 4-16 How to list events available to GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d p100 -e -l
Domain 0
    83886081 tex0_cache_sector_queries - queries
    83886082 tex1_cache_sector_queries - queries
    83886083 tex0_cache_sector_misses - misses
    83886084 tex1_cache_sector_misses - misses
  Domain 1
    83886182 active_cycles - active_cycles
    83886183 elapsed_cycles_sm - elapsed_cycles_sm
  Domain 2
    83886085 fb_subp0_read_sectors - sectors
    83886086 fb_subp1_read_sectors - sectors
    83886087 fb_subp0_write_sectors - sectors
    83886088 fb_subp1_write_sectors - sectors
<... Output Omitted ...>
  Domain 3
    83886134 gld_inst_8bit - gld_inst_8bit
    83886135 gld_inst_16bit - gld_inst_16bit
    83886136 gld_inst_32bit - gld_inst_32bit
    83886137 gld_inst_64bit - gld_inst_64bit
    83886138 gld_inst_128bit - gld_inst_128bit
<... Output Omitted ...>
  Domain 4
    83886166 active_cycles_in_trap - active_cycles_in_trap
  Domain 5
    83886144 prof_trigger_00 - pmtrig0
    83886145 prof_trigger_01 - pmtrig1
<... Output Omitted ...>
```

Example 4-17 shows a sample of the metrics that are available for the NVIDIA P100 GPU.

Example 4-17 How to list metrics available to GPM profile tool

```
$ /opt/ibmhpc/ppedev.hpct/bin/gpmlist -d p100 -m -l
    19922945 inst_per_warp - Instructions per warp
        83886156 inst_executed - inst_executed
        83886152 warps_launched - warps_launched
    19922946 branch_efficiency - Branch Efficiency
        83886177 branch - branch
        83886176 divergent_branch - branch
    19922947 warp_execution_efficiency - Warp Execution Efficiency
        83886157 thread_inst_executed - thread_inst_executed
        83886156 inst_executed - inst_executed
<... Output Omitted ...>
```

As with other HPC Toolkit tools, GPM is also flexible regarding the instrument-profile-visualize cycle possibilities. It can also be used with the HPM tool.

The following environment variables are used to configure the profiler:

- `GPM_METRIC_SET`: Sets the metrics to be profiled.
- `GPM_EVENT_SET`: Sets the hardware events to be profiled. Cannot be exported with `GPM_METRIC_SET`.
- `GPM_VIZ_OUTPUT=y`: Enables the creation of visualization files that are used by hpctView visualizer (disabled by default).
- `GPM_STDOUT=n`: Suppresses the profiler messages to standard output (stdout). Sends messages to `stdout` by default.
- `GPM_ENABLE_TRACE=y`: Turns on trace mode (is off by default).

Example 4-18 shows a profiling session of a hybrid MPI and CUDA C application that is named `a.out`. The lines ranging 1 - 26 are the content of the `gpm.sh` script, which exports the GPM control variables (lines 13 - 15) that instruct the system to calculate the `sm_efficiency` metric. Then, they evoke the wrapper that is named `gpm_wrap.sh`. In turn, the wrapper (lines 29 - 35) script preloads (line 33) the GPM library. The remained lines are messages that are printed by GPM on standard output.

Example 4-18 Profiling with GPU Performance Monitor tool

```
 1 $ cat gpm.sh
 2
 3 #!/bin/bash
 4
 5
 6
 7 export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda-7.5/lib64:/opt/ibmhpc/pecurrent/m
pich/gnu/lib64:/opt/ibmhpc/pecurrent/gnu/lib64/
 8
 9
10
11 # GPU Performance Monitor - variables
12
13 export GPM_METRIC_SET=sm_efficiency
14
15 export GPM_VIZ_OUTPUT=y
```

```
16
17
18
19 export MP_CUDA_AWARE=yes
20
21 export MP_RESD=poe
22
23 export MP_PROCS=2
24
25 poe ./gpm_wrap.sh
26
27 $ cat gpm_wrap.sh
28
29 #!/bin/bash
30
31
32
33 export LD_PRELOAD=/opt/ibmhpc/ppedev.hpct/lib64/preload/libgpm.so:$LD_PRELOAD
34
35 ./a.out
36
37 $ ./gpm.sh
38
39
40
41  GPM (IBM HPC Toolkit for PE Developer Edition) results
42
43
44
45 Device 0 (Tesla K80):
46
47 ------------------------------------------------
48
49   Symbol Name: cudaMalloc
50
51   ------------------------------------------------
52
53               Memory Kind: cudaMalloc
54
55              Bytes Copied: 4096
56
57
58
59   Symbol Name: cudaMemcpy
60
61   ------------------------------------------------
62
63               Memory Kind: cudaMemcpyHostToDevice
64
65              Bytes Copied: 4096
66
67
68
69   Kernel Name: _Z12vecIncKernelPii
70
```

```
 71   ------------------------------------------
 72
 73          GPU Kernel Time: 0.000005 seconds
 74
 75             W/clock Time: 0.012338 seconds
 76
 77            sm_efficiency: 31.155739%
 78
 79
 80
 81   Symbol Name: cudaMemcpy
 82
 83   ------------------------------------------
 84
 85              Memory Kind: cudaMemcpyDeviceToHost
 86
 87             Bytes Copied: 4096
 88
 89
 90
 91   Totals for device 0
 92
 93   ------------------------------------------
 94
 95           Total Memory Copied: 8192 bytes
 96
 97         Total GPU Kernel Time: 0.000005 seconds
 98
 99            Total W/clock Time: 0.012338 seconds
100
101                 sm_efficiency: 31.155739%
102
```

As a result, the tool reports the calculated metric or counted event on per-task files. Example 4-19 shows the report that is generated for execution in Example 4-18 on page 169.

Example 4-19 Report generated by GPU Performance Monitoring tool

```
$ cat hpct_0_0.gpm.a.out.txt

 GPM (IBM HPC Toolkit for PE Developer Edition) results

  Totals for device 0
  ------------------------------------------
          Total Memory Copied: 8192 bytes
        Total GPU Kernel Time: 0.000005 seconds
           Total W/clock Time: 0.012338 seconds
                sm_efficiency: 31.155739%
```

For more information about the GPM tool, see the Using GPU hardware counter profile page of the IBM Knowledge Center website.

IBM HPC Toolkit hpctView

The hpctView is a front end view for the Parallel Performance Toolkit run time and command-line tools. It provides a GUI to instrument the parallel executable file, which you can run to collect data and visualize information from your development workstation (desktop or notebook).

The tool can be run from the developer workstation. Version 2.3 is supported on Mac OS 10.9 (Mavericks) or later, Microsoft Windows 64 bit, and any 64 bit Linux distribution.

Find the hpctView as a tarball file distributed in toolkit distribution media. Then, proceed as shown in the following example to install it and run it on Linux workstation:

```
$ tar xvzf hpctView-2.2.0-0-linux-gtk-x86_64.tar.gz
$ cd hpctView
$ ./hpctView &
```

The hpctView application implements the complete cycle of instrument-run-visualize (see 4.6.1, “Parallel Performance Toolkit” on page 159) that is required to use the Parallel Performance Toolkit tools. As an alternative, profile and trace data files can be generated by using the command-line tools and libraries then loaded into hpctView just for visualization and analysis.

In addition to the hpctView stand-alone application, the toolkit includes plug-ins to enable hpctView within Eclipse Integrated Development Environment (IDE). For more information, see 4.6.3, “Eclipse for Parallel Application Developers” on page 174.

The first step to profile and trace an application is to prepare the program execution. To instrument the binary, complete the following steps in hpctView:

1. In the Instrumentation pane (left side pane), select the profiler for which you want to instrument the binary. The options are HPM, MPI, OpenMP, and MIO.
2. Click **File** → **Open Executable**. A window opens that includes a connection to the remote machine. Select the file to be instrumented, as shown Figure 4-30.

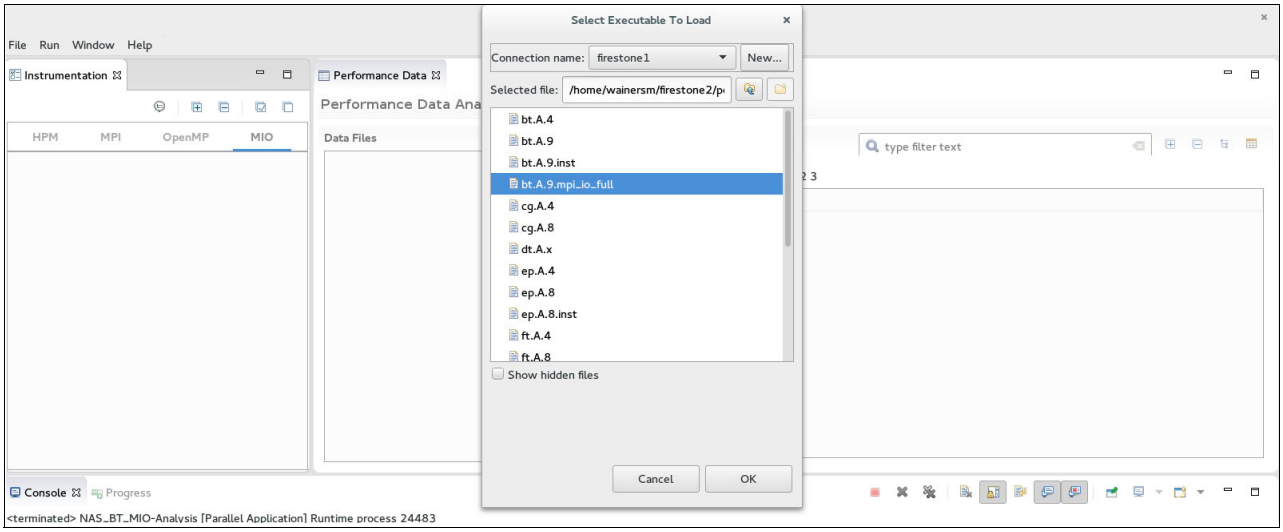


Figure 4-30 hpctView: Select binary for instrumentation

3. Click **New** to create a connection to the remote system where the binary is hosted if it does not exist.

4. Select the binary. Click **OK**.

   The binary content is analyzed and sections that correspond to the instrumentation portions of the application code are listed in tree format in the Instrumentation pane. Different instrumentation options are displayed for each type of profiling tool. For example, Figure 4-31 shows the options (Function body, Function Call, HPM Region) for the instrumentation of a binary to be used with the HPM tool. Select the instrumentation points according to your needs.

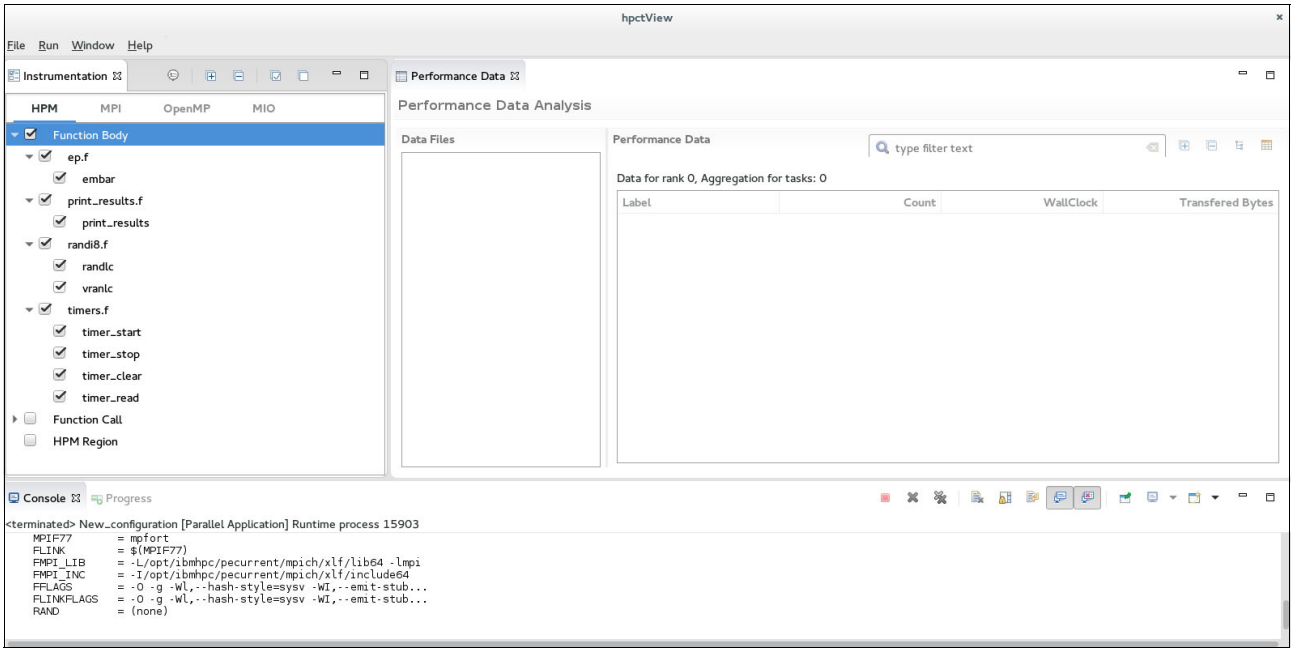


Figure 4-31 hpctView: Binary instrumentation

5. Click the instrument executable icon in the Instrumentation pane. If the operation succeeds, a file that is named `<binary>.inst` is saved in the same folder as the original binary.

From within hpctView, you can then start the instrumented binary by using IBM Spectrum MPI or submitting a batch job to IBM Spectrum Load Sharing Facility (LSF). The Parallel Performance Toolkit provides many environment variables to control the tools behavior (for example, the amount of data to be collected).

After the instrumented binary is run, a view of the corresponding tool is automatically opened. Now, the many options that are available to make analysis of the data collected as, for example, interval trace viewer can be used.

For more information about hpctView use, see the Using the hpctView application page of the IBM Knowledge Center website.

4.6.2 Parallel application debuggers

The GNU Debugger (GDB) is a known debugger for serial and multi-threaded application that can also be used to debug parallel SPMD programs. By using the IBM Spectrum MPI, the GDB can be attached to individual MPI processes or start many debug instances with the **`mpirun`** command. Both techniques require complex setup, do not scale well, and often do not deliver suitable results.

The use of specialized debuggers for parallel applications is recommended. The following products support debugging of C, C++, and Fortran applications in IBM Power Systems:

- Allinea DDT debugger
- RogueWave TotalView

These parallel debuggers are tightly integrated with IBM Spectrum MPI. The use of the **`mpirun --debug`** command starts DDT or TotalView, if it is reachable in the system's PATH.

For more information about DDT and TotalView integration with IBM Spectrum MPI, see the following IBM Knowledge Center pages:

- Debugging applications with the Allinea DDT debugger and IBM Spectrum MPI
- Debugging applications with the TotalView debugger and IBM Spectrum MPI

4.6.3 Eclipse for Parallel Application Developers

Eclipse for Parallel Application Developers provides an entire IDE. It bundles some open source projects under the Eclipse umbrella that includes C/C++ Development Tools (CDT) and PTP. It provides a complete environment for coding, compiling, starting, analyzing, and debugging applications that are written in C, C++, and Fortran. It uses MPI, OpenMP, PAMI, and OpenSHMEM.

The specialized editors feature syntax highlighting, code assistant, place markers for compiler errors, auto-completion, and many other features to improve productivity. They also include static analysis tools that are used to identify common coding mistakes, such as MPI barriers mismatch.

You can build programs by using Eclipse managed makefile (integrated with IBM XL and GNU compilers), Makefile, Autotools, and custom scripts and commands. The integration with XL and GNU compilers also includes build output parsers that can correlate errors to source code files.

Remotely started applications on IBM Spectrum MPI (Open MPI) or IBM Spectrum LSF are supported.

Its parallel debugger provides specific debugging features for parallel applications that distinguish it from the Eclipse debugger for serial applications. Its parallel debugger can also start and remotely debug parallel applications through Open MPI and LSF.

For more information about the topics that are introduced in this section, see the Eclipse PTP user guide at the following website by selecting the **Parallel Tools Platform (PTP) User Guide**.

To install Eclipse, download its tarball file and then uncompress it on your development workstation. For more information, Visit the Eclipse download website, click **Download Packages**, and select **Eclipse for Parallel Application Developers**.

Download the Eclipse tarball file for your workstation operating system and architecture. At the time of this writing, the latest Eclipse version was 4.6.2 (also known as Eclipse Neon.2).

Remote synchronized development model

The Eclipse for Parallel Application developers implements a remote development working model in which the code editing is done locally within a workbench (GUI). However, other tasks that are usually required to be performed on the server side, such as build, launch, debug, and profile, are carried out remotely.

The model uses a synchronized project type to keep local and remote copies of the working directory updated so that code programming is minimally affected by network latency or slow response time in the editor. The C, C++, and Fortran editors appear as though you are developing locally on the Power Systems machine even though all of the resources are on the server side. By using this approach, you do not need to rely on a cross-compilation environment, which is often difficult to set up. This approach provides an efficient method for remote development.

Parallel debugger

The Eclipse PTP parallel debugger extends the default C/C++ Eclipse debugger for serial application to scale parallel programs. It combines the information that is obtained from the many processes and threads that a parallel job is composed of into a single viewer. Therefore, you can browse the entities separately, if needed.

Debugging operations can be carried out on any arbitrary subset of processes within a parallel job; for example, as step in and out, and stop at breakpoint. A different type of breakpoint, which is called a *parallel breakpoint*, can be applied in these collections of processes.

The debugger can start and remote debug parallel programs by using several types of workload managers, including the Open MPI and Spectrum LSF.

Using IBM hpctView within Eclipse

The IBM Parallel Performance Toolkit provides plug-ins for hpctView (see 4.6.1, “Parallel Performance Toolkit” on page 159) be installed on top of Eclipse.

Find the plug-ins in Parallel Performance Toolkit distribution media. Then, uncompress the file as shown in the following example:

```
$ mkdir ppedev_update-2.3.0
$ unzip ppedev_update-2.3.0-0.zip -d ppedev_update-2.3.0
```

Next, install the plug-ins by using the Eclipse install software by completing the following steps:

1. At tool bar menu, select **Help** → **Install New Software**.
2. Click **Add** to create a repository location from where Eclipse can find plug-ins to be installed.
3. Enter the location where hpctView update site files were uncompressed. Click **OK** to add the repository configuration.
4. In the selection tree that is appears, select its root element to install all of the plug-ins. Click **Next**. Then, click **Next**.
5. Read and accept the license agreement to continue the installation.
6. Click **Finish** to install the plug-ins.

For more information about installing the plug-ins, see the Updating and installing software section of the Eclipse workbench user guide website. In the left side navigation tree at the website, click **Workbench User Guide** → **Tasks** → **Updating and installing software**.

To use hpctView, open its perspective within Eclipse by completing the following steps:

1. Click **Window** → **Perspective** → **Open Perspective** → **Other**.
2. In the perspective window that opens, select **HPCT** and click **OK**.

4.6.4 NVIDIA Nsight Eclipse Edition for CUDA C/C++

The NVIDIA Nsight Eclipse Edition is a complete IDE for development of CUDA C and C++ applications that run on NVIDIA GPUs. It provides developers with an Eclipse-based GUI that includes tools for a complete cycle of development: Code writing, compiling, debugging, and profiling and tuning.

The following development models are available:

- Local: The complete development cycle is carried out at the machine where Nsight is running. The program that is produced is targeted for the local host architecture and GPU.
- Remote: The complete development cycle is carried out at a remote host where Nsight is running.

In the next sections, we describe the basic use of Nsight for remote development of CUDA C applications for IBM Power Systems. The examples that are described in this section use a workstation with Linux Fedora 23 64-bit that is running the NVIDIA Nsight Eclipse Edition that is included with the CUDA Toolkit 8.0.

Not all of Nsight's functions are described in this book. Instead, we show how to create, compile, and debug projects. For more information, see the Nsight Eclipse Edition website.

Running NVIDIA Nsight Eclipse Edition

The Nsight GUI can be started as shown in the following example:

```
$ export PATH=/usr/local/cuda-8.0/bin:$PATH
$ nsight &
```

Creating a project for remote development

Before you begin, check the requirements and complete the following steps:

1. Ensure that the machine (usually the cluster login node) you are going to use for remote development has Git installed. Nsight uses **`git`** commands to synchronize local and remote copies of the project files.
2. Connect to the remote machine to configure user name and email that is used by Git, as shown in the following example:

   ```
   $ git config --global user.name "Wainer dos Santos Moschetta"
   $ git config --global user.email "wainersm@br.ibm.com"
   ```

3. Ensure that the directory that is going to hold the project files in the remote machines is created, as shown in the following example:

   ```
   $ mkdir -p ~/wainer/cudaBLAS
   ```

In the Nsight GUI, complete the following steps to create a CUDA C/C++ project:

1. On the main menu tool bar, click **File** → **New** → **CUDA C/C++ Project** to open the window that is shown in Figure 4-32.

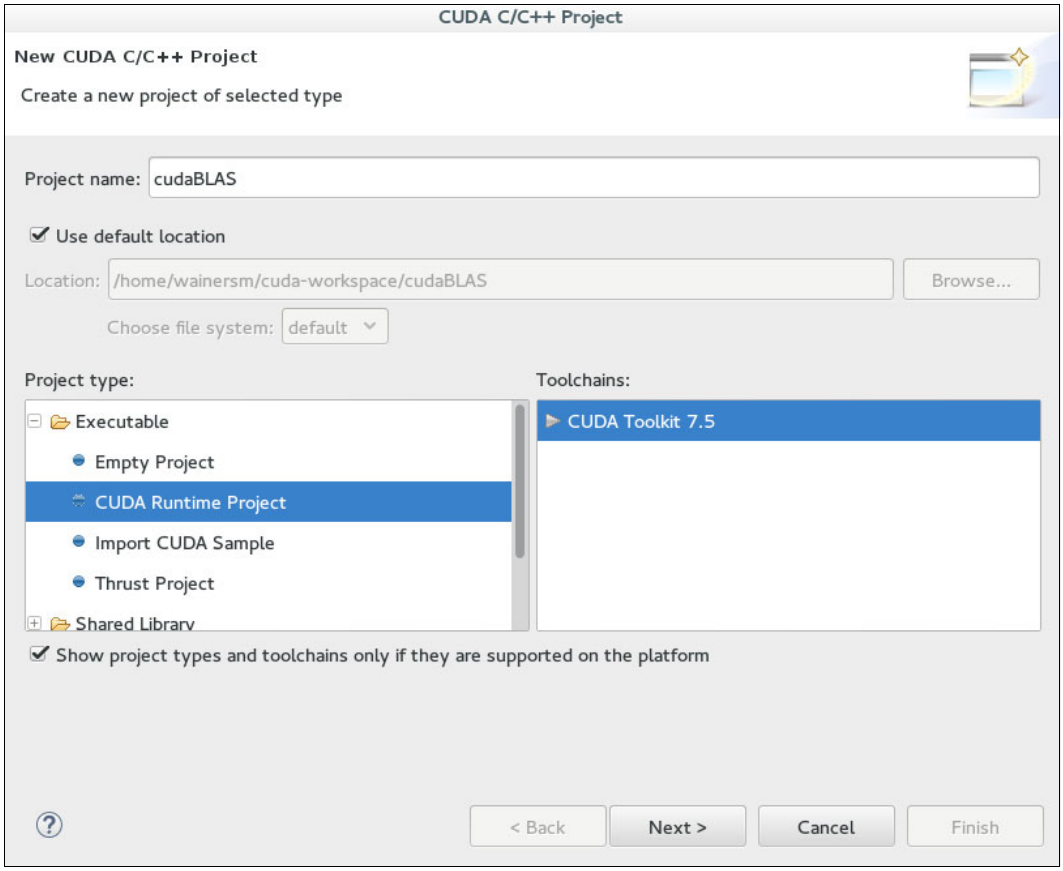


Figure 4-32 Nsight: New CUDA C/C++ project wizard

2. Click **Next**. Then, click **Next** again.

3. Select **3.7** for the **Generate PTX code** and **Generate GPU code** options, as shown in Figure 4-33. Click **Next**.

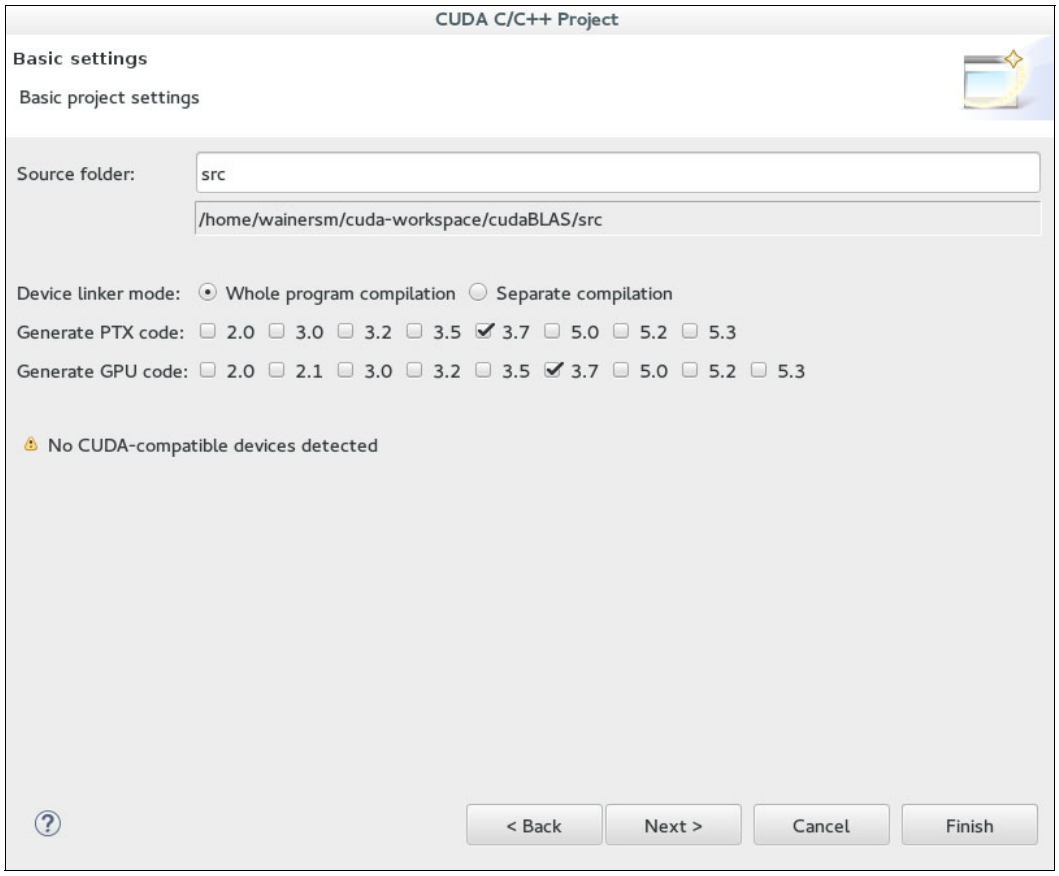


Figure 4-33 Nsight: New CUDA C/C++ project basic settings

4. Click **Manage** (as shown in Figure 4-34) to configure a new connection to the remote machine.

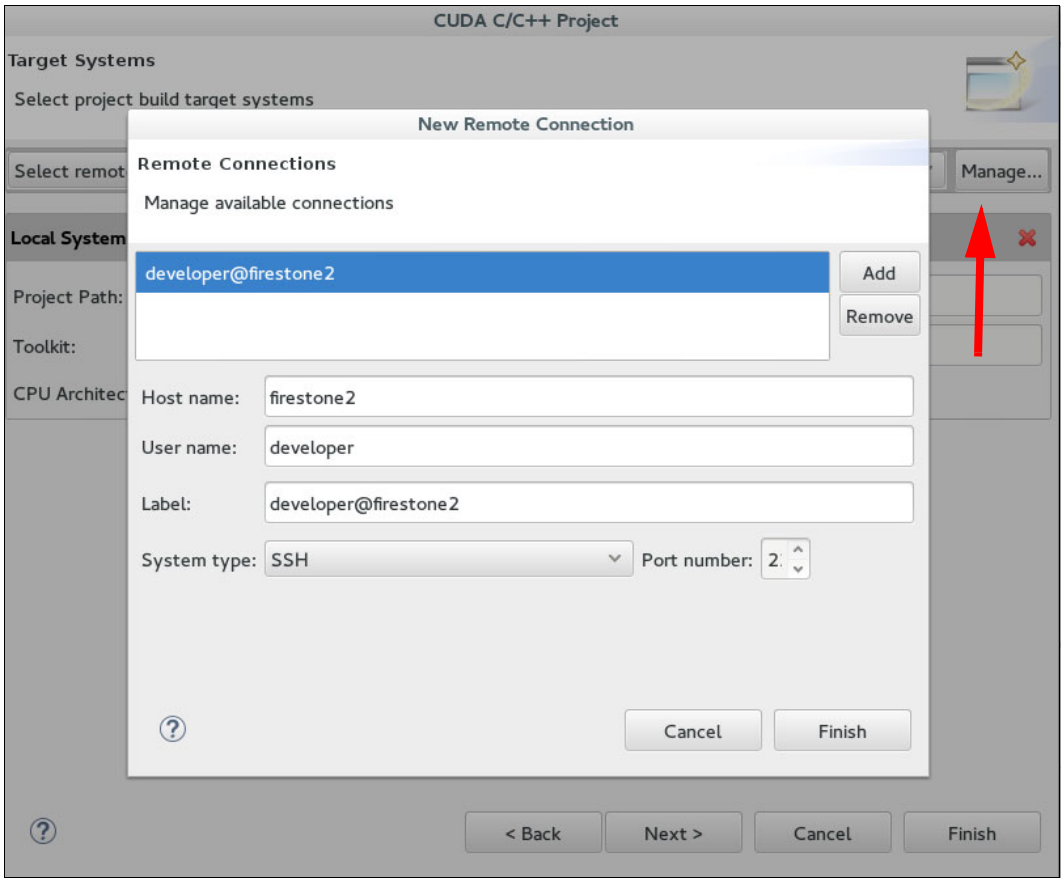


Figure 4-34 Nsight: New CUDA C/C++ project remote connection setup

5. As shown in Figure 4-34, enter the connection information fields (host name and user name). Click **OK**.

6. Enter the Project path and Toolkit information and set the CPU Architecture fields for the newly created connection (see Figure 4-35). You must remove the Local System configuration. Click **Next**. Then, click **Finish**.

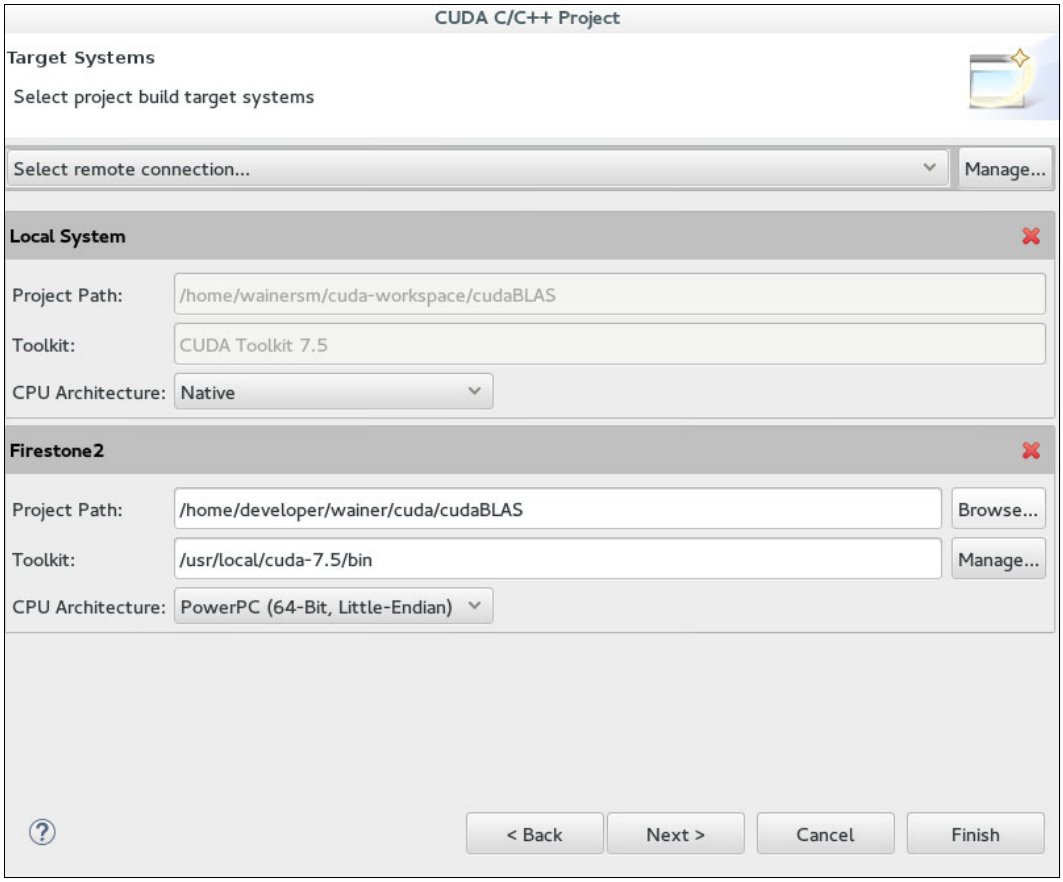


Figure 4-35 Nsight - new CUDA C/C++ project further remote configuration setup

After you successfully create the project, complete the following steps to configure the Nsight to automatically synchronize the local and remote project files:

1. Create a source code file (for example, `main.c`).
2. Right-click the project in the Project Explorer pane and select **Synchronize** → **Set active**. Then, choose the option that matches the name of the connection configuration to the remote machine.
3. Right-click the project again and select **Synchronize** → **Sync Active now** to perform the first synchronization between local and remote folders. (A synchronization can be performed at any time.) Remember that Nsight is configured to perform a synchronization after or before some tasks (for example, before compiling the project).

4. As an optional step, set the host compiler that is evoked by **nvcc**. Right-click the project name and select **Properties**. Expand **Build**, and then choose **Settings**. Click the **Tool Settings** tab and select **Build Stages**. Enter the necessary information in the Compiler Path and Preprocessor options fields (see Figure 4-36). Change the content of the **Compiler Path** under the Miscellaneous section of the NVCC Linker window.

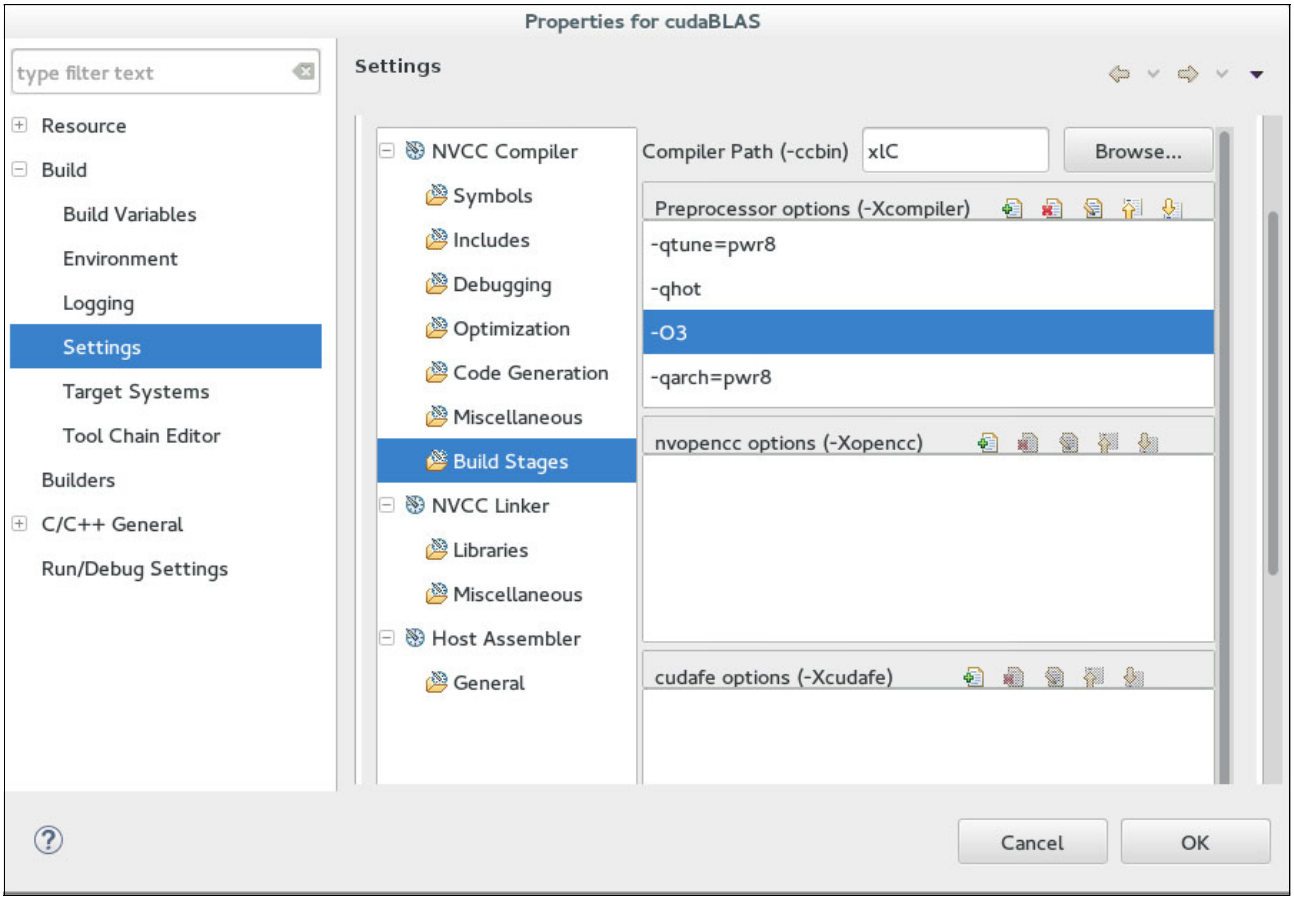


Figure 4-36 Nsight - new CUDA C/C++ project build settings

After completing these steps, the new project is ready for CUDA C/C++ programming. You can use many of the features that are provided by the Nsight C and C++ code editor, such as syntax highlighting, code completion, static analysis, and error markers.

Compiling the application

To build the application, complete the following steps:

1. Right-click the project in the Project Explorer pane and select **Run As** → **Remote C/C++ Application**.
2. Check that all the information is correct and change any information, if needed. Then, click **Run**.

Debugging the application

The steps to start the Nsight debugger are similar to the steps that are used to run the application. However, the executable file is started by using the **cuda-gdb** command, which in turn is backed by the GNU GDB debugger tool.

For more information about the **cuda-gdb** command see 4.6.5, “Command-line tools for CUDA C/C++” on page 182.

To debug the application, complete the following steps:

1. Right-click project name in the Project Explorer pane and select **Debug As** → **Remote C/C++ Application**.
2. A window opens in which you are prompted for permission to open the Eclipse Debug perspective. Click **Yes**.

By default, the debugger stops at the first instruction on the main method. Figure 4-37 shows the Nsight debugger.

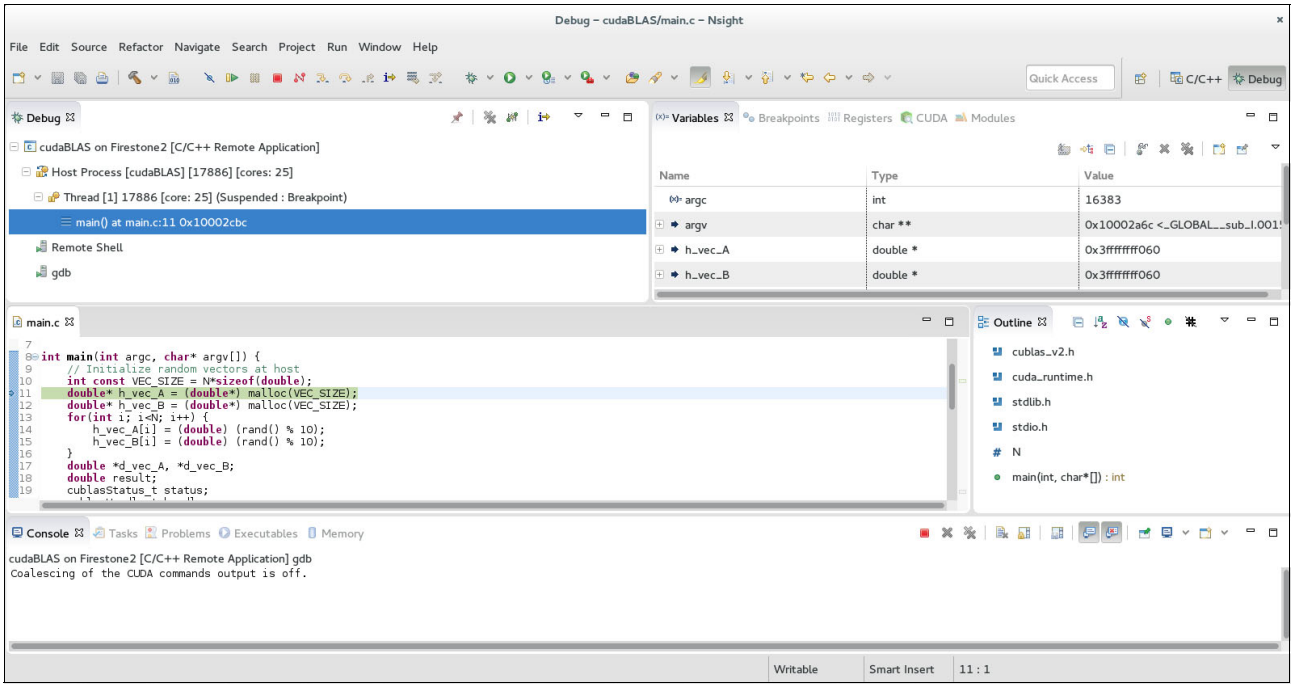


Figure 4-37 Nsight: CUDA C/C++ debugger

4.6.5 Command-line tools for CUDA C/C++

The NVIDIA Toolkit 8.0 provides the following tools for debugging problems on CUDA C and C++ applications.

CUDA-MEMCHECK

The CUDA-MAMCHECK tool detects memory-related flaws on programs. It dynamically instruments the program executable file on run time and can check allocation and deallocation and accesses on global and shared memories.

In addition to memory checking, CUDA-MEMCHECK also can inspect the application to catch the following types of errors:

- Race condition: Check for race condition on access of shared and local variables. Uses the **--tool racecheck** option.
- initcheck: Check for use of non-initialized memory. Uses the **--tool initcheck** option.
- synccheck: Check for synchronization errors. Uses the **--tool synccheck** option.

Notice that for complete visualization of the source code file and the line number where detected problems occur, the binary must be compiled by using the **nvcc -g -lineinfo** options.

For more information about the CUDA-MEMCHECK tool, see the CUDA-MEMCHECK user's manual.

CUDA GDB

CUDA application portions of the code runs on CPU (host) and GPUs (devices), and programs can often have thousands of CUDA threads that are spread across many GPUs. Although traditional debuggers are effective to work with CPU code, they are not properly built to deal with both.

The cuda-gdb tool is an implemented debugger extension of GNU GDB that is designed to deal with the scenario that CUDA hybrid model imposes.

nvprof tool

The nvprof tool is an extensive command-line profiling tool that is distributed with CUDA Toolkit. Designed to be flexible, profiles all processes in the entire system, only a specific application, or only some elements of it (for example, kernels, streams, and contexts). Also, it allows you to select which GPU device must be profiled (default is all on the system).

By default, **nvprof** time profiles the calls to kernel functions and CUDA API, as shown in Example 4-20.

Example 4-20 Time profiling kernels and CUDA API

```
$ nvprof --log-file profile.out ./MonteCarloMultiGPU 2&> /dev/null
$ cat profile.out
==73495== NVPROF is profiling process 73495, command: ./MonteCarloMultiGPU 2
==73495== Profiling application: ./MonteCarloMultiGPU 2
==73495== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 79.29%  97.066ms         4  24.267ms  24.169ms  24.396ms
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
 20.67%  25.306ms         4  6.3264ms  6.2893ms  6.3711ms
rngSetupStates(curandStateXORWOW*, int)
  0.03%  37.344us         4  9.3360us  8.9600us  10.048us  [CUDA memcpy HtoD]
  0.01%  12.928us         4  3.2320us  2.8480us  3.5840us  [CUDA memcpy DtoH]

==73495== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 85.47%  547.62ms         4  136.90ms  131.25ms  143.30ms  cudaStreamCreate
  3.77%  24.160ms         4  6.0401ms  195.00us  22.889ms  cudaEventSynchronize
  2.03%  12.999ms        12  1.0832ms  1.0394ms  1.1578ms  cudaMalloc
  1.84%  11.818ms        16  738.65us  714.22us  818.61us  cudaGetDeviceProperties
  1.19%  7.6318ms         4  1.9079ms  1.8529ms  1.9930ms  cudaMallocHost
  1.18%  7.5904ms         4  1.8976ms  1.8906ms  1.9076ms  cudaHostAlloc
  1.04%  6.6817ms         8  835.22us  17.319us  1.6565ms  cudaLaunch
  0.98%  6.2936ms         4  1.5734ms  14.881us  6.2399ms  cudaDeviceSynchronize
  0.86%  5.5070ms         8  688.37us  673.41us  748.69us  cudaFreeHost
  0.66%  4.2604ms         4  1.0651ms  1.0615ms  1.0672ms  cuDeviceTotalMem
  0.48%  3.0439ms       364  8.3620us     394ns  303.12us  cuDeviceGetAttribute
  0.38%  2.4584ms        12  204.87us  194.19us  230.03us  cudaFree
  0.04%  251.96us         4  62.989us  61.403us  64.379us  cuDeviceGetName
```

```
  0.02%  132.76us         8  16.594us  11.935us  31.873us  cudaMemcpyAsync
  0.01%  77.114us        28  2.7540us  1.6600us  15.832us  cudaSetDevice
  0.01%  53.252us         4  13.313us  12.214us  16.493us  cudaStreamDestroy
  0.00%  23.942us         4  5.9850us  5.1930us  7.5630us  cudaEventRecord
  0.00%  21.598us        28     771ns     662ns  1.2580us  cudaSetupArgument
  0.00%  19.164us         4  4.7910us  4.4390us  5.1540us  cudaEventCreate
  0.00%  17.214us         4  4.3030us  2.7130us  5.3020us  cudaEventDestroy
  0.00%  8.7520us         8  1.0940us     744ns  1.6830us  cudaConfigureCall
  0.00%  7.1430us        12     595ns     422ns     795ns  cuDeviceGet
  0.00%  7.1380us         8     892ns     707ns  1.1000us  cudaGetLastError
  0.00%  6.2230us         3  2.0740us     719ns  4.6540us  cuDeviceGetCount
  0.00%  4.3670us         1  4.3670us  4.3670us  4.3670us  cudaGetDeviceCount
```

Time profiling is a good start point towards improving an application. As shown in Example 4-20, it unveils that `MonteCarloOneBlockPerOption` kernel ran in a total of 97.066 ms (79.29%), while `cudaStreamCreate API` calls used 547.62 ms (85.47%).

Another way to view the application behavior that is shown in Example 4-20 is to determine the execution critical path combined with the information that is gathered from CPU (CUDA API calls) and GPU (kernel evocations) executions. The **`nvprof`** dependency analysis mode[8] produces a report for the critical path when the option `--dependency-analysis` is used (see Example 4-21).

Example 4-21 Reporting the execution critical path

```
$ nvprof --log-file profile.out --dependency-analysis ./MonteCarloMultiGPU 2&>
/dev/null
$ cat profile.out
==73532== NVPROF is profiling process 73532, command: ./MonteCarloMultiGPU 2
==73532== Profiling application: ./MonteCarloMultiGPU 2
==73532== Profiling result:
Time(%)      Time     Calls       Avg       Min       Max  Name
 79.32%  97.266ms         4  24.316ms  24.010ms  24.473ms
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
 20.64%  25.310ms         4  6.3274ms  6.2963ms  6.3661ms
rngSetupStates(curandStateXORWOW*, int)
  0.03%  36.544us         4  9.1360us  8.7680us  9.6000us  [CUDA memcpy HtoD]
  0.01%  12.992us         4  3.2480us  2.8160us  3.6800us  [CUDA memcpy DtoH]

==73532== API calls:
Time(%)      Time     Calls       Avg       Min       Max  Name
 86.40%  590.17ms         4  147.54ms  138.74ms  162.78ms  cudaStreamCreate
  3.51%  23.995ms         4  5.9988ms  38.611us  23.186ms  cudaEventSynchronize
  1.92%  13.088ms        12  1.0907ms  1.0670ms  1.1258ms  cudaMalloc
  1.70%  11.595ms        16  724.69us  713.95us  737.43us  cudaGetDeviceProperties
  1.13%  7.6917ms         4  1.9229ms  1.9016ms  1.9583ms  cudaMallocHost
  1.12%  7.6606ms         4  1.9152ms  1.8882ms  1.9460ms  cudaHostAlloc
  0.97%  6.6572ms         8  832.15us  17.940us  1.6505ms  cudaLaunch
  0.92%  6.2873ms         4  1.5718ms  13.461us  6.2349ms  cudaDeviceSynchronize
  0.81%  5.5394ms         8  692.43us  675.34us  759.93us  cudaFreeHost
  0.62%  4.2598ms         4  1.0650ms  1.0618ms  1.0691ms  cuDeviceTotalMem
  0.45%  3.0713ms       364  8.4370us     390ns  308.84us  cuDeviceGetAttribute
  0.36%  2.4474ms        12  203.95us  193.04us  230.29us  cudaFree
```

[8] New in nvprof of NVIDIA CUDA Toolkit 8.0.

```
  0.04%  254.18us         4  63.544us  61.981us  65.150us  cuDeviceGetName
  0.02%  135.26us         8  16.907us  11.896us  31.121us  cudaMemcpyAsync
  0.01%  73.833us        28  2.6360us  1.5100us  15.121us  cudaSetDevice
  0.01%  52.310us         4  13.077us  11.761us  16.365us  cudaStreamDestroy
  0.00%  24.634us         4  6.1580us  5.3050us  7.7970us  cudaEventRecord
  0.00%  21.074us        28     752ns     666ns  1.1720us  cudaSetupArgument
  0.00%  18.341us         4  4.5850us  4.2260us  5.2930us  cudaEventCreate
  0.00%  15.729us         4  3.9320us  2.6210us  5.1060us  cudaEventDestroy
  0.00%  8.5760us         8  1.0720us     801ns  1.9340us  cudaConfigureCall
  0.00%  7.3530us        12     612ns     441ns     844ns  cuDeviceGet
  0.00%  6.8210us         8     852ns     689ns  1.0040us  cudaGetLastError
  0.00%  6.4790us         3  2.1590us     681ns  4.9340us  cuDeviceGetCount
  0.00%  4.4320us         1  4.4320us  4.4320us  4.4320us  cudaGetDeviceCount

==73532== Dependency Analysis:
Critical path(%)  Critical path  Waiting time  Name
          83.03%   590.169908ms           0ns  cudaStreamCreate
           4.14%    29.393820ms           0ns  <Other>
           3.38%    24.010192ms           0ns
MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
           1.84%    13.088064ms           0ns  cudaMalloc
           1.63%    11.595084ms           0ns  cudaGetDeviceProperties
           1.08%     7.691710ms           0ns  cudaMallocHost
           1.08%     7.660636ms           0ns  cudaHostAlloc
           0.89%     6.296326ms           0ns  rngSetupStates(curandStateXORWOW*,
int)
           0.78%     5.539428ms           0ns  cudaFreeHost
           0.70%     4.998013ms           0ns  cudaLaunch
           0.60%     4.259809ms           0ns  cuDeviceTotalMem_v2
           0.43%     3.071325ms           0ns  cuDeviceGetAttribute
           0.34%     2.447406ms           0ns  cudaFree
           0.04%   254.177000us           0ns  cuDeviceGetName
           0.02%   108.747000us           0ns  cudaMemcpyAsync
           0.01%    58.902000us           0ns  cudaSetDevice
           0.01%    52.310000us           0ns  cudaStreamDestroy_v5050
           0.00%    19.329000us           0ns  cudaEventRecord
           0.00%    18.341000us           0ns  cudaEventCreate
           0.00%    17.468000us           0ns  cudaSetupArgument
           0.00%    15.729000us           0ns  cudaEventDestroy
           0.00%     9.600000us           0ns  [CUDA memcpy HtoD]
           0.00%     7.747000us           0ns  cudaConfigureCall
           0.00%     7.353000us           0ns  cuDeviceGet
           0.00%     6.479000us           0ns  cuDeviceGetCount
           0.00%     5.051000us           0ns  cudaGetLastError
           0.00%     4.432000us           0ns  cudaGetDeviceCount
           0.00%     3.680000us           0ns  [CUDA memcpy DtoH]
           0.00%            0ns   23.940847ms  cudaEventSynchronize
           0.00%            0ns    6.228596ms  cudaDeviceSynchronize
```

Example 4-22 shows application information by using the nvprof flags of `--cpu-profiling`, which enable capture operations that are run in CPU.

Example 4-22 Collecting information about operations executed in CPU

```
$ nvprof --log-file profile.out --cpu-profiling on --cpu-profiling-thread-mode
separated ./MonteCarloMultiGPU 2&> /dev/null
$ cat profile.out

======== CPU profiling result (bottom up):
Time(%)      Time  Name
======== Thread 17592186332928
 65.66%     650ms  cuDevicePrimaryCtxRetain
 65.66%     650ms  |
cudart::contextStateManager::initPrimaryContext(cudart::device*)
 65.66%     650ms  |   cudart::contextStateManager::initDriverContext(void)
 65.66%     650ms  |
cudart::contextStateManager::getRuntimeContextState(cudart::contextState**, bool)
 65.66%     650ms  |       cudart::doLazyInitContextState(void)
 65.66%     650ms  |         cudart::cudaApiStreamCreate(CUstream_st**)
 65.66%     650ms  |           cudaStreamCreate
 65.66%     650ms  |             _ZL11multiSolverP11TOptionPlani
 17.17%     170ms  cuInit
 17.17%     170ms  | cudart::__loadDriverInternalUtil(void)
 17.17%     170ms  |   __pthread_once
 17.17%     170ms  |     cudart::cuosOnce(int*, void (*) (void))
 17.17%     170ms  |       cudart::globalState::initializeDriver(void)
 17.17%     170ms  |         cudaGetDeviceCount
 17.17%     170ms  |           main
  3.03%      30ms  ???
  3.03%      30ms  | getcontext@@GLIBC_2.17
  2.02%      20ms  cuMemFreeHost
  2.02%      20ms  | cudart::driverHelper::freeHost(void*)
  2.02%      20ms  |   cudart::cudaApiFreeHost(void*)
  2.02%      20ms  |     cudaFreeHost
  2.02%      20ms  |       closeMonteCarloGPU
  2.02%      20ms  cuLaunchKernel
  2.02%      20ms  | _ZN6cudartL19cudaApiLaunchCommonEPKvb
  2.02%      20ms  |   cudaLaunch
  2.02%      20ms  |     cudaError cudaLaunch<char>(char*)
  2.02%      20ms  cuDeviceGetAttribute
  2.02%      20ms  | cudart::device::updateDeviceProperties(void)
  2.02%      20ms  |   cudart::cudaApiGetDeviceProperties(cudaDeviceProp*, int)
  2.02%      20ms  |     cudaGetDeviceProperties
  1.01%      10ms  |       _ZL11multiSolverP11TOptionPlani
  1.01%      10ms  |       adjustGridSize(int, int)
  1.01%      10ms  random
  1.01%      10ms  | rand
  1.01%      10ms  |   randFloat(float, float)
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuMemHostAlloc
  1.01%      10ms  | cudart::driverHelper::mallocHost(unsigned long, void**,
unsigned int)
  1.01%      10ms  |   cudart::cudaApiHostAlloc(void**, unsigned long, unsigned
int)
```

```
  1.01%      10ms  |     cudaHostAlloc
  1.01%      10ms  |       _ZL14cudaMallocHostPPvmj
  1.01%      10ms  __exp_finite
  1.01%      10ms  | exp
  1.01%      10ms  |   BlackScholesCall
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuDeviceTotalMem_v2
  1.01%      10ms  | cudart::deviceMgr::enumerateDevices(void)
  1.01%      10ms  |   cudart::globalState::initializeDriverInternal(void)
  1.01%      10ms  |     cudart::globalState::initializeDriver(void)
  1.01%      10ms  |       cudaGetDeviceCount
  1.01%      10ms  |         main
  1.01%      10ms  ???
  1.01%      10ms  | cudart::contextState::loadCubin(bool*, void**)
  1.01%      10ms  | |
cudart::globalModule::loadIntoContext(cudart::contextState*)
  1.01%      10ms  | |   cudart::contextState::applyChanges(void)
  1.01%      10ms  | |
cudart::contextStateManager::getRuntimeContextState(cudart::contextState**, bool)
  1.01%      10ms  | |       cudart::doLazyInitContextState(void)
  1.01%      10ms  | |         cudart::cudaApiStreamCreate(CUstream_st**)
  1.01%      10ms  | |           cudaStreamCreate
  1.01%      10ms  | |             _ZL11multiSolverP11TOptionPlani
  1.01%      10ms  cuMemAlloc_v2
  1.01%      10ms  | cudart::driverHelper::mallocPtr(unsigned long, void**)
  1.01%      10ms  |   cudart::cudaApiMalloc(void**, unsigned long)
  1.01%      10ms  |     cudaMalloc
  1.01%      10ms  |       initMonteCarloGPU
  1.01%      10ms  __log_finite
  1.01%      10ms  | log
  1.01%      10ms  |   BlackScholesCall
  1.01%      10ms  |     main
  1.01%      10ms  |       generic_start_main.isra.0
  1.01%      10ms  cuEventSynchronize
  1.01%      10ms    cudart::cudaApiEventSynchronize(CUevent_st*)
  1.01%      10ms      cudaEventSynchronize
  1.01%      10ms        _ZL11multiSolverP11TOptionPlani
======== Thread 21990246248880
 86.87%     860ms  ???
 86.87%     860ms  | start_thread
 86.87%     860ms  | | clone
======== Thread 23089793855920
 68.69%     680ms  ???
 68.69%     680ms  | start_thread
 68.69%     680ms  | | clone
======== Thread 24189306532272
 52.53%     520ms  ???
 52.53%     520ms  | start_thread
 52.53%     520ms  | | clone
======== Thread 25288817111472
 36.36%     360ms  ???
 36.36%     360ms  | start_thread
 36.36%     360ms  | | clone
======== Thread 17592299352496
```

```
 90.91%     900ms  ???
 90.91%     900ms  | start_thread
 90.91%     900ms  | | clone

======== Data collected at 100Hz frequency
```

Some tools, such as oprofile and perf, can obtain data out of the PMU in CPU. Also, nvprof can profile hardware events that are triggered by the GPU and calculate metrics. Use the `--events` or `--metrics` flags to pass a list of events or metrics to the profiler.

Although some available metrics calculate aspects of the GPU efficiency, others can unveil bottlenecks in specific units (for example, the case of metrics from events involving NVLink communication. Example 4-23 shows the receive (`nvlink_receive_throughput`) and transmit (`nvlink_transmit_throughput`) throughput metrics for each GPU in the system.

Example 4-23 NVIDIA nvprof: report metrics for NVLink

```
$ nvprof --log-file profile.out --metrics
nvlink_transmit_throughput,nvlink_receive_throughput  ./MonteCarloMultiGPU 2&>
/dev/null
$ cat profile.out
==73977== NVPROF is profiling process 73977, command: ./MonteCarloMultiGPU 2
==73977== Profiling application: ./MonteCarloMultiGPU 2
==73977== Profiling result:
==73977== Metric result:
Invocations                                Metric Name
Metric Description         Min         Max         Avg
Device "Tesla P100-SXM2-16GB (0)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  1.1570MB/s  1.1570MB/s  976.56KB/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  681.24KB/s  681.24KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  567.64KB/s  567.64KB/s  0.00000B/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  285.74KB/s  285.74KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (1)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  177.71KB/s  177.71KB/s  0.00000B/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  244.35KB/s  244.35KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  308.39KB/s  308.39KB/s  0.00000B/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  190.42KB/s  190.42KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (2)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  177.16KB/s  177.16KB/s  0.00000B/s
```

```
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  231.30KB/s  231.30KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  568.43KB/s  568.43KB/s  0.00000B/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  284.86KB/s  284.86KB/s  0.00000B/s
Device "Tesla P100-SXM2-16GB (3)"
    Kernel: rngSetupStates(curandStateXORWOW*, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  1.1607MB/s  1.1607MB/s  976.56KB/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  703.26KB/s  703.26KB/s  0.00000B/s
    Kernel: MonteCarloOneBlockPerOption(curandStateXORWOW*, __TOptionData const *,
__TOptionValue*, int, int)
          1                nvlink_transmit_throughput                NVLink
Transmit Throughput  309.12KB/s  309.12KB/s  0.00000B/s
          1                 nvlink_receive_throughput                 NVLink
Receive Throughput  186.25KB/s  186.25KB/s  0.00000B/s
```

Use the `--query-metrics` or `--query-events` flags to list all of the metrics and events that are available for profiling.

Another useful function of **`nvprof`** is to generate traces of the application execution. Those traces can be then imported in the NVIDIA Visual Profiler tool. For more information, see the NVIDIA Visual Profiler website.

For more information about the nvprof tool, see the Profiler User's Guide.

Part 2

Administrator’s guide

This part provides administrators with information about how to deploy the hardware and software in the cluster. This part also includes a cluster monitoring and health check of the cluster to help administrators manage the day-to-day operation and health of the environment.

The following chapters are included in this part:

5

Node and software deployment

This chapter describes the software deployment of an Extreme Cluster/Cloud Administration Toolkit (xCAT) cluster with the IBM High Performance Computing (HPC) software. The cluster runs on Red Hat Enterprise Linux (RHEL) Server 7.3 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on an IBM Power System S822LC server. An IBM Power System S812LC server is the management server.

This chapter includes the following topics:

5.1 Software stack

The following software stack components and versions are referenced in this chapter:

- Extreme Cluster/Cloud Administration Toolkit (xCAT) 2.12.4
- Red Hat Enterprise Linux (RHEL) Server 7.3
- Compute Unified Device Architecture (CUDA) Toolkit 8 (8.0.54)
- Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux 3.4 (3.4-2.0.0.1)
- XL C/C++ Compiler for Linux V13.1.5
- XL Fortran Compiler for Linux V15.1.5
- Advance Toolchain 10.0
- PGI Compiler 16.10
- Spectrum MPI (SMPI) 10.1.0.2
- Parallel Performance Toolkit (PPT) 2.3
- Engineering and Scientific Subroutine Library (ESSL) 5.5
- Parallel ESSL (PESSL) 5.3
- Spectrum Scale (formerly GPFS) 4.2.x.3
- Spectrum LSF (formerly Platform LSF) 10.1.0.1
- Scale-Out LC System Firmware (OP820.01/PNOR OP8_v1.11_2.19/BMC 2.13.00058)
- Intelligent Platform Management Interface (IPMI) tool (IPMItool) 1.8.15

For more information about the currently supported and recommended software versions, see the Wikis page of the IBM developerWorks website.

5.2 System management

This chapter describes system management functions (or operations) for the IBM Power System S822LC server that is based on the IPMI protocol, and uses the IPMItool utility. Some functions require IPMItool version 1.8.15 or later, such as system firmware upgrade.

This section includes the following topics:

- Frequently used commands for IPMItool
- Configure the boot order in the Petitboot bootloader
- Upgrade the system firmware of the IBM Power System S822LC server

5.2.1 Frequently used commands with the IPMItool

The **ipmitool** command features the following most noteworthy options:

- `-H <address>`: The host name of the baseboard management controller (BMC).
- `-U <username>`: The remote server user name (default is ADMIN in POWER8).
- `-P <password>`: The remote server password (default is admin in POWER8).
- `-I <interface>`: The interface type of the connection (defaults to lan, lanplus is needed for some commands. In any case, lanplus is the preferred interface).

The following **ipmitool** commands are most useful for monitoring BMCs:

- `chassis` retrieves chassis status and sets the power states.
- `sol` facilitates Serial-over-Lan connections.
- `mc` interacts with the management controller.
- `lan` configures the network interface.

By using the **chassis** command, you can power the node on and off, and get the status of the node power. Example 5-1 shows the node in the off state and powered on. Use the `cycle` keyword to perform a power cycle (off, then on).

Example 5-1 The chassis command

```
$ ipmitool -H <IP> -U ADMIN -P admin chassis status
System Power         : off
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
Power Restore Policy : always-off
Last Power Event     : command
Chassis Intrusion    : inactive
Front-Panel Lockout  : inactive
Drive Fault          : false
Cooling/Fan Fault    : false
Front Panel Control  : none

$ ipmitool -H <IP> -U ADMIN -P admin chassis power on
Chassis Power Control: Up/On

$ ipmitool -H <IP> -U ADMIN -P admin chassis status
System Power         : on
...
```

The **sol** command is used to establish a serial connection to the specified node. This command requires the **lanplus** interface to function, and only one serial connection can be started at any time. This tool is typically used when the OS of the node is no longer reachable through ssh. Example 5-2 shows how to establish a sol connection to a node with a connection.

Example 5-2 The sol command

```
$ ipmitool -H <IP> -U ADMIN -P admin -I lanplus sol activate
Info: SOL payload already active on another session
$ ipmitool -H <IP> -U ADMIN -P admin -I lanplus sol deactivate
$ ipmitool -H <IP> -U ADMIN -P admin -I lanplus sol activate
[SOL Session operational.  Use ~? for help]

p8r1n1 login:
```

You can also close an active console session with the following keystrokes:

- On a non-Secure Shell (SSH) connection:

  `Enter, ~ (tilde[1]), . (period)`

  Note: This command can close an SSH connection, which can leave the console session open.

- On an SSH connection:

  `Enter, ~ (tilde), ~ (tilde), . (period)`

[1] On keyboards with dead keys (for example, on some non-English languages), the tilde mark requires two keystrokes: Tilde and space.

Note: This command leaves the SSH connection open and closes the console session.

The **mc** command is used to perform management console functions. Typically, this command is used only when the BMC is no longer responsive to commands (such as **chassis**).

Example 5-3 shows the most common use of this command to reset the BMC.

Example 5-3 The mc reset command

```
$ ipmitool -H <IP> -U ADMIN -P admin  mc reset cold
Sent cold reset command to MC
```

The **lan** command is used to configure the BMC network interface. The following important arguments can be included:

- Display the BMC Ethernet Port network configuration:

  ```
  $ ipmitool <arguments> lan print 1
  ```

- Set the BMC Ethernet Port for Dynamic Host Configuration Protocol (DHCP) IP address:

  ```
  $ ipmitool <arguments> lan set 1 ipsrc dhcp
  ```

- Set the BMC Ethernet Port for Static IP address:

  ```
  $ ipmitool <arguments> lan set 1 ipsrc static
  $ ipmitool <arguments> lan set 1 ipaddr a.b.c.d
  $ ipmitool <arguments> lan set 1 netmask e.f.g.h
  $ ipmitool <arguments> lan set 1 defgw i.j.k.l
  ```

For more information about other **ipmitool** commands, see 6.3, “Using the BMC for node monitoring” on page 300.

5.2.2 Boot order configuration

The Petitboot bootloader can automatically boot (or *autoboot*) from several types of devices in a certain order (that is, falling back to later devices if earlier devices cannot be booted from).

This scenario requires a specific configuration of the device boot order (specifically, for Genesis-based node discovery, and diskful installation, which are described in 5.3, “xCAT overview” on page 201). It is important for Petitboot to first attempt to boot from the network devices by way of DHCP. If the network devices are not available, it attempts to boot from the disk devices. In such order, the node can obtain its network and boot configuration from the xCAT management node (for example, the Genesis image or diskless installation), or fall back to boot an OS from disk if network boot is not specified (for example, diskful installation).

To configure the boot order in the Petitboot bootloader, complete the following steps:

1. Power on the system, as shown in the following example:

   ```
   $ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
       chassis power on
   ```

2. Open the console, as shown in the following example:

   ```
   $ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
       sol activate
   ```

3. Wait for the Petitboot panel (see Example 5-4).

Example 5-4 Petitboot panel

```
Petitboot (v1.2.6-8fa93f2)                    8335-GTB 0000000000000000

System information
System configuration
Language
Rescan devices
Retrieve config from URL
*Exit to shell

Enter=accept, e=edit, n=new, x=exit, l=language, h=help
Welcome to Petitboot
```

4. Configure boot order for Petitboot. In the Petitboot panel, select **System configuration**.
5. In the Petitboot System Configuration panel (see Example 5-5), complete the following steps:
   a. In the Boot order section, complete the following steps:
      i. Select **Clear**.
      i. Select **Add Device**.
      ii. In the Select a boot device to add panel (see Example 5-6 on page 198), select **Any Network device**, and click **OK**.
      iii. Select **Add Device** again.
      iv. In the Select a boot device to add panel again, select **Any Disk device**, and click **OK**.
      v. Verify that the Boot order section is identical to Example 5-7 on page 198.
   b. In the Network section, select the **DHCP on a specific interface** option.

> **Note:** You can select the **DHCP on all active interfaces** option, but that option might slow the boot process unnecessarily if multiple network ports are cabled and active.

   c. In the Device section, select the network interface for accessing the xCAT Management Node (if you selected **DHCP on a specific interface** in the Network section).
   d. Click **OK**.

Example 5-5 Petitboot System Configuration panel, DHCP on a specific interface setting

```
Petitboot System Configuration

Boot Order:    (None)

               [     Add Device     ]
               [  Clear & Boot Any  ]
               [       Clear        ]

Network:       ( ) DHCP on all active interfaces
               (*) DHCP on a specific interface
               ( ) Static IP configuration
```

```
Device:       ( ) enP5p7s0f0 [70:e2:84:14:09:ae, link up]
              (*) enP5p7s0f1 [70:e2:84:14:09:af, link up]

DNS Server(s): _____________________________ (eg. 192.168.0.2)
              (if not provided by DHCP server)

Disk R/W:     ( ) Prevent all writes to disk
              (*) Allow bootloader scripts to modify disks

              [    OK    ] [   Help   ] [  Cancel  ]

tab=next, shift+tab=previous, x=exit, h=help
```

Example 5-6 shows the panel to select a boot device.

Example 5-6 Select a boot device to add panel (any Network Device setting)

```
Select a boot device to add

        ( ) net:  enP1p3s0f0 [mac: 98:be:94:59:fa:24]
        ( ) net:  enP1p3s0f1 [mac: 98:be:94:59:fa:25]
        (*) Any Network device
        ( ) Any Disk device
        ( ) Any USB device
        ( ) Any CD/DVD device
        ( ) Any Device

        [    OK    ] [  Cancel  ]

 tab=next, shift+tab=previous, x=exit, h=help
```

Example 5-7 shows the petitboot system configuration option panel.

Example 5-7 Petitboot System Configuration panel, Boot Order configuration

```
Petitboot System Configuration

Boot Order:   (0) Any Network device
              (1) Any Disk device

              [     Add Device     ]
              [  Clear & Boot Any  ]
              [       Clear        ]

<...>
```

On the next boot, the Petitboot bootloader can automatically boot from the network and disk devices in the specified order. On this boot, no automatic boot attempt is made because of user intervention.

5.2.3 System firmware upgrade

The IPMItool version 1.8.15 or later can be used to upgrade the system firmware of the IBM Power System S822LC. Run the **ipmitool** command from the same or a close local area network (LAN) to the target system or BMC to avoid network instability problems.

To upgrade the system firmware, complete the following steps:

1. Download the system firmware image:
   a. Go to the IBM Support website for the 8335-GTB server.
   b. In the results list, click the wanted version (for example, **OP8_v1.11_2.19**). Usually, the latest version is suggested for the general case.
   c. Proceed with the sign-in process.
   d. Select a download option (for example, HTTPS).
   e. Click the **8335_<*version*>_update.hpm** link to download the file (for HTTPS; for other methods, follow the instructions that are provided in the website).
   f. (Optional) Click the **Description** link for more information about the firmware version.
2. Install the system firmware image:
   a. Power off the system by using the following command:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power off
Chassis Power Control: Down/Off
```

   Wait until the system is powered off. You can verify it by using the following command:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power status
Chassis Power is off
```

   b. Reset the BMC by using the following command:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    mc reset cold
Sent cold reset command to MC
```

   Wait until the BMC is back online. You can verify it by using the following command (repeat as necessary):

```
$ ping -c1 bmc-address
<...>
1 packets transmitted, 1 received, 0% packet loss, time 0ms
<...>
```

   c. Protect the BMC memory content (for example, network configuration) during upgrade by using the following command:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    raw 0x32 0xba 0x18 0x00
```

   d. Upgrade the system firmware image (see Example 5-8 on page 200). Consider the following points about the process:
      - You might be asked to confirm the operation; if so, press **y** and **Enter**.
      - The output can vary depending on the old and new firmware versions that are used.
      - If segmentation fault errors occur, change the **-z** argument to 25000.
      - If you lose the network configuration, establish a serial connection to the internal serial port and configure it by using the **ipmitool** command on the 127.0.0.1 IP address.

      For more information and about errors, see the Description link in the system firmware image download page.

Example 5-8 One step of the system firmware upgrade with the ipmitool command

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    -z 30000 hpm upgrade 8335_<version>_update.hpm force
Setting large buffer to 30000

PICMG HPM.1 Upgrade Agent 1.0.9:

Validating firmware image integrity...OK
Performing preparation stage...
Services  be affected during upgrade. Do you wish to continue? (y/n): y
OK

Performing upgrade stage:

--------------------------------------------------------------------------------
|ID  | Name     |                     Versions                          | %  |
|    |          |      Active     |      Backup     |      File       |    |
|----|----------|-----------------|-----------------|-----------------|----|
|*  2|BIOS      |   0.00 00000000 | ---.-- -------- |   1.00 3E010701 |100%|
|    |Upload Time: 00:27          | Image Size: 33554584 bytes                |
|*  0|BOOT      |   2.13 7B4E0100 | ---.-- -------- |   2.13 AB660100 |100%|
|    |Upload Time: 00:00          | Image Size:  262296 bytes                 |
|*  1|APP       |   2.13 7B4E0100 | ---.-- -------- |   2.13 AB660100 |100%|
|    |Upload Time: 00:16          | Image Size: 33292440 bytes                |
--------------------------------------------------------------------------------
(*) Component requires Payload Cold Reset

Firmware upgrade procedure successful
```

e. Power on the system by using the following command:

Note: The system firmware upgrade completes only *after* the system is powered on.

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    chassis power on
Chassis Power Control: Up/On
```

f. Open the console, by using the following command:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
sol activate
```

g. Wait for the Petitboot panel (see Example 5-4 on page 197).

Note: Only approximately 20 ISTEP lines are required (earlier than Petitboot).

If the IPMI console is not responsive to any keys, try to reset the BMC again.

3. Verify that the system firmware version matches the wanted or downloaded version, as shown in the following example:

```
$ ipmitool -I lanplus -H bmc-address -U ipmi-username -P ipmi-password \
    fru print 47
Product Name          : OpenPOWER Firmware
 Product Version       : IBM-garrison-ibm-OP8_v1.11_2.19
 Product Extra         :     op-build-6ce5903
```

```
Product Extra         :     buildroot-81b8d98
Product Extra         :     skiboot-5.3.7
Product Extra         :     hostboot-1f6784d-02b09df
Product Extra         :     linux-4.4.24-openpower1-5d537af
Product Extra         :     petitboot-v1.2.6-8fa93f2
Product Extra         :     garrison-xml-3db7b6e
Product Extra         :     occ-69fb587
Product Extra         :     hostboot-bina
```

5.3 xCAT overview

This section provides an overview of the architecture and concepts that are involved in a cluster that is administered with xCAT (that is, an *xCAT cluster*), and the scenario that is described in this chapter.

xCAT manages the nodes in a cluster by using continuous configuration and event handling. The xCAT database contains the required information to perform the configuration steps. Several services and commands (for example, DHCP server, and xCAT administration commands) trigger and respond to events, such as node booting, node discovery, and installing an operating system (OS) and other software.

For more information, see the following xCAT resources:

- xCAT website
- Extreme Cloud/Cluster Administration Toolkit documentation

5.3.1 xCAT cluster: Nodes and networks

An xCAT cluster is several nodes that are interconnected by one or more networks.

The type of node depends on its function in the cluster (for example, management, compute, login, or service node). The type of network depends on its traffic and interconnected nodes (for example, operating system-level management and provisioning, service processor-level hardware management, application-level intercommunication, and public/Internet access).

The following types of nodes are available in an xCAT cluster:

- Management node

  This node performs administrative operations on compute nodes; for example, power control, software deployment (OS provisioning, application installation, and updates), configuration, command execution, and monitoring.

- Compute nodes

  This node perform operations that are specified by the management node; for example, OS provisioning and command execution, and runs the runtime and application software. Compute nodes are sometimes referred to as *nodes*.

- Login nodes

  This node is an interface to the users of the clusters, which enables tasks, such as job submission and source-code compilation.

- Service nodes

  This node performs operations that are delegated by the management node on groups of nodes, and respond to requests from a set of other nodes, acting as intermediary management nodes on large clusters. An xCAT cluster with service nodes is known as a *hierarchical cluster*.

The following types of networks are available in an xCAT cluster:

- Management network:
  - Used for in-band operations (that is, through the system's network interface); for example, node discovery, OS provisioning, and management.
  - Interconnects the management node, service nodes (if any), login nodes (if any), and compute nodes, all by way of the in-band network interface controller (NIC).
  - Possibly segregated into separate virtual LANs (VLANs) for each service node.
  - Usually an Ethernet network of high transfer speed, depending on the number of compute nodes and frequency of OS provisioning, software download, and installation operations.
- Service network:
  - Used for out-of-band operations (that is, through the BMC's network interface); for example, power control, console sessions, and platform-specific functions.
  - Interconnects the management node and service nodes (by way of the in-band NIC), and the other nodes (by way of the out-of-band NIC).
  - Possibly combined with the management network (same physical network).
  - Usually an Ethernet network, but does not need to use as high a transfer speed as the management node because the network traffic of out-of-band operations is often of a smaller size and lower frequency.
- Application network:
  - Used by applications that are running on compute nodes.
  - Interconnects the compute nodes.
  - Usually an InfiniBand network for HPC applications.
- Optional: Site (public) network:
  - Used for directly accessing the management node, and other nodes.
  - Interconnects the site gateway, and nodes (by way of in-band NIC).
  - Can provide the cluster with access to the internet.
  - Can be combined with the management node (same physical network).
  - Usually an Ethernet network.

5.3.2 xCAT database: Objects and tables

The xCAT database contains all information that relates to the cluster. This information is defined by an administrator or automatically obtained from the cluster, such as nodes, networks, services, and configuration for any services that are used by xCAT (for example, DNS, DHCP, HTTP, and NFS).

All data is stored in tables and can be manipulated directly as tables (for example, `nodelist`, `networks`, `nodegroup`, `osimage`, or `site`) or logically as objects (or object definitions) of certain types (for example, `node`, `network`, `group`, `osimage`, or `site`) by using the following commands:

- Objects:
  - View: `lsdef`
  - Create: `mkdef`
  - Change: `chdef`
  - Remove: `rmdef`
- Tables:
  - View: `tabdump`
  - Change: `tabedit` or `chtab`

On certain tables or object attributes (typically on per-node attributes), you can use *regular expressions* to set an attribute's value according to the name of the respective object (for example, the IP address of compute nodes).

The xCAT database is stored in a SQLite database by default. Powerful open source database engines, such as MySQL, MariaDB, and PostgreSQL are also supported for a large clusters.

For more information, see the xCAT database manual page by using the following command:

```
$ man 5 xcatdb
```

5.3.3 xCAT node booting

The xCAT management node can control the boot method (or device) of the compute nodes by using the following methods:

- Change the temporary boot device configuration of the bootloader by way of IPMI.
- Change the network boot configuration that is provided to the bootloader by way of DHCP.
- Do not provide a network boot configuration to the bootloader, which allows it to boot from other device than network adapters.

The methods that are based in the network boot configuration require the correct automatic boot (or autoboot) configuration in the bootloader, and dynamic configuration of DHCP and Trivial File Transfer Protocol (TFTP) servers with general and per-node settings.

New or undiscovered nodes can be booted into node discovery, and known or discovered nodes into arbitrary stages (for example, OS provisioning, installed OS, basic shell environment, or node discovery again).

For that purpose, the nodes' bootloader must be configured to automatically boot with the following device order: Primarily, from the network interface on the xCAT management network (to obtain network and boot configuration by way of DHCP/TFTP from the xCAT management node), and secondarily from local disks. For more information, see 5.2.2, "Boot order configuration" on page 196.

On that foundation, any boot image can be specified by the xCAT management node through DHCP to a node, retrieved by the node through TFTP, and booted. If no boot image is specified, the node proceeds to boot from disk, which can contain an OS installation that is already provisioned.

5.3.4 xCAT node discovery

The xCAT node discovery (or node discovery process) consists of booting a new (or undiscovered) node and allowing the management node to identify (or *discover*) the node by using some specific method.

For example, during boot, the node can be offered a special-purpose boot image by the management node. This process is called *genesis* (basically, a Linux *kernel* and a custom initial RAM file system, or *initramfs*). Genesis collects identification and hardware information about the node, informs the management node about it, and waits for instructions. The management node can then configure and control the discovered node (for example, based on object definitions).

The following node discovery methods are available in xCAT, ranging between more manual and more automatic:

- Manual node definition (not really "discovery")
- Machine Type and Model, and Serial number-based (MTMS) discovery (adopted in this chapter)
- Sequential discovery
- Switch-based discovery

For more information about node discovery methods, see the xCAT Hardware Discovery & Define Node documentation page.

For example, the MTMS-based discovery can be summarized in the following sequence:

1. The new or undiscovered node is powered on, and the bootloader requests a network address and boot configuration.
2. The xCAT management node does not recognize the node; that is, the Media Access Control (MAC) address in the request. It provides the node with a temporary network address and pointers to the node discovery boot image.
3. The node applies the network configuration, downloads the node discovery boot image, and boots it.
4. The boot image collects hardware information (for example, the system's machine type and model, serial number, network interfaces' MAC address, and processor and memory information) and reports it back to the xCAT management node.
5. The xCAT management node attempts to match part of the reported hardware information to a node object definition (currently, the system's machine-type and model, and serial number).

6. If a match is identified, the xCAT management node then configures that node according to its object definition (for example, network configuration and next stages to perform) and updates that object definition with any new hardware information.

The node can then respond to in-band or OS-level management operations through SSH on its IP address (which is available on the running Genesis image).

After node discovery, the new or undiscovered node becomes a known or discovered node and supports in-band operations; for example, OS provisioning, software installation, and configuration, from the xCAT management node.

Note: The Genesis image contains the IPMItool utility, which is available for running IPMI commands both out-of-band (by way of the BMC IP address) and in-band (by way of the internal connection between the system and BMC, independently of network configuration).

This function is specially useful if problems occur when the BMC network is configured, which can render it unresponsive out-of-band, but still responsive in-band.

5.3.5 xCAT BMC discovery

The xCAT BMC discovery occurs automatically after node discovery. It can be summarized in the following sequence:

1. The xCAT management node attempts to match the node object definition to a temporary BMC object definition (with machine type and model, and serial number information) for new or undiscovered BMCs.
2. If a match is identified, the xCAT management node configures the BMC according to the BMC attributes in the node object definition (for example, network configuration) and removes the temporary BMC object.

The node can then respond to out-of-band management operations through IPMI on its BMC's IP address.

The BMC becomes a discovered BMC, and the corresponding node supports out-of-band operations (for example power control and monitoring) by way of its object definition in the xCAT management node.

5.3.6 xCAT OS installation types: Disks and state

The xCAT management node can support the following methods for provisioning an OS and providing persistent state (data) to the compute nodes according to availability of disks and persistency requirements:

- Diskful and Stateful

  The OS is installed to disk and loaded from disk. Changes are written to disk (persistent).

- Diskless and Stateless

  The OS is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and discarded (not persistent).

- Diskless and Statelite

  An intermediary type between stateless and stateful. The OS is installed by using a different and contained method in the management node and loaded from the network. Changes are written to memory and can be stored (persistent) or discarded (not persistent).

For more information about OS provisioning and state persistency, see the xCAT IBM Power LE / OpenPOWER documentation page.

5.3.7 xCAT network interfaces: Primary and additional

In xCAT terminology, a network adapter or interface[2] in a node can be *primary* or *additional* (also known as *secondary*). Only one primary network adapter exists, and zero or more additional network adapters can exist.

The primary network interface of a node connects to the xCAT management network (that is, to the management node), and is used to provision, boot, and manage that node.

An additional network interface connects to an xCAT network other than the xCAT management network and xCAT service network. Therefore, it is used by xCAT application networks, xCAT site networks or public networks, or for other purposes.

For more information, see the xCAT Configure Additional Network Interfaces - confignics documentation page.

5.3.8 xCAT software kits

The xCAT provides support to a software package format that is called *xCAT Software Kits* (also known as *xCAT kits* or *kits*) that is tailored for installing software in xCAT clusters. Kits are used with some products of the IBM HPC Software stack.

An xCAT software kit can include a software product's distribution packages, configuration information, scripts, and other files. It also includes xCAT-specific information for installing the appropriate product pieces, which are referred to as the *kit components,* to a particular node according to its environment and role in the xCAT cluster.

The kits are distributed as *tarballs*, which are files with the `.tar` extension. The kits can be either *complete* or *incomplete* (which are also known as *partial*), which indicates whether a kit contains the packages of a product (complete) or not (incomplete/partial).

The incomplete or partial kits are indicated by file names with the `NEED_PRODUCT_PKGS` string, which can be converted to complete kits when combined with the product's distribution packages. Incomplete or partial kits are usually distributed separately from the product's distribution media.

After a complete kit is added to the xCAT management node, its kit components can be assigned or installed to new and existing OS installations.

For more information about xCAT Software Kits, see the xCAT Software Kits documentation page.

[2] The term *network interface* can be more precise and unambiguous because a single network *adapter* can provide multiple network *interfaces* (for example, one interface per port, or virtual interfaces) to the OS.

5.3.9 xCAT synchronizing files

Synchronizing (sync) files to the nodes is a feature of xCAT used to distribute specific files from the management node to the new-deploying or deployed nodes.

This function is supported for diskful or diskless nodes. Generally, the specific files are usually the system configuration files for the nodes in the `/etc` directory, like `/etc/hosts`, `/etc/resolve.conf`. It can also be the application programs configuration files for the nodes. The advantages of this function are: It can parallel sync files to the nodes or nodegroup for the installed nodes and it can automatically sync files to the newly installing node after the installation. Additionally, this feature also supports the flexible format to define the synced files in a configuration file, called `synclist`.

The synclist file can be a common file for a group of nodes that use the same profile or osimage, or can be the special one for a particular node. Considering that the location of the synclist file is used to find the synclist file, the common synclist is put in a specific location for Linux nodes or specified by the osimage.

For more information about xCAT file synchronization, see the xCAT Synchronizing Files documentation page.

5.3.10 xCAT version

This section describes xCAT version 2.12.4, which includes the following functions:

- RHEL Server 7.3 support for PowerPC 64-bit Little-Endian (ppc64le):
  - Provisioning types: Diskful/stateful and diskless/stateless
  - Support for CUDA Toolkit for NVIDIA GPUs
  - Support for Mellanox OFED for Linux for Mellanox InfiniBand adapters
  - Support for kits with the IBM HPC Software
  - Support for non-virtualized (or bare-metal) mode
- Power System S822LC server with OPAL firmware:
  - Hardware discovery for BMCs
  - Hardware management with IPMI

For more information, see the xCAT 2.12.4 release website.

Update: After this publication was written, xCAT 2.13 was announced. For more information, see the xCAT 2.13 Release Notes website.

5.3.11 xCAT scenario

This chapter adopts the following xCAT networks and network addresses (it follows the network setup that is described in Figure 1-1 on page 5):

- Network address scheme: `10.`*`network-number`*`.0.0/16 (16-bit network mask)`
- Management (OS) network (10 Gigabit Ethernet): `10.`*`1`*`.0.0/16`
- Service (BMC) network (1 Gigabit Ethernet): `10.`*`2`*`.0.0/16`
- High-performance interconnect (InfiniBand): `10.`*`10`*`.0.0/16`
- Site network (1 Gigabit Ethernet): `9.x.x.x/24`

Note: Depending on the adapters in the systems, the number and type of network interfaces can differ.

The management and service networks are not combined in a single network interface for this setup. For 8335-GTB servers, it is recommended to use two physical cables, which allows the BMC port to be *dedicated*. Therefore, one port is used by the OS only.

For more information, see the xCAT noboot documentation website.

The network switch VLAN configuration can ensure that the management node can access all nodes' in-band and out-of-band (BMC) network interfaces, and that the non-management nodes can access only in-band network interfaces (but not out-of-band network interfaces).

The IP addresses are assigned to the nodes according to the following scheme:

- IP address: `10.network-number.rack-number.node-number-in-rack`
- xCAT management node: `10.network-number.0.1`
- Temporary IP addresses (the dynamic range): `10.network-number.254.sequential-number`

The host names (or node names) are assigned to the POWER8 (thus, *p8*) compute nodes according to the following naming scheme:

- Node naming scheme: `p8r<rack-number>n<node-number-in-rack>`
- For example, for five racks and six systems per rack: `p8r1n1, p8r1n2, ..., p8r2n1, p8r2n2,... p8r5n6`

The following IPMI credentials are adopted for the BMCs:

- Username: `ADMIN`
- Password: `admin`

5.4 Initial xCAT Management Node installation on S812LC

This section describes the deployment of the xCAT Management Node with RHEL Server 7.3 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on the IBM Power System S812LC server. The S812LC is a single socket POWER8 server with two rear 3.5-inch disks and up to 12 3.5-inch front disks. It is the perfect management server for a S822LC node HPC Cluster.

After you complete the steps that are described in this section, the Management Node is ready for the configuration and execution of the xCAT Node Discovery process on other nodes in the cluster.

For more information, see the xCAT Extreme Cloud/Cluster Administration Toolkit documentation website. At the website, see the following resources:

- *Installation Guide for Red Hat Enterprise Linux* (click **Install Guides** → **Installation Guide for Red Hat Enterprise Linux**)
- *Configure xCAT* section (click **Admin Guide** → **Manage Clusters** → **IBM Power LE / OpenPOWER** → **Configure xCAT**)

5.4.1 RHEL server

You can install RHEL server by using one of the following methods:

- Virtual media (with the BMC ASM interface)
- Network boot server
- USB device
- Network installation (for example, by way of HTTP)

Note: It is not possible to use optical media because optical drives are not supported.

This chapter describes the installation process that uses two USB flash drives. This method is recommended because it is the easiest method and has the least dependencies. The OS is installed on the two rear disks by using this method.

For more information about other installation methods, see the following resources:

- The RHEL 7 Installation Guide at the Red Hat documentation website
- The Installing Linux on OPAL POWER8 systems documentation at the IBM Knowledge Center website

Prerequisites

The following components are required for the USB flash drive installation:

- A computer that is running Linux (recommended), macOS, or Windows
- Access to RHEL server 7.3 ISO
- Two USB flash drives (one drive must be at least 4 GB)
- One of the following means to navigate in petitboot:
  - Network connection to BMC and ipmitool (recommended)
  - Serial connection
  - VGA monitor and USB keyboard

Configure BMC IP

Before installing the system, configure the BMC so that the installation can be performed remotely from your desk:

1. Initial the BMC configuration by using one of the following methods:
   a. Connect a USB keyboard and a VGA monitor to the server.
   b. Use a notebook with a serial connection to the server. For more information, see the Connecting to your server with a serial console connection page of the IBM Knowledge Center website.
2. Power on your server by pressing the power button on the front of your system. Your system powers on to the Petitboot panel. This process takes approximately 1 - 2 minutes to complete.

   Note: Do not walk away from your system. When Petitboot loads, your monitor becomes active and you must push any key to interrupt the boot process.

3. At the Petitboot bootloader main menu, select **Exit to shell**.

4. Set the BMC network settings for the first channel:
   a. Set mode to static:
      ```
      $ ipmitool lan set 1 ipsrc static
      ```
   b. Set BMC IP address:
      ```
      $ ipmitool admin lan set 1 ipaddr <ipaddr>
      ```
   c. Set netmask:
      ```
      $ ipmitool admin lan set 1 netmask <netmask>
      ```
   d. Set default gateway server:
      ```
      $ ipmitool lan set 1 defgw ipaddr <gateway_server>
      ```
5. Verify the new configuration by using the following command:
   ```
   $ ipmitool lan print 1
   <...>
   IP Address Source       : Static Address
   IP Address              : 9.x.x.x
   Subnet Mask             : 255.255.255.0
   MAC Address             : 70:e2:84:14:09:ad
   <...>
   Default Gateway IP      : 9.x.x.x
   Default Gateway MAC     : 98:be:94:00:47:84
   Backup Gateway IP       : 0.0.0.0
   Backup Gateway MAC      : 00:00:00:00:00:00
   802.1q VLAN ID          : Disabled
   802.1q VLAN Priority    : 0
   <...>
   ```
6. To activate the new configuration, you must reset the BMC by using the following command:
   ```
   $ ipmitool mc reset cold
   ```
7. Wait approximately 3 minutes. Then, attempt to ping the BMC IP to test the new configuration.

Note: If your ping does not return successfully, complete the following steps:

1. Power your system off by using the **`ipmitool power off`** command.
2. Unplug the power cords from the back of the system. Wait 30 seconds and then, apply power to boot BMC.

Acquiring network device name

Complete the following steps to acquire the network device name for the RHEL installation:

1. Open the Serial Over LAN (SOL) console to the previously configured BMC, as shown in the following example (the default BMC username and password is: `ADMIN:admin`):
   ```
   $ ipmitool -I lanplus -H <bmc-ip> -U <username> -P <password> sol activate
   ```

Note: Alternatively, you can use the serial connection at the node. For more information, see “Configure BMC IP” on page 209.

2. Check that the system is turned on, as shown in the following example:
   ```
   $ ipmitool -I lanplus -H <bmc-ip> -U <username> -P <password> chassis power on
   ```

3. Wait for the Petitboot panel and choose **System information**, as shown in Example 5-9.

Example 5-9 Petitboot system information panel

```
Petitboot System Information
 ................................................................................
 Network interfaces
 enp1s0f0:
  MAC:  98:be:94:68:ab:e8
  link: down

 enp1s0f1:
  MAC:  98:be:94:68:ab:e9
  link: down

 enp1s0f2:
  MAC:  98:be:94:68:ab:ea
  link: up

 enp1s0f3:
  MAC:  98:be:94:68:ab:eb
  link: down

 tunl0:
  MAC:  00:00:00:00:08:00
 ................................................................................
 x=exit, h=help
```

Look for the `link: up` to locate which adapters have cables that are connected. In our example, `enp1s0f2` is the adapter that is connected to the site LAN (all other cables are not connected yet). The network device name is needed as described in “Editing kickstarter file for second USB flash drive” on page 212.

4. Leave the serial console open because it is needed during the process to prepare the USB flash drives, which is described next. To close the console, use the ~.

Preparing the USB flash drives

We created a kickstarter file for you that helps to complete the basic installation and configuration (“Editing kickstarter file for second USB flash drive” on page 212) process. This kickstarter file creates a software RAID1 (mdraid) on both rear disks, disables the firewall and selinux (required by xCAT), and configures the initial network interface.

Prepare the following USB flash drives:

- USB flash drive with the main RHEL server 7.3 installation ISO
- USB flash drive with the kickstarter configuration file

You must edit the kickstarter file and adapt it for your environment. For more information, see “Editing kickstarter file for second USB flash drive” on page 212.

Note: You can add another partition to the first USB stick and use this partition for the kickstarter configuration. We want to keep the configuration simple; therefore, we use two separate USB flash drives.

Copying RHEL ISO to first USB flash drive

The following procedure assumes that you are using a Linux or a macOS system. Complete the following steps:

1. Download the RHEL server 7.3 installation ISO from Red Hat.
2. Use the **dd** command to copy the ISO file to the first USB flash drive, as shown in the following example:

```
$ dd if=/home/user/Downloads/RHEL-7.3-20161019.0-Server-ppc64le-dvd1.iso
of=/dev/<usb_device> bs=512k
```

Note: For more information about other operating systems, such as Windows, see the Red Hat Making Installation USB Media website.

Editing kickstarter file for second USB flash drive

Get the `smn_md.cfg` kickstarter file at the following website (for more information, see Appendix B, “Additional material” on page 377).

Open the `smn_md.cfg` file by using an editor of your choice and edit the first parameters, such as language, time zone, root password, and network for your environment, as shown in Example 5-10.

Example 5-10 Edit section of smn_md.cfg kickstarter file

```
#########################
#  Kickstarter file for  #
#  IBM S812LC server     #
#########################

####################
# Edit the following #
####################

#System language
lang en_US

#System keyboard
keyboard us

#System timezone
timezone US/Eastern

#Root password. Default is cluster
rootpw --iscrypted $1$tv1B/ROT$OSaZEF5R.mkdI29iZci2O0

#Network information
network --bootproto=static --ip=192.168.0.1 --netmask=255.255.255.0
--gateway=192.168.0.254 --nameserver=192.168.0.100,192.168.0.200 --device=enp1s0f2
network --hostname=xcat-mn.domain.com


##################################
# Do not touch anything below this #
##################################
<...>
```

You also might have to change the network device name. For more information, see 5.4.1, “RHEL server” on page 209.

Generate a root password hash by using the following command:

```
$ openssl passwd -1 <root_pw>
```

You also can use the preconfigured hash, which sets `cluster` as the root password.

Finally, copy the edited kickstarter file to the second FAT32 formatted USB flash drive. It is recommended that you set the label of your USB drive to something informative, such as “KICKSTART”.

Record the second USB flash drive UUID by using the following **`blkid`** command:

```
$ blkid
/dev/sdc2: SEC_TYPE="msdos" UUID="BAE0-6F47" TYPE="vfat"
```

Starting the installation

Complete the following steps to start the installation process:

1. Put both USB keys only in the blue USB ports.
2. Open your serial console window or reconnect to the console.
3. Restart the server by using the following **`ipmitool`** command:

   ```
   $ ipmitool -I lanplus -H <bmc-ip> -U <username> -P <password> chassis power
   cycle
   ```

4. Wait for the Petitboot Panel. Choose **Install Red Hat Enterprise Linux 7.3 (64-bit kernel)** and press E to edit the boot arguments, as shown in the following example:

   ```
   <...>
   [USB: sdn / 2016-10-19-18-33-07-00]
       Rescue a Red Hat Enterprise Linux system (64-bit kernel)
       Test this media & install Red Hat Enterprise Linux 7.3  (64-bit kernel)
    *  Install Red Hat Enterprise Linux 7.3 (64-bit kernel)
   <...>
   ```

   Note: Select **Rescan devices** if USB device does not appear.

5. Change the boot arguments as shown in the following example:

   ```
   ro inst.ks=hd:UUID=<UUID_KICKSTARTER>:/smn_md.cfg
   inst.stage2=hd:UUID=<UUID_RHEL7_ISO>
   ```

   Note: `<UUID_KICKSTARTER>` is your second USB flash drive with kickstarter file that is described in “Editing kickstarter file for second USB flash drive” on page 212. `<UUID_RHEL7_ISO>` is your first USB flash drive with RHEL7 ISO on it. It is the string in “[]” brackets after the device name in the edit menu that is shown in Example 5-11.

   Example 5-11 shows the full string for our example.

Example 5-11 Boot argument string

```
Device:          (*) sdm [2016-10-19-18-33-07-00]
                 ( ) Specify paths/URLs manually

 Kernel:          /ppc/ppc64/vmlinuz
 Initrd:          /ppc/ppc64/initrd.img
```

```
 Device tree:
 Boot arguments: ro inst.ks=hd:UUID=BAE0-6F47:/smn_md.cfg
inst.stage2=hd:UUID=2016-10-19-18-33-07-00
```

6. Click **OK**. Then, press Enter, and Enter again to start the Installation.

   The installation proceeds automatically. After the installation is finished, the system reboots.

   You must choose the correct boot target, starting with `Disk` in petitboot if you do not unplug the USB flash drive during the reboot, as shown in the following example:

```
[Disk: md126 / 62c0a482-733c-4e74-ac7c-22ee6483ab3d]
    Red Hat Enterprise Linux Server (0-rescue-31154c160e114634900069a33764d7d8)
 *  Red Hat Enterprise Linux Server (3.10.0-514.el7.ppc64le) 7.3 (Maipo)
```

7. After final disk boot, login by using SSH with your root credentials as configured in your kickstarter file.

Configuring RHEL Server 7.3 package repository

To install more packages and satisfy package dependencies for xCAT, configure a `yum` package repository for the RHEL Server 7.3 packages.

You can configure the system for the RHEL regular package update channels, or at least, the RHEL Server 7.3 DVD 1 ISO. For simplicity, this chapter describes RHEL Server 7.3 DVD 1 ISO.

Complete the following steps to configure the package repository for RHEL Server 7.3 DVD 1:

1. Copy RHEL Server 7.3 ISO content from USB flash drive to server, as shown in the following example:

```
$ mkdir /mnt/usb
$ mount -o ro /dev/disk/by-uuid/2016-10-19-18-33-07-00 /mnt/usb
$ cp -r /mnt/usb /mnt/rhel7.3-dvd1
$ chmod 500 /mnt/rhel7.3-dvd1
$ umount /mnt/usb
```

2. Configure repository:

   a. Install the RPM GPG key, as shown in the following example:

```
$ rpm --import /mnt/rhel7.3-dvd1/RPM-GPG-KEY-redhat-release
```

   b. Create the yum package repository file, as shown in the following example:

```
$ cat <<EOF >/etc/yum.repos.d/rhel7.3-dvd1.repo
[rhel-7.3-dvd1]
name=RHEL 7.3 Server DVD1
baseurl=file:///mnt/rhel7.3-dvd1
enabled=1
gpgcheck=1
EOF
```

You can verify that the package repository is configured correctly by using the following command:

```
$ yum repolist
yum repolist
Loaded plugins: product-id, search-disabled-repos, subscription-manager
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
```

```
repo id                                repo name
status
rhel-7.3-dvd1                          RHEL 7.3 Server DVD1
3.454
repolist: 3.454
```

Front disk formatting

Consider the following points as you configure the front drives for data:

- `/install` for xCAT must be at least 500 GB because it consists of all your images and software packages.
- Keep track of logs and databases.
- Use LVM for volume management to be flexible.

The front drives can be formatted by using one of the following methods:

- Use the specific built-in HW RAID controller. For more information, see the manufacturer's manual.
- Use LVM mirroring. For more information, see the Red Hat Creating Mirrored Volumes documentation.

5.4.2 xCAT packages

The xCAT is a collection of packages that are available for download at the xCAT project page of the xCAT website.

The xCAT packages are organized into the following package repositories:

- xCAT Core Packages: Packages with the xCAT.

  This package repository is available in the following streams (or types):

  - Release (or Stable) builds: The latest, officially released (general availability) version of xCAT.
  - Snapshot Builds: Unreleased changes for the next refresh of current version of xCAT.
  - Development Builds: Unreleased changes for the next version of xCAT.
- xCAT Dependency Packages: Required packages that are not provided by the Linux distribution.

xCAT is installed by using one of the following methods:

- Manually with online or offline RPM package repositories for RHEL, SUSE Linux Enterprise Server (SLES), and Debian packages (for Ubuntu).
- xCAT version 2.12.1 added an installation tool called `go-xcat`. This tool simplifies the xCAT installation and update procedure and configures all required repositories automatically.

This section describes performing the installation by using the new `go-xcat` tool. Complete the following steps:

1. Download the `go-xcat` tool, as shown in the following example:

```
$ wget
https://raw.githubusercontent.com/xcat2/xcat-core/master/xCAT-server/share/xcat
/tools/go-xcat -O - >/tmp/go-xcat
$ chmod +x /tmp/go-xcat
```

2. Run the tool without parameters to configure both xCAT repositories and install the latest version of xCAT, as shown in the following example:

Note: You can specify a specific version of xCAT to install with `-x <version>`.

```
$ /tmp/go-xcat install
Operating system:   linux
Architecture:       ppc64le
Linux Distribution: rhel
Version:            7.3

Reading repositories ...... done

xCAT is going to be installed.
Continue? [y/n] y
<... long yum output ...>
Complete!

xCAT has been installed!
========================

If this is the first time xCAT has been installed, run the following
commands to set environment variables into your PATH:

    for sh,
        `source /etc/profile.d/xcat.sh'
    or csh,
        `source /etc/profile.d/xcat.csh'
```

For an offline installation, you can specify the path to download `xcat-core` and `xcat-dep` packages, as shown in the `go-xcat` help output in Example 5-12.

Example 5-12 go-xcat help output

```
$ /tmp/go-xcat -h
go-xcat version 1.0.10

Usage: go-xcat [OPTION]... [ACTION]
Install xCAT automatically

Options:
Mandatory arguments to long options are mandatory for short options too.
  -h, --help                    display this help and exit
  --xcat-core=[URL]             use a different URL or path for the xcat-core
                                repository
  --xcat-dep=[URL]              use a different URL or path for the xcat-dep
                                repository
  -x, --xcat-version=[VERSION]  specify the version of xCAT; cannot use with
                                --xcat-core
  -y, --yes                     answer yes for all questions

Actions:
  install                       installs all the latest versions of xcat-core
                                and xcat-dep packages from the repository
  update                        updates installed xcat-core packages to the
                                latest version from the repository
```

```
Examples:
  go-xcat
  go-xcat install
  go-xcat update
  go-xcat --yes install
  go-xcat -x 2.12 -y install
  go-xcat --xcat-version=devel install
  go-xcat --xcat-core=/path/to/xcat-core.tar.bz2 \
          --xcat-dep=/path/to/xcat-dep.tar.bz2 install
  go-xcat --xcat-core=http://xcat.org/path/to/xcat-core.tar.bz2 \
          --xcat-dep=http://xcat.org/path/to/xcat-dep.tar.bz2 install

xCAT (Extreme Cloud/Cluster Administration Toolkit): <http://xcat.org/>
Full documentation at: <http://xcat-docs.readthedocs.io/en/stable/>
```

3. Verify that the package repositories are configured correctly, as shown in the following example:

```
$ yum repolist
<...>
repo id           repo name                 status
rhel-7.3-dvd1     RHEL 7.3 Server DVD1      3.454
xcat-core         xCAT 2 Core packages      20
xcat-dep          xCAT 2 depedencies        32
repolist: 3.506
```

4. Verify that the xCAT service is running, as shown in the following example:

```
$ systemctl status xcatd
xcatd.service - LSB: xcatd
   Loaded: loaded (/etc/rc.d/init.d/xcatd)
   Active: active (running) since Do 2016-11-17 15:02:35 EST; 15min ago
<...>
```

5. Verify the version information and node type, as shown in the following example:

```
$ source /etc/profile.d/xcat.sh
Version 2.12.4 (git commit 64ec2d41285d9a8770d8d9ef909f251ecbb5100b, built Thu
Nov 10 23:59:02 EST 2016)
This is a Management Node
dbengine=SQLite
```

Note: The **source** command for the `xcat.sh` file is required on current login shells only.

5.4.3 Configuring more network interfaces

The xCAT requires a static IP network configuration for the Management Node. The site network interface is configured during the kickstarter process. For all other networks follow the instructions that are presented in this section.

Note: This requirement applies to any xCAT networks with communication between the Management Node and other nodes (for example, Management and Service Networks).

You can configure an Ethernet network interface with static IP address in one of many ways. This section describes the method that uses `sysconfig` or `ifcfg` files, and the **`nmcli`** command (Network Manager Command Line Interface).

For more information, see the RHEL 7 Networking Guide at the Red Hat documentation website.

This section uses content from the following sections of the RHEL 7 Networking Guide:

- Section 1.9: Network configuration by using sysconfig files
- Section 2.4.1: Configuring a network interface with ifcfg files

To configure a network interface with static IP address, complete the following steps for the management (OS) network:

1. Create the `/etc/sysconfig/network-scripts/ifcfg-<network-interface>` file.

   For the scenario that is described in this chapter, the file resembles the following example:

   ```
   $ cat <<EOF >/etc/sysconfig/network-scripts/ifcfg-enp1s0f0
   DEVICE=enp1s0f0
   ONBOOT=yes
   BOOTPROTO=none
   IPADDR=10.1.0.1
   PREFIX=16
   IPV6INIT=yes
   EOF
   ```

2. Verify that the network configuration is not in effect immediately (no IPv4 or IPv6 address):

   ```
   $ ip addr show enp1s0f0
   4: enp1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen
   1000
       link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
   ```

3. Reload the network configuration for that network interface by using the **`nmcli`** command.

   The network configuration is loaded automatically on system boot:

   ```
   $ nmcli connection load /etc/sysconfig/network-scripts/ifcfg-enp1s0f0
   ```

4. Verify that the network configuration is in effect (including an IPv6 link-local address):

   ```
   $ ip addr show enp1s0f0
   4: enp1s0f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen
   1000
       link/ether 98:be:94:59:fa:26 brd ff:ff:ff:ff:ff:ff
       inet 10.1.0.1/16 brd 10.1.255.255 scope global enp1s0f0
          valid_lft forever preferred_lft forever
       inet6 fe80::9abe:94ff:fe59:fa26/64 scope link
          valid_lft forever preferred_lft forever
   ```

For the service (BMC) network, as described in 5.3.11, “xCAT scenario” on page 207, a physically separate service network must be available.

Perform these steps for the service network as well. In our case, this network is connected to the 1 Gigabit Ethernet port `enp1s0f3` and uses the `10.2.0.0/16` network.

5.4.4 Host name and aliases

Although the kickstarter file configured the host name, you must verify that everything is set up correctly.

Verify the short and long (fully qualified domain name) host names are detected:

```
$ hostname --short
xcat-mn

$ hostname --long
xcat-mn.xcat-cluster
```

Configure the host name to be resolved to the IP address in the management (OS) network by completing the following steps:

1. Add the host aliases in the `/etc/hosts` file:

```
$ echo "10.1.0.1 $(hostname -s) $(hostname -f)" >> /etc/hosts
$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.1.0.1    xcat-mn xcat-mn.xcat-cluster
```

2. Verify that the short host name resolves to the long host name, and that the **`ping`** test works:

```
$ ping -c1 xcat-mn
PING xcat-mn.xcat-cluster (10.1.0.1) 56(84) bytes of data.
64 bytes from xcat-mn.xcat-cluster (10.1.0.1): icmp_seq=1 ttl=64 time=0.025 ms

--- xcat-mn.xcat-cluster ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.025/0.025/0.025/0.000 ms
```

5.4.5 xCAT networks

The xCAT networks configuration is stored the `networks` table, and is available as objects of the `network` type.

The use of the **`makenetworks`** command populates the `networks` table based on the current configuration of the network interfaces. It is run automatically during the installation of xCAT packages.

You can use the following commands to create, list, modify, and remove, `network` objects, and list, and edit, the `networks` table:

- `mkdef -t network`
- `lsdef -t network`
- `chdef -t network`
- `rmdef -t network`
- `tabdump networks`
- `tabedit networks`

To configure the xCAT networks, populate the `networks` table by using the **`makenetworks`** command, remove any non-xCAT networks, rename the xCAT networks (optional), and create any other xCAT networks.

For our scenario, complete the following steps:

1. Populate the `networks` table by using the **`makenetworks`** command:

```
$ makenetworks

$ lsdef -t network
10_1_0_0-255_255_0_0  (network)
10_2_0_0-255_255_0_0  (network)
9_x_x_x-255_255_255_0  (network)
fd55:xxxx:xxxx:33e::/64  (network)
```

2. Remove any non-xCAT networks:

```
$ rmdef -t network 9_x_x_x-255_255_255_0
1 object definitions have been removed.

$ rmdef -t network fd55:xxxx:xxxx:33e::/64
1 object definitions have been removed.

$ lsdef -t network
10_1_0_0-255_255_0_0  (network)
10_2_0_0-255_255_0_0  (network)
```

3. (Optional) Rename the xCAT networks:

```
$ chdef -t network 10_1_0_0-255_255_0_0 -n net-mgmt
Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

$ chdef -t network 10_2_0_0-255_255_0_0 -n net-svc
Changed the object name from 10_1_0_0-255_255_0_0 to net-mgmt.

$ lsdef -t network
net-mgmt  (network)
net-svc (network)
```

4. Create any other xCAT networks:

```
$ mkdef -t network net-app-ib net=10.10.0.0 mask=255.255.0.0
1 object definitions have been created or modified.

$ lsdef -t network
net-app-ib (network)
net-mgmt  (network)
net-svc  (network)
```

Note: The Application Networks lack the `mgtifname` attribute (*`management interface name`*) because the Management Node is not connected to them; that is, only some other nodes are, such as the Compute Nodes, and Login Nodes.

However, Application Networks must be defined in the Management Node for it to perform their network configuration on other nodes (by way of the Management Network).

5. Verify the xCAT networks configuration.

   You can use the **lsdef** command to list the configuration of a specific xCAT network or networks. The following example lists the Management Network and Application Network for InfiniBand port 1:

```
$ lsdef -t network net-mgmt,net-app-ib
Object name: net-mgmt
    gateway=<xcatmaster>
    mask=255.255.0.0
    mgtifname=enp1s0f0
    net=10.1.0.0
    tftpserver=10.1.0.1
Object name: net-app-ib
    mask=255.255.0.0
    net=10.10.0.0
```

   You can use the **tabdump** command to list the configuration of all xCAT networks:

```
$ tabdump networks
#netname,net,mask,mgtifname,<...>
"net-mgmt","10.1.0.0","255.255.0.0","enp1s0f0",<...>
"net-svc","10.2.0.0","255.255.0.0","enp1s0f3",<...>
"net-app-ib","10.10.0.0","255.255.0.0",,,,,,,,,,,,,,,
```

5.4.6 DNS server

The xCAT configures the DNS server based on attributes of the `site` table, the `/etc/hosts` file, and node definitions. The **makedns** command applies the configuration.

The following attributes of the `site` table are used to configure the DNS server:

- `dnsinterfaces`: Host name (optional) and network interfaces for the DNS server to listen on.
- `domain`: DNS domain name for the cluster.
- `forwarders`: DNS servers for resolving non-cluster names, that is, the site's or external DNS servers.
- `master`: IP address of the xCAT management node on the management (OS) network, as known by the nodes.
- `nameservers`: DNS servers that are used by the compute nodes (usually, the IP address of the management node). The value *<xcatmaster>* indicates that the management node or service node that is managing a node (automatically defined to the correct IP address in the respective xCAT network) is more portable.

For more information, see the manual page of the `site` table by using the following command:

```
$ man 5 site
```

The use of the **makedns** command generates the following configuration files for the DNS server, and reloads it:

- `/etc/named.conf`: Main configuration file (generated by **makedns -n**).
- `/var/named/*`: Zone files for network names and addresses (generated by **makedns -n**)

To configure the DNS server, complete the following steps:

1. Check the following basic configuration parameters in your site table:

```
$ lsdef -t site -i dnsinterfaces,domain,forwarders,master,nameservers
Object name: clustersite
    domain=xcal-cluster
    forwarders=9.x.x.x,9.x.x.x
    master=10.1.0.1
    nameservers=10.1.0.1
```

   Note: To get a detailed description for each parameter, run the `tabdump -d site` command.

2. The `dnsinterfaces` attribute of the `site` table is not configured yet. Set the attribute by using the following **`chdef`** command:

```
$ chdef -t site dnsinterfaces='xcat-mn|enp1s0f0'
1 object definitions have been created or modified.
```

3. Generate new configuration files for the DNS server by using the **`makedns -n`** command.

   The DNS server is automatically started or restarted, as shown in the following example:

```
$ makedns -n
Handling xcat-mn.xcat-cluster in /etc/hosts.
Handling localhost in /etc/hosts.
Handling localhost in /etc/hosts.
Getting reverse zones, this  take several minutes for a large cluster.
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this  take several minutes for a large cluster.
Completed updating DNS records.
```

   Note: It is important that no errors are reported in the output of the **`makedns`** command. The proper functioning of the DNS server is essential to several features in the xCAT (for example, node discovery).

   If any errors are reported, check the messages, the contents of the `/etc/hosts` file, and any node definitions (by using the **`lsdef`** command) for errors or inconsistencies.

4. Verify that the DNS server is resolving internal and external names with the **`host`** command by completing the following steps:

   a. Install the `bind-utils` package (not installed with the minimal installation package set):

```
$ yum install bind-utils
```

   b. Verify the name resolution of internal names. For example, the Management Node (that is, its short and long host names are associated, and are resolved to its IP address in the Management Network):

```
$ host xcat-mn 10.1.0.1
Using domain server:
Name: 10.1.0.1
Address: 10.1.0.1#53
Aliases:
```

```
xcat-mn.xcat-cluster has address 10.1.0.1
```

c. Verify the name resolution of external names:

```
$ host example.com 10.1.0.1
Using domain server:
Name: 10.1.0.1
Address: 10.1.0.1#53
Aliases:

example.com has address 93.184.216.34
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

5.4.7 DHCP server

The xCAT configures the DHCP server based on attributes of the `site` and `networks` tables, and the node definitions (for reservation of IP address leases based on MAC address). The **`makedhcp`** command applies the configuration.

The following attributes of the `site` and `networks` tables are used to configure the DHCP server:

- `dhcpinterfaces` (`site` table)

  Host name (optional) and network interfaces on which the DHCP server listens.

- `dynamicrange` (`networks` table)

  Range of IP addresses that are *temporarily* assigned during the node discovery process, which are required in the xCAT management and service networks.

The use of the **`makedhcp`** command generates the following configuration files for the DHCP server, and reloads it:

- `/etc/dhcp/dhcpd.conf`: Main configuration file (generated by **`makedhcp -n`**)
- `/var/lib/dhcpd/dhcpd.leases`: IP address leases (generated by **`makedhcp -a`**)

Because we use two physically separate networks for OS and BMC, we must add the **`:noboot`** tag to the service (BMC) network in `dhcpinterfaces`. This tag prevents the distribution of the genesis image in this network. For more information, see 5.3.11, “xCAT scenario” on page 207. In this example `enp1s0f0` is connected to the dedicated management (OS) network and `enp1s0f3` is connected to the dedicated service (BMC) network.

To configure of the DHCP server, complete the following steps:

1. Set the `dhcpinterfaces` attribute of the `site` table by using the **`chdef`** command (note the `:nooboot` flag), as shown in the following example for the scenario in this chapter:

   ```
   $ chdef -t site dhcpinterfaces='xcat-mn|enp1s0f0,xcat-mn|enp1s0f3:noboot'
   1 object definitions have been created or modified.
   ```

   You can verify the attributes by using the **`lsdef`** command, as shown in the following example:

   ```
   $ lsdef -t site -i dhcpinterfaces
   Object name: clustersite
       dhcpinterfaces=xcat-mn|enp1s0f0,xcat-mn|enp1s0f3:noboot
   ```

2. Remove the genesis configuration file from the `:noboot` tagged network with by using the **`mknb`** command:

   Note: The **`mknb`** (make network boot) command generates the network boot configuration file (specified in the `/etc/dhcp/dhcpd.conf` file) and genesis image files. It is run automatically when the xCAT packages are installed.

   ```
   $ mknb ppc64
   Creating genesis.fs.ppc64.gz in /tftpboot/xcat
   ```

   Note: At the time of writing, a bug exists in xCAT 2.12.4. The **`mknb`** command fails if `dhcpinterfaces` contains a nodegroup definition with "|".

   A fix for this bug was opened and merged. For more information, see the Fix mknb: Undefined subroutine noderange page.

   xCAT 2.13 has this fix included.

3. Generate the configuration file for the DHCP server by using the **`makedhcp -n`** command. The DHCP server is automatically started or restarted, as shown in the following example:

   ```
   $ makedhcp -n
   Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak

   The dhcp server must be restarted for OMAPI function to work
   Warning: No dynamic range specified for 10.1.0.0. If hardware discovery is
   being used, a dynamic range is required.
   Warning: No dynamic range specified for 10.2.0.0. If hardware discovery is
   being used, a dynamic range is required.
   ```

   Despite the message that is related to the need to restart the DHCP server, it is automatically restarted, as shown in the `/var/log/messages` file:

   ```
   $ tail /var/log/messages
   <...>
   <...> xcat[...]: xCAT: Allowing makedhcp -n for root from localhost
   <...> systemd: Starting DHCPv4 Server Daemon...
   <...> dhcpd: Internet Systems Consortium DHCP Server 4.2.5
   <...>
   ```

4. Generate the leases file for the DHCP server by using the **`makedhcp -a`** command. This step is only required if any node definitions exist (they do not yet exist in the scenario in this chapter):

   ```
   $ makedhcp -a
   ```

5.4.8 IPMI authentication credentials

The xCAT configures the authentication credentials for IPMI commands (for example, power management, console sessions, BMC discovery, and network configuration) based on attributes from (in this order) node definitions, node groups definitions, and the `passwd` table.

To configure IPMI authentication credentials on individual nodes or on node groups, set the `bmcusername` and `bmcpassword` attributes on the node or node group object with the **`chdef`** command:

```
$ chdef <node or group> bmcusername=<IPMI username> bmcpassword=<IPMI password>
```

If the IPMI authentication credentials are common across some or all of the systems' BMCs, you can set the common credentials in the `passwd` table. Any different credentials can be set in the respective node or node group objects.

For more information, see the Configure passwords page of the xCAT documentation website.

To configure the IPMI authentication credentials, complete the following steps:

1. Set the `username` and `password` attributes of the `ipmi` key/row in the `passwd` table by using the **`chtab`** or **`tabedit`** commands, as shown in the following example for the scenario in this chapter:

   ```
   $ chtab key=ipmi passwd.username=ADMIN passwd.password=admin
   ```

2. Verify the setting by using the **`tabdump`** command:

   ```
   $ tabdump -w key==ipmi passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "ipmi","ADMIN","admin",,,,
   ```

   You can use the **`tabdump`** command without filter arguments to list the configuration of all entries in the `passwd` table:

   ```
   $ tabdump passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "omapi","xcat_key","<...>=",,,,
   "ipmi","ADMIN","admin",,,,
   ```

   Note: If the IPMI authentication credentials are not set or invalid, some IPMI-based commands can show errors, as shown in the following example:

   ```
   $ rpower node status
   node: Error: Unauthorized name
   ```

5.5 xCAT node discovery

This section describes the xCAT node discovery (or hardware discovery) process. It covers the configuration steps that are required in the Management Node, and instructions for performing the discovery of nodes in the cluster. For more information, see 5.3.4, "xCAT node discovery" on page 204.

The xCAT provides the following methods for node discovery:

- Manual definition

  This method includes manual hardware information collection and node object definition. This example includes required node-specific and xCAT/platform-generic attributes:

  ```
  $ mkdef node1 \
     groups=all,s822lc \
     ip=10.1.1.1 mac=6c:ae:8b:6a:d4:e installnic=mac primarynic=mac \
     bmc=10.2.1.1 bmcusername=ADMIN bmcpassword=admin \
     mgt=ipmi cons=ipmi netboot=petitboot
  ```

- Machine Type and Model, and Serial Number (MTMS)-based discovery

  This process automatically collects MTM and S/N information from the node's BMC and OS (genesis), and matches it with a minimal manually defined node object (with `mtm` and `serial` attributes). This process automatically stores the hardware information in the matched node object.

- Sequential discovery

  This method automatically stores hardware information in a list of minimal node objects (with no attributes) in a sequential manner, in the order that the nodes are booted.

- Switch-based discovery

  Automatically identifies the network switch and port for the node (with the SNPMv3 protocol), and matches it with a minimal manually defined node object (with `switch` and `switchport` attributes). This process automatically stores the hardware information in the matched node object.

This section describes the MTMS-based discovery method.

5.5.1 Verification of network boot configuration and genesis image files

The xCAT node discovery requires the node to boot the genesis image from the network. (For more information, see 5.3.4, “xCAT node discovery” on page 204.) It is important to verify that the files for network boot configuration and genesis image are correctly in place.

To verify and generate the files for network boot configuration and genesis image, complete the following steps:

1. Verify the location of the platform-specific network boot configuration file in the `/etc/dhcp/dhcpd.conf` file.

   The scenario adopted in this chapter uses OPAL firmware:

   ```
   $ grep -A1 OPAL /etc/dhcp/dhcpd.conf
      } else if option client-architecture = 00:0e { #OPAL-v3
         option conf-file = "http://10.1.0.1/tftpboot/pxelinux.cfg/p/10.1.0.0_16";
   ```

   Note: The network boot configuration and genesis image files are required only for the xCAT Management Network.

2. Verify that the file exists:

   ```
   $ ls /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   ```

   Note: The specified file might not exist in some cases, such as when the **mknb** command is not run after a change in the xCAT networks configuration. If this issue occurs, run `mknb ppc64` again.

3. Verify the content of the file. It must contain pointers to the platform-specific Genesis image files:

   ```
   $ cat /tftpboot/pxelinux.cfg/p/10.1.0.0_16
   default "xCAT Genesis (10.1.0.1)"
      label "xCAT Genesis (10.1.0.1)"
      kernel http://10.1.0.1:80//tftpboot/xcat/genesis.kernel.ppc64
      initrd http://10.1.0.1:80//tftpboot/xcat/genesis.fs.ppc64.gz
      append "quiet xcatd=10.1.0.1:3001 "
   ```

4. Verify that the files of the Genesis image exist:

   ```
   $ ls -lh /tftpboot/xcat/genesis.{kernel,fs}.ppc64*
   -rw-r--r-- 1 root root 48M  8. Dez 20:36 /tftpboot/xcat/genesis.fs.ppc64.gz
   -rwxr-xr-x 1 root root 23M 25. Okt 09:31 /tftpboot/xcat/genesis.kernel.ppc64
   ```

Tip: To increase the verbosity of the node discovery in the nodes (which is useful for educational and debugging purposes), remove the `quiet` argument from the `append` line in the network boot configuration file by using the following command:

```
$ sed 's/quiet//' -i /tftpboot/pxelinux.cfg/p/10.1.0.0_16
```

5.5.2 Configuring the DHCP dynamic range

The xCAT node discovery requires temporary IP addresses for nodes and BMCs until the association with the respective node objects and permanent network configuration occur. The IP address range that is reserved for that purpose is known as *dynamic range*, which is an attribute of network objects and is reflected in the configuration file of the DHCP server.

You must provide temporary IP addresses for the Management and Service Networks to handle the *in-band* network interface (used by the Petitboot bootloader and Genesis image) and *out-of-band* network interface (used by the BMC).

Depending on your network environment and configuration, the Management Node can use different or shared network interfaces for the Management and Service Networks. This consideration is important because the dynamic range is defined *per network interface* in the configuration file of the DHCP server. As described in 5.3.11, "xCAT scenario" on page 207, the following networks are recommended:

- For different network interfaces, set the `dynamicrange` attribute on the Management and Service Networks.
- For shared network interface, set the `dynamicrange` attribute in either one of the Management or Service Networks.

To set the dynamic range, complete the following steps:

1. Set the `dynamicrange` attribute for the Management Network by using the **`chdef`** command, as shown in the following example for the scenario in this chapter:

   ```
   $ chdef -t network net-mgmt dynamicrange=10.1.254.1-10.1.254.254
   1 object definitions have been created or modified.
   ```

   You can verify it by using the **`lsdef`** command:

   ```
   $ lsdef -t network net-mgmt -i dynamicrange
   Object name: net-mgmt
       dynamicrange=10.1.254.1-10.1.254.254
   ```

2. Set the `dynamicrange` attribute for the Service Network by using the **`chdef`** command (only required for different network interfaces), as shown in the following example for the scenario in this chapter:

   ```
   $ chdef -t network net-svc dynamicrange=10.2.254.1-10.2.254.254
   1 object definitions have been created or modified.
   ```

   You can verify it by using the **`lsdef`** command:

   ```
   $ lsdef -t network net-svc -i dynamicrange
   Object name: net-svc
       dynamicrange=10.2.254.1-10.2.254.254
   ```

3. Generate the configuration file for the DHCP server by using the **`makedhcp -n`** command.

   ```
   $ makedhcp -n
   Renamed existing dhcp configuration file to  /etc/dhcp/dhcpd.conf.xcatbak
   ```

4. Verify the dynamic range in the configuration of the DHCP server:

```
$ grep 'network\|subnet\|_end\|range' /etc/dhcp/dhcpd.conf
```

5.5.3 Configuring BMCs to DHCP mode

The xCAT node discovery requires the BMCs' network configuration to occur in DHCP mode (until the association with the respective node objects and permanent network configuration occur).

Note: If the BMCs' network configuration cannot be changed (for example, because of network restrictions or maintenance requirements), skip the required steps for the BMC network configuration in the node discovery process. For more information, see 5.5.3, "Configuring BMCs to DHCP mode" on page 228, and 5.5.4, "Definition of temporary BMC objects" on page 230. Then, manually set the `bmc` attribute of one or more nodes to the respective BMC IP address.

To set the network configuration of the BMCs to DHCP mode, complete the following tasks:

- For BMCs that are in DHCP mode, no action required.
- For BMCs with Static (IP) Address mode and known IP address:

  a. Set the `IP Address Source` attribute to `DHCP` by using the **`ipmitool`** command.

  Notice that the IP Address Source attribute is set to `Static Address`:

```
$ ipmitool -I lanplus -H <ip> -U <user> -P <password> lan print 1
<...>
IP Address Source       : Static Address
IP Address              : 192.168.101.29
Subnet Mask             : 255.255.255.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

  Set it to DHCP:

```
$ ipmitool -I lanplus -H <ip> -U <user> -P <password> lan set 1 ipsrc dhcp
```

  Notice that the network configuration changes do not take effect immediately:

```
$ ipmitool -I lanplus -H <ip> -U <user> -P <password> lan print 1
<...>
IP Address Source       : DHCP Address
IP Address              : 192.168.101.29
Subnet Mask             : 255.255.255.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

  b. Reboot the BMC by using the **`ipmitool`** command, which is required for the network configuration changes to take effect:

```
$ ipmitool -I lanplus -H <ip> -U <user> -P <password> mc reset cold
```

  c. Wait for the BMC to perform initialization and network configuration.

  To determine when the BMC is back online and acquired an IP address through DHCP, you can watch the `/var/log/messages` file for DHCP server log messages, as shown in the following example:

```
$ tail -f /var/log/messages
<...>
<...> dhcpd: DHCPDISCOVER from 70:e2:84:14:02:54 via enp1s0f3
```

```
<...> dhcpd: DHCPOFFER on 10.2.254.1 to 70:e2:84:14:02:54 via enp1s0f3
<...> dhcpd: DHCPREQUEST for 10.2.254.1 (10.1.0.1) from 70:e2:84:14:02:54
via enp1s0f3
<...> dhcpd: DHCPACK on 10.2.254.1 to 70:e2:84:14:02:54 via enp1s0f3
<...>
```

It is also possible to determine when the BMC is back online with an approach that is based on its IPv6 link-local address, which does not change across power cycles, network configuration steps, and so on (see Example 5-13). To identify the IPv6 link-local address of each BMC in a local network, see the next bullet, “For BMCs with unknown IP address”.

Example 5-13 Waiting for the BMC with the ping6 command and IPv6 link-local address

```
$ while ! ping6 -c 1 <IPv6-link-local-address>%<network-interface>; do echo
Waiting; done; echo Finished
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) ...
<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254)...
<...>
Waiting
<...>
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) ...
64 bytes from fe80::72e2:84ff:fe14:254: icmp_seq=1 ttl=64 time=0.669 ms
<...>
Finished
```

d. Verify that the network configuration changes are in effect by using the **ipmitool** command:

```
$ ipmitool -I lanplus -H <ip> -U <user> -P <password> lan print 1
<...>
IP Address Source       : DHCP Address
IP Address              : 10.2.254.1
Subnet Mask             : 255.255.0.0
MAC Address             : 70:e2:84:14:02:54
<...>
```

- For BMCs with unknown IP address (in DHCP or Static Address mode):

  a. Install the `nmap` package:

  ```
  $ yum install nmap
  ```

  b. Discover one or more IPv6 link-local addresses of one or more BMCs by using the **nmap** command (see Example 5-14).

  Example 5-14 Discovering the IPv6 link-local address of BMCs with the nmap command

  ```
  $ nmap -6 --script=targets-ipv6-multicast-echo -e enp1s0f3

  Starting Nmap 6.40 ( http://nmap.org ) at <...>
  Pre-scan script results:
  | targets-ipv6-multicast-echo:
  |   IP: fe80::9abe:94ff:fe59:f0f2  MAC: 98:be:94:59:f0:f2  IFACE: enp1s0f3
  |   IP: fe80::280:e5ff:fe1b:fc99   MAC: 00:80:e5:1b:fc:99  IFACE: enp1s0f3
  |   IP: fe80::280:e5ff:fe1c:9c3    MAC: 00:80:e5:1c:09:c3  IFACE: enp1s0f3
  |   IP: fe80::72e2:84ff:fe14:259   MAC: 70:e2:84:14:02:59  IFACE: enp1s0f3
  |   IP: fe80::72e2:84ff:fe14:254   MAC: 70:e2:84:14:02:54  IFACE: enp1s0f3
  ```

```
|_  Use --script-args=newtargets to add the results as targets
WARNING: No targets were specified, so 0 hosts scanned.
Nmap done: 0 IP addresses (0 hosts up) scanned in 2.37 seconds
```

c. Perform the steps that are described in the “Static (IP) Address mode and known IP address” case. Replace the BMC’s IPv4 address in the **ipmitool** command for the IPv6 link-local address with the network interface as `zone index`, which is separated by the percent (%) symbol, as shown in the following example, (see Example 5-13 on page 229):

<IPv6-link-local-address>%<network-interface>

5.5.4 Definition of temporary BMC objects

The xCAT node discovery requires temporary node objects for BMCs until the association with the respective node objects and permanent network configuration occurs.

The temporary BMC objects are created by using the **bmcdiscover** command, which scans an IP address range for BMCs and collects information, such as machine type and model, serial number, and IP address. It can provide that information as objects in the xCAT database or the respective *stanzas* (a text-based description format with object name, type, and attributes). The objects are named after their MTM and S/N information (which are obtained through IPMI).

The temporary BMC objects are automatically removed during the node discovery process after the respective node objects are matched and ready to refer to the BMCs. It is a simple means to have xCAT objects to refer to the BMCs still using temporary IP addresses (not yet associated with the respective node objects). This configuration allows for running xCAT commands (for example, power management, and console sessions) before the network configuration of the BMC occurs.

The **bmcdiscover** command has the following requirements:

- The BMCs’ IP addresses to be within a known range (satisfied with the dynamic range configuration, as described in 5.5.2, “Configuring the DHCP dynamic range” on page 227, and BMCs in DHCP mode, as described in 5.5.3, “Configuring BMCs to DHCP mode” on page 228).
- The IPMI authentication credentials to be defined in the `passwd` table (satisfied in 5.4.8, “IPMI authentication credentials” on page 224) or by using command arguments.

The **bmcdiscover** command can be used with the following arguments:

- `-z`: Provides object definition stanza.
- `-t`: Provides object with attributes for BMC node type and hardware type.
- `-w`: Writes objects to the xCAT database.

To define temporary objects for the BMCs, complete the following steps:

1. Run the **bmcdiscover** command on the dynamic range of IP addresses:

```
$ bmcdiscover --range 10.2.254.1-254 -t -w
node-8335-gtb-0000000000000000:
        objtype=node
        groups=all
        bmc=10.2.254.1
        cons=ipmi
        mgt=ipmi
        mtm=8335-GTB
```

```
serial=0000000000000000
nodetype=mp
hwtype=bmc
```

Note: The serial number is set to zeros in the early system revision that is used for this book. This information is present on other systems (for example, from normal customer orders).

2. Verify that the BMC objects are listed as node objects by using the **lsdef** command.

   Note the attributes/values `nodetype=mp` and `hwtype=bmc`:

```
$ lsdef
node-8335-gtb-0000000000000000  (node)

$ lsdef node-8335-gtb-0000000000000000
Object name: node-8335-gtb-0000000000000000
    bmc=10.2.254.1
    cons=ipmi
    groups=all
    hwtype=bmc
    mgt=ipmi
    mtm=8335-GTB
    nodetype=mp
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
```

When BMC objects and IPMI authentication credentials are defined, you can run xCAT commands on the BMC objects, as shown in the following examples:

- `rpower` for power management
- `rcons` for console sessions (requires the `makeconservercf` command first)
- `rsetboot` for boot-method selection

Note: It is always possible to run **ipmitool** commands to the BMC IP address as well.

5.5.5 Defining node objects

The xCAT node discovery requires minimal node objects that can *match* the MTMS information, which can be collected either automatically by the Genesis image (by using the **mtm** and **serial** attributes) or manually. For more information, see 5.2.1, “Frequently used commands with the IPMItool” on page 194.

The node discovery also attempts to match the information with the temporary BMC objects to associate the node objects with their respective BMCs and perform the network configuration of the BMCs.

This section describes creating a node group to set the attributes that are common among nodes or based on regular expressions. For more information about xCAT regular expressions, see *Groups and Regular Expressions in Tables: Using Regular Expressions in the xCAT Tables* at the xCAT documentation website.

To define a node group, complete the following steps:

1. Create the `s822lc` node group by using the **`mkdef`** command:

```
$ mkdef -t group s822lc \
  ip='|p8r(\d+)n(\d+)|10.1.($1+0).($2+0)|' \
  bmc='|p8r(\d+)n(\d+)|10.2.($1+0).($2+0)|' \
  mgt=ipmi \
  cons=ipmi
Warning: Cannot determine a member list for group 's822lc'.
1 object definitions have been created or modified.
```

2. Verify the node group by using the **`lsdef`** command:

```
$ lsdef -t group s822lc
Object name: s822lc
    bmc=|p8r(\d+)n(\d+)|10.2.($1+0).($2+0)|
    cons=ipmi
    grouptype=static
    ip=|p8r(\d+)n(\d+)|10.1.($1+0).($2+0)|
    members=
    mgt=ipmi
```

To create a node that is part of the node group, complete the following steps:

1. Create a node by using the **`mkdef`** command, and include the node group in the `groups` attribute. You can also modify a node and make it part of a group by using the **`chdef`** command:

```
$ mkdef p8r1n1 groups=all,s822lc
1 object definitions have been created or modified.
```

   To create many nodes at the same time, you can use a node range, as shown in the following example:

```
$ mkdef p8r[1-5]n[1-6] groups=all,s822lc
30 object definitions have been created or modified.
```

2. Verify that the node inherits the node group's attributes by using the **`lsdef`** command. Notice that the attributes that are based on regular expressions are evaluated according to the name of the node. Some other attributes are set by xCAT by default:

```
$ lsdef p8r1n1
Object name: p8r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,s822lc
    ip=10.1.1.1
    mgt=ipmi
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
```

To set the `mtm` and `serial` attributes to match the BMC object, complete the following steps:

1. Set the attributes by using the **`chdef`** command. You can also set the attributes when creating the node object by using the **`mkdef`** command:

```
$ chdef p8r1n1 mtm=8335-GTB serial=0000000000000000
1 object definitions have been created or modified.
```

> **Note:** The `mtm` and `serial` attributes are case-sensitive.

2. Verify the attributes by using the **lsdef** command:

```
$ lsdef p8r1n1
Object name: p8r1n1
    bmc=10.2.1.1
    cons=ipmi
    groups=all,s822lc
    ip=10.1.1.1
    mgt=ipmig
    mtm=8335-GTB
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
```

3. Enable the `bmcsetup` script for all nodes in the `s822lc` group to set the BMC IP to static on first boot:

```
$ chdef -t group s822lc chain="runcmd=bmcsetup"
```

5.5.6 Configuring host table, DNS, and DHCP servers

The xCAT requires the node objects to be present and up-to-date in configuration files for the host table, DNS server, and DHCP server.

Note: This step is important for the node discovery process, which otherwise can show errors that are difficult to trace to specific misconfiguration steps.

The configuration files must be updated after the following changes are made, which are reflected in the configuration files:

- Adding or removing node objects
- Adding, modifying, or removing host names, IP addresses, or aliases for network interfaces

The order of the commands is relevant, as some commands depend on changes that are performed by other commands. For more information, see the manual pages of the **makehosts**, **makedns**, and **makedhcp** commands:

```
$ man makehosts
$ man makedns
$ man makedhcp
```

To update the configuration files with the node objects, complete the following steps:

1. Update the host table by using the **makehosts** command.

```
$ makehosts s822lc
```

2. Verify that the node objects are present on the host table:

```
$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.1.0.1    xcat-mn.xcat-cluster xcat-mn
10.1.1.1    p8r1n1 p8r1n1.xcat-cluster
```

3. Update the DNS server configuration by using the **makedns** command:

```
$ makedns -n s822lc
Handling p8r1n1 in /etc/hosts.
Getting reverse zones, this  take several minutes for a large cluster.
```

```
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this  take several minutes for a large cluster.
Completed updating DNS records.
```

4. Verify that the DNS server can resolve the name of the node by using the **host** command:

```
$ host p8r1n1 10.1.0.1
Using domain server:
Name: 10.1.0.1
Address: 10.1.0.1#53
Aliases:

p8r1n1.xcat-cluster has address 10.1.1.1
```

5. Update the DHCP server's configuration file by using the **makedhcp** command:

```
$ makedhcp -n
```

6. Update the DHCP server's leases file by using the **makedhcp** command:

```
$ makedhcp -a
```

5.5.7 Booting into Node discovery

Finally, you can boot the nodes into node discovery with power on (or cycle) if the boot order configuration is correct. For more information, see 5.2.2, "Boot order configuration" on page 196.

You can watch the progress of the node discovery process in the `/var/log/messages` file, which is described with comments in Example 5-15.

Example 5-15 Contents and comments for the `/var/log/messages` file during node discovery

```
$ tail -f /var/log/messages
...

Petitboot (DHCP client acquires an IP address, and releases it before booting):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPOFFER on 10.1.254.4 to 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPREQUEST for 10.1.254.4 (10.1.0.1) from 98:be:94:59:f0:f2 via
enp1s0f0
... dhcpd: DHCPACK on 10.1.254.4 to 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPRELEASE of 10.1.254.4 from 98:be:94:59:f0:f2 via enp1s0f0 (found)

Genesis (DHCP client acquires an IP address):

... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPOFFER on 10.1.254.5 to 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPREQUEST for 10.1.254.5 (10.1.0.1) from 98:be:94:59:f0:f2 via
enp1s0f0
... dhcpd: DHCPACK on 10.1.254.5 to 98:be:94:59:f0:f2 via enp1s0f0

Genesis (Communication with the xCAT Management Node; some error messages and
duplicated steps are apparently OK):
```

```
... xcat[30280]: xCAT: Allowing getcredentials x509cert
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: Discovery Error: Could not find any node.
... xcat[17646]: xcatd: Processing discovery request from 10.1.254.5

Genesis (The respective BMC object is identified, used for configuring the BMC
according to the node object, and then removed):

... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gtb-0000000000000000.
... xcat[39159]: xCAT: Allowing rspconfig to node-8335-gtb-0000000000000000
password= for root from localhost
... xcat[39168]: xCAT: Allowing chdef node-8335-gtb-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gtb-0000000000000000.
... xcat[39175]: xCAT: Allowing rspconfig to node-8335-gtb-0000000000000000
ip=10.2.1.1 for root from localhost
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gtb-0000000000000000.
... xcat[39184]: xCAT: Allowing rmdef node-8335-gtb-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: p8r1n1 has been
discovered
... xcat[17646]: Discovery info: configure password for
pbmc_node:node-8335-gtb-0000000000000000.
... xcat[39217]: xCAT: Allowing chdef node-8335-gtb-0000000000000000 bmcusername=
bmcpassword= for root from localhost
... xcat[17646]: Discover info: configure ip:10.2.1.1 for
pbmc_node:node-8335-gtb-0000000000000000.
... xcat[17646]: Discovery info: remove pbmc_node:node-8335-gtb-0000000000000000.

Genesis (DHCP client releases the temporary IP address, and acquires the permanent
IP address):

... dhcpd: DHCPRELEASE of 10.1.254.5 from 98:be:94:59:f0:f2 via enp1s0f0 (found)
... dhcpd: DHCPDISCOVER from 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPOFFER on 10.1.1.1 to 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPREQUEST for 10.1.1.1 (10.1.0.1) from 98:be:94:59:f0:f2 via enp1s0f0
... dhcpd: DHCPACK on 10.1.1.1 to 98:be:94:59:f0:f2 via enp1s0f0

Genesis (Cleanup of the BMC discovery):

... xcat[39230]: xCAT: Allowing rmdef node-8335-gtb-0000000000000000 for root from
localhost
... xcatd: Discovery worker: fsp instance: nodediscover instance: Failed to notify
10.1.254.5 that it's actually p8r1n1.

Genesis (Further communication with the xCAT Management Node):

... xcat[39233]: xCAT: Allowing getcredentials x509cert from p8r1n1
... xcat: credentials: sending x509cert
```

The Genesis image remains waiting for further instructions from the xCAT Management Node and is accessible through SSH.

Note: During the BMC network configuration steps, network connectivity can be lost (including the IPv6 link-local address). In this case, reset the connection by using in-band IPMI with the **ipmitool** command that is included in the Genesis image. This limitation might be addressed in a future xCAT or firmware version.

The steps to reset the BMC by using in-band IPMI on the Genesis image and wait for the BMC to come back online are shown in Example 5-16.

Example 5-16 Resetting the BMC via in-band IPMI on the Genesis image

```
Reset the BMC via in-band IPMI on the Genesis image:

$ ssh p8r1n1 'ipmitool mc reset cold'

Wait for the BMC to come back online:
(This example is based on the link-local IPv6 address; you can also use the IPv4
address assigned on node discovery; e.g., ping 10.2.1.1)

$ while ! ping6 -c 1 -q fe80::72e2:84ff:fe14:254%enp1s0f3; do echo Waiting; done;
echo; echo Finished
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enp1s0f3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

Waiting
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enp1s0f3 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

<...>
Waiting
PING fe80::72e2:84ff:fe14:254%enp1s0f3(fe80::72e2:84ff:fe14:254) 56 data bytes

--- fe80::72e2:84ff:fe14:254%enp1s0f3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.606/0.606/0.606/0.000 ms

Finished
```

You can watch the node discovery on the node console through IPMI. The BMC IP address can change as a result of the process. Therefore, use the BMC IPv6 link-local address (which does not change) for that purpose (see Example 5-17).

Example 5-17 Node discovery on the console via IPMI with the rcons on ipmitool commands

```
With the rcons command:
(require initial configuration with the makeconservercf command):
$ makeconservercf
$ rcons node-8335-gtb-0000000000000000

With the ipmitol command (via IPv6 link-local address):
```

```
$ ipmitool -I lanplus -H fe80::72e2:84ff:fe14:254%enp1s0f3 -U ADMIN -P admin sol
activate
```

You can verify that the node object now contains attributes that are obtained during the node discovery process (for example, hardware characteristics), and other xCAT attributes with the **lsdef** command (see Example 5-18).

Example 5-18 Node object with attributes obtained during node discovery

```
$ lsdef p8r1n1
Object name: p8r1n1
    arch=ppc64
    bmc=10.2.1.1
    cons=ipmi
    cpucount=192
    cputype=POWER8 (raw), altivec supported
    disksize=sda:1000GB,sdb:1000GB
    groups=all,s822lc
    ip=10.1.1.1
    mac=98:be:94:59:f0:f2
    memory=261482MB
    mgt=ipmi
    mtm=8335-GTB
    netboot=petitboot
    nodetype=mp
    postbootscripts=otherpkgs
    postscripts=syslog,remoteshell,syncfiles
    serial=0000000000000000
    status=standingby
    statustime=11-25-2016 23:13:50
    supportedarchs=ppc64
```

5.6 xCAT Compute Nodes (stateless)

This section describes the deployment of an xCAT Compute Node with the IBM HPC software that is running on RHEL Server 7.3 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode on the IBM Power System S822LC server.

The steps that are described in this section cover the stateless (diskless) installation. The image includes only runtime libraries to keep the image size as small as possible. For more information about a stateful image with full compiler support, see 5.7, “xCAT Login Nodes (stateful)” on page 285.

5.6.1 Network interfaces

The network interface that is associated with the management network is known as a *primary network interface* (or adapter). The network interfaces that are associated with other networks are known as *secondary* (or additional) *network interfaces* (or adapters).

For more information, see the Configure Additional Network Interfaces - confignics xCAT documentation page.

Primary network interface

The primary network interface is the network interface that is connected to the Management Network. The following attributes are important for this network interface:

- `primarynic`

  This attribute identifies the primary network interface. Set it to `mac` to use the network interface with the MAC address specified by the `mac` attribute (collected during node discovery).

- `installnic`

  This attribute identifies the network interface that is used for OS installation, which is usually the primary network interface; therefore, it is recommended that it is set it to `mac`.

To set the attributes for the primary network interface to `mac`, complete the following steps:

1. Set the `primarynic` and `installnic` by using the **`chdef`** command:

```
$ chdef -t group s822lc \
    installnic=mac \
    primarynic=mac
1 object definitions have been created or modified.
```

2. Verify the attributes by using the **`lsdef`** command:

```
$ lsdef -t group s822lc -i installnic,primarynic
Object name: s822lc
    installnic=mac
    primarynic=mac
```

Secondary network interfaces

The xCAT uses the information from the `nics` table to configure the network interfaces in the nodes to be part of the xCAT networks that are defined in the `networks` table. For example, the following attributes are used:

- `nicips` for IP addresses
- `nicnetworks` for xCAT networks (defined in the `networks` table)
- `nictypes` for the type of networks (for example, Ethernet or InfiniBand)
- (optional) `nichostnamesuffixes` for appending per-network suffixes to host names

The attribute format for the `nics` table uses several types of field separators, which allows for each field to relate to multiple network interfaces with multiple values per network interface (for example, IP addresses and host names). The format is a comma-separated list of `interface!values` pairs (that is, one pair per network interface), where *`values`* is a pipe-separated list of values (that is, all values assigned to that network interface).

For values with regular expressions, include the xCAT regular expression delimiters or pattern around the value (that is, leading pipe, regular expression pattern, separator pipe, value, and trailing pipe).

Consider the following example:

- Two network interfaces: `eth0` and `eth1`
- Two IP addresses each (`eth0` with `192.168.0.1` and `192.168.0.2`; `eth1` with `192.168.0.3` and `192.168.0.4`):

```
nicips='eth0!192.168.0.1|192.168.0.2,eth1!192.168.0.3|192.168.0.4'
```

- Two host name suffixes each (`eth0` with `-port0ip1` and `-port0ip2`; `eth1` with `-port1ip1` and `-port1ip2`):

  ```
  nichostnamesuffixes='eth0!-port0ip1|-port0ip2,eth1!-port1ip1|-port1ip2'
  ```

To configure the InfiniBand network interfaces of the compute nodes, complete the following steps:

1. Set the attributes of the nics table in the node group by using the **`chdef`** command:

   ```
   $ chdef -t group s822lc \
       nictypes='ib0!Infiniband' \
       nicnetworks='ib0!net-app-ib' \
       nichostnamesuffixes='ib0!-ib' \
       nicips='|p8r(\d+)n(\d+)|ib0!10.10.($1+0).($2+0)|'
   1 object definitions have been created or modified.
   ```

   Note: The following definition results in the same IP:

   ```
   nicips='|ib0!10.10.($2+0).($3+0)|'
   ```

2. Verify the attributes by using the **`lsdef`** command. They are represented with per-interface subattributes (that is, `<attribute>.<interface>=<values-for-the-interface>`):

   ```
   $ lsdef --nics -t group s822lc
   Object name: s822lc
       nichostnamesuffixes.ib0=-ib
       nicips.ib0=10.10.($1+0).($2+0)
       nicnetworks.ib0=net-app-ib
       nictypes.ib0=Infiniband
   ```

   Note: To view the full nicips value, `--verbose` must be added to the **`lsdef`** command.

3. Verify that the attributes that are based on regular expressions have correct values on a particular node by using the **`lsdef`** command:

   ```
   lsdef --nics p8r1n1
   Object name: p8r1n1
       nichostnamesuffixes.ib0=-ib
       nicips.ib0=10.10.1.1
       nicnetworks.ib0=net-app-ib
       nictypes.ib0=Infiniband
   ```

4. Update the configuration files for the host table, DNS server, and DHCP server by using the **`makehosts`**, **`makedns`**, and **`makedhcp`** commands:

   ```
   $ makehosts s822lc

   $ cat /etc/hosts
   127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
   ::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
   10.1.0.1    xcat-mn.xcat-cluster xcat-mn
   10.1.1.1    p8r1n1 p8r1n1.xcat-cluster
   10.10.1.1   p8r1n1-ib p8r1n1-ib.xcat-cluster
   ```

Note: If a network interface prefix or suffix is renamed or removed, you can update the host table by removing the node entry (or nodes or group entries), and adding it (them) back by using the following commands:

```
$ makehosts -d nodes
$ makehosts nodes
```

```
$ makedns -n s822lc
Handling p8r1n1 in /etc/hosts.
Handling p8r1n1-ib in /etc/hosts.
Getting reverse zones, this  take several minutes for a large cluster.
Completed getting reverse zones.
Updating zones.
Completed updating zones.
Restarting named
Restarting named complete
Updating DNS records, this  take several minutes for a large cluster.
Completed updating DNS records.

$ makedhcp -n
$ makedhcp -a
```

5. Add the `confignics` script to the list of postscripts for configuring the network interfaces. The InfiniBand network interfaces (2-port adapter) require the argument `--ibaports=2`:

```
$ chdef -t group s822lc --plus postscripts='confignics --ibaports=2'
1 object definitions have been created or modified.

$ lsdef -t group s822lc -i postscripts
Object name: s822lc
    postscripts=confignics --ibaports=2
```

   For more information about the `confignics` script and the configuration of InfiniBand adapters, see the following xCAT documentation pages:

   – Configure Additional Network Interfaces - confignics
   – IB Network Configuration

Public or site network connectivity (optional)

The connectivity to the public or site networks for the compute nodes can be provided by using one of the following methods:

- By way of the Management Node by using network address translation (NAT).
- By way of Compute Nodes by using another xCAT network.

To perform the required network configuration, follow the steps of either method.

Management Node method

This method requires that the `gateway` attribute of the xCAT Management Network is set to the Management node (default setting, with value "`<xcatmaster>`"), and NAT rules be configured in the firewall. To connect, complete the following steps:

1. Verify that the `gateway` attribute of the Management Network is set to `<xcatmaster>` by using the **`lsdef`** command:

```
$ lsdef -t network net-mgmt -i gateway
Object name: net-mgmt
    gateway=<xcatmaster>
```

If not, set it by using the **chdef** command:

```
$ chdef -t network net-mgmt gateway='<xcatmaster>'
```

2. Verify the `routers` option of the DHCP server configuration file (based on the `gateway` attribute) in the Management Network reflects the IP address of the Management Node:

```
$ grep 'subnet\|routers' /etc/dhcp/dhcpd.conf
  subnet 10.1.0.0 netmask 255.255.0.0 {
    option routers  10.1.0.1;
  } # 10.1.0.0/255.255.0.0 subnet_end
  subnet 10.2.0.0 netmask 255.255.0.0 {
    option routers  10.2.0.1;
  } # 10.2.0.0/255.255.0.0 subnet_end
```

If not, regenerate the DHCP server configuration files by using the **makedhcp** command:

```
$ makedhcp -n
$ makedhcp -a
```

3. Verify the default route on the compute nodes is set to the IP address of the Management Node by using the **ip** command:

```
$ xdsh p8r1n1 'ip route show | grep default'
p8r1n1: default via 10.1.0.1 dev enp1s0f0
```

If not, restart the network service by using the **systemctl** command:

```
$ xdsh p8r1n1 'systemctl restart network'
```

4. Configure the `iptables` firewall rules for NAT in the `rc.local` script. Configure it to start automatically on boot and start the service manually this time.

Consider the following points regarding the network scenario in this chapter:

- Management network on network interface `enp1s0f0` is in the management node.
- Public/site network on network interface `enp1s0f2` is in the management node.

```
$ cat <<EOF >>/etc/rc.d/rc.local
iptables -t nat --append POSTROUTING --out-interface enp1s0f2 -j MASQUERADE
iptables --append FORWARD --in-interface enp1s0f0 -j ACCEPT
EOF

$ chmod +x /etc/rc.d/rc.local
$ systemctl start rc-local
```

Note: The default firewall in RHEL Server 7.3 is `firewalld`, which is disabled by xCAT.

For more information about firewalld, see the Using Firewalls RHEL documentation page.

5. You can verify the network connectivity on a running node (if any) by using the **ping** command:

```
$ xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=4.07 ms
<...>
```

Compute Nodes method

This method requires the `gateway` attribute of the xCAT Management Network *not* to be set, and another xCAT network (for the public or site network) to define the network interface, address, and gateway to be used. Complete the following steps:

1. Reset the `gateway` attribute of the Management Network by using the **`chdef`** command:

```
$ chdef -t network net-mgmt gateway=''

$ lsdef -t network net-mgmt -i gateway
Object name: net-mgmt
    gateway=
```

2. Define a new xCAT network for the public or site network (for example, `net-site`):

```
$ mkdef -t network net-site net=9.x.x.x mask=255.255.255.0
1 object definitions have been created or modified.

# lsdef -t network
net-app-ib (network)
net-mgmt  (network)
net-site  (network)
net-svc  (network)

$ lsdef -t network net-site
Object name: net-site
    mask=255.255.255.0
    net=9.x.x.x
```

3. Modify the `nics` table to include the attributes of the new xCAT network by using the **`tabedit`** command or the **`chdef`** command. The network scenario that is described in this chapter uses a public or site network on network interface `enp1s0f1` in the compute nodes:

```
$ tabedit nics

OR:

$ chdef -t group s822lc \
    nictypes='enp1s0f1!Ethernet,ib0!Infiniband' \
    nicnetworks='enp1s0f1!net-site,ib0!net-app-ib' \
    nichostnamesuffixes='enp1s0f1!-site,ib0!-ib' \
    nicips='|p8r(\d+)n(\d+)|enp1s0f1!9.x.x.(181-$2),ib0!10.10.($1+0).($2+0)|' \
    nicextraparams='enp1s0f1!GATEWAY=9.x.x.254'
1 object definitions have been created or modified.
```

4. Verify the attributes by using the **`tabdump`** command or **`lsdef`** command:

```
$ tabdump nics
#node,nicips,nichostnamesuffixes,nichostnameprefixes,nictypes,niccustomscripts,
nicnetworks,nicaliases,nicextraparams,comments,disable
"s822lc","|p8r(\d+)n(\d+)|enp1s0f1!9.x.x.(181-$2),ib0!10.10.($1+0).($2+0)|","en
p1s0f1!-site,ib0!-ib","enp1s0f1!Ethernet,ib0!Infiniband","enp1s0f1!net-site,i
b0!net-app-ib",,"enp1s0f1!GATEWAY=9.x.x.254",,

$ lsdef --nics -t group s822lc
Object name: s822lc
<...>
    nicextraparams.enp1s0f1=GATEWAY=9.x.x.254
<...>
```

```
    nichostnamesuffixes.enp1s0f1=-site
<...>
    nicips.enp1s0f0=9.x.x.(181-$2)
<...>
    nicnetworks.enp1s0f1=net-site
<...>
    nictypes.enp1s0f1=Ethernet

$ lsdef --nics p8r1n1
Object name: p8r1n1
<...>
    nicextraparams.enp1s0f1=GATEWAY=9.x.x.254
<...>
    nichostnamesuffixes.enp1s0f1=-site
<...>
    nicips.enp1s0f1=9.x.x.180
<...>
    nicnetworks.enp1s0f1=net-site
<...>
    nictypes.enp1s0f1=Ethernet
```

5. Update the host table by using the **makehosts** command:

```
$ makehosts -d s822lc
$ makehosts s822lc

$ cat /etc/hosts
<...>
10.1.1.1 p8r1n1 p8r1n1.xcat-cluster
10.10.1.1 p8r1n1-ib p8r1n1-ib.xcat-cluster
9.x.x.180 p8r1n1-site p8r1n1-site.xcat-cluster
<...>
```

6. Update the DNS server configuration by using the **makedns** command:

```
$ makedns -n s822lc
<...>
Handling p8r1n1-site in /etc/hosts.
<...>
Completed updating DNS records.
```

7. Update the DHCP server configuration by using the **makedhcp** command:

```
$ makedhcp -n
$ makedhcp -a
```

8. You can update the network configuration on a running node (if any) by using the **confignics** script of the **updatenode** command:

```
$ updatenode p8r1n1 -P confignics
```

9. You can verify the network connectivity on a running node (if any) by using the **ping** command:

```
$ xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.6.2 RHEL server

The xCAT stores the configuration for installing OSs in objects of type `osimage` (that is, *operating system image*). The **copycds** command can be used to create `osimage` objects that are based on an OS installation disk image (for example, an ISO file).

Setting the password for the root user

To set the root password, complete the following steps:

1. Set the `username` and `password` attributes of the `system` key or row in the `passwd` table by using the **chtab** command:

   ```
   $ chtab key=system passwd.username=root passwd.password=cluster
   ```

2. Verify the attributes by using the **tabdump** command:

   ```
   $ tabdump -w key==system passwd
   #key,username,password,cryptmethod,authdomain,comments,disable
   "system","root","cluster",,,,
   ```

   Note: If the attributes are not correctly set, the **nodeset** command shows the following error message:

   ```
   # nodeset p8r1n1 osimage=rh73-compute-stateless
   p8r1n1: Error: Unable to find requested filed <password> from table
   <passwd>, with key <key=system,username=root>
   Error: Some nodes failed to set up install resources on server
   xcat-mn.xcat-cluster, aborting
   ```

Creating an osimage object

Initially, no `osimage` objects are available:

```
$ lsdef -t osimage
Could not find any object definitions to display.
```

To create `osimage` objects for the RHEL Server 7.3 installation disk image, complete the following steps:

1. Run the **copycds** command on the RHEL Server 7.3 ISO file:

   ```
   $ copycds /mnt/RHEL-7.3-20161019.0-Server-ppc64le-dvd1.iso
   Copying media to /install/rhels7.3/ppc64le
   Media copy operation successful
   ```

2. Verify that the `osimage` objects are present by using the **lsdef** command.

   ```
   $ lsdef -t osimage
   rhels7.3-ppc64le-install-compute  (osimage)
   rhels7.3-ppc64le-install-service  (osimage)
   rhels7.3-ppc64le-netboot-compute  (osimage)
   rhels7.3-ppc64le-stateful-mgmtnode  (osimage)
   ```

   For more information about the `osimage` objects, see 5.3.6, “xCAT OS installation types: Disks and state” on page 205.

3. Create a copy of the original `osimage` object, named `rh73-compute-stateless`.

   It is optional, but useful in case multiple `osimage` objects are maintained (for example, for multiple or different configurations of the same OS).

You can use the **lsdef -z** command, which provides the object stanza, modify it (for example, by using the **sed** command), and create an object that is based on it by using the **mkdef -z** command:

```
$ osimage=rh73-compute-stateless

$ lsdef -t osimage rhels7.3-ppc64le-netboot-compute -z \
| sed "s/^[^# ].*:/$osimage:/" \
| mkdef -z
1 object definitions have been created or modified.
```

4. Verify the copy `osimage` object by using the **lsdef** command:

```
$ lsdef -t osimage
rh73-compute-stateless  (osimage)
rhels7.3-ppc64le-install-compute  (osimage)
rhels7.3-ppc64le-install-service  (osimage)
rhels7.3-ppc64le-netboot-compute  (osimage)
rhels7.3-ppc64le-stateful-mgmtnode  (osimage)
```

5. To view more information about the new `osimage`, append the name of the object to the command in the previous step, as shown in Example 5-19.

Example 5-19 Detailed osimage information

```
lsdef -t osimage $osimage
Object name: rh73-compute-stateless
    exlist=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.exlist
    imagetype=linux
    osarch=ppc64le
    osdistroname=rhels7.3-ppc64le
    osname=Linux
    osvers=rhels7.3
    otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
    pkgdir=/install/rhels7.3/ppc64le
    pkglist=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist

postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.postinstall
    profile=compute
    provmethod=netboot
    rootimgdir=/install/netboot/rhels7.3/ppc64le/compute
```

Note: The OS can be installed (without other components of the software stack) by using the **nodeset** command:

```
$ nodeset p8r1n1 osimage=rh73-compute-stateless
```

Changing the `pkglist` attribute

The use of multiple package lists is convenient for independently organizing the required packages for each component of the software stack. Although, the xCAT currently does not support multiple package lists in the `pkglist` attribute, it does support a package list to reference the contents of other package lists. This feature provides a way to achieve multiple package lists.

The following types of entries can be used in the `pkglist` file:

- RPM name without version numbers
- group/pattern name marked with a "`@`" (for stateful install only)

- RPMs to removed after the installation marked with a “-” (for stateful install only)
- `#INCLUDE: <full file path>#` to include other pkglist files
- `#NEW_INSTALL_LIST#` to signify that the following rpms will be installed with a new rpm install command (zypper, yum, or rpm as determined by the function that uses this file)

To change the `pkglist` attribute for a custom package list, complete the following steps:

1. Verify the current `pkglist` attribute, and assign it to the `old_list` variable:

```
$ lsdef -t osimage rh73-compute-stateless -i pkglist
Object name: rh73-compute-stateless
    pkglist=/install/custom/netboot/rhel/rh73-compute.pkglist

$ old_pkglist=/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist
```

2. In this example, we include the default RHEL Server 7.3 `pkglist` and add the `numactl` package to it. Create the list that includes the old list. Then, add `numactl`:

```
$ new_pkglist=/install/custom/netboot/rh/rh73-compute.pkglist

$ mkdir -p $(dirname $new_pkglist)
$ echo "# RHEL Server 7.3 (original pkglist)" >> $new_pkglist
$ echo "#INCLUDE:${old_pkglist}#" >> $new_pkglist
$ echo "numactl" >> $new_pkglist
```

> **Note:** All of your custom files must be in the `/install/custom` directory for easier distinction.

3. Verify the contents of the new list. The trailing “#” character at the end of the line is important for correct `pkglist` parsing:

```
$ cat $new_pkglist
# RHEL Server 7.3 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
numactl
```

4. Change the `pkglist` attribute to the new list by using the **`chdef`** command:

```
$ chdef -t osimage rh73-compute-stateless pkglist=$new_pkglist
1 object definitions have been created or modified.
```

5. Verify the `pkglist` attribute by using the **`lsdef`** command:

```
$ lsdef -t osimage rh73-compute-stateless -i pkglist
Object name: rh73-compute-stateless
    pkglist=/install/custom/netboot/rhel/rh73-compute.pkglist
```

Changing the `rootimgdir` attribute

Each stateless image must have its own `rootimgdir`. The image is stored in the `rootimgdir=/install/netboot/rhels7.3/ppc64le/rh73-compute-stateless` directory. It is recommended to match the last level directory name with the name of the `osimage`.

Change the `rootimgdir` to `rh73-compute-stateless`:

```
$ lsdef -t osimage rh73-compute-stateless -i rootimgdir
Object name: rh73-compute-stateless
    rootimgdir=/install/netboot/rhels7.3/ppc64le/compute

$ chdef -t osimage rh73-compute-stateless
rootimgdir=/install/netboot/rhels7.3/ppc64le/rh73-compute-stateless
```

```
1 object definitions have been created or modified.
```

5.6.3 CUDA Toolkit

The CUDA Toolkit can be installed with the RPM packages that are contained in the local repository package that is available for download in the NVIDIA website. The packages are marked for installation with the `otherpkgdir` and `otherpkglist` mechanism to be installed during the Linux distribution installation.

Note: For stateless, installing the CUDA packages *must* be done in the `otherpkglist` and *not* the `pkglist` as with stateful.

For more information, see the xCAT RHEL 7.2 LE documentation page.

To install the CUDA Toolkit, complete the following steps:

1. Install the `createrepo` package for creating package repositories:

   ```
   $ yum install createrepo
   ```

2. Download and extract the CUDA Toolkit RPM package:

   Note: The CUDA Toolkit is available for download at the NVIDIA CUDA downloads website.

   At the website, click **(Operating Systems) Linux** → **(Architecture) ppc64le** → **(Distribution) RHEL** → **(Version) 7** → **(Installer Type) rpm (local)** → **Download**.

   ```
   $ dir=/tmp/cuda
   $ mkdir -p $dir
   $ cd $dir

   $ curl -sOL https://.../cuda-repo-rhel7-8-0-local-8.0.54-1.ppc64le.rpm
   $ rpm2cpio cuda-repo-rhel7-8-0-local-8.0.54-1.ppc64le.rpm | cpio -id
   2430295 blocks
   ```

3. Copy the extracted package repository to a new `/install/software` directory:

   Note: All other software packages are in the central `/install/software` directory.

   ```
   $ dir_cuda=/install/software/cuda/8.0
   $ mkdir -p $dir_cuda

   $ ls -1d $dir_cuda/*
   /install/software/cuda/8.0/cuda-8.0-54-1.ppc64le.rpm
   <...>
   /install/software/cuda/8.0/repodata
   <...>

   $ cd ..
   $ rm -r /tmp/cuda
   ```

4. The CUDA Toolkit contains RPMs that depend on the `dkms` package. This `dkms` package is provided by Extra Packages for Enterprise Linux (EPEL). For more information about EPEL, see fedora's EPEL wiki page. Download the `dkms` RPM package from EPEL:

```
$ dir_deps=/install/software/cuda/deps
$ mkdir -p $dir_deps
$ cd $dir_deps

$ epel_url="https://dl.fedoraproject.org/pub/epel/7/ppc64le"
$ dkms_path="${epel_url}/d/$(curl -sL ${epel_url}/d | grep -o
'href="dkms-.*\.noarch\.rpm"' | cut -d '"' -f 2)"
$ curl -sOL "$dkms_path"

$ ls -1
dkms-2.2.0.3-34.git.9e0394d.el7.noarch.rpm
```

5. Create a package repository for it by using the **createrepo** command:

```
$ createrepo .
<...>

$ ls -1
dkms-2.2.0.3-34.git.9e0394d.el7.noarch.rpm
repodata
```

6. Create symlinks from the `otherpkglist` directory to software directory:

```
$ lsdef -t osimage rh73-compute-stateless -i otherpkgdir
Object name: rh73-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le

$ otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
$ cd $otherpkgdir

$ ln -s $dir_cuda cuda-8.0
$ ln -s $dir_deps cuda-deps

$ ls -l
lrwxrwxrwx 1 root root 26 22. Nov 15:55 cuda-8.0 -> /install/software/cuda/8.0
lrwxrwxrwx 1 root root 27 22. Nov 15:55 cuda-deps ->
/install/software/cuda/deps
```

7. Create the `otherpkglist` file with relative paths from `otherpkgdir` to `cuda-runtime` and `dkms` package:

```
$ lsdef -t osimage rh73-compute-stateless -i pkglist,otherpkgdir,otherpkglist
Object name: rh73-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
    otherpkglist=
    pkglist=/install/custom/netboot/rhel/rh73-compute.pkglist

$ otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
$ otherpkglist=/install/custom/netboot/rhel/rh73-compute.otherpkglist

$ cd $otherpkgdir; find -L * -type f \( -name "cuda-runtime*" -o -name "dkms*"
\)
cuda-8.0/cuda-runtime-8-0-8.0.54-1.ppc64le.rpm
cuda-deps/dkms-2.2.0.3-34.git.9e0394d.el7.noarch.rpm
```

```
$ cat <<EOF >>$otherpkglist
$ CUDA
cuda-deps/dkms
cuda-8.0/cuda-runtime-8-0
EOF
```

8. Set the `otherpkglist` attribute for the `osimage`:

```
$ chdef -t osimage rh73-compute-stateless --plus otherpkglist=$otherpkglist
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-compute-stateless -i otherpkgdir,otherpkglist
Object name: rh73-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
    otherpkglist=/install/custom/netboot/rhel/rh73-compute.otherpkglist
```

5.6.4 Mellanox OFED

To install the Mellanox OFED for Linux, complete the following steps.

For more information, see the following xCAT documentation resources:

- Mellanox OFED Installation Script - Preparation page
- IB Network Configuration page

1. Download and copy the ISO file to the xCAT installation directory:

> **Note:** The Mellanox OFED is available for download from the Mellanox downloads website.
>
> At the website, click **IBM HPC and Technical Clusters** → **8335-GTB** → **MLNX_OFED_LINUX-3.4-2.0.0.1-rhel7.3-ppc64le.iso**
>
> Because you must accept a user agreement, the package must be downloaded by using a browser.

```
$ dir=/install/software/mofed
$ mkdir -p $dir

$ Download via browser, then scp to xcat management server
$ scp MLNX_OFED_LINUX-3.4-2.0.0.1-rhel7.3-ppc64le.iso
xcat-mn:/install/software/mofed
```

2. Copy the `mlnxofed_ib_install.v2` script into the `postscripts` directory (with the required file name `mlnxofed_ib_install`). This script is a sample script intended to assist with the installation of the Mellanox OFED drivers:

```
$ script=/install/postscripts/mlnxofed_ib_install
$ cp /opt/xcat/share/xcat/ib/scripts/Mellanox/mlnxofed_ib_install.v2 $script
$ chmod +x $script
$ ls -l $script
-rwxr-xr-x 1 root root 16269 22. Nov 10:51
/install/postscripts/mlnxofed_ib_install
```

3. Create a custom `postinstall` script that runs `mlnxofed_ib_install` and include it in the `osimage` definition.
4. Include the `--add-kernel-support` argument to the list of default arguments (`--without-32bit --without-fw-update --force`), according to the xCAT documentation to rebuild kernel modules for the installed kernel version.

   Also, clear the `/tmp` directory after successful installation:

```
$ file=MLNX_OFED_LINUX-3.4-2.0.0.1-rhel7.3-ppc64le.iso
$ args='--add-kernel-support --without-32bit --without-fw-update --force'

$ lsdef -t osimage rh73-compute-stateless -i postinstall
Object name: rh73-compute-stateless

postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.postinstall
$
postinstall=/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.postinstall
$ new_postinstall=/install/custom/netboot/rhel/rh73-compute.postinstall
$ cp $postinstall $new_postinstall

$ echo "# Install InfiniBand MOFED" >> $new_postinstall
$ echo "/install/postscripts/mlnxofed_ib_install -p $dir/$file -m $args -end-
-i \$installroot -n genimage" >> $new_postinstall
$ echo "rm -rf \$installroot/tmp/MLNX_OFED_LINUX*" >> $new_postinstall

$ cat $new_postinstall
   <...>
   # Install InfiniBand MOFED
   /install/postscripts/mlnxofed_ib_install -p
   /install/software/mofed/MLNX_OFED_LINUX-3.4-2.0.0.1-rhel7.3-ppc64le.iso -m
   --add-kernel-support --without-32bit --without-fw-update --force -end- -i
   $installroot -n genimage
   rm -rf $installroot/tmp/MLNX_OFED_LINUX*

$ chdef -t osimage rh73-compute-stateless postinstall=$new_postinstall
1 object definitions have been created or modified.
$ lsdef -t osimage rh73-compute-stateless -i postinstall
Object name: rh73-compute-stateless
    postinstall=/install/custom/netboot/rhel/rh73-compute.postinstall
```

   Note: The `-i $installroot` argument is the installation root directory and is passed to the postinstallation script at run time as `$1`.

5. Certain dependency packages are required to install Mellanox OFED correctly. Add the InfiniBand package list that is provided by xCAT to the package list:

```
$ ib_list=/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist

$ lsdef -t osimage rh73-compute-stateless -i pkglist
Object name: rh73-compute-stateless
    pkglist=/install/custom/netboot/rhel/rh73-compute.pkglist
$ pkglist=/install/custom/netboot/rhel/rh73-compute.pkglist
$ cat $pkglist
   # RHEL Server 7.3 (original pkglist)
   #INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
   numactl
```

```
$ echo "# InfiniBand MOFED Dependencies (sample pkglist)" >> $pkglist
$ echo "#INCLUDE:${ib_list}#" >> $pkglist

$ cat $pkglist
   # RHEL Server 7.3 (original pkglist)
   #INCLUDE:/opt/xcat/share/xcat/netboot/rh/compute.rhels7.ppc64le.pkglist#
   numactl
   # InfiniBand MOFED Dependencies (sample pkglist)
   #INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#
```

5.6.5 XL C/C++ runtime libraries

To install the XL C/C++ Compiler xCAT software kit, complete the following steps. For more information, see the following xCAT IBM XL Compilers documentation page:

1. Download the partial kit (distributed by the xCAT project):

```
$ mkdir /tmp/xlc
$ cd /tmp/xlc

$ curl -sOL
https://xcat.org/files/kits/hpckits/2.12/rhels7.3/ppc64le/xlc-13.1.5-0-ppc64le.
NEED_PRODUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages by using the **buildkit** command:

```
$ dir=/install/software/compilers/xlc

$ ls -1 $dir
libxlc-13.1.5.0-161028a.ppc64le.rpm
libxlc-devel.13.1.5-13.1.5.0-161028a.ppc64le.rpm
libxlmass-devel.8.1.5-8.1.5.0-161021a.ppc64le.rpm
libxlsmp-4.1.5.0-161025.ppc64le.rpm
libxlsmp-devel.4.1.5-4.1.5.0-161025.ppc64le.rpm
xlc-13.1.5-0-ppc64le.tar.bz2
xlc.13.1.5-13.1.5.0-161028a.ppc64le.rpm
xlc.compiler-compute-13.1.5-0.noarch.rpm
xlc-license.13.1.5-13.1.5.0-161028a.ppc64le.rpm
xlc.license-compute-13.1.5-0.noarch.rpm
xlc.rte-compute-13.1.5-0.noarch.rpm


$ buildkit addpkgs xlc-13.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlc/xlc-13.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
Please wait.
<...>
Kit tar file /tmp/xlc/xlc-13.1.5-0-ppc64le.tar.bz2 successfully built.

$ ls -1
xlc-13.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlc-13.1.5-0-ppc64le.tar.bz2
```

3. Add the kit to xCAT by using the **addkit** command:

```
$ lsdef -t kit
Could not find any object definitions to display.
```

```
$ addkit xlc-13.1.5-0-ppc64le.tar.bz2
Adding Kit xlc-13.1.5-0-ppc64le
Kit xlc-13.1.5-0-ppc64le was successfully added.

$ lsdef -t kit
xlc-13.1.5-0-ppc64le  (kit)

$ cd ..
$ rm -r /tmp/xlc
```

4. Verify its `kitcomponent` objects (and `description` fields) by using the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==xlc-13.1.5-0-ppc64le -i description
Object name: xlc.compiler-compute-13.1.5-0-rhels-7-ppc64le
    description=XLC13 for compiler kitcomponent
Object name: xlc.license-compute-13.1.5-0-rhels-7-ppc64le
    description=XLC13 license kitcomponent
Object name: xlc.rte-compute-13.1.5-0-rhels-7-ppc64le
    description=XLC13 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **`addkitcomp`** command:

```
$ lsdef -t osimage rh73-compute-stateless -i kitcomponents,otherpkglist
Object name: rh73-compute-stateless
    kitcomponents=
    otherpkglist=/install/custom/netboot/rhel/rh73-compute.otherpkglist

$ addkitcomp --adddeps -i rh73-compute-stateless \
xlc.rte-compute-13.1.5-0-rhels-7-ppc64le
Assigning kit component xlc.rte-compute-13.1.5-0-rhels-7-ppc64le to osimage
rh73-compute-stateless
Kit components xlc.rte-compute-13.1.5-0-rhels-7-ppc64le were added to osimage
rh73-compute-stateless successfully

$ lsdef -t osimage rh73-compute-stateless -i kitcomponents,otherpkglist
Object name: rh73-compute-stateless

kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le

otherpkglist=/install/osimages/rh73-compute-stateless/kits/KIT_DEPLOY_PARAMS.ot
herpkgs.pkglist,/install/custom/netboot/rhel/rh73-compute.otherpkglist,/install
/osimages/rh73-compute-stateless/kits/KIT_COMPONENTS.otherpkgs.pkglist
```

Note: For more information about the use of a subset of compute nodes for compiling applications, see 5.7, “xCAT Login Nodes (stateful)” on page 285.

5.6.6 XL Fortran runtime libraries

To install the XL Fortran Compiler xCAT kit, complete the following steps. For more information, see the xCAT IBM XL Compilers documentation page:

1. Download the partial kit (distributed by the xCAT project):

```
$ mkdir /tmp/xlf
$ cd /tmp/xlf

$ curl -sOL
https://xcat.org/files/kits/hpckits/2.12/rhels7.3/ppc64le/xlf-15.1.5-0-ppc64le.
NEED_PRODUCT_PKGS.tar.bz2
```

2. Build the complete kit by combining the partial kit and the product packages that are distributed in the installation media by using the **buildkit** command:

```
$ dir=/install/software/compilers/xlf

$ ls -1 $dir
libxlf-15.1.5.0-161028a.ppc64le.rpm
libxlf-devel.15.1.5-15.1.5.0-161028a.ppc64le.rpm
libxlmass-devel.8.1.5-8.1.5.0-161021a.ppc64le.rpm
libxlsmp-4.1.5.0-161025.ppc64le.rpm
libxlsmp-devel.4.1.5-4.1.5.0-161025.ppc64le.rpm
xlf-15.1.5-0-ppc64le.tar.bz2
xlf.15.1.5-15.1.5.0-161028a.ppc64le.rpm
xlf.compiler-compute-15.1.5-0.noarch.rpm
xlf-license.15.1.5-15.1.5.0-161028a.ppc64le.rpm
xlf.license-compute-15.1.5-0.noarch.rpm
xlf.rte-compute-15.1.5-0.noarch.rpm

$ buildkit addpkgs xlf-15.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2 --pkgdir $dir
Extracting tar file /tmp/xlf/xlf-15.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2.
Please wait. <...>
Kit tar file /tmp/xlf/xlf-15.1.5-0-ppc64le.tar.bz2 successfully built.

$ ls -1
xlf-15.1.5-0-ppc64le.NEED_PRODUCT_PKGS.tar.bz2
xlf-15.1.5-0-ppc64le.tar.bz2
```

3. Add the kit to xCAT by using the **addkit** command:

```
$ addkit xlf-15.1.5-0-ppc64le.tar.bz2
Adding Kit xlf-15.1.5-0-ppc64le
Kit xlf-15.1.5-0-ppc64le was successfully added.

$ lsdef -t kit
xlc-13.1.5-0-ppc64le  (kit)
xlf-15.1.5-0-ppc64le  (kit)

$ cd ..
$ rm -r /tmp/xlf
```

4. Verify its `kitcomponent` objects (and `description` fields) by using the **lsdef** command:

```
$ lsdef -t kitcomponent -w kitname==xlf-15.1.5-0-ppc64le -i description
Object name: xlf.compiler-compute-15.1.5-0-rhels-7-ppc64le
    description=XLF15 for compiler kitcomponent
Object name: xlf.license-compute-15.1.5-0-rhels-7-ppc64le
```

```
    description=XLF15 license kitcomponent
Object name: xlf.rte-compute-15.1.5-0-rhels-7-ppc64le
    description=XLF15 for runtime kitcomponent
```

5. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **addkitcomp** command:

```
$ addkitcomp --adddeps -i rh73-compute-stateless \
xlf.rte-compute-15.1.5-0-rhels-7-ppc64le
Assigning kit component xlf.rte-compute-15.1.5-0-rhels-7-ppc64le to osimage
rh73-compute-stateless
Kit components xlf.rte-compute-15.1.5-0-rhels-7-ppc64le were added to osimage
rh73-compute-stateless successfully

$ lsdef -t osimage rh73-compute-stateless -i kitcomponents
Object name: rh73-compute-stateless

kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le,xlf.license-compute-15.1.5-0-rhels-7-ppc64le,xlf.rte-compu
te-15.1.5-0-rhels-7-ppc64le
```

5.6.7 Advance Toolchain runtime libraries

To install the Advance Toolchain, complete the following steps:

Note: For more information about the latest download links, see the Supported Linux Distributions section of the IBM Advance Toolchain for PowerLinux™ Documentation website.

1. Download the product packages:

```
$ dir=/install/software/compilers/at
$ mkdir -p $dir
$ cd $dir

$ wget
'ftp://ftp.unicamp.br/pub/linuxpatch/toolchain/at/redhat/RHEL7/at10.0/*-10.0-1.
ppc64le.rpm'
<...>

$ ls -1
advance-toolchain-at10.0-devel-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-devel-debuginfo-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-mcore-libs-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-mcore-libs-debuginfo-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-perf-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-perf-debuginfo-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-runtime-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-runtime-at9.0-compat-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-runtime-debuginfo-10.0-1.ppc64le.rpm
advance-toolchain-at10.0-selinux-10.0-1.ppc64le.rpm
advance-toolchain-golang-at-10.0-1.ppc64le.rpm
```

2. Create a package repository by using the **createrepo** command:

```
$ createrepo .
<...>
```

3. Add packages to `otherkglist` of `osimage`:

```
$ lsdef -t osimage rh73-compute-stateless -i otherpkglist,otherpkgdir
Object name: rh73-compute-stateless
    otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le

otherpkglist=/install/osimages/rh73-compute-stateless/kits/KIT_DEPLOY_PARAMS.ot
herpkgs.pkglist,/install/custom/netboot/rhel/rh73-compute.otherpkglist,/install
/osimages/rh73-compute-stateless/kits/KIT_COMPONENTS.otherpkgs.pkglist

$ otherpkgdir=/install/post/otherpkgs/rhels7.3/ppc64le
$ otherpkglist=/install/custom/netboot/rhel/rh73-compute.otherpkglist

$ cd $otherpkgdir
$ ln -s $dir at

$ ls -l
lrwxrwxrwx 1 root root 20 22. Nov 16:45 at -> /install/software/compilers/at
lrwxrwxrwx 1 root root 26 22. Nov 15:55 cuda-8.0 -> /install/software/cuda/8.0
lrwxrwxrwx 1 root root 27 22. Nov 15:55 cuda-deps ->
/install/software/cuda/deps
lrwxrwxrwx 1 root root 69 22. Nov 15:07 xlc-13.1.5-0-rhels-7-ppc64le ->
/install/kits/xlc-13.1.5-0-ppc64le/repos/xlc-13.1.5-0-rhels-7-ppc64le
lrwxrwxrwx 1 root root 69 22. Nov 15:24 xlf-15.1.5-0-rhels-7-ppc64le ->
/install/kits/xlf-15.1.5-0-ppc64le/repos/xlf-15.1.5-0-rhels-7-ppc64le

$ echo "# Advance Toolchain" >> $otherpkglist
$ echo "at/advance-toolchain-at10.0-runtime" >> $otherpkglist
$ echo "at/advance-toolchain-at10.0-mcore-libs" >> $otherpkglist

$ cat $otherpkglist
   # CUDA
   cuda-deps/dkms
   cuda-8.0/cuda-runtime-8-0
   # Advance Toolchain
   at/advance-toolchain-at10.0-runtime
   at/advance-toolchain-at10.0-mcore-libs
```

Note: Advance Toolchain is approximately 2 GB after installation. For stateless nodes, it is recommended to install this package in a shared file system to save main memory on the compute node.

5.6.8 PGI runtime libraries

To install PGI runtime libraries for OpenACC, complete the following steps:

Note: The PGI Compilers are available for download at the PGI Community Edition page of the PGI Compilers and Tools website.

At the website, click **Linux OpenPOWER** to begin the download process.

Because you must accept a user agreement, must download this package with a browser.

No PGI Compiler runtime libraries-only package are available. Therefore, libraries must be copied manually.

1. Download and copy downloaded package to management node:

```
$ dir=/install/software/compilers/pgi
$ mkdir -p $dir

# Download via browser, then scp to xcat management server
$ scp pgilinux-2016-1610-ppc64le.tar.gz xcat-mn:/install/software/compilers/pgi
```

2. Extract the package:

```
$ cd $dir
$ tar -xf pgilinux-2016-1610-ppc64le.tar.gz

$ ls -1
documentation.html
install
install_components
pgilinux-2016-1610-ppc64le.tar.gz

$ libdir=install_components/linuxpower/16.10/lib
$ ls -1 $dir/$libdir
$ ls -1 $dir/$libdir/*.so*
/install/software/compilers/pgi/install_components/linuxpower/16.10/lib/libacca
pimp.so
/install/software/compilers/pgi/install_components/linuxpower/16.10/lib/libacca
pi.so
<...>
```

3. Include libraries in the `osimage` definition with the help of a postscript that is running in `postinstall`. The postscript copies the libraries to the root image:

```
$ lsdef -t osimage rh73-compute-stateless -i postinstall
Object name: rh73-compute-stateless
    postinstall=/install/custom/netboot/rhel/rh73-compute.postinstall
$ postinstall=/install/custom/netboot/rhel/rh73-compute.postinstall

$ installscript=/install/postscripts/install_pgilibs
$ cat <<EOF > $installscript
   #!/bin/bash
   # This script installs PGI libraries
   # It should run in postinstall

   installroot=\$1
   pgilibs=/opt/pgi/lib

   echo "Installing PGI libraries ..."
   mkdir -p \${installroot}\${pgilibs}
   cp -rav $dir/$libdir/*.so* \${installroot}\${pgilibs}

   # Add library path to ld config
   echo \$pgilibs > \${installroot}/etc/ld.so.conf.d/pgi.conf
   EOF

chmod +x $installscript

$ echo "# Install PGI libraries" >> $postinstall
$ echo "$installscript \$installroot" >> $postinstall
```

4. A `postscript` that is running after boot is needed to update runtime library bindings accordingly. This update cannot be done in `postinstall` because it is not running in `chroot`. Create such a script and add it to the `node` or `nodegroup` definition:

```
$ configscript=/install/postscripts/config_pgilibs
$ cat <<EOF > /install/postscripts/config_pgilibs
   #!/bin/bash
   # This script will update run time library bindings

   ldconfig
   EOF

$ chmod +x $configscript

$ chdef -t node p8r1n1 --plus postscripts=config_pgilibs
1 object definitions have been created or modified.

$ lsdef -t node p8r1n1 -i postscripts
Object name: p8r1n1
    postscripts=syslog,remoteshell,syncfiles,confignics -s,confignics
--ibaports=2,setroute,config_pgilibs
```

5.6.9 SMPI

To install IBM Spectrum MPI, complete the following steps:

Note: IBM Spectrum MPI substitutes IBM PE Runtime Environment.

1. Add the kit (distributed with the product media) to xCAT by using the **`addkit`** command:

```
$ addkit /path/to/smpi-files/ibm_smpi_kt-10.1.0.2-rh73-ppc64le.tar.bz2
Adding Kit ibm_smpi_kt-10.1.0.2-rh7-ppc64le
Kit ibm_smpi_kt-10.1.0.2-rh7-ppc64le was successfully added.

$ lsdef -t kit
ibm_smpi_kt-10.1.0.2-rh7-ppc64le  (kit)
xlc-13.1.5-0-ppc64le  (kit)
xlf-15.1.5-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) by using the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==ibm_smpi_kt-10.1.0.2-rh7-ppc64le -i
description
Object name: ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le
    description=IBM Spectrum MPI full installation
Object name: ibm_spectrum_mpi_license-10.1.0.2-rh7-rhels-7-ppc64le
    description=IBM Spectrum MPI license
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh73-compute-stateless \
ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le
Assigning kit component ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le to
osimage rh73-compute-stateless
Kit components ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le were added to
osimage rh73-compute-stateless successfully
```

```
$ lsdef -t osimage rh73-compute-stateless -i kitcomponents
Object name: rh73-compute-stateless

kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le,xlf.license-compute-15.1.5-0-rhels-7-ppc64le,xlf.rte-compu
te-15.1.5-0-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.1.0.2-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le
```

5.6.10 PPT

To install IBM Parallel Performance Toolkit, complete the following steps:

Note: IBM Parallel Performance Toolkit substitutes IBM PE Developer Edition. However, the xCAT software kit is still named `ppedev`.

1. Add the kit (distributed with the product media) to xCAT by using the **`addkit`** command:

```
$ addkit /path/to/ppt-files/ppedev-2.3.0-0.tar.bz2
Adding Kit ppedev-2.3.0-0
Kit ppedev-2.3.0-0 was successfully added.

$ lsdef -t kit
ibm_smpi_kt-10.1.0.2-rh7-ppc64le  (kit)
ppedev-2.3.0-0  (kit)
xlc-13.1.5-0-ppc64le  (kit)
xlf-15.1.5-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) by using the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==ppedev-2.3.0-0 -i description
Object name: ppedev.compute-2.3.0-0-rhels-7.3-ppc64le
    description=Parallel Performance Toolkit for compute nodes
Object name: ppedev.license-2.3.0-0-rhels-7.3-ppc64le
    description=Parallel Performance Toolkit license package
Object name: ppedev.login-2.3.0-0-rhels-7.3-ppc64le
    description=Parallel Performance Toolkit for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh73-compute-stateless \
ppedev.compute-2.3.0-0-rhels-7.3-ppc64le
Assigning kit component ppedev.compute-2.3.0-0-rhels-7.3-ppc64le to osimage
rh73-compute-stateless
Kit components ppedev.compute-2.3.0-0-rhels-7.3-ppc64le were added to osimage
rh73-compute-stateless successfully

$ lsdef -t osimage rh73-compute-stateless -i kitcomponents
Object name: rh73-compute-stateless

kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le,xlf.license-compute-15.1.5-0-rhels-7-ppc64le,xlf.rte-compu
te-15.1.5-0-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.1.0.2-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le,ppedev.license-2.3.0-0-rh
els-7.3-ppc64le,ppedev.compute-2.3.0-0-rhels-7.3-ppc64le
```

Note: To perform development/profiling tasks on a compute node, install the following `kitcomponent`:

`ppedev.login-2.3.0-0-rhels-7.3-ppc64le`

5.6.11 ESSL

To install ESSL, complete the following steps:

Note: ESSL version 5.5.0.0 needs a CUDA version >= 8.0.54.

1. Add the kit (distributed with the product media) to xCAT by using the **`addkit`** command:

```
$ addkit /path/to/essl-files/essl-5.5.0-0-ppc64le.tar.bz2
Adding Kit essl-5.5.0-0-ppc64le
Kit essl-5.5.0-0-ppc64le was successfully added.

$ lsdef -t kit
essl-5.5.0-0-ppc64le  (kit)
ibm_smpi_kt-10.1.0.2-rh7-ppc64le  (kit)
ppedev-2.3.0-0  (kit)
xlc-13.1.5-0-ppc64le  (kit)
xlf-15.1.5-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) by using the **`lsdef`** command:

```
$ lsdef -t kitcomponent -w kitname==essl-5.5.0-0-ppc64le -i description
Object name: essl-computenode-3264rte-5.5.0-0-rhels-7.3-ppc64le
    description=essl for compute nodes with 3264 rte only
Object name: essl-computenode-3264rtecuda-5.5.0-0-rhels-7.3-ppc64le
    description=essl for compute nodes with 3264 rte cuda only
Object name: essl-computenode-5.5.0-0-rhels-7.3-ppc64le
    description=essl for compute nodes
Object name: essl-computenode-6464rte-5.5.0-0-rhels-7.3-ppc64le
    description=essl for compute nodes with 6464 rte only
Object name: essl-computenode-nocuda-5.5.0-0-rhels-7.3-ppc64le
    description=essl for compute nodes
Object name: essl-license-5.5.0-0-rhels-7.3-ppc64le
    description=essl license for compute nodes
Object name: essl-loginnode-5.5.0-0-rhels-7.3-ppc64le
    description=essl for login nodes
Object name: essl-loginnode-nocuda-5.5.0-0-rhels-7.3-ppc64le
    description=essl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **`addkitcomp`** command:

```
$ addkitcomp --adddeps -i rh73-compute-stateless \
essl-computenode-5.5.0-0-rhels-7.3-ppc64le
Assigning kit component essl-computenode-5.5.0-0-rhels-7.3-ppc64le to osimage
rh73-compute-stateless
Kit components essl-computenode-5.5.0-0-rhels-7.3-ppc64le were added to osimage
rh73-compute-stateless successfully

$ lsdef -t osimage rh73-compute-stateless -i kitcomponents
Object name: rh73-compute-stateless
```

```
kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le,xlf.license-compute-15.1.5-0-rhels-7-ppc64le,xlf.rte-compu
te-15.1.5-0-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.1.0.2-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le,ppedev.license-2.3.0-0-rh
els-7.3-ppc64le,ppedev.compute-2.3.0-0-rhels-7.3-ppc64le,essl-license-5.5.0-0-r
hels-7.3-ppc64le,essl-computenode-5.5.0-0-rhels-7.3-ppc64le
```

5.6.12 PESSL

To install PESSL, complete the following steps:

1. Add the kit (distributed with the product media) to xCAT by using the **addkit** command:

```
$ addkit /path/to/pessl-files/pessl-5.3.0-0-ppc64le.tar.bz2
Adding Kit pessl-5.3.0-0-ppc64le.tar.bz2
Kit pessl-5.3.0-0-ppc64le was successfully added.

$ lsdef -t kit
essl-5.5.0-0-ppc64le  (kit)
ibm_smpi_kt-10.1.0.2-rh7-ppc64le  (kit)
pessl-5.3.0-0-ppc64le  (kit)
ppedev-2.3.0-0  (kit)
xlc-13.1.5-0-ppc64le  (kit)
xlf-15.1.5-0-ppc64le  (kit)
```

2. Verify its `kitcomponent` objects (and `description` fields) by using the **lsdef** command:

```
$ lsdef -t kitcomponent -w kitname==pessl-5.3.0-0-ppc64le -i description
Object name: pessl-computenode-3264rte-5.3.0-0-rhels-7.3-ppc64le
    description=pessl for compute nodes with ESSL non-cuda runtime
Object name: pessl-computenode-5.3.0-0-rhels-7.3-ppc64le
    description=pessl for compute nodes
Object name: pessl-computenode-nocuda-5.3.0-0-rhels-7.3-ppc64le
    description=pessl for compute nodes
Object name: pessl-license-5.3.0-0-rhels-7.3-ppc64le
    description=pessl license for compute nodes
Object name: pessl-loginnode-5.3.0-0-rhels-7.3-ppc64le
    description=pessl for login nodes
Object name: pessl-loginnode-nocuda-5.3.0-0-rhels-7.3-ppc64le
    description=pessl for login nodes
```

3. Add the following `kitcomponent` (and dependencies) to the `osimage` object by using the **addkitcomp** command:

```
$ addkitcomp --adddeps -i rh73-compute-stateless \
pessl-computenode-5.3.0-0-rhels-7.3-ppc64le
Assigning kit component pessl-computenode-5.3.0-0-rhels-7.3-ppc64le to osimage
rh73-compute-stateless
Kit components pessl-computenode-5.3.0-0-rhels-7.3-ppc64le were added to
osimage rh73-compute-stateless successfully

$ lsdef -t osimage rh73-compute-stateless -i kitcomponents
Object name: rh73-compute-stateless

kitcomponents=xlc.license-compute-13.1.5-0-rhels-7-ppc64le,xlc.rte-compute-13.1
.5-0-rhels-7-ppc64le,xlf.license-compute-15.1.5-0-rhels-7-ppc64le,xlf.rte-compu
te-15.1.5-0-rhels-7-ppc64le,ibm_spectrum_mpi_license-10.1.0.2-rh7-rhels-7-ppc64
le,ibm_spectrum_mpi-full-10.1.0.2-rh7-rhels-7-ppc64le,ppedev.license-2.3.0-0-rh
```

```
els-7.3-ppc64le,ppedev.compute-2.3.0-0-rhels-7.3-ppc64le,essl-license-5.5.0-0-r
hels-7.3-ppc64le,essl-computenode-5.5.0-0-rhels-7.3-ppc64le,pessl-license-5.3.0
-0-rhels-7.3-ppc64le,pessl-computenode-5.3.0-0-rhels-7.3-ppc64le
```

5.6.13 Spectrum Scale (formerly GPFS)

This section described how to install the packages for Spectrum Scale 4.2.x.0 and the package updates for Spectrum Scale 4.2.x.3, and to build and install the GPFS Portability Layer (GPL) packages.

The process to build the GPL requires a provisioned node (for example, compute, login, or management node) with Spectrum Scale packages installed (specifically, `gpfs.gpl`). Therefore, if the management node is not used to build the GPL, you cannot install Spectrum Scale with the GPL (requirement) in a single provisioning stage for one of the nodes, which is used to build the GPL. It is possible, afterward, provided the built GPL package is made available for download in the management node (such as other Spectrum Scale packages) for installation on other nodes.

For more information about the installation process, see the Installing GPFS on Linux nodes page in the IBM Knowledge Center website. To install Spectrum Scale 4.2.x.3 (on top of 4.2.x.0), complete the following steps:

1. Install a script for environment configuration.
2. Install the packages for Spectrum Scale 4.2.1.0.
3. Install the packages for Spectrum Scale 4.2.1.3.
4. Build the GPL.

Installing a script for environment configuration

To install the script for environment configuration, complete the following steps:

1. Create a script to set the `PATH` environment variable:

```
$ script=/install/postscripts/gpfs-path

$ cat <<EOF >$script
#!/bin/bash

profile='/etc/profile.d/gpfs.sh'
echo 'export PATH=\$PATH:/usr/lpp/mmfs/bin' >\$profile
EOF

$ chmod +x $script

$ ls -l $script
-rwxr-xr-x 1 root root 99 Nov 30 17:24 /install/postscripts/gpfs-path
```

2. Include it in the `postscripts` list of the `osimage` object by using the **`chdef`** command:

```
$ lsdef -t osimage rh73-hpc-diskful -i postscripts
Object name: rh73-hpc-diskful
    postscripts=nvidia-power-limit

$ chdef -t osimage rh73-hpc-diskful --plus postscripts='gpfs-path'
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-hpc-diskful -i postscripts
Object name: rh73-hpc-diskful
    postscripts=nvidia-power-limit,gpfs-path
```

Installing the packages for Spectrum Scale 4.2.x.0

To install the packages, complete the following steps:

1. Extract the product packages of Spectrum Scale 4.2.x.0:

```
$ dir=/install/post/otherpkgs/rhels7.3/ppc64le/gpfs-4210
$ mkdir -p $dir

$ /path/to/Spectrum_Scale_install-4.2.1.0_ppc64le_standard --silent --dir $dir
<...>
Extracting Product RPMs to /install/post/otherpkgs/rhels7.3/ppc64le/gpfs-4210
<...>
```

2. Create the respective package repository by using the **createrepo** command:

```
$ createrepo $dir
<...>

$ ls -1 $dir
gpfs.base_4.2.1-0_ppc64el.deb
gpfs.base-4.2.1-0.ppc64le.rpm
gpfs.docs_4.2.1-0_all.deb
gpfs.docs-4.2.1-0.noarch.rpm
gpfs.ext_4.2.1-0_ppc64el.deb
gpfs.ext-4.2.1-0.ppc64le.rpm
gpfs.gpl_4.2.1-0_all.deb
gpfs.gpl-4.2.1-0.noarch.rpm
gpfs.gskit_8.0.50-40_ppc64el.deb
gpfs.gskit-8.0.50-40.ppc64le.rpm
gpfs.hadoop-2-connector-4.2.1-0.ppc64le.rpm
gpfs.msg.en-us_4.2.1-0_all.deb
gpfs.msg.en_US-4.2.1-0.noarch.rpm
license
manifest
repodata
```

3. Create the respective package list with a combination of commands:

```
$ list=/install/custom/rhels7.3/gpfs-4210.otherpkgs.pkglist

$ rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" | grep -v hadoop >$list

$ cat $list
gpfs-4210/gpfs.base
gpfs-4210/gpfs.docs
gpfs-4210/gpfs.ext
gpfs-4210/gpfs.gpl
gpfs-4210/gpfs.gskit
gpfs-4210/gpfs.msg.en_US
```

4. Include the respective package list in the `otherpkglist` attribute of the `osimage` object by using the **chdef** command:

```
$ chdef -t osimage rh73-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-hpc-diskful -i otherpkglist
Object name: rh73-hpc-diskful
```

```
    otherpkglist=/install/osimages/rh73-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh73-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.3/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.3/gpfs-4210.otherpkgs.pkglist
```

Installing the packages for Spectrum Scale 4.2.x.3

To install the packages, complete the following steps:

1. Extract the product packages of Spectrum Scale 4.2.x.3:

```
$ dir=/install/post/otherpkgs/rhels7.3/ppc64le/gpfs-4213
$ mkdir -p $dir

$ /root/software/gpfs/Spectrum_Scale_Standard-4.2.1.3-ppc64LE-Linux-update
--silent --dir $dir
<...>
Product rpms successfully extracted to
/install/post/otherpkgs/rhels7.3/ppc64le/gpfs-4213
```

2. Create the respective package repository by using the **createrepo** command:

```
$ createrepo $dir
<...>

$ ls -1 $dir
gpfs.base_4.2.1-3_ppc64el_update.deb
gpfs.base-4.2.1-3.ppc64le.update.rpm
gpfs.docs_4.2.1-3_all.deb
gpfs.docs-4.2.1-3.noarch.rpm
gpfs.ext_4.2.1-3_ppc64el_update.deb
gpfs.ext-4.2.1-3.ppc64le.update.rpm
gpfs.gpl_4.2.1-3_all.deb
gpfs.gpl-4.2.1-3.noarch.rpm
gpfs.gskit_8.0.50-47_ppc64el.deb
gpfs.gskit-8.0.50-47.ppc64le.rpm
gpfs.hadoop-connector_2.7.0-2_ppc64el.deb
gpfs.hadoop-connector-2.7.0-2.ppc64le.rpm
gpfs.msg.en-us_4.2.1-3_all.deb
gpfs.msg.en_US-4.2.1-3.noarch.rpm
manifest
repodata
```

3. Create the respective package list with a combination of commands:

> **Note:** Use the `#NEW_INSTALL_LIST#` directive to perform package installation of Spectrum Scale 4.2.x.0 and 4.2.x.3 in different stages to ensure the update process (from version 4.2.x.0 to version 4.2.x.3) occurs correctly.
>
> For more information, see the xCAT File Format for .otherpkgs.pkglist File documentation page.

```
$ list=/install/custom/rhels7.3/gpfs-4213.otherpkgs.pkglist

$ echo '#NEW_INSTALL_LIST#' > $list
$ rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" | grep -v hadoop >>$list
```

```
$ cat $list
   #NEW_INSTALL_LIST#
   gpfs-4213/gpfs.base
   gpfs-4213/gpfs.docs
   gpfs-4213/gpfs.ext
   gpfs-4213/gpfs.gpl
   gpfs-4213/gpfs.gskit
   gpfs-4213/gpfs.msg.en_US
```

4. Include the respective package list in the `otherpkglist` attribute of the `osimage` object by using the **chdef** command:

```
$ chdef -t osimage rh73-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-hpc-diskful -i otherpkglist
Object name: rh73-hpc-diskful
    otherpkglist=/install/osimages/rh73-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh73-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.3/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.3/gpfs-4210.otherpkgs.pkglist,/install/custom/rhels7.3/gpfs-4213.otherpkgs.pkg
list
```

Building the GPL

The process to build the GPL requires a working node (for example, a management, login, or compute node) with Spectrum Scale packages (specifically, `gpfs.gpl`) installed.

The node must be capable of building out-of-tree kernel modules (that is, with development packages, such as `gcc`, `make`, and `kernel-devel` installed), and run the same kernel packages version and processor architecture as the target compute nodes for the GPL produced binaries.

For example, you can build the GPL in the management node or an installed compute node (if any), if it is a system based on POWER8 that is running RHEL Server 7.3 for ppc64le, and matches the kernel packages version of the target compute nodes. This section describes the build process with an installed compute node.

Note: The GPL must be rebuilt and reinstalled if the kernel packages are updated. This process requires the `kernel-devel` package for the respective update version.

To perform the GPL build, complete the following steps:

1. Verify the Spectrum Scale installation (`PATH` environment variable and RPM packages):

```
$ xdsh p8r2n2 'echo $PATH' | grep mmfs
p8r2n2: /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/lpp/mmfs/bin:
/opt/ibutils/bin

$ xdsh p8r2n2 'rpm -qa | grep ^gpfs'
p8r2n2: gpfs.msg.en_US-4.2.1-3.noarch
p8r2n2: gpfs.gpl-4.2.1-3.noarch
p8r2n2: gpfs.gskit-8.0.50-47.ppc64le
p8r2n2: gpfs.docs-4.2.1-3.noarch
p8r2n2: gpfs.ext-4.2.1-3.ppc64le
p8r2n2: gpfs.base-4.2.1-3.ppc64le
```

2. Build the GPL.

   Verify that the build process writes the resulting GPL binary RPM package (`gpfs.gplbin-<kernel version>.<architecture>-<spectrum scale version>.rpm`), and finishes with an exit code of `zero` (success):

```
$ xdsh p8r2n2 --stream 'cd /usr/lpp/mmfs/src && make Autoconfig && make World
&& make InstallImages && make rpm'
<...>
p8r2n2: Verifying that tools to build the portability layer exist....
p8r2n2: cpp present
p8r2n2: gcc present
p8r2n2: g++ present
p8r2n2: ld present
<...>
p8r2n2: Wrote:
/root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-3.10.0-327.el7.ppc64le-4.2.1-3.ppc64le.
rpm
<...>
p8r2n2: + exit 0
```

3. Copy the package and create the respective package repository with the **createrepo** command:

```
$ dir=/install/post/otherpkgs/rhels7.3/ppc64le/gpfs-gpl
$ mkdir -p $dir

$ scp p8r2n2:/root/rpmbuild/RPMS/ppc64le/gpfs.gplbin-*.rpm $dir
gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le.rpm   <...>

$ createrepo $dir
<...>

$ ls -1 $dir
gpfs.gplbin-3.10.0-327.el7.ppc64le-4.2.1-3.ppc64le.rpm
repodata
```

4. Create the respective package list by using the following commands:

```
$ list=/install/custom/rhels7.3/gpfs-gpl.otherpkgs.pkglist

$ echo '#NEW_INSTALL_LIST#' >$list
$ rpm -qip $dir/*.rpm | awk '/^Name/ { print $3 }' | sed "s:^:$(basename
$dir)/:" >>$list

$ cat $list
#NEW_INSTALL_LIST#
gpfs-gpl/gpfs.gplbin-3.10.0-327.el7.ppc64le
```

5. Include the respective package list in the `otherpkglist` attribute of the `osimage` object by using the **chdef** command.

   This step allows the GPL package to be installed automatically (without rebuild steps) during the provisioning stage of other nodes:

```
$ chdef -t osimage rh73-hpc-diskful --plus otherpkglist=$list
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-hpc-diskful -i otherpkglist
Object name: rh73-hpc-diskful
```

```
    otherpkglist=/install/osimages/rh73-hpc-diskful/kits/KIT_DEPLOY_PARAMS.othe
rpkgs.pkglist,/install/osimages/rh73-hpc-diskful/kits/KIT_COMPONENTS.otherpkgs.
pkglist,/install/custom/rhels7.3/at8.0.otherpkgs.pkglist,/install/custom/rhels7
.3/gpfs-4210.otherpkgs.pkglist,/install/custom/rhels7.3/gpfs-4213.otherpkgs.pkg
list,/install/custom/rhels7.3/gpfs-gpl.otherpkgs.pkglist
```

Note: You can install the GPL in an installed and running compute node by using the **otherpkgs** script of the **updatenode** command:

```
$ updatenode p8r3n3 -P otherpkgs
<...>
p8r3n3: pkgsarray:  gpfs-gpl/gpfs.gplbin-3.10.0-327.el7.ppc64le, 2
<...>

$ xdsh p8r3n3 'rpm -qa | grep ^gpfs.gpl'
p8r3n3: gpfs.gpl-4.2.1-3.noarch
p8r3n3: gpfs.gplbin-3.10.0-327.el7.ppc64le-4.1.1-3.ppc64le
```

For more information about configuration and usage, see the Steps to establishing and starting your GPFS cluster page at the IBM Knowledge Center website.

5.6.14 IBM Spectrum LSF

This section describes how to install, update, and enable Spectrum LSF (formerly Platform LSF) on the compute nodes, and to configure some features. The Spectrum LSF installation and the user applications (which is run by using jobs) must be available on a parallel file system for all nodes to concurrently access the same code and data correctly. For the scenario that is described in this section, a Spectrum Scale file system is available at `/gpfs/gpfs_fs0`.

The process consists of installing and updating Spectrum LSF in one node (that is, only once), and then enabling it on all nodes. The installation or update is only required in one node because it is available in the parallel file system accessible by all nodes. However, the enablement is required on all nodes because it performs configuration steps and enables startup services.

Therefore, the process to install or update Spectrum LSF requires a provisioned node (for example, a compute or login node) with access to the parallel file system that is available to the other nodes. It is not possible to install and enable Spectrum LSF in a single provisioning stage for one of the nodes, which is used to install Spectrum LSF to the parallel file system. It is possible afterward if it is installed and available in the parallel file system for access by other nodes, which must enable only Spectrum LSF and be added to the Spectrum LSF cluster. The option with single provisioning stage is not covered in this section, which still requires enabling Spectrum LSF and adding the nodes to the cluster manually.

To install Spectrum LSF, complete the following steps:

1. Install Spectrum LSF.
2. Update Spectrum LSF.
3. Enable Spectrum LSF.
4. Add nodes.
5. Configure extra HPC and IBM PE support features.
6. Configure GPU support features.

Installing Spectrum LSF

Complete the following steps on the management node, targeting a provisioned compute node (for example, p8r1n1, defined as `lsf_master`):

1. Create the directories for the installation and distribution directories of Spectrum LSF to be mounted from a Spectrum Scale file system (for example, `/gpfs/gpfs_fs0`):

```
$ gpfs_dir='/gpfs/gpfs_fs0/lsf'
$ gpfs_top="$gpfs_dir/top"
$ gpfs_distrib="$gpfs_dir/distrib"

$ lsf_top='/usr/share/lsf'
$ lsf_distrib='/usr/share/lsf_distrib'

$ lsf_master='p8r1n1'

$ xdsh $lsf_master "mkdir -p $lsf_top $gpfs_top && echo '$gpfs_top $lsf_top
none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_top"
p8r1n1: mount: /gpfs/gpfs_fs0/lsf/top bound on /usr/share/lsf.

$ xdsh $lsf_master "mkdir -p $lsf_distrib $gpfs_distrib && echo '$gpfs_distrib
$lsf_distrib none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_distrib"
p8r1n1: mount: /gpfs/gpfs_fs0/lsf/distrib bound on /usr/share/lsf_distrib.
```

2. Copy the installation tarballs and the entitlement file to the distribution directory:

```
$ cd /path/to/lsf-install-files

$ ls -1
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
lsf.entitlement

$ scp \
    lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z \
    lsf9.1.3_lnx310-lib217-ppc64le.tar.Z \
    lsf.entitlement \
    $lsf_master:$lsf_distrib
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
100%  116MB  58.2MB/s   00:02
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
100%  228MB  76.1MB/s   00:03
lsf.entitlement
100%  167     0.2KB/s   00:00
```

   Verify the files are present in the correct directory:

```
$ ssh $lsf_master "cd $lsf_distrib; pwd; ls -1"
/usr/share/lsf_distrib
lsf9.1.3_lnx310-lib217-ppc64le.tar.Z
lsf9.1.3_lsfinstall_linux_ppc64le.tar.Z
lsf.entitlement
```

3. Create the administrator user for Spectrum LSF:

```
$ lsf_username='lsfadmin'
$ lsf_password='<password>'

$ xdsh $lsf_master "useradd -m -s /bin/bash $lsf_username && echo
"$lsf_username:$lsf_password" | chpasswd; su -l $lsf_username -c whoami"
```

```
p8r1n1: lsfadmin

$ xdsh $lsf_master "su -l $lsf_username -c 'cat ~/.profile >> ~/.bash_profile'"
```

4. Create the configuration file for the installation (`install.config`). It must be placed in the same directory as the `install_lsf` and `lsf_startup` scripts.

   The following configuration is used for the scenario in this chapter:

   - Top directory: `/usr/share/lsf`
   - Distribution directory: `/usr/share/lsf_distrib`
   - Entitlement file: `lsf.entitlement` (in the distribution directory)
   - Administrator username: `lsfadmin`
   - Cluster name: `lsf-cluster`
   - Master/server nodes: `p8r1n1` (that is, `$lsf_master`)
   - Non-master/server nodes: None at this time

```
$ lsf_cluster='lsf-cluster'
$ lsf_entitlement="$lsf_distrib/lsf.entitlement"

$ cat <<EOF >/install/postscripts/install.config
LSF_TOP="$lsf_top"
LSF_TARDIR="$lsf_distrib"
LSF_ENTITLEMENT_FILE="$lsf_entitlement"
LSF_ADMINS="$lsf_username"
LSF_CLUSTER_NAME="$lsf_cluster"
LSF_MASTER_LIST="$lsf_master"
LSF_ADD_SERVERS=""
EOF
```

5. Verify that the file contents are correct:

```
$ cat /install/postscripts/install.config
LSF_TOP="/usr/share/lsf"
LSF_TARDIR="/usr/share/lsf_distrib"
LSF_ENTITLEMENT_FILE="/usr/share/lsf_distrib/lsf.entitlement"
LSF_ADMINS="lsfadmin"
LSF_CLUSTER_NAME="lsf-cluster"
LSF_MASTER_LIST="p8r1n1"
LSF_ADD_SERVERS=""
```

6. Include the `ed` package in the package list:

```
$ lsdef -t osimage rh73-hpc-diskful -i pkglist
Object name: rh73-hpc-diskful
    pkglist=/install/custom/install/rh/rh73-hpc.pkglist

$ list=/install/custom/install/rh/rh73-hpc.pkglist

$ cat <<EOF >>$list

# Spectrum LSF
ed
EOF
```

7. Verify that the package list is correct:

```
$ cat $list
# RHEL Server 7.2 (original pkglist)
```

```
#INCLUDE:/opt/xcat/share/xcat/install/rh/compute.rhels7.pkglist#

# CUDA Toolkit 7.5 for RHEL 7.2 (original pkglist)
#INCLUDE:/opt/xcat/share/xcat/install/rh/cudafull.rhels7.ppc64le.pkglist#

# Infiniband with Mellanox OFED for RHEL 7.2 (original pkglist)
createrepo
#INCLUDE:/opt/xcat/share/xcat/ib/netboot/rh/ib.rhels7.ppc64le.pkglist#

# PE RTE IVP
ksh

# Spectrum LSF
ed
```

8. Install the packages from the `pkglist` attribute by using the `ospkgs` script for the **updatenode** command:

```
$ updatenode $lsf_master --scripts ospkgs
p8r1n1: xcatdsklspost: downloaded postscripts successfully
p8r1n1: <...> Running postscript: ospkgs
<...>
p8r1n1: Postscript: ospkgs exited with code 0
p8r1n1: Running of postscripts has completed.
```

9. Verify that the **ed** command is available:

```
$ xdsh $lsf_master 'ed --version | head -n1'
p8r1n1: GNU Ed 1.9
```

10. Install Spectrum LSF on the node by using the `install_lsf` script for the **updatenode** command.

11. Verify that the `exit code` of the script is `zero` (success).

```
$ updatenode $lsf_master --scripts install_lsf
p8r1n1: xcatdsklspost: downloaded postscripts successfully
p8r1n1: <...> Running postscript: install_lsf
<...>
p8r1n1: INFO: Installation script DONE.
p8r1n1: INFO: Updating LSF Cluster Configuration Files lsf.conf and lsb.hosts
p8r1n1: Postscript: install_lsf exited with code 0
p8r1n1: Running of postscripts has completed.
```

Note: If the **ed** command is not available, the following error can occur:

```
$ updatenode $lsf_master --scripts install_lsf
<...>
p8r1n1: ERROR: Fail to install LSF. Check Install.log and Install.err in
/usr/share/lsf_distrib/lsf9.1.3_lsfinstall.
<...>

$ xdsh $lsf_master "cat /usr/share/lsf_distrib/*/Install.err"
<...>
p8r1n1:     Cannot find UNIX command " ed".
<...>
```

Updating Spectrum LSF

Several updates are available for Spectrum LSF in the form of fixes (or patches).

For more information about applying fixes to Spectrum LSF, see the Manage cluster patches and versions on UNIX and Linux page of the IBM Knowledge Center website.

To download and apply fixes for Spectrum LSF, complete the following steps:

1. Download the fixes:
   a. Go to the IBM Support website.
   b. In Product Finder field, enter `LSF`.
   c. In the results list, select **LSF 9.1.3**.
   d. Under Downloads, select **Downloads (fixes & PTFs)**.

      A window opens (title: **Refine my fix list**; subtitle: **LSF 9.1.3**).
   e. In Version fix level, select **9.1.3**.
   f. In Filter by operating system, select **Linux Power PC 64 Little Endian**.
   g. Click **Continue**.

      The list of fixes is shown, and more information is available in Show fix details. For example, the following information is shown at the time of this writing):

      - Interim fix: `lsf-9.1.3-build347817`

        Abstract: P101215. Fix to enhance NVIDIA GPU integration with LSF 9.1.3 in Linux x64 environment. LSF makes use of the `cgroup` device subsystem to enforce the GPU in this patch. LSF can disable auto boost for the job with exclusive thread/process multi-GPU requirements. LSF can power off the GPU if it is not in use.
      - Interim fix: `lsf-9.1.3-build368515`

        Abstract: `P101357`. This fix updates the `lsb_readjobinfo` API to set a job.
      - Fix pack: `lsf-9.1.3.3-spk-2015-Jun-build346694`

        Abstract: `LSF Version 9.1.3 Fix Pack 3`. This Fix Pack includes all fixed issues and solutions that are included in previous LSF Version 9.1.3 Fix Packs and addresses new issues and solutions from February 1, 2015 - June 8, 2015. For more information about the issues and solutions in this Fix Pack, see the LSF 9.1.3 Fix Pack 3 Fixed Bugs List (`lsf9.1.3.3_fixed_bugs.pdf`).
      - Interim fix: `lsf-9.1.3-build362511`

        Abstract: `P101353`. Fix to ensure launch job run successfully over 32 nodes.
   h. In the results list, select the wanted fixes, and click **Continue**. Usually, all of the fixes are suggested for the general case.
   i. Proceed with the sign-in process.
   j. Select a download option (for example, **HTTPS**).
   k. Copy the download link of each file (for HTTPS; for other methods, follow the instructions that are provided at the website) or download the files and transfer to the wanted node later.
2. Apply the fixes.

   Perform the following commands in one of the nodes with access to the parallel file system with the Spectrum LSF installation (for example, open an SSH connection to `p8r1n1`):

   a. Go to the patch installation directory:

```
$ patch_dir=/usr/share/lsf/9.1/install/patches/
$ mkdir -p $patch_dir
$ cd $patch_dir
```

b. Download (or copy) one or more fixes.

You can use the following download links:

```
$ curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-347817.tar.Z
$ curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-368515.tar.Z
$ curl -sOL https://<...>/lsf9.1.3_lnx310-lib217-ppc64le-346694.tar.Z
$ curl -sOL https://<...>/lsf9.1.3_linux3.10-glibc2.17-ppc64le-362511.tar.Z

$ ls -1
lsf9.1.3_linux3.10-glibc2.17-ppc64le-347817.tar.Z
lsf9.1.3_linux3.10-glibc2.17-ppc64le-362511.tar.Z
lsf9.1.3_linux3.10-glibc2.17-ppc64le-368515.tar.Z
lsf9.1.3_lnx310-lib217-ppc64le-346694.tar.Z
```

c. Run the **patchinstall** command on the fixes:

```
$ ../patchinstall *-362511.* *-346694.* *-368515.* *-347817.*
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-362
511.tar.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3625
11.tar.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_lnx310-lib217-ppc64le-346694.ta
r.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_lnx310-lib217-ppc64le-346694.tar
.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-368
515.tar.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3685
15.tar.Z.
<...>
Installing package
"/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-347
817.tar.Z"...
<...>
Are you sure you want to update your cluster with this patch? (y/n) [y]
<...>
   Done installing
/usr/share/lsf/9.1/install/patches/lsf9.1.3_linux3.10-glibc2.17-ppc64le-3478
17.tar.Z.
```

```
This patch has updated binaries or library files that affect running
daemons.
To make the changes take effect, you must restart your cluster.

Exiting...
```

d. Verify the installed fixes by using the **pversions** command:

```
$ /usr/share/lsf/9.1/install/pversions

IBM Platform LSF 9.1.3
----------------------
  binary type: linux3.10-glibc2.17-ppc64le, Apr 01 2015, Build 335772
  installed: Nov 30 2015
  patched:  Fix P101353, build 362511, installed Dec 01 2015
            Fix , build 346694, installed Dec 01 2015
            Fix P101357, build 368515, installed Dec 01 2015
            Fix P101215, build 347817, installed Dec 01 2015
```

e. Restart several services in the master node:

```
$ lsadmin limrestart all
<...>
Do you really want to restart LIMs on all hosts? [y/n] y
<...>

$ lsadmin resrestart all
Do you really want to restart RES on all hosts? [y/n] y
<...>

$ badmin hrestart all
<...>
Restart slave batch daemon on all the hosts? [y/n] y
<...>

$ badmin mbdrestart
<...>
Do you want to restart MBD? [y/n] y
<...>
```

Enabling Spectrum LSF

Complete the following steps on the management node and target all compute nodes:

1. Modify the `lsf_startup` script to correctly find the Spectrum LSF version that is based on its installation path (this modification is done only once, in the management node):

   Note: This issue is expected to be resolved in a future xCAT version (xCAT issue #495).

   a. Create a copy of the original script:

   ```
   $ cp -a /install/postscripts/lsf_startup
   /install/postscripts/lsf_startup.bkp
   ```

   b. Modify the script by using the following expression to the **sed** command. The expression is a single, long line (without line breaks):

   ```
   $ sed \
   '/^LSF_VERSION=/ a LSF_VERSION="$(find /$LSF_TOP -path "*/install/hostsetup"
   ```

```
| grep -o "/[^/]\\+/install/hostsetup" | cut -d/ -f2)"' \
-i /install/postscripts/lsf_startup
```

c. Verify the differences between the original and the modified scripts:

```
$ diff /install/postscripts/lsf_startup.bkp /install/postscripts/lsf_startup
34a35
> LSF_VERSION="$(find /$LSF_TOP -path "*/install/hostsetup" | grep -o
"/[^/]\+/install/hostsetup" | cut -d/ -f2)"
```

2. Run the `lsf_startup` script in the nodes by using the **`updatenode`** command (see Example 5-20).

   You can use a node group (for example, `s822lc`) instead of the `$lsf_master` variable after the nodes in the group are online.

   Verify that the `exit code` of the script is `zero` (success):

Example 5-20 Running the lsf_startup script with the updatenode command

```
$ updatenode $lsf_master --scripts lsf_startup
p8r1n1: xcatdsklspost: downloaded postscripts successfully
p8r1n1: <...> Running postscript: lsf_startup
p8r1n1: INFO: Run hostsetup on each node.
p8r1n1: Logging installation sequence in /usr/share/lsf/log/Install.log
p8r1n1:
p8r1n1: ------------------------------------------------------------
p8r1n1:     L S F    H O S T S E T U P    U T I L I T Y
p8r1n1: ------------------------------------------------------------
p8r1n1: This script sets up local host (LSF server, client, or slave)
environment.
p8r1n1: Setting up LSF server host "p8r1n1" ...
p8r1n1: Checking LSF installation for host "p8r1n1.xcat-cluster" ... Done
p8r1n1: Installing LSF RC scripts on host "p8r1n1.xcat-cluster" ... Done
p8r1n1: LSF service ports are defined in /usr/share/lsf/conf/lsf.conf.
p8r1n1: Checking LSF service ports definition on host "p8r1n1.xcat-cluster" ...
Done
p8r1n1: You are installing IBM Platform LSF -  Standard Edition.
p8r1n1:
p8r1n1: ... Setting up LSF server host "p8r1n1" is done
p8r1n1: ... LSF host setup is done.
p8r1n1: INFO: Set LSF environment for root and LSF_ADMINS
p8r1n1: INFO: Start LSF Cluster.
p8r1n1: Starting up LIM on <p8r1n1> ...... done
p8r1n1: Starting up RES on <p8r1n1> ...... done
p8r1n1: Starting up slave batch daemon on <p8r1n1> ...... done
p8r1n1: Postscript: lsf_startup exited with code 0
p8r1n1: Running of postscripts has completed.
```

3. Verify that the Spectrum LSF commands are available for the LSF administrator user, and that the cluster is listed by using the **`lsclusters`** command:

```
$ xdsh $lsf_master "su -l $lsf_username -c lsclusters"
p8r1n1: CLUSTER_NAME   STATUS   MASTER_HOST    ADMIN    HOSTS  SERVERS
p8r1n1: lsf-cluster    ok       p8r1n1         lsfadmin     1        1
```

Adding nodes

To add nodes to the cluster, complete the following steps on the management node, targeting a provisioned compute node (for example, p8r3n2, defined as `lsf_node`). For more information, see the Adding a host page of the IBM Knowledge Center website:

1. Create the directories for the installation and distribution directories of Spectrum LSF to be mounted from a Spectrum Scale file system (for example, `/gpfs/gpfs_fs0`):

```
$ gpfs_dir='/gpfs/gpfs_fs0/lsf' # then /top and /distrib
$ gpfs_top="$gpfs_dir/top"
$ gpfs_distrib="$gpfs_dir/distrib"

$ lsf_top='/usr/share/lsf'
$ lsf_distrib='/usr/share/lsf_distrib'

$ lsf_node='p8r3n2'

$ xdsh $lsf_node "mkdir -p $lsf_top $gpfs_top && echo '$gpfs_top $lsf_top none
defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_top"
p8r3n2: mount: /gpfs/gpfs_fs0/lsf/top bound on /usr/share/lsf.

$ xdsh $lsf_node "mkdir -p $lsf_distrib $gpfs_distrib && echo '$gpfs_distrib
$lsf_distrib none defaults,bind 0 0' >>/etc/fstab && mount -v $lsf_distrib"
p8r3n2: mount: /gpfs/gpfs_fs0/lsf/distrib bound on /usr/share/lsf_distrib.
```

2. Add the node to the `lsf.cluster.<cluster-name>` file.

   You can edit the file from any node that can access the parallel file system with the Spectrum LSF installation directory.

   You can add a line for the new node (`p8r3n2`) based on an existing node (`p8r2n2`), for example:

```
$ vi /usr/share/lsf/conf/lsf.cluster.lsf-cluster
<...>
Begin   Host
HOSTNAME  model    type        server r1m  mem  swp  RESOURCES    #Keywords
<...>
p8r1n1   !   !   1   3.5   ()   ()   (mg)
p8r3n2   !   !   1   3.5   ()   ()   (mg)
End     Host
```

3. Restart the services on a master node (for example, `p8r1n1`):

```
<management-node> $ lsf_master=p8r1n1
<management-node> $ ssh $lsf_master

<master-node> $ su lsfadmin

<master-node> $ lsadmin reconfig

Checking configuration files ...
No errors found.

Restart only the master candidate hosts? [y/n] y
Restart LIM on <p8r1n1> ...... done

<master-node> $ badmin mbdrestart

Checking configuration files ...
```

```
There are warning errors.

Do you want to see detailed messages? [y/n] y
Checking configuration files ...
<...>
-------------------------------------------------------
No fatal errors found.
<...>
Do you want to restart MBD? [y/n] y
MBD restart initiated
```

Verify the new node is listed by the **lshosts** command (with no hardware details yet):

```
<master-node> $ lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r3n2      UNKNOWN UNKNOWN_   1.0     -      -      -    Yes (mg)
```

4. Create the administrator user for Spectrum LSF:

```
$ lsf_username='lsfadmin'
$ lsf_password='<password>'

$ xdsh $lsf_node "useradd -m -s /bin/bash $lsf_username && echo
"$lsf_username:$lsf_password" | chpasswd; su -l $lsf_username -c whoami"
p8r3n2: lsfadmin

$ xdsh $lsf_node "su -l $lsf_username -c 'cat ~/.profile >> ~/.bash_profile'"
```

5. Run the `lsf_startup` script in the nodes by using the **updatenode** command:

```
$ updatenode $lsf_node --scripts lsf_startup
```

6. Start the services on the node (you can reboot the node, instead):

```
$ xdsh $lsf_node "su -l $lsf_username -c 'lsadmin limstartup' "
p8r3n2: Starting up LIM on <p8r3n2> ...... done


$ xdsh $lsf_node "su -l $lsf_username -c 'lsadmin resstartup' "
p8r3n2: Starting up RES on <p8r3n2> ...... done


$ xdsh $lsf_node "su -l $lsf_username -c 'badmin hstartup' "
p8r3n2: Starting up slave batch daemon on <p8r3n2> ...... done
```

7. Verify that the new node is listed by running the **lshosts** command (with hardware details now):

```
$ lsf_master=p8r1n1

$ xdsh $lsf_master "su -l $lsf_username -c lshosts"
p8r1n1: HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1: p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r1n1: p8r3n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

Configuring extra HPC and IBM PE support features

When Spectrum LSF is installed, you can set the `CONFIGURATION_TEMPLATE` property on `install.config` file to use one of the following configuration templates:

- `DEFAULT`: Used for mixed type of work loads. Although it provides good performance overall, it is not tuned for any specific type of cluster.
- `PARALLEL`: Adds extra support to large parallel jobs, including specific configurations to IBM Parallel Environment (PE).
- `HIGH_THROUGHPUT`: Tunes for high throughput (high rate of mainly short jobs).

As an alternative, you can use `DEFAULT` template, but then configure it manually according to your needs. For example, the following configuration tasks enable the cluster as it was installed with a `PARALLEL` template. As the Spectrum LSF administrator user, complete the following steps:

1. Edit the `lsf.shared` configuration file to add following resources in the `Resource` section:

```
ibmmpi     Boolean ()       ()          (IBM POE MPI)
adapter_windows Numeric    30    N    (free adapter windows on css0 on IBM SP)
nrt_windows     Numeric    30    N    (The number of free nrt windows on IBM
systems)
poe             Numeric    30    N    (poe availability)
css0            Numeric    30    N    (free adapter windows on css0 on IBM SP)
csss            Numeric    30    N    (free adapter windows on csss on IBM SP)
dedicated_tasks Numeric    ()    Y    (running dedicated tasks)
ip_tasks        Numeric    ()    Y    (running IP tasks)
us_tasks        Numeric    ()    Y    (running US tasks)
```

2. Map out new resources on the `lsf.cluster.`*<cluster_name>* configuration file, in the `ResourceMap` section:

```
poe                 [default]
adapter_windows     [default]
nrt_windows         [default]
dedicated_tasks     (0@[default])
ip_tasks            (0@[default])
us_tasks            (0@[default])
```

3. Reconfigure and restart LIM daemons:

```
$ lsadmin reconfig
```

4. Configure reservation usage of adapter and nrt windows in the `lsb.resources` configuration file. In the `ReservationUsage` section:

```
Begin ReservationUsage
RESOURCE           METHOD
adapter_windows    PER_TASK
nrt_windows        PER_TASK
End ReservationUsage
```

5. (Optional) Create a queue for PE jobs in the `lsb.queues` configuration file. The following sample includes the `hpc_ibm` and `hpc_ibm_tv` queues:

```
Begin Queue
QUEUE_NAME   = hpc_ibm
PRIORITY     = 30
NICE         = 20
#RUN_WINDOW   = 5:19:00-1:8:30 20:00-8:30
#r1m         = 0.7/2.0    # loadSched/loadStop
#r15m         = 1.0/2.5
```

```
#pg            = 4.0/8
#ut            = 0.2
#io            = 50/240
#CPULIMIT      = 180/hostA  # 3 hours of host hostA
#FILELIMIT     = 20000
#DATALIMIT     = 20000      # jobs data segment limit
#CORELIMIT     = 20000
#TASKLIMIT     = 5          # job processor limitEnd of change
#USERS         = all        # users who can submit jobs to this queue
#HOSTS         = all        # hosts on which jobs in this queue can run
#PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v Hey
RES_REQ = select[ poe > 0 ]
EXCLUSIVE = Y
REQUEUE_EXIT_VALUES = 133 134 135
DESCRIPTION  = IBM Platform LSF 9.1 for IBM. This queue is to run POE jobs
ONLY.
End Queue

Begin Queue
QUEUE_NAME   = hpc_ibm_tv
PRIORITY     = 30
NICE         = 20
#RUN_WINDOW   = 5:19:00-1:8:30 20:00-8:30
#r1m          = 0.7/2.0    # loadSched/loadStop
#r15m          = 1.0/2.5
#pg            = 4.0/8
#ut            = 0.2
#io            = 50/240
#CPULIMIT      = 180/hostA  # 3 hours of host hostA
#FILELIMIT     = 20000
#DATALIMIT     = 20000      # jobs data segment limit
#CORELIMIT     = 20000
#TASKLIMIT     = 5          # job processor limitEnd of change
#USERS         = all        # users who can submit jobs to this queue
#HOSTS         = all        # hosts on which jobs in this queue can run
#PRE_EXEC      = /usr/local/lsf/misc/testq_pre >> /tmp/pre.out
#POST_EXEC     = /usr/local/lsf/misc/testq_post |grep -v Hey
RES_REQ = select[ poe > 0 ]
REQUEUE_EXIT_VALUES = 133 134 135
TERMINATE_WHEN = LOAD PREEMPT WINDOW
RERUNNABLE = NO
INTERACTIVE = NO
DESCRIPTION  = IBM Platform LSF 9.1 for IBM debug queue. This queue is to run
POE jobs ONLY.
End Queue
```

6. Reconfigure and restart batch daemons:

```
$ badmin reconfig
```

For more information about this configuration, see the Spectrum LSF manuals that are available at the Enable LSF HPC Features page of the IBM Knowledge Center website.

Configuring GPU support features

Spectrum LSF can be configured to manage graphics processing unit (GPU) resources so that they can be used on areas, such as monitoring, requirement expressions, and usage reservation on job submission.

The following resources are made available as external load indexes by the `elim.gpu` External Load Information Manager (ELIM) program:

- `ngpus`: Total number of GPUs.
- `ngpus_shared`: Number of GPUs in share mode.
- `ngpus_excl_t`: Number of GPUs in exclusive thread mode.
- `ngpus_excl_p`: Number of GPUs in exclusive process mode.

Before you proceed with configuration to export those resources, verify that the `elim.gpu` program is deployed on the directory that is pointed to by the `LSF_SERVERDIR` environment variable. This process is automatically started by the LIM daemon and can be checked by using following command:

```
$ ps -aux | grep elim.gpu
```

Complete the following steps to configure Spectrum LSF:

1. Edit the `lsf.shared` configuration file to add the following resources in the `Resource` section:

```
ngpus      Numeric 60   N          (Number of GPUs)
ngpus_shared  Numeric 60   N          (Number of GPUs in Shared Mode)
ngpus_excl_t  Numeric 60   N          (Number of GPUs in Exclusive Thread Mode)
ngpus_excl_p  Numeric 60   N          (Number of GPUs in Exclusive Process
Mode)
```

2. Map out new resources in the `lsf.cluster.`*`<cluster_name>`* configuration file in the `ResourceMap` section:

```
ngpus             ([default])
ngpus_shared      ([default])
ngpus_excl_t      ([default])
ngpus_excl_p      ([default])
```

3. Enable reservation usage of those resources in the `lsb.resources` configuration file and in the `ReservationUsage` section:

```
ngpus_shared     PER_HOST      N
ngpus_excl_t     PER_HOST      N
ngpus_excl_p     PER_HOST      N
```

4. Reconfigure and restart LIM and the batch daemons:

```
$ lsadmin reconfig
$ badmin reconfig
```

5. Confirm whether LIM now collects the information about GPUs:

```
$ lshosts -l
```

The process that is used to make other GPU-specific resources that are made available by the `elim.gpu.ext` and `elim.gpu.topology` elim programs is not described in this section. For more information about this configuration process, see the Define GPU resources page of the IBM Knowledge Center website.

5.6.15 Synchronize configuration files

It is recommended to add all configuration files to your `osimage synclists` definition. Therefore, all files are up-to-date and in sync across all nodes and images. When you generate or regenerate a stateless image or install a stateful image, the syncfiles are updated. You also can force the syncfile process by using the **`updatenode <node> -F`** command.

For more information, see the Synchronizing Files xCAT documentation page.

This example shows the synchronization of `/etc/hosts` to reduce host name and IP resolution time.

Note: If Service nodes are used to manage you nodes (hierarchy), you should ensure that the service nodes were synchronized with the latest files from the Management Node before installing.

If you have a group of compute nodes (compute) that are going to be installed that are serviced by a service node, run the `updatenode` **`compute -f`** command before the installation to synchronize the current files to the service node. The node range is the compute node names and updatenode will determine which service nodes need updating.

Complete the following steps:

1. Create required folders:

```
$ syncdir=/install/custom/syncfiles/common
$ mkdir -p $syncdir

$ cd $syncdir
$ mkdir etc
$ cd etc
```

2. Create the hosts file (link to `/etc/hosts` in this case):

```
$ ln -s /etc/hosts hosts
$ pwd
/install/custom/syncfiles/common/etc
$ ls -l
lrwxrwxrwx 1 root root 10  1. Dez 15:24 hosts -> /etc/hosts
```

3. Create the `synclist` file:

Note: For more information about the synclist file syntax, see the The Format of synclist file page of the xCAT documentation website.

```
$ synclist=/install/custom/netboot/rhel/rh73-compute.synclist

$ echo "/install/custom/syncfiles/common/etc/* -> /etc/" >> $synclist

$ cat $synclist
/install/custom/syncfiles/common/etc/* -> /etc/
```

4. Add the synclist to the osimage definition:

```
$ lsdef -t osimage rh73-compute-stateless -i synclists
Object name: rh73-compute-stateless
    synclists=
```

```
$ chdef -t osimage rh73-compute-stateless synclists=$synclist
1 object definitions have been created or modified.

$ lsdef -t osimage rh73-compute-stateless -i synclists
Object name: rh73-compute-stateless
    synclists=/install/custom/netboot/rhel/rh73-compute.synclist
```

For more configuration files for a specific node type, create a directory in `/install/custom/syncfiles` and add the directory to the corresponding `synclist` file, as shown in the following example:

`/install/custom/syncfiles/compute/opt.`

As of xCAT 2.12, nodes and node groups can be added to the synclist file. For more information, see the Support nodes in synclist file page of the xCAT documentation website.

5.6.16 Generating and packing the image

Because this image is a stateless image, you must generate and pack the image before providing it. The following steps are needed for a stateless image only:

1. Generate the stateless image by using the **`genimage`** command. It uses the `osimage` definition information to generate the image:

   ```
   $ genimage rh73-compute-stateless
   Generating image:
   <...>
   the initial ramdisk for statelite is generated successfully.
   ```

   This process can take some time. The installation process runs in a chroot environment on the management node. Monitor the building process carefully and watch for possible errors.

2. Check the content of the generated image:

   ```
   $ lsdef -t osimage rh73-compute-stateless -i rootimgdir
   Object name: rh73-compute-stateless
       rootimgdir=/install/netboot/rhels7.3/ppc64le/rh73-compute-stateless
   $ rootimgdir=/install/netboot/rhels7.3/ppc64le/rh73-compute-stateless

   $ ls -l $rootimgdir/rootimg
   lrwxrwxrwx  1 root root    7 30. Nov 12:30 bin -> usr/bin
   dr-xr-xr-x  3 root root  296 30. Nov 12:34 boot
   drwxr-xr-x  2 root root   42  1. Dez 16:33 dev
   drwxr-xr-x 75 root root 8192  1. Dez 16:35 etc
   drwxr-xr-x  2 root root   10 10. Mär 2016  home
   lrwxrwxrwx  1 root root    7 30. Nov 12:30 lib -> usr/lib
   lrwxrwxrwx  1 root root    9 30. Nov 12:30 lib64 -> usr/lib64
   drwxr-xr-x  2 root root   10 10. Mär 2016  media
   drwxr-xr-x  2 root root   10 10. Mär 2016  mnt
   drwxr-xr-x 11 root root  180  1. Dez 16:32 opt
   drwxr-xr-x  2 root root   10 30. Nov 12:29 proc
   dr-xr-x---  2 root root   78 30. Nov 12:34 root
   drwxr-xr-x 12 root root  234  1. Dez 16:22 run
   lrwxrwxrwx  1 root root    8 30. Nov 12:30 sbin -> usr/sbin
   drwxr-xr-x  2 root root   10 10. Mär 2016  srv
   drwxr-xr-x  2 root root   10 30. Nov 12:29 sys
   ```

```
drwxrwxrwt  7 root root  245  1. Dez 16:34 tmp
drwxr-xr-x 14 root root  222 30. Nov 12:45 usr
drwxr-xr-x 19 root root 4096 30. Nov 12:45 var
drwxr-xr-x  6 root root 4096  1. Dez 16:35 xcatpost
```

Note: After your image is generated, you can **chroot** to the image, install any other software, or modify the files manually. However, it is recommended that the installation steps are done by using all of the mechanisms that are described in 5.3.6, “xCAT OS installation types: Disks and state” on page 205 because it ensures that a reproducible image is produced.

3. Pack the generated image with the **packimage** command:

```
$ packimage rh73-compute-stateless
Packing contents of
/install/netboot/rhels7.3/ppc64le/rh73-compute-stateless/rootimg
archive method:cpio
compress method:gzip
```

Note: The packimage process can be accelerated by installing the parallel compression tool pigz on the management node. For more information, see the Accelerating the diskless initrd and rootimg generating page of the xCAT documentation website.

5.6.17 Node provisioning

To start provisioning a compute node (or node group), and load the OS and software stack according to the osimage and node group objects, complete the following steps:

1. Define the osimage attribute of a node (or node group) by using the **nodeset** command:

```
$ nodeset p8r1n1 osimage=rh73-compute-stateless
p8r1n1: netboot rhels7.3-ppc64le-compute
```

2. You can verify the changes to the nodes attributes by using the **lsdef** command:

```
$ lsdef p8r1n1
Object name: p8r1n1
    <...>
    initrd=xcat/osimage/rh73-compute-stateless/initrd-stateless.gz
    <...>
kcmdline=imgurl=http://!myipfn!:80//install/netboot/rhels7.3/ppc64le/rh73-compu
te-stateless/rootimg.cpio.gz XCAT=!myipfn!:3001 NODE=p8r1n1 FC=0
BOOTIF=70:e2:84:14:09:ae
    kernel=xcat/osimage/rh73-compute-stateless/kernel
    <...>
    os=rhels7.3
    <...>
    profile=compute
    provmethod=rh73-compute-stateless
    <...>
```

3. Set the boot method to network by using the **rsetboot** command (optional):

```
$ rsetboot p8r1n1 net
```

Note: This process is performed automatically by the **nodeset** command, and is not required if the bootloader configuration for automatic boot is correct. For more information, see 5.2.2, “Boot order configuration” on page 196.

4. (Optional) Reboot the node (or node group) by using the **rpower** command:

```
$ rpower p8r1n1 reset
```

> **Note:** This process is performed automatically (within some time) if the Genesis image for node discovery is still running in the compute node (waiting for instructions from the Management Node).

5. You can watch the node's console by using the **rcons** command:

```
$ rcons p8r1n1
```

6. You can monitor the node boot progress in the node object and the `/var/log/messages` log file:

```
$ lsdef  p8r1n1 -i status
Object name: p8r1n1
    status=powering-on

$ lsdef  p8r1n1 -i status
Object name: p8r1n1
    status=netbooting

lsdef  p8r1n1 -i status
Object name: p8r1n1
    status=booted

$tail -f /var/log/messages | grep p8r1n1
<...>
```

5.6.18 Postinstallation verification

In this section, we describe the steps that are used to verify that the software stack is correctly provisioned.

> **Note:** For more information about checks, see Chapter 6, "Cluster monitoring and health checking" on page 289.

Verifying the CUDA Toolkit

Verify all GPUs are listed by using the **nvidia-smi** command:

```
$ xdsh p8r1n1 'nvidia-smi --list-gpus'
p8r1n1: GPU 0: Tesla P100-SXM2-16GB (UUID:
GPU-f8af01bb-b803-fe8e-b99b-692c5f0dd1bc)
p8r1n1: GPU 1: Tesla P100-SXM2-16GB (UUID:
GPU-8d457a4f-8b65-357f-8a53-0df1e14747c7)
p8r1n1: GPU 2: Tesla P100-SXM2-16GB (UUID:
GPU-8ec0ffa5-f00c-cf0f-1893-ad189a808ec2)
p8r1n1: GPU 3: Tesla P100-SXM2-16GB (UUID:
GPU-c614020d-48eb-48cc-8a12-7e77312f7312)
```

Verifying the Mellanox OFED

To verify the installation of Mellanox OFED, complete the following steps:

1. Verify that the `openibd` service is correctly loaded and active by using the **`systemctl`** command:

```
xdsh p8r1n1 'systemctl status openibd'
p8r1n1: * openibd.service - openibd - configure Mellanox devices
p8r1n1:    Loaded: loaded (/usr/lib/systemd/system/openibd.service; enabled;
vendor preset: disabled)
p8r1n1:    Active: active (exited) since Tue 2016-11-29 21:43:14 UTC; 3min 46s
ago
p8r1n1:      Docs: file:/etc/infiniband/openib.conf
p8r1n1:   Process: 6936 ExecStop=/etc/init.d/openibd stop (code=exited,
status=0/SUCCESS)
p8r1n1:   Process: 7190 ExecStart=/etc/init.d/openibd start bootid=%b
(code=exited, status=0/SUCCESS)
p8r1n1:  Main PID: 7190 (code=exited, status=0/SUCCESS)
p8r1n1:    CGroup: /system.slice/openibd.service
p8r1n1:
p8r1n1: <...> systemd[1]: Starting openibd - configure Mellanox devices...
p8r1n1: <...> openibd[7190]: Loading HCA driver and Access Layer:[  OK  ]
p8r1n1: <...> systemd[1]: Started openibd - configure Mellanox devices.
```

2. Verify the information and status of the InfiniBand adapter and ports by using the **`ibstat`** command:

```
xdsh p8r1n1 ibstat
p8r1n1: CA 'mlx5_0'
p8r1n1:     CA type: MT4115
p8r1n1:     Number of ports: 1
p8r1n1:     Firmware version: 12.17.2010
p8r1n1:     Hardware version: 0
p8r1n1:     Node GUID: 0xe41d2d03006751b2
p8r1n1:     System image GUID: 0xe41d2d03006751b2
p8r1n1:     Port 1:
p8r1n1:         State: Active
p8r1n1:         Physical state: LinkUp
p8r1n1:         Rate: 100
p8r1n1:         Base lid: 30
p8r1n1:         LMC: 0
p8r1n1:         SM lid: 1
p8r1n1:         Capability mask: 0x2651e848
p8r1n1:         Port GUID: 0xe41d2d03006751b2
p8r1n1:         Link layer: InfiniBand
p8r1n1: CA 'mlx5_1'
p8r1n1:     CA type: MT4115
p8r1n1:     Number of ports: 1
p8r1n1:     Firmware version: 12.17.0222
p8r1n1:     Hardware version: 0
p8r1n1:     Node GUID: 0xe41d2d0300f2a584
p8r1n1:     System image GUID: 0xe41d2d0300f2a584
p8r1n1:     Port 1:
p8r1n1:         State: Active
p8r1n1:         Physical state: LinkUp
p8r1n1:         Rate: 100
p8r1n1:         Base lid: 38
p8r1n1:         LMC: 0
p8r1n1:         SM lid: 1
p8r1n1:         Capability mask: 0x2651e848
```

```
p8r1n1:         Port GUID: 0xe41d2d0300f2a584
p8r1n1:         Link layer: InfiniBand
```

Verifying installed runtime libraries and MPI

Verify the installed version of all runtime libraries and MPI by using the **rpm** command:

```
$ xdsh p8r1n1 'rpm -qa | \
   grep -E "xlc|xlf|advance-toolchain|ppedev|smpi|essl|pessl"'
p8r1n1: ibm_smpi-10.1.0.2-rh7.ppc64le
p8r1n1: pessl-license-5.3.0-0.noarch
p8r1n1: essl.3264.rte-5.5.0-0.ppc64le
p8r1n1: essl.rte-5.5.0-0.ppc64le
p8r1n1: xlf.rte-compute-15.1.5-0.noarch
p8r1n1: ppedev_mrnet-2.3.0-0.ppc64le
p8r1n1: pessl.3264.rte-5.3.0-0.ppc64le
p8r1n1: essl.rte.common-5.5.0-0.ppc64le
p8r1n1: pessl.msg-5.3.0-0.ppc64le
p8r1n1: essl.6464.rte-5.5.0-0.ppc64le
p8r1n1: ppedev.compute-2.3.0-0.noarch
p8r1n1: pessl.license-5.3.0-0.ppc64le
p8r1n1: ppedev_runtime-2.3.0-0.ppc64le
p8r1n1: libxlc-13.1.5.0-161028a.ppc64le
p8r1n1: libxlf-15.1.5.0-161028a.ppc64le
p8r1n1: pessl.rte.common-5.3.0-0.ppc64le
p8r1n1: essl.3264.rtecuda-5.5.0-0.ppc64le
p8r1n1: xlc-license.13.1.5-13.1.5.0-161028a.ppc64le
p8r1n1: pessl-computenode-5.3.0-0.noarch
p8r1n1: gcc-gfortran-4.8.5-11.el7.ppc64le
p8r1n1: ibm_smpi_lic_s-10.1-rh7.ppc64le
p8r1n1: xlf-license.15.1.5-15.1.5.0-161028a.ppc64le
p8r1n1: essl-computenode-5.5.0-0.noarch
p8r1n1: advance-toolchain-at10.0-runtime-10.0-1.ppc64le
p8r1n1: ppedev.license-2.3.0-0.noarch
p8r1n1: xlf.license-compute-15.1.5-0.noarch
p8r1n1: advance-toolchain-at10.0-mcore-libs-10.0-1.ppc64le
p8r1n1: pessl.common-5.3.0-0.ppc64le
p8r1n1: ppedev_license-2.3.0-0.ppc64le
p8r1n1: essl.common-5.5.0-0.ppc64le
p8r1n1: essl-license-5.5.0-0.noarch
p8r1n1: libgcc-4.8.5-11.el7.ppc64le
p8r1n1: essl.license-5.5.0-0.ppc64le
p8r1n1: essl.msg-5.5.0-0.ppc64le
p8r1n1: xlc.license-compute-13.1.5-0.noarch
p8r1n1: xlc.rte-compute-13.1.5-0.noarch
```

Verifying Spectrum Scale

Complete the following steps:

1. Verify the Spectrum Scale packages are installed by using the **rpm** command:

```
$ xdsh p8r1n1 'rpm -qa | grep ^gpfs'
p8r1n1: gpfs.gskit-8.0.50-47.ppc64le
p8r1n1: gpfs.gpl-4.1.1-3.noarch
p8r1n1: gpfs.ext-4.1.1-3.ppc64le
p8r1n1: gpfs.docs-4.1.1-3.noarch
p8r1n1: gpfs.msg.en_US-4.1.1-3.noarch
```

```
p8r1n1: gpfs.base-4.1.1-3.ppc64le
```

2. Verify that the node does not yet belong to any cluster by using the **mmlscluster** command:

```
$ xdsh p8r1n1 'mmlscluster'
p8r1n1: mmlscluster: This node does not belong to a GPFS cluster.
p8r1n1: mmlscluster: Command failed. Examine previous error messages to
determine cause.
```

(Optional) Checking public and site network connectivity

Verify the connectivity to external and non-cluster nodes by using the **ping** command:

```
$ xdsh p8r1n1 'ping -c1 example.com'
p8r1n1: PING example.com (93.184.216.34) 56(84) bytes of data.
p8r1n1: 64 bytes from 93.184.216.34: icmp_seq=1 ttl=46 time=3.94 ms
<...>
```

5.7 xCAT Login Nodes (stateful)

The process that is used to deploy an xCAT login node is similar to the xCAT compute node deployment. The main difference is the particular software stack components that are used on each node type. Also, other methods need some change because login nodes often use a `stateful` image.

Therefore, this section does not describe the deployment instructions for login nodes. Instead, it describes some examples of the differences in the software stack components and methods between login and compute nodes.

The following differences are typically considered for the login nodes:

- Use the `rhels7.3-ppc64le-`**`install`**`-compute osimage` as an initial template.
- Consider the configuration of RAID and disk partitions. For more information, see the following pages of the xCAT documentation website:
  - Configure RAID before deploying the OS
  - Configure Disk Partition
- CUDA Toolkit often is not required because the extra compute-related hardware is not present on login nodes.
- The Mellanox OFED installation process is slightly different. Use the `postscripts node` attribute instead of the `postinstall osimage` attribute to start the mlnxofed_ib_install script. For more information, see the Diskful Installation page of the XCAT documentation website.
- The XL C/C++ and Fortran compilers use the login nodes to compile applications. Therefore, the compiler package must be installed. Add the following appropriate xCAT software kits:

```
Object name: xlc.compiler-compute-13.1.5-0-rhels-7-ppc64le
   description=XLC13 for compiler kitcomponent

Object name: xlf.compiler-compute-15.1.5-0-rhels-7-ppc64le
   description=XLF15 for compiler kitcomponent
```

- Advance Toolchain needs at least the `advance-toolchain-at10.0-`**`devel`**`-10.0-1.ppc64le.rpm` package for compiler support. It is also recommended to install `advance-toolchain-at10.0-`**`perf`**`-10.0-1.ppc64le.rpm`.

- The PGI compiler installation is different because the entire PGI compiler stack is needed. Complete the following steps to install it:

  a. Ensure that your `pkglist` includes the following packages because they are a requirement for PGI compilers:

     ```
     gcc
     gcc-c++
     ```

  b. Create a `postscript` as shown in the following example to perform an unattended installation:

     ```
     $ script=/install/postscripts/install_pgi_login
     $ cat /install/postscripts/install_pgi_login
        #!/bin/bash

        INSTALL_TAR="/install/software/compilers/pgi/pgilinux-2016-1610-ppc64le.t
        ar.gz"
        INSTALL_PATH="/opt/pgi"
        PGI_PATH="/opt/pgi/linuxpower/16.10"
        PROFILE_FILENAME="/etc/profile.d/xcat-pgi.sh"

        # environment variables for unattended install
        export PGI_SILENT="true"
        export PGI_ACCEPT_EULA="accept"
        export PGI_INSTALL_DIR="$INSTALL_PATH"
        export PGI_INSTALL_TYPE="single"

        # get tar and extract it
        dir=/tmp/pgi_install
        mkdir $dir
        cd $dir
        wget -l inf -N --waitretry=10 --random-wait --retry-connrefused -t 10 -T
        60 -nH --no-parent "http://$MASTER/$INSTALL_TAR" 2> wget.log
        tar -xf "${INSTALL_TAR##*/}"

        # do installation
        ./install

        # set the paths required for pgi
        echo "export PGI=$INSTALL_PATH" > "${PROFILE_FILENAME}"
        echo "export PATH=$PGI_PATH/bin:\$PATH" >> "${PROFILE_FILENAME}"

        cd ..
        # comment for debugging
        rm -rf $dir

     $ chmod +x $script
     $ chdef -t <login_node> --plus postscripts=install_pgi_login
     ```

- The kits for PPT, ESSL, and PESSL provide kit components that are specifically for login nodes, for example:

```
Object name: ppedev.login-2.3.0-0-rhels-7.3-ppc64le
   description=Parallel Performance Toolkit for login nodes

Object name: essl-loginnode-5.5.0-0-rhels-7.3-ppc64le
   description=essl for login nodes

Object name: pessl-loginnode-5.3.0-0-rhels-7.3-ppc64le
   description=pessl for login nodes
```

- The parallel file system (IBM Spectrum Scale) system can allow the login node to access the data that is provided to or produced by the applications that are running in the compute nodes.
- The job scheduler (Spectrum LSF) is typically required to submit jobs from the login nodes to the compute nodes.

6

Cluster monitoring and health checking

Monitoring is an important task that helps in maintaining cluster resources and ensures its serviceability to users. Monitoring involves resource control and health check tasks on the various nodes (compute, login, and services), networking devices, and storage systems.

This chapter introduces some tools and resources that can be employed to support monitoring of a high-performance computing (HPC) cluster.

This chapter includes the following topics:

6.1 Basic commands

This chapter describes the following basic commands that can be used to check the overall hardware status of a cluster. (This information is for reference only.) For information, see 6.5, “Diagnostic and health check framework” on page 310:

- The **rpower** command controls the power of the nodes remotely. The most typically used power commands include: on, off, boot, and status.

  The **rpower** command is run from the xCAT management node, as shown in Example 6-1.

  Example 6-1 The rpower command

```
# rpower <nodelist> [power control option]
$ rpower c931f04p30,c931f04p32,c931f04p34 on
c931f04p30: on
c931f04p32: on
c931f04p34: on
```

- The **rinv** command shows inventory details regarding the various components of the specified node: firmware versions, hardware serial numbers, and hardware types.

  Use this command to verify details regarding the hardware inventory. This command is run from the xCAT management node, as shown in Example 6-2.

  Example 6-2 The rinv command

```
# rinv <nodelist> [inventory types]
$ rinv c931f04p30 firmware
c931f04p30: BMC Firmware: 2.13
```

- The **rvitals** command shows the temperature, voltage, and general vital statistics of the nodes that are specified in the command.

  Use this command to determine the environmental data regarding your node. This command is run from the xCAT management node, as shown in Example 6-3.

  Example 6-3 The rvitals command

```
# rvitals <nodelist> [vital types]
$ rvitals c931f04p30 temp
c931f04p30: Ambient Temp: 16 C (61 F)
c931f04p30: CPU 1 VDD Temp: 46 C (115 F)
c931f04p30: CPU 2 VDD Temp: 40 C (104 F)
...
```

- The **nodestat** command shows the current running status for the supplied nodes: Running, noping, or any number of deployment statuses.

  Use this command to determine the state of your nodes after a reboot or if the node appears to be inaccessible. This command is run from the xCAT management node, as shown in Example 6-4.

  Example 6-4 The nodestat command

```
# nodestat <nodelist>
$ nodestat c931f04p30,c931f04p32
c931f04p30: sshd
c931f04p32: sshd
```

- The **nodels** command allows you to query the xCAT database for that node. After listing the nodes supply either the table or table.column.

  Use this command to determine the last recorded details for the nodes supplied. This command is run from the xCAT management node as shown in Example 6-5.

Example 6-5 The nodels command

```
# nodels <nodelist> [table.column]
$ nodels c931f04p[30,32] nodelist.status
c931f04p30: powering-on
c931f04p32: powering-on
```

The following commands are basic Linux commands that are used to query the hardware inventory. These commands must be run on the compute node or by using the **xdsh** command:

- The **lscpu** command shows basic information about the processor, such as the number of sockets, cores, threads, NUMA nodes, and cache size. Sample output of this command is shown in Example 6-6.

Example 6-6 The lscpu command

```
$ lscpu
Architecture:          ppc64le
Byte Order:            Little Endian
CPU(s):                160
On-line CPU(s) list:   0-159
Thread(s) per core:    8
Core(s) per socket:    10
Socket(s):             2
NUMA node(s):          2
Model:                 8335-GTB
L1d cache:             64 K
L1i cache:             32 K
L2 cache:              512 K
L3 cache:              8192 K
NUMA node0 CPU(s):     0-79
NUMA node1 CPU(s):     80-159
```

- The **free** command shows the amount of RAM a node, as shown in Example 6-7.

Example 6-7 The free command

```
$ free -h
              total        used        free      shared  buff/cache   available
Mem:           1.0T        6.4G        1.0T        21M        2.0G        1.0T
Swap:           15G          0B         15G
```

- By using the **ibstat** command, you can check information that is related to the InfiniBand adapter, including firmware, link status, and link speed. Sample output of this command is shown in Example 6-8.

Example 6-8 The ibstat command

```
$ ibstat
CA 'mlx5_0'
   CA type: MT4115
   Number of ports: 1
   Firmware version: 12.16.1006
   Hardware version: 0
   Node GUID: 0x7cfe900300576fcc
   System image GUID: 0x7cfe900300576fcc
   Port 1:
      State: Active
      Physical state: LinkUp
      Rate: 100
      Base lid: 6
      LMC: 0
      SM lid: 1
      Capability mask: 0x2651e848
      Port GUID: 0x7cfe900300576fcc
      Link layer: InfiniBand
```

For more information about NVIDIA GPUs, see 6.4, “Using nvidia-smi tool for GPU monitoring” on page 302.

6.2 IBM Spectrum LSF tools for job monitoring

IBM Spectrum Load Sharing Facility (LSF) provides a comprehensive set of tools that can be implemented for monitoring the cluster. This section describes how to use some of these tools to complete common tasks. The following commands are used in this section:

- `lsclusters`: Lists all configured clusters.
- `lsid`: Shows the current Spectrum LSF version number, the cluster name, and master host name.
- `lshosts`: Displays hosts information.
- `lsload`: Displays load-per-host information.
- `bhosts`: Displays information about batches.
- `bjobs`: Displays information about jobs.
- `bqueues`: Displays information about batch queues.
- `bparams`: Displays batch parameters.
- `badmin`: Provides a set of administrative subcommands for batches.
- `lsadmin`: Provides a set of administrative subcommands for hosts.

For more information about all of the available commands and usage guides, see the IBM Spectrum LSF Command Reference page at the Spectrum LSF manual website.

6.2.1 General information about clusters

The **lsclusters** command shows information about the clusters that are managed by the Spectrum LSF master host server, as shown in Example 6-9.

Example 6-9 Spectrum LSF lsclusters command to gather information about the clusters

```
$ lsclusters
CLUSTER_NAME   STATUS   MASTER_HOST              ADMIN    HOSTS  SERVERS
lsf-cluster    ok       p8r1n1                lsfadmin        2        2
```

As shown in Example 6-10, the **lsid** command is used to display the cluster name, master host server, and version of Spectrum LSF.

Example 6-10 Spectrum LSF lsid command to display cluster name and master host

```
$ lsid
IBM Platform LSF Standard 10.1.0.1, Oct 31 2016
Copyright IBM Corp. 1992, 2014. All rights reserved.
US Government Users Restricted Rights - Use, duplication or disclosure restricted
by GSA ADP Schedule Contract with IBM Corp.

My cluster name is lsf-cluster
My master name is p8r1n1
```

Use the **badmin** command with the **showstatus** flag (as shown in Example 6-11) to get an overview of servers, users, groups, and jobs in the cluster. Notice that the **badmin** command is a suite of commands to manage batch-related configuration and daemons. For more information about its features, see “Managing batch services” on page 298 and “Managing the LIM and Remote Execution Server services” on page 300.

Example 6-11 Spectrum LSF badmin command to show master host batch daemon status

```
$ badmin showstatus
LSF runtime mbatchd information
    Available local hosts (current/peak):
        Clients:                0/0
        Servers:                2/2
           CPUs:                2/2
         Cores:                 40/40
         Slots:                 unlimited/unlimited

    Number of servers:          2
        Ok:                     2
        Closed:                 0
        Unreachable:            0
        Unavailable:            0

    Number of jobs:             1
        Running:                1
        Suspended:              0
        Pending:                0
        Finished:               0

    Number of users:            4
    Number of user groups:      1
```

```
Number of active users:     1

Latest mbatchd start:       Thu Jan 10 21:21:33 2017
Active mbatchd PID:         4617

Latest mbatchd reconfig:    -
```

6.2.2 Getting information about hosts

The Spectrum LSF master host obtains static and dynamic information about the subordinate hosts of the cluster. Static information means properties that are difficult to change, such as system memory, disk capacity, topology, and the number of CPUs. In contrast, dynamic information depends on current workload on the system, such as the amount of memory that is used, the amount of swap memory, and jobs that are scheduled to run.

By default, the commands that are highlighted in this section display information about all hosts in the cluster, unless a host name is specified. Also, a subset of hosts can be selected by using *resources requirement expressions*. Example 6-12 shows an expression (specified as an argument of the **-R** option) to select hosts whose model matches `POWER8`.

Example 6-12 Spectrum LSF lshosts command to show subordinate hosts selected by the expression

```
$ lshosts -R "select[model==POWER8]"
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r2n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

Resource requirements can also be used in scheduling of jobs and with other Spectrum LSF commands. For more information about the language to express requirements, see the About resource requirement strings section of the Spectrum LSF manual.

By default, the **lshost** command displays static information about all hosts that are configured in the local cluster as shown in Example 6-13. The columns `cpuf`, `ncpus`, `maxmem`, and `maxswp` display the CPU performance factor, number of CPUs, maximum memory, and maximum swap memory.

Example 6-13 Spectrum LSF lshosts command shows static information of hosts

```
$ lshosts
HOST_NAME      type    model  cpuf ncpus maxmem maxswp server RESOURCES
p8r1n1      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
p8r2n2      LINUXPP   POWER8 250.0   160   256G   3.9G    Yes (mg)
```

More information about individual hosts can be queried by passing the **-l** option to the **lshost** command, as shown in Example 6-14. Notice that in addition to the static information, the **-l** option reports the current load status of the host.

Example 6-14 Spectrum LSF lshosts command displays static information about a specified host

```
$ lshosts -l p8r2n2

HOST_NAME:  p8r2n2
type         model  cpuf ncpus ndisks maxmem maxswp maxtmp  rexpri server nprocs ncores nthreads
LINUXPPC64  POWER8 250.0   160      1   256G   3.9G 949020M       0    Yes       1     20        8

RESOURCES: (mg)
```

```
RUN_WINDOWS:  (always open)

LOAD_THRESHOLDS:
  r15s   r1m  r15m   ut    pg    io   ls   it   tmp   swp   mem   pnsd pe_network
     -   3.5     -    -     -     -    -    -     -     -     -      -          -
```

The **lshosts** command with **-T** option shows the nonuniform memory access (NUMA) topology of the hosts. In Example 6-15, the **lshosts** command reports a two-node (0 and 8) S822LC host (p8r2n2), with 128 GB of memory available per node (for a total of 256 GB). It also shows 10 processor cores per NUMA node, each with 8 CPUs (SMT8 mode).

Example 6-15 Spectrum LSF lshosts command shows NUMA topology of a specified host

```
$ lshosts -T p8r2n2
Host[256G] p8r2n2
    NUMA[0: 128G]
        core(0 1 2 3 4 5 6 7)
        core(8 9 10 11 12 13 14 15)
        core(16 17 18 19 20 21 22 23)
        core(24 25 26 27 28 29 30 31)
        core(32 33 34 35 36 37 38 39)
        core(40 41 42 43 44 45 46 47)
        core(48 49 50 51 52 53 54 55)
        core(56 57 58 59 60 61 62 63)
        core(64 65 66 67 68 69 70 71)
        core(72 73 74 75 76 77 78 79)
    NUMA[8: 128G]
        core(80 81 82 83 84 85 86 87)
        core(88 89 90 91 92 93 94 95)
        core(96 97 98 99 100 101 102 103)
        core(104 105 106 107 108 109 110 111)
        core(112 113 114 115 116 117 118 119)
        core(120 121 122 123 124 125 126 127)
        core(128 129 130 131 132 133 134 135)
        core(136 137 138 139 140 141 142 143)
        core(144 145 146 147 148 149 150 151)
        core(152 153 154 155 156 157 158 159)
```

In contrast to the **lshost** command that provides only static information, the **lsload** command gives dynamic information for the hosts. Example 6-16 shows load information for all hosts in the cluster.

Example 6-16 Spectrum LSF lsload command reports dynamic information for the hosts

```
$ lsload
HOST_NAME       status  r15s   r1m  r15m   ut    pg  ls    it   tmp   swp   mem
p8r2n2              ok   0.1   0.0   0.0   0%   0.0   1     0  920G  3.9G  246G
p8r1n1              ok   1.0   0.0   0.1   0%   0.0   0  8518  920G  3.9G  245G
```

The report features the following column headings:

- status reports the status of the host. This column features the following possible values:
  - ok: The host is ready to accept remote jobs.
  - -ok: The LIM daemon is running, but the RES daemon is unreachable.
  - busy: The host is overloaded.
  - lockW: The host is locked by its run window.

  - `lockU`: The host is locked by the LSF administrator or root.
  - `unavail`: The host is down or the LIM daemon is not running.
- `r15s`, `r1m`, and `r15m` indicate that the CPU load averaged exponentially over the last 15 seconds, 1 minute, and 15 minutes, respectively.
- `ut` is the CPU utilization time that is averaged exponentially over the last minute.
- `pg` is the memory paging rate that is averaged exponentially over the last minute.
- `ls` is the number of current login users.
- `it` shows the idle time of the host.
- `tmp` is the amount of free space in the `/tmp` directory.
- `swp` is the amount of free swap space.
- `mem` is the amount of memory available on the host.

Other options can be used with the **`lsload`** command to display different information. For example, the **`-l`** option displays network resources information for scheduling IBM Parallel Environment (PE) jobs and also the disk I/O rate.

6.2.3 Getting information about jobs and queues

Some commands are available to monitor the workload in the cluster and give information about jobs and queue status. The **`bhosts`** command reports jobs statistics per-host, which is useful to see the overall cluster workload, as shown in Example 6-17.

Example 6-17 Spectrum LSF bhosts to report on hosts batch jobs status

```
$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
p8r1n1             ok              -      -      1      1      0      0      0
p8r2n2             ok              -      -      0      0      0      0      0
```

To show unfinished jobs in the entire cluster, use the **`bjobs`** command (see Example 6-18). You can use the **`-a`** option with the **`bjobs`** command to view all recently finished jobs and those jobs that are still running.

Example 6-18 Spectrum LSF bjobs command to show unfinished jobs

```
$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
209     wainers RUN   normal     p8r1n1      p8r1n1      serial     Jan 10 17:18
```

The **`bjobs`** command can also be used to report more information about a job, as shown in Example 6-19.

Example 6-19 Spectrum LSF bjobs command to show detailed information about a job

```
$ bjobs -l 209

Job <209>, Job Name <serial>, User <wainersm>, Project <default>, Status <RUN>,
                     Queue <normal>, Command <#!/bin/sh; #BSUB -o %J.out -e %J
                    .err;#BSUB -J serial; ./noop>, Share group charged </waine
                    rsm>
Thu Apr 14 17:18:41: Submitted from host <p8r1n1>, CWD <$HOME/lsf/noop>, Output
                     File <209.out>, Error File <209.err>;
Thu Apr 14 17:18:42: Started 1 Task(s) on Host(s) <p8r1n1>, Allocated 1 Slot(s)
```

```
                     on Host(s) <p8r1n1>, Execution Home </home/wainersm>, Exe
                    cution CWD </home/wainersm/lsf/noop>;
Thu Apr 14 17:22:41: Resource usage collected.
                    MEM: 22 Mbytes;  SWAP: 0 Mbytes;  NTHREAD: 5
                    PGID: 82953;  PIDs: 82953 82954 82958 82959


 MEMORY USAGE:
 MAX MEM: 22 Mbytes;  AVG MEM: 22 Mbytes

 SCHEDULING PARAMETERS:
           r15s   r1m  r15m   ut      pg    io   ls    it    tmp    swp    mem
 loadSched   -     -     -     -       -     -    -     -     -      -      -
 loadStop    -     -     -     -       -     -    -     -     -      -      -

 RESOURCE REQUIREMENT DETAILS:
 Combined: select[type == local] order[r15s:pg]
 Effective: select[type == local] order[r15s:pg]
```

The **bqueues** command displays information about the queues of the jobs that are available in a cluster along with use statistics. The command list queues are sorted by priority (PRIO). It also reports their status (STATUS), maximum number of job slots (MAX), maximum jobs slots per users (JL/U), maximum job slots per processors (JL/P), maximum job slots per slot (JL/H), number of tasks for jobs (NJOBS), and number of jobs pending (PEN), running (RUN) and suspended (SUSP).

Example 6-20 shows the output of a **bqueues** command.

Example 6-20 Spectrum LSF bqueues command to show available job queues

```
$ bqueues
QUEUE_NAME      PRIO STATUS          MAX JL/U JL/P JL/H NJOBS  PEND   RUN  SUSP
admin             50  Open:Active      -    -    -    -     0     0     0     0
owners            43  Open:Active      -    -    -    -     0     0     0     0
priority          43  Open:Active      -    -    -    -     0     0     0     0
night             40  Open:Inact       -    -    -    -     0     0     0     0
chkpnt_rerun_qu   40  Open:Active      -    -    -    -     0     0     0     0
short             35  Open:Active      -    -    -    -     0     0     0     0
license           33  Open:Active      -    -    -    -     0     0     0     0
normal            30  Open:Active      -    -    -    -     1     0     1     0
interactive       30  Open:Active      -    -    -    -     0     0     0     0
hpc_linux         30  Open:Active      -    -    -    -     0     0     0     0
hpc_linux_tv      30  Open:Active      -    -    -    -     0     0     0     0
idle              20  Open:Active      -    -    -    -     0     0     0     0
```

6.2.4 Administering the cluster

High-performance computing (HPC) cluster administration involves many non-trivial tasks, although tools, such as Spectrum LSF, simplify the process. This section describes a small part of the Spectrum LSF capabilities and tools. For more information, see the Administering and Configuring IBM Platform LSF page of the IBM Knowledge Center website.

The Spectrum LSF configuration directory has a few files in which global and per-cluster basis configurations are defined. Its path depends on the installation that is performed by the administrator and the LSF_ENVDIR environment variable value, as shown in the following example:

```
$ echo $LSF_ENVDIR
/usr/share/lsf/conf
```

The bare minimum Spectrum LSF configuration directory includes the following files:

- lsf.shared: Contains the cluster names and definitions that can be referenced by configuration files.
- lsf.conf: Contains global configurations that are shared among clusters defined on lsf.shared.
- lsf.cluster.<cluster-name>: Contains per cluster name-specific configurations, such as the list of hosts and their attributes, administrators, and resources mapping.

Other configuration files are available; for example, lsf.queue is the file where batch queues are defined.

Typically, any change requires restarting the Spectrum LSF servers and daemons; therefore, these changes are often followed by running the **badmin reconfig** and the **lsadmin reconfig** commands so that the new configurations take effect.

Managing batch services

To check all batch configuration parameters, use the **bparams** command. Without any options, it displays basic information, as shown in Example 6-21. MBD_SLEEP_TIME is the jobs dispatch interval.

Example 6-21 Spectrum LSF bparams command to show basic batch configuration parameters

```
$ bparams
Default Queues:  normal interactive
MBD_SLEEP_TIME used for calculations:  10 seconds
Job Checking Interval:  7 seconds
Job Accepting Interval:  0 seconds
```

Example 6-22 lists configured batch parameters with the **bparams -l** command. Because the list of parameters is long, many of them are omitted from the output, as shown in Example 6-22.

Example 6-22 Spectrum LSF bparams to show detailed batch configuration parameters

```
$ bparams -l

System default queues for automatic queue selection:
    DEFAULT_QUEUE = normal interactive

Amount of time in seconds used for calculating parameter values:
    MBD_SLEEP_TIME = 10 (seconds)

The interval for checking jobs by slave batch daemon:
    SBD_SLEEP_TIME = 7 (seconds)

The interval for a host to accept two batch jobs:
    JOB_ACCEPT_INTERVAL = 0 (* MBD_SLEEP_TIME)
```

```
The idle time of a host for resuming pg suspended jobs:
    PG_SUSP_IT = 180 (seconds)

The amount of time during which finished jobs are kept in core memory:
    CLEAN_PERIOD = 3600 (seconds)

The maximum number of retries for reaching a slave batch daemon:
    MAX_SBD_FAIL = 3

<... Omitted output ...>
Resets the job preempted counter once this job is requeued, migrated, or rerun:
    MAX_JOB_PREEMPT_RESET = Y

Disable the adaptive chunking scheduling feature:
    ADAPTIVE_CHUNKING = N
```

Spectrum LSF uses default batch parameters unless the (optional) `lsb.params` file is deployed in `$LSF_ENVDIR/<cluster-name>/configdir` as a per-cluster configuration file. After any change to `lsb.params`, run the **`badmin reconfig`** command to reconfigure the mbatchd daemon.

The **`badmin`** administrative tool provides some debugging subcommands that can help determine problems with batch daemons. The following commands can be run with the Spectrum LSF administrator user:

- `sbddebug`: Debug the slave batch daemon
- `mbddebug`: Debug the master batch daemon
- `schddebug`: Debug the scheduler batch daemon

As a case study, Example 6-23 shows the output of the **`bhosts`** command, in which the `p8r2n2` host is marked as unreachable.

Example 6-23 Spectrum LSF bhosts command to determine problem on batch services

```
$ bhosts
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
p8r1n1             ok              -      -      0      0      0      0      0
p8r2n2             unreach         -      -      0      0      0      0      0
```

From the master host (`p8r1n1`), **`badmin mbddebug`** nor **`badmin schddebug`** reported errors. Now, **`badmin sbddebug`** shows that the batch daemon is unreachable:

```
$ badmin sbddebug p8r2n2
failed : Slave batch daemon (sbatchd) is unreachable now on host p8r2n2
```

These issues are fixed by restarting the slave batch daemon, as shown in Example 6-24.

Example 6-24 Spectrum LSF badmin command to start slave batch daemon

```
$ badmin hstartup p8r2n2
Start up slave batch daemon on <p8r2n2> ? [y/n] y
Start up slave batch daemon on <p8r2n2> ...... done
```

As a result, the slave batch daemon is back online again:

```
$ bhosts p8r2n2
HOST_NAME          STATUS       JL/U    MAX  NJOBS    RUN  SSUSP  USUSP    RSV
p8r2n2             ok              -      -      0      0      0      0      0
```

Managing the LIM and Remote Execution Server services

Spectrum LSF deploys the Load Information Manager (LIM) and Remote Execution Server servers in each host, where they run as daemons. These servers play the following roles in the Spectrum LSF system:

- LIM collects load and configuration information about the host.
- Remote Execution Server provides execution services. It allows secure and transparent execution of jobs and tasks on the host.

The **lsadmin** tool provides flags to manage the LIM and Remote Execution Server in a single host or groups. Operations to start, stop, and restart the daemons are carried out by the **limstartup**, **limshutdown**, **limrestart**, **resstartup**, **resshutdown** and **resrestart** flags.

The **lsadmin** command provides a useful flag to check the correctness of LIM and Remote Execution Server configurations called `chkconfig`:

```
$ lsadmin ckconfig

Checking configuration files ...
No errors found.
```

The **badmin** command includes **limdebug** and **resdebug** flags for debugging problems with LIM and RES daemons.

6.3 Using the BMC for node monitoring

xCAT provides several tools to monitor the BMC, as described in 6.1, “Basic commands” on page 290. Some commands are described in 5.2.1, “Frequently used commands with the IPMItool” on page 194.

This chapter describes several techniques that can be used for monitoring the node with the BMC.

The **sdr** command can be used to query the various sensors on the node. The **sdr** command can retrieve metrics for several sensors, including temperature, voltage, power consumption, firmware status, and component functions. Example 6-25 shows how to use the **sdr** command to get the sensor data from the Temperature sensors.

For more information about sensor types, see the output of **sdr type**.

Example 6-25 The sdr command

```
$ ipmitool -H <IP> -U ADMIN -P admin sdr type Temperature
CPU Diode 1      | C8h | ok  |  3.0 | 37 degrees C
CPU Diode 2      | CBh | ok  |  3.0 | 36 degrees C
CPU1 Temp        | 0Bh | ok  |  3.0 | 46 degrees C
CPU2 Temp        | 0Dh | ok  |  3.1 | 43 degrees C
DIMM1 Temp       | 69h | ok  | 32.0 | 37 degrees C
DIMM2 Temp       | 6Ah | ok  | 32.1 | 37 degrees C
DIMM3 Temp       | 6Bh | ok  | 32.2 | 35 degrees C
DIMM4 Temp       | 6Ch | ok  | 32.3 | 36 degrees C
Mem Buf Temp 1   | C0h | ok  | 209.0 | 46 degrees C
...
Ambient Temp     | 0Ah | ok  |  7.0 | 32 degrees C
CPU Core Temp 1  | 89h | ok  | 208.0 | 45 degrees C
...
```

```
GPU Temp 1       | BCh | ns  | 216.0 | No Reading
GPU Temp 2       | BDh | ns  | 216.1 | No Reading
GPU Temp 3       | BEh | ns  | 216.2 | No Reading
GPU Temp 4       | BFh | ns  | 216.3 | No Reading
CPU 1 VDD Temp   | E4h | ok  |  3.0 | 41 degrees C
CPU 2 VDD Temp   | E5h | ok  |  3.1 | 41 degrees C
```

The **fru** command gathers data for the various field replaceable units (FRU) that are owned by the node. Each entry is expected to have a manufacturer code, name, part number, and serial number. If a component is missing, an Unknown FRU message is expected in the output. Example 6-26 shows the output from the use of the **fru** command to identify missing components.

Example 6-26 The fru command

```
$ ipmitool -H <IP> -U ADMIN -P admin fru print
...
FRU Device Description : DIMM32 (ID 43)
 Product Manufacturer  : 2c80
 Product Name          : 0c
 Product Part Number   : 18ASF1G72PDZ-2G3B1
 Product Version       : 31
 Product Serial        : ******

FRU Device Description : GPU1 (ID 44)
 Unknown FRU header version 0x00
...
FRU Device Description : PSU2 (ID 49)
 Product Manufacturer  : EMER
 Product Name          : Texan1300W
 Product Part Number   : 01AF314
 Product Version       : 0.12
 Product Serial        : ******
 Product Asset Tag     : 0.1
...
$ ipmitool -H 10.4.4.22 -U ADMIN -P admin fru print 2>&1 | grep "^$" -B1 | grep
"^FRU"
FRU Device Description : Builtin FRU Device (ID 0)
FRU Device Description : GPU1 (ID 44)
FRU Device Description : GPU2 (ID 45)
FRU Device Description : FAN1 (ID 46)
FRU Device Description : FAN2 (ID 50)
FRU Device Description : FAN3 (ID 51)
FRU Device Description : FAN4 (ID 52)
FRU Device Description : GPU3 (ID 53)
FRU Device Description : GPU4 (ID 54)
```

The following commands also are useful:

- To display the machine type and model, serial number, and other information:

  `$ ipmitool <arguments> fru print 3`

- Display system firmware version information:

  `$ ipmitool <arguments> fru print 47`

6.4 Using nvidia-smi tool for GPU monitoring

NVIDIA provides a tool to control the status and health of GPUs that is called the System Management Interface (**nvidia-smi**). The tool shows different levels of information, depending on the generation of your card. Some options can be disabled and enabled when this tool is used.

Example 6-27 shows default output for the example system with four NVIDIA Tesla P100 cards.

Example 6-27 Default nvidia-smi output

```
$ nvidia-smi
Thu Nov 10 21:29:53 2016
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 361.93.01              Driver Version: 361.93.01                 |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P100-SXM2...  Off  | 0002:01:00.0     Off |                    0 |
| N/A   42C    P0    30W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P100-SXM2...  Off  | 0003:01:00.0     Off |                    0 |
| N/A   38C    P0    29W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Tesla P100-SXM2...  Off  | 0006:01:00.0     Off |                    0 |
| N/A   43C    P0    30W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   3  Tesla P100-SXM2...  Off  | 0007:01:00.0     Off |                    0 |
| N/A   39C    P0    27W / 300W |      0MiB / 16280MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

6.4.1 Information about jobs on GPU

When the system has jobs that use graphics processing unit (GPU) calls inside them, you can find information (name and PID of tasks, GPU number for each process, and GPU memory usage) about it in the bottom section of the default **nvidia-smi** call. This section is shown in Example 6-28 for a sample MPI run with 10 MPI tasks, where each task uses only one GPU with a number that is assigned in round-robin order.

Example 6-28 Process section of nvidia-smi output during MPI run

```
+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0     23102    C   ./sample_mpi                                  971MiB |
|    1     23103    C   ./sample_mpi                                  834MiB |
|    2     23104    C   ./sample_mpi                                  971MiB |
|    3     23105    C   ./sample_mpi                                  834MiB |
|    0     23106    C   ./sample_mpi                                  971MiB |
|    1     23107    C   ./sample_mpi                                  834MiB |
|    2     23108    C   ./sample_mpi                                  971MiB |
|    3     23109    C   ./sample_mpi                                  834MiB |
|    0     23110    C   ./sample_mpi                                  971MiB |
|    1     23111    C   ./sample_mpi                                  834MiB |
+-----------------------------------------------------------------------------+
```

6.4.2 All GPU details

To show all of the information about GPUs inside your node, use the **-q** option. To list only data about specific GPU, specify the ID with the **-i** option. Example 6-29 on page 304 shows the second GPU that uses these options.

Example 6-29 Detailed information about second GPU in system using nvidia-smi

```
$ nvidia-smi -q -i 1

==============NVSMI LOG==============

Timestamp                           : Thu Nov 10 21:31:49 2016
Driver Version                      : 361.93.01

Attached GPUs                       : 4
GPU 0003:01:00.0
    Product Name                    : Tesla P100-SXM2-16GB
    Product Brand                   : Tesla
    Display Mode                    : Disabled
    Display Active                  : Disabled
    Persistence Mode                : Disabled
    Accounting Mode                 : Disabled
    Accounting Mode Buffer Size     : 1920
    Driver Model
        Current                     : N/A
        Pending                     : N/A
    Serial Number                   : ************
    GPU UUID                        : GPU-23eded49-4de4-c4d3-8ef7-********
    Minor Number                    : 1
    VBIOS Version                   : 86.00.1C.00.01
    MultiGPU Board                  : No
    Board ID                        : 0x30100
    GPU Part Number                 : 900-2H403-0400-000
    Inforom Version
        Image Version               : H403.0201.00.04
        OEM Object                  : 1.1
        ECC Object                  : 4.1
        Power Management Object     : N/A
    GPU Operation Mode
        Current                     : N/A
        Pending                     : N/A
    PCI
        Bus                         : 0x01
        Device                      : 0x00
        Domain                      : 0x0003
        Device Id                   : 0x15F910DE
        Bus Id                      : 0003:01:00.0
        Sub System Id               : 0x116B10DE
        GPU Link Info
            PCIe Generation
                Max                 : 3
                Current             : 3
            Link Width
                Max                 : 16x
                Current             : 8x
        Bridge Chip
            Type                    : N/A
            Firmware                : N/A
        Replays since reset         : 0
        Tx Throughput               : 0 KB/s
        Rx Throughput               : 0 KB/s
```

```
Fan Speed                         : N/A
Performance State                 : P0
Clocks Throttle Reasons
    Idle                          : Not Active
    Applications Clocks Setting   : Active
    SW Power Cap                  : Not Active
    HW Slowdown                   : Not Active
    Sync Boost                    : Not Active
    Unknown                       : Not Active
FB Memory Usage
    Total                         : 16280 MiB
    Used                          : 0 MiB
    Free                          : 16280 MiB
BAR1 Memory Usage
    Total                         : 16384 MiB
    Used                          : 2 MiB
    Free                          : 16382 MiB
Compute Mode                      : Default
Utilization
    Gpu                           : 0 %
    Memory                        : 0 %
    Encoder                       : 0 %
    Decoder                       : 0 %
Ecc Mode
    Current                       : Enabled
    Pending                       : Enabled
ECC Errors
    Volatile
        Single Bit
            Device Memory         : 0
            Register File         : 0
            L1 Cache              : N/A
            L2 Cache              : 0
            Texture Memory        : 0
            Texture Shared        : 0
            Total                 : 0
        Double Bit
            Device Memory         : 0
            Register File         : 0
            L1 Cache              : N/A
            L2 Cache              : 0
            Texture Memory        : 0
            Texture Shared        : 0
            Total                 : 0
    Aggregate
        Single Bit
            Device Memory         : 0
            Register File         : 0
            L1 Cache              : N/A
            L2 Cache              : 0
            Texture Memory        : 0
            Texture Shared        : 0
            Total                 : 0
        Double Bit
            Device Memory         : 0
```

```
            Register File       : 0
            L1 Cache            : N/A
            L2 Cache            : 0
            Texture Memory      : 0
            Texture Shared      : 0
            Total               : 0
    Retired Pages
        Single Bit ECC          : 0
        Double Bit ECC          : 0
        Pending                 : No
    Temperature
        GPU Current Temp        : 38 C
        GPU Shutdown Temp       : 85 C
        GPU Slowdown Temp       : N/A
    Power Readings
        Power Management        : Supported
        Power Draw              : 29.58 W
        Power Limit             : 300.00 W
        Default Power Limit     : 300.00 W
        Enforced Power Limit    : 300.00 W
        Min Power Limit         : 150.00 W
        Max Power Limit         : 300.00 W
    Clocks
        Graphics                : 1328 MHz
        SM                      : 1328 MHz
        Memory                  : 715 MHz
        Video                   : 1189 MHz
    Applications Clocks
        Graphics                : 1328 MHz
        Memory                  : 715 MHz
    Default Applications Clocks
        Graphics                : 1328 MHz
        Memory                  : 715 MHz
    Max Clocks
        Graphics                : 1480 MHz
        SM                      : 1480 MHz
        Memory                  : 715 MHz
        Video                   : 1480 MHz
    Clock Policy
        Auto Boost              : N/A
        Auto Boost Default      : N/A
    Processes                   : None
```

To print only specific information about the GPUs, use the **-d** option with the names of interesting sections. Example 6-30 shows only the details about temperature, power, and clocks by using this option.

Example 6-30 Details of GPU by using the -d option

```
$ nvidia-smi -q -i 1 -d POWER,TEMPERATURE,CLOCK

==============NVSMI LOG==============

Timestamp                           : Thu Nov 10 22:16:11 2016
Driver Version                      : 361.93.02

Attached GPUs                       : 4
GPU 0003:01:00.0
    Temperature
        GPU Current Temp            : 34 C
        GPU Shutdown Temp           : 85 C
        GPU Slowdown Temp           : 82 C
    Power Readings
        Power Management            : Supported
        Power Draw                  : 30.98 W
        Power Limit                 : 300.00 W
        Default Power Limit         : 300.00 W
        Enforced Power Limit        : 300.00 W
        Min Power Limit             : 150.00 W
        Max Power Limit             : 300.00 W
    Power Samples
        Duration                    : 2.36 sec
        Number of Samples           : 119
        Max                         : 31.00 W
        Min                         : 30.98 W
        Avg                         : 30.98 W
    Clocks
        Graphics                    : 405 MHz
        SM                          : 405 MHz
        Memory                      : 715 MHz
        Video                       : 835 MHz
    Applications Clocks
        Graphics                    : 1328 MHz
        Memory                      : 715 MHz
    Default Applications Clocks
        Graphics                    : 1328 MHz
        Memory                      : 715 MHz
    Max Clocks
        Graphics                    : 1480 MHz
        SM                          : 1480 MHz
        Memory                      : 715 MHz
        Video                       : 1480 MHz
    SM Clock Samples
        Duration                    : 17.33 sec
        Number of Samples           : 6
        Max                         : 1328 MHz
        Min                         : 405 MHz
        Avg                         : 1288 MHz
    Memory Clock Samples
```

```
        Duration                    : 17.33 sec
        Number of Samples           : 6
        Max                         : 715 MHz
        Min                         : 715 MHz
        Avg                         : 715 MHz
    Clock Policy
        Auto Boost                  : N/A
        Auto Boost Default          : N/A
```

If you want to log the data, the **--query-gpu** option is available, which takes a comma-separated list of properties. In combination with **--format=csv,nounits -f <path_to_csv>**, it writes the output to a `.csv` file. Adding **-l 10** results in data collection each 10 seconds, as shown in Example 6-31. Then, this file can be used to draw graphs or report statistics.

For a list of all available data fields, run the **nvidia-smi --help-query-gpu** command.

Example 6-31 Getting .csv of GPU performance data

```
$ nvidia-smi
--query-gpu=timestamp,index,temperature.gpu,power.draw,clocks.gr,clocks.mem
--format=csv,nounits -f /tmp/gpu.csv -l 10
^C
$ cat /tmp/gpu.csv
timestamp, index, temperature.gpu, power.draw [W], clocks.current.graphics [MHz],
clocks.current.memory [MHz]
2016/11/10 22:27:55.718, 0, 35, 30.00, 405, 715
2016/11/10 22:27:55.721, 1, 34, 30.98, 405, 715
2016/11/10 22:27:55.724, 2, 38, 30.48, 405, 715
2016/11/10 22:27:55.726, 3, 32, 29.48, 405, 715
2016/11/10 22:28:05.730, 0, 35, 29.98, 405, 715
2016/11/10 22:28:05.733, 1, 34, 30.98, 405, 715
2016/11/10 22:28:05.735, 2, 38, 30.48, 405, 715
2016/11/10 22:28:05.738, 3, 33, 29.48, 405, 715
```

6.4.3 Compute modes

The **nvidia-smi** command can change compute modes of GPUs by using following command:

nvidia-smi -i <GPU_number> -c <compute_mode>

NVIDIA GPUs support the following compute modes:

- PROHIBITED: The GPU cannot compute applications and no contexts are allowed.
- EXCLUSIVE_THREAD: Only one process can be assigned to the GPU at a time and perform work from only one thread of this process.
- EXCLUSIVE_PROCESS: Only one process can be assigned to the GPU at a time and different process threads can submit jobs to the GPU concurrently.
- DEFAULT: Multiple processes can use the GPU simultaneously and different process threads can submit jobs to the GPU concurrently.

6.4.4 Persistence mode

Persistence mode is a mode of the NVIDIA driver that helps to keep GPU initialized, even when no processes are accessing the cards. This mode requires more power, but shortens delays that occur at each start of GPU jobs. It is useful when you have a series of short runs.

To enable persistence mode for all GPUs, use following command:

```
nvidia-smi -pm 1
```

After the command is issued, you get the output for your system as shown in Example 6-32.

Example 6-32 Enable persistence mode for all GPUs

```
$ nvidia-smi -pm 1
Enabled persistence mode for GPU 0000:03:00.0.
Enabled persistence mode for GPU 0000:04:00.0.
Enabled persistence mode for GPU 0002:03:00.0.
Enabled persistence mode for GPU 0002:04:00.0.
All done.
```

To enable persistence mode only for a specific GPU, use the **-i** option. Example 6-33 shows how to enable this mode for the first GPU.

Example 6-33 Enable persistence mode for specific GPU

```
$ nvidia-smi -i 0 -pm 1
Enabled persistence mode for GPU 0000:03:00.0.
All done.
```

To disable persistence mode for all or a specific GPU, use the **0** value for the **-pm** option as shown in Example 6-34.

Example 6-34 Disable persistence mode for all or specific GPU

```
$ nvidia-smi -pm 0
Disabled persistence mode for GPU 0000:03:00.0.
Disabled persistence mode for GPU 0000:04:00.0.
Disabled persistence mode for GPU 0002:03:00.0.
Disabled persistence mode for GPU 0002:04:00.0.
All done.

$ nvidia-smi -i 0 -pm 0
Disabled persistence mode for GPU 0000:03:00.0.
All done.
```

The NVIDIA Persistence Daemon also is available. This daemon is used if you want to use a persistence state in production. For more information, see the Persistence Daemon section of the NVIDIA GPU Management and Deployment website.

6.4.5 More information

Another useful **nvidia-smi** command option is the **topo** option, which when used with the **-m** flag shows topological information about the system, as shown in Example 6-35.

Example 6-35 Show system topology

```
$ nvidia-smi topo -m
        GPU0    GPU1    GPU2    GPU3    mlx5_0 mlx5_1 CPU Affinity
GPU0     X      NV2     SOC     SOC     SOC     SOC     0-79
GPU1            NV2      X      SOC     SOC     SOC     SOC  0-79
GPU2     SOC     SOC      X      NV2     SOC     SOC     80-159
GPU3     SOC     SOC     NV2      X      SOC     SOC     80-159
mlx5_0   SOC     SOC     SOC     SOC      X      PIX
mlx5_1   SOC     SOC     SOC     SOC     PIX      X

Legend:

  X   = Self
  SOC  = Connection traversing PCIe as well as the SMP link between CPU
sockets(for example, QPI)
  PHB  = Connection traversing PCIe as well as a PCIe Host Bridge (typically the
CPU)
  PXB  = Connection traversing multiple PCIe switches (without traversing the PCIe
Host Bridge)
  PIX  = Connection traversing a single PCIe switch
  NV#  = Connection traversing a bonded set of # NVLinks
```

For more information about the options of the **nvidia-smi** command, see NVIDIA's nvidia-smi - NVIDIA System Management Interface program documentation.

6.5 Diagnostic and health check framework

The Diagnostic and health check framework that is provided by IBM is an application that examines the hardware and organizes the results. This process is done by running various tests across multiple nodes simultaneously and analyzing the resulting errors and data.

The framework is configuration file-driven and easily customized and extended. The framework uses xCAT to communicate with the remote nodes. The framework is in development (more checks will follow soon) and some features that are described in this section can change in the future.

Note: The framework is part of Cluster System Management (CSM), which is an entire software stack that was developed for the Department of Energy's Coral project to manage and administer an HPC cluster. By default, the framework uses some of CSM's features to check the node health and save the result in the CSM database.

Not every user runs a CSM environment; therefore, it can be used without CSM. Instead of writing the results to the database, it writes to a log file. Also, users must ensure that intrusive tests do not interfere with user jobs.

For more information about how to configure the framework in stand-alone mode, see 6.5.2, "Configuration" on page 312.

Before describing the functional flow of the diagnostic framework, consider the following definitions:

- Test

  A single executable file that is running on a single or multiple nodes that exercises one portion of the system.

- Test Group

  An abstract collection of test cases that are based on specific hardware coverage; for example, memory, processor, GPU, and network.

- Test Bucket

  A collection of test cases that is configured per some principle. For example, a bucket might be based on node health. Alternatively, it might be based on the thoroughness of the health check or in the network stress it provides. A test bucket often contains multiple test groups.

The diagnostic and health check framework can run on the following types of tests:

- Non-intrusive tests

  These tests query information about hardware components, their configuration, and status. These nonintrusive tests are run at system installation, upgrade, and periodically to ensure that no unexpected system changes occur. These tests can run in parallel with user jobs and are intended to monitor the status of hardware and software.

- Intrusive tests

  These tests must run dedicated. These test also cannot run in parallel with user jobs, must be properly marked, and are not used for scheduled jobs. It is recommended that these tests run at initial system installation and whenever a system change is made, such as a software or hardware upgrade, to spot possible performance problems.

6.5.1 Installation

The diagnostic and health check framework is provided as an RPM package. It contains the framework and some predefined tests. The tests might need to be customized. We assume default directories for tests that start scripts/binaries installed by third-party RPMs that might not match the user installation.

Because some tests must run as non-root user, we recommend the use of a non-root user to run diagnostic tests constantly. A non-root user can be configured to run Diagnostics. For more information, see the Granting Users xCAT Privileges page of the xCAT documentation website.

Download the RPM and install it by using the following command on the management node and all compute nodes you want to check:

```
$ rpm -ihv ibm-csm-hcdiag-0.1.1-1.noarch.rpm
```

It might be a good idea to add the **`hcdiag_run.py`** binary to your **`$PATH`** variable for easier access. One way to add this binary is to create a symbolic link in `/usr/local/bin`:

```
$ ln -s /opt/ibm/csm/hcdiag/bin/hcdiag_run.py /usr/local/bin/hcdiag_run
```

Note: It does not make sense to include the **`hcdiag_query.py`** script if you do not use a CSM database because this script does not work without such a database.

To validate your basic installation run the following command:

```
$ hcdiag_run --test test_simple --target <nodename> --nocsm
Health Check Diagnostics version beta1, running on Linux
3.10.0-481.el7.ppc64le, p8r1n1 machine.
Using configuration file /opt/ibm/csm/hcdiag/etc/hcdiag.properties.
Using tests configuration file /opt/ibm/csm/hcdiag/etc/test.properties.
Health Check Diagnostics, run id 1703071647072233, initializing...
[...]
test_simple PASS on 1 node(s):
c931f04p32
================================================================================
Health Check Diagnostics ended at 2017-03-07-16:47:11.098701, exit code 0.
```

Installing more RPMs

Some tests need more binaries to perform their tasks. The following RPMs must be installed on the compute nodes to run all tests:

- IBM Spectrum MPI, IBM ESSL, and IBM XL Fortran (installed automatically)
- CUDA (installed automatically) and Data Center GPU Manager package
- HTX package

6.5.2 Configuration

Two important subdirectories are available in the hcdiag installation directory `/opt/ibm/csm/hcdiag` on the management node, as shown in Example 6-36. Consider the following points:

- All configuration files are in the `etc` subdirectory
- All health and diagnostic test scripts are in the `tests` subdirectory

Example 6-36 Relative directory structure of /opt/ibm/csm/hcdiag directory

```
··· etc
·   ··· hcdiag.properties
·   ··· test.properties
·   ··· threshold.properties (not used yet)
··· tests
    ··· daxpy
    ··· dcgm-diag
    ··· dgemm
    ··· jlink
    ··· nvvs
    ··· ppping
    ··· README
    ··· test_2
    ··· test_simple
```

All configuration files use the following format:

```
[Section1]
key1 = value
key2 = value2

[tests.test1]
key1 = value
```

The global configuration file is called `hcdiag.properties`. The default settings can be used for standard xCAT installations.

One important parameter is the `xcat_fanout` parameter. It configures the number of concurrent remote shell command processes (default is 64). For clusters with more than 64 nodes per service node, you can increase this number to run checks faster. IBM intends to release a configuration option to disable the CSM functions globally.

Another important setting in the `hcdiag.propeties` file is the variable `csm`. If set to `yes`, the framework runs with the CSM environment, unless it is overwritten by command line. If set to `no`, the framework runs with no chasm environment, unless it is overwritten by command line. The default value is `yes`. Therefore, if you do not have a CSM environment, you must change this default value to `no`.

The `test.properties` configuration file configures all available tests, test groups, and buckets. Example 6-37 shows some examples for each category.

The following attributes are valid for the `[tests]` section:

- `description`: Short description of the test (documentation only).
- `group`: Group to which the test belongs.
- `timeout`: Time (in seconds) that xCAT waits for output from any running remote targets. Default is 10 seconds.
- `target`: The type of the node to which the test applies. Valid values are `compute` and `management`.
- `executable`: The full path of the executable file. This attribute is a mandatory.
- `args`: Arguments for the executable file, if any.
- `cluster`: Test tells the framework if this test is a single node test or a cluster test.

The `[bucket]` section features the following attributes:

- `description`: A short description of the bucket.
- `tests`: A comma-separated list of the tests that are part of the bucket.

Example 6-37 The test.properties cfg file

```
[tests.daxpy]
description = DAXPY measures aggregate memory bandwidth in MB/s.
executable  = /opt/ibm/csm/hcdiag/tests/daxpy/daxpy.sh
group       = memory
timeout     = 100
targetType  = Compute

[...]

[bucket.performance]
description = A default set of tests designed to provide comprehensive performance
test.
tests       = dgemm, daxpy, jlink
```

As shown in Example 6-37, the test group does not have its own section. The `group` keyword of the `[tests.testname]` sections defines available test groups. You can add your own tests, test groups, and buckets to this file or edit existing files. If you want to add your own tests, you must configure those tests in this file.

6.5.3 Usage

Example 6-38 shows the use of the **hcdiag_run.py** command. The most important arguments are marked in **bold**.

Note: The **hcdiag_query.py** command cannot be used without a CSM database.

Example 6-38 The hcdiag_run.py --help output

```
$ hcdiag_run --help
usage: hcdiag_run [-h]
                  (--test t [t ...] | --bucket b | --list item [item ...])
                  [--target n [n ...]] [--noallocation] [--nocsm]
                  [--fanout fanout_value] [--diagproperties filename]
                  [--testproperties filename] [--thresholdproperties filename]
                  [--logdir dir] [--stoponerror action] [-v level]

optional arguments:
  -h, --help            show this help message and exit
  --test t [t ...]      test to run
  --bucket b, -b b      bucket to run
  --list item [item ...], -l item [item ...]
                        list available: test, group, bucket | all
  --target n [n ...], -t n [n ...]
                        target on which to run health check/diagnostic
  --noallocation        do not allocate the target nodes
  --nocsm               do not allocate the target nodes
  --fanout fanout_value, -f fanout_value
                        maximum number of concurrent remote shell command
                        processes
  --diagproperties filename
                        diag properties file
  --testproperties filename
                        test properties file
  --thresholdproperties filename
                        threshold properties file
  --logdir dir          root directory for the log files
  --stoponerror action  define action if test fail {no: continue, node: stop
                        at node level, system: stop the run}
  -v level, --verbose level
                        output verbosity {debug, info, warn, error, critical}
```

To run a test, you specify the test or bucket (multiple tests) name tat you want to run with a xCAT node range, as shown in Example 6-39. The framework outputs a run ID, some information about the nodes, and the result summary. The output is saved to the `logdir` as configured in `hcdiag.properties` file (default is `/tmp`). For a more detailed test result, you can review the run ID directory as shown in Example 6-39.

Example 6-39 Run of a simple non-intrusive test on two nodes

```
$ hcdiag_run --test test_memsize --target p8r1n1,p8r1n2 --nocsm
[...]
Health Check Diagnostics, run id 1703081638126144, initializing...
[...]
test_memsize PASS on node p8r1n1, serial number: XXXXXXX.
```

```
test_memsize PASS on node p8r1n2, serial number: XXXXXXX.

============================== Results summary  ==============================

16:38:13 ========================================================================

test_memsize PASS on 2 node(s):

c931f04p32,c931f04p34
================================================================================

$ cat /tmp/1703081638126144/test_memsize/*
 MemTotal of 251GB is greater or equal to 250GB
 ./test_memsize.sh test PASS, RC=0
 Remote_command_rc = 0
 MemTotal of 251GB is greater or equal to 250GB
 ./test_memsize.sh test PASS, RC=0
 Remote_command_rc = 0
```

If the test is visible to the target (framework is installed on a shared file system), the framework runs it. If the test is not visible to the target, the framework copies it to the target before running it. Other binaries, such as HTX, dgemm, or daxpy, are run from the binary installation directory that is on the target node.

6.5.4 Adding tests

Adding your own tests to the framework is an easy process. In this section, we describe how to add your own simple system memory size test.

Each test must have its own subdirectory. The name of the subdirectory must match the name of the test in the `tests.properties` file. The test can be a script or some executable binary. The return code of the test script determines whether the test passed (`RC == 0`) or failed (`RC != 0`). The output is irrelevant because you can output anything you decide.

For example, our new test is called `test_memsize`., as shown in Example 6-40.

Example 6-40 test_memsize.sh

```
#!/bin/bash
# check MemTotal in GB

readonly TESTNAME=$0

[ $# -ne 1 ] && echo "Usage: $TESTNAME <total memsize in GB>" && exit
readonly EXPECTED_MEMSIZE=$1

memsize_gb=$(awk '/^MemTotal:/{printf "%d",$2/1048576}' /proc/meminfo)

if [ $memsize_gb -ge $EXPECTED_MEMSIZE ]; then
        rc=0
        echo "MemTotal of ${memsize_gb}GB is greater or equal to
${EXPECTED_MEMSIZE}GB"
        echo "$TESTNAME test PASS, RC=$rc"
else
        rc=1
```

```
        echo "MemTotal of ${memsize_gb}GB is less than expected
${EXPECTED_MEMSIZE}GB"
        echo "$TESTNAME test FAIL, RC=$rc"
fi

exit $rc
```

First, we must create the test in the `test_memsize` directory called `test_memsize`:

```
cd /opt/ibm/csm/hcdiag/tests
mkdir test_memsize
vi test_memsize/test_memsize.sh # see Example 6-40

chmod +x test_memsize/test_memsize.sh
```

Then, we add an entry for the test in the `tests.properties` file with the wanted argument (expected minimum memory size in GB):

```
[tests.test_memsize]
description = This tests checks the available system memory size
executable  = /opt/ibm/csm/hcdiag/tests/test_memisze/test_memsize.sh
args = 250
```

Now, the new test can run, as shown in Example 6-39 on page 314.

Part 3

Evaluation and system planning guide

This part provides the developers, architects, and IT decision makers with information about how to understand the software and hardware components of a high-performance computing solution.

The part also provides a chapter to help the readers evaluate and plan the system role out.

The following chapters are included in this part:

7

Hardware components

The IBM Power System S822LC for High Performance Computing (HPC) server (8335-GTB) is the first Power Systems offering with NVIDIA NVLink Technology. It removes GPU computing bottlenecks by employing the high-bandwidth and low-latency NVLink interface from CPU-to-GPU and GPU-to-GPU. This configuration unlocks new performance abilities and applications for accelerated computing.

Power System LC servers are products of a codesign with OpenPOWER Foundation ecosystem members. The S822LC for HPC server innovation partners includes IBM, NVIDIA, Mellanox, Canonical, Wistron, and other partners.

In this chapter, we describe these hardware components.

This chapter includes the following topics:

7.1 Server features

The S822LC for HPC server offers a modular design to scale from single racks to hundreds of racks, simplify of ordering, and provide a strong innovation road map for GPUs. The server offers two processor sockets for a total of 16 cores at 3.259 GHz or 20 cores at 2.860 GHz in a 19-inch rack-mount, 2U (EIA units) drawer configuration. All of the cores are activated.

Figure 7-1 shows the front view of an S822LC for HPC server.

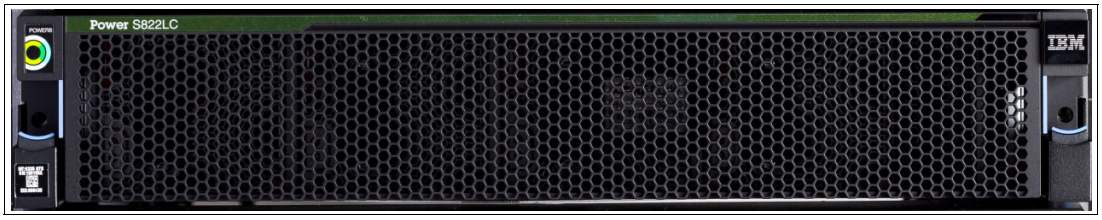


Figure 7-1 Front view of the S822LC for HPC server

The server chassis contains two processor modules that are attached directly to the board. Each POWER8 processor module can have up to 12 cores and has a 64-bit architecture, up to 512 KB of L2 cache per core, and up to 8 MB of L3 cache per core. Clock speeds of 3.259 GHz or 2.860 GHz are available.

The S822LC for HPC server provides eight dual inline memory module (DIMM) memory slots. Memory features that are supported are 16 GB (#EM55), 32 GB (#EM56), 64 GB (#EM57), and 128 GB (#EM58), which allows for a maximum of 1024 GB DDR4 system memory. Memory operates at the double data rate type four (DDR4) data rate.

Figure 7-2 shows the physical locations of the main server components.

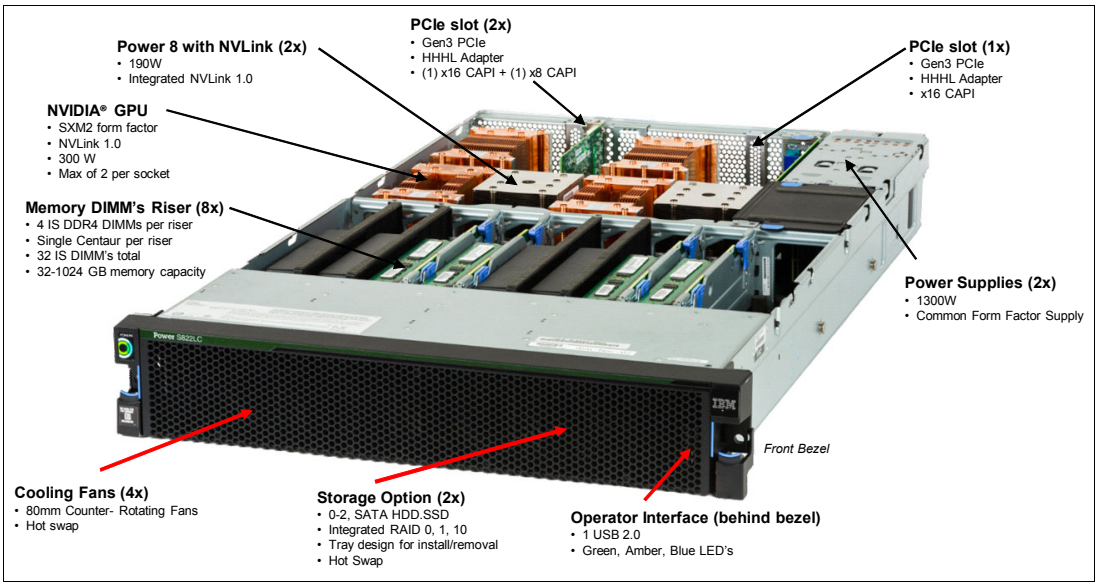


Figure 7-2 Location of server main components

The server supports four NVIDIA Tesla P100 GPU (#EC4C, #EC4D, or #EC4F), based on the NVIDIA SXM2 form factor connectors. The features are first-pair air cooled, second-pair air cooled, and water cooled (all four GPUs require #ER2D).

The S822LC for HPC server (8355-GTB) includes the following standard features:

- 19-inch rack-mount (2U) chassis
- Two POWER8 processor modules:
  - 10-core 3.259 GHz processor module
  - 12-core 2.860 GHz processor module
  - Up to 1024 GB of 1333 MHz DDR4 error correction code (ECC) memory
- Two small form factor (SFF) bays for two hard disk drives (HDDs) or two solid-state drives (SSDs) that support:
  - Two 1 TB 7200 RPM NL Serial Advanced Technology Attachment (SATA) disk drives (#ELD0)
  - Two 2 TB 7200 RPM NL SATA disk drives (#ES6A)
  - Two 480 GB SATA SSDs (#ELS5)
  - Two 960 GB SATA SSDs (#ELS6)
  - Two 1.92 TB SATA SSDs (#ELSZ)
  - Two 3.84 TB SATA SSDs (#ELU0)
- Integrated SATA controller
- Three Peripheral Component Interconnect Express (PCIe) Gen 3 slots:
  - One PCIe x8 Gen3 Low Profile slot, Coherent Accelerator Processor Interface (CAPI) enabled
  - Two PCIe x16 Gen3 Low Profile slot, CAPI enabled
- Four NVIDIA Tesla P100 GPU (#EC4C, #EC4D, or #EC4F), based on the NVIDIA SXM2 form factor connectors
- Integrated features:
  - IBM EnergyScale™ technology
  - Hot-swap and redundant cooling
  - One front USB 2.0 port for general use
  - One rear USB 3.0 port for general use
  - One system port with RJ45 connector
- Two power supplies

Note: A Hardware Management Console is not supported on the S822LC for HPC server (8335-GTB).

7.1.1 Minimum features

The minimum initial order for the S822LC for HPC server (8355-GTB) must include the following minimum features:

- Two processor modules with at least 20 cores
- 128 GB of memory (eight 16 GB memory DIMMs)
- Two #EC4C compute-intensive accelerators (NVIDIA GP100)
- Two power supplies and power cords
- An operating system indicator
- A rack integration indicator
- A Language Group Specify

Linux is the supported operating system. The Integrated 1 Gb Ethernet port can be used as the base LAN port.

7.1.2 System cooling

Air or water cooling depends on the GPU that is installed.

To order water cooled, feature #ER2D must be selected as the initial order. Otherwise, the server is built to be air cooled.

Rack requirement: The IBM 7965-94Y rack with feature #ER22 or #ER23 installed supports the water cooling option for the S822LC for HPC server.

Cold plates to cool two processor modules and four GPUs, such as #EC4F, are included. Water lines carrying cool water in and warm water out are also included. This feature is installed in the system unit when the server is manufactured and is not installed in the field.

When shipped from IBM, an air-cooled server cannot be changed into a water-cooled server; and a water-cooled server cannot be changed into an air-cooled server.

Customer setup is not supported for water-cooled systems.

The GPU air-cooled and water-cooled servers have the following ordering differences:

- With an air-cooled server, an initial order can be ordered with two GPUs (quantity of two of feature #EC4C and a quantity of zero of feature #EC4D) or four GPUs (quantity of two of feature #EC4C plus a quantity of two of feature #EC4D).
- With a water-cooled server (#ER2D), a quantity of four feature #EC4F GPUs must be ordered.

Figure 7-3 shows a server with water-cooled GPUs.

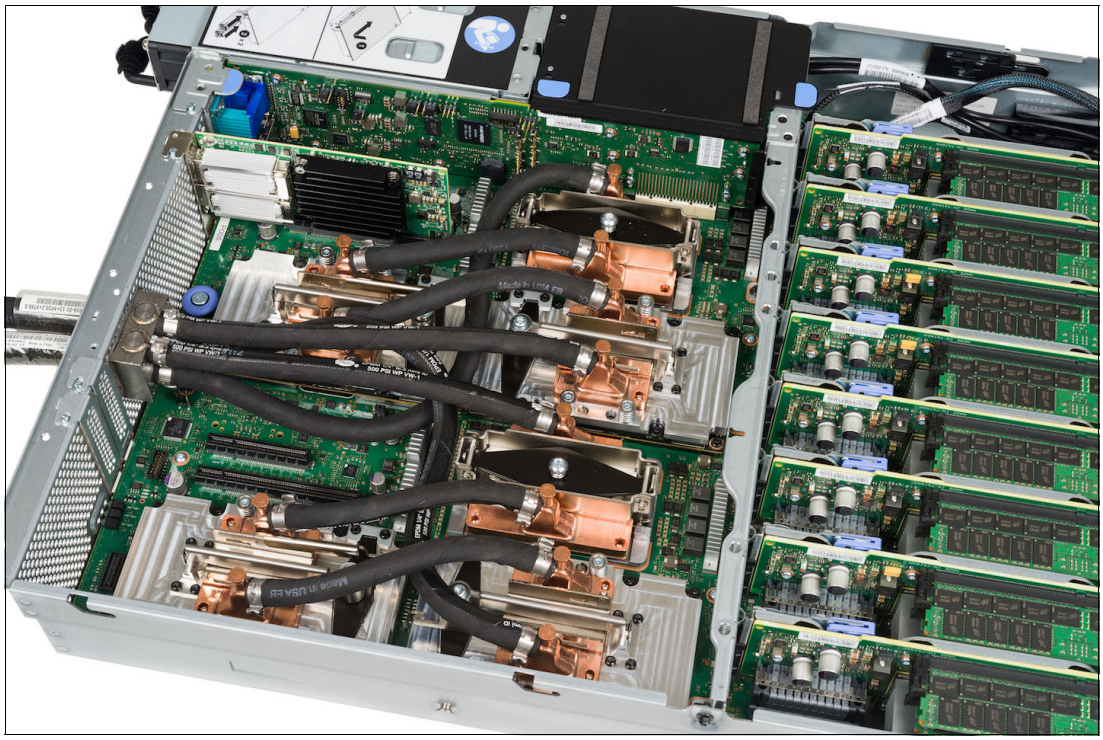


Figure 7-3 S822LC for HPC server with water-cooled GPU

Note: If #ER2D is ordered, you must order a #EJTX fixed rail kit. Ordering #ER2D with #EJTY slide rails is not supported.

For more information about the water cooling option, see the Model 8335-GTB water cooling option (Feature code E2RD) page of the IBM Knowledge Center website.

7.2 NVIDIA Tesla P100

NVIDIA's new NVIDIA Tesla P100 accelerator (see Figure 7-4) takes GPU computing to the next level. This section describes the Tesla P100 accelerator.

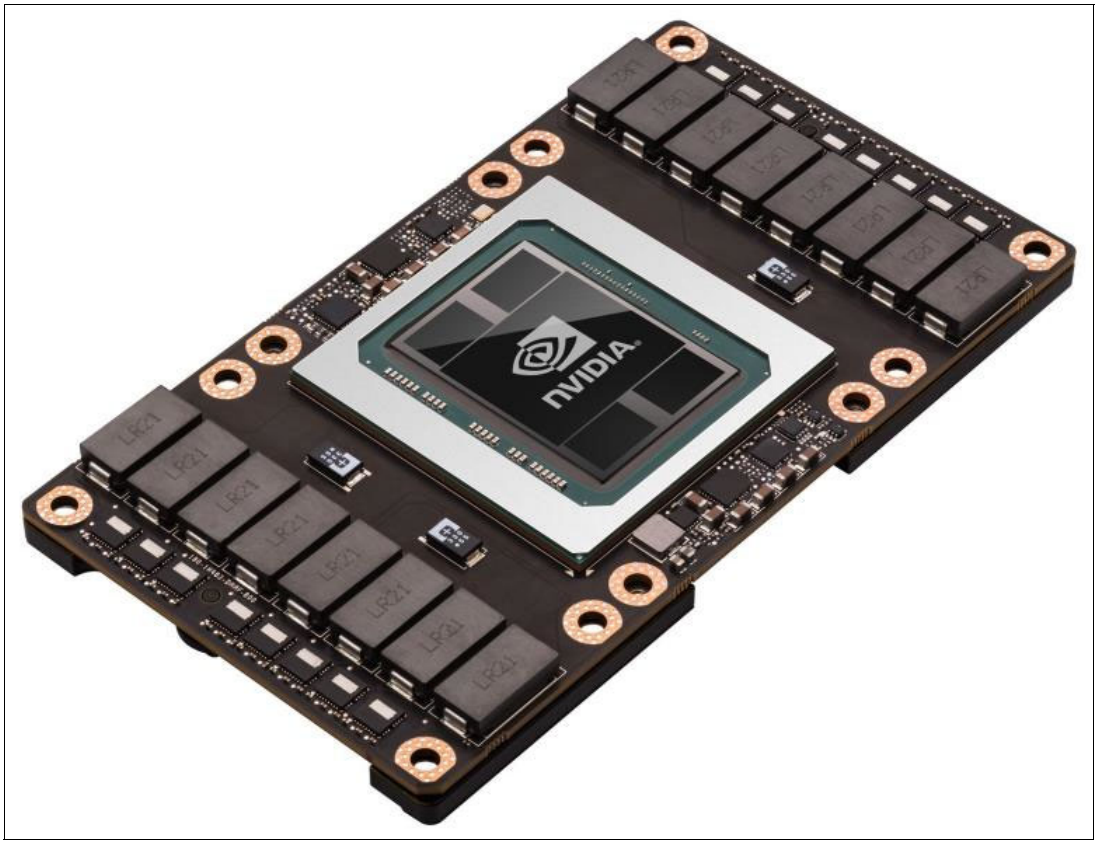


Figure 7-4 NVIDIA Tesla P100 accelerator

The Tesla P100 is the most powerful and the most architecturally complex GPU accelerator architecture ever built. It has a 15.3 billion transistor GPU, a new high-performance interconnect that greatly accelerates GPU peer-to-peer and GPU-to-CPU communications, new technologies to simplify GPU programming, and exceptional power efficiency.

The Tesla P100 includes the following key features:

- Extreme performance

  Powering high-performance computing, deep learning, and many more GPU computing areas.

- NVLink

  NVIDIA's new high-speed, high-bandwidth interconnect for maximum application scalability.

- HBM2

  Fast, high-capacity, extremely efficient chip-on-wafer-on-substrate (CoWoS) stacked memory architecture.

- Unified memory, compute preemption, and new artificial intelligence (AI) algorithms

  Significantly improved programming model and advanced AI software that is optimized for the Pascal architecture.

- 16 nm FinFET

  Enables more features, higher performance, and improved power efficiency.

The Tesla P100 is built to deliver exceptional performance for the most demanding compute applications. It delivers the following performance benefits:

- 5.3 TFLOPS of double-precision floating point (FP64) performance
- 10.6 TFLOPS of single-precision (FP32) performance
- 21.2 TFLOPS of half-precision (FP16) performance

In addition to the numerous areas of HPC that NVIDIA GPUs accelerated for years, deep learning became an important area of focus for GPU acceleration. NVIDIA GPUs are at the forefront of deep neural networks (DNNs) and AI. They are accelerating DNNs in various applications by a factor of 10x to 20x, compared to CPUs and reducing training times from weeks to days.

In the past three years, NVIDIA GPU-based computing platforms helped speed up deep learning network training times by a factor of 50x. In the past two years, the number of companies NVIDIA collaborates with on deep learning increased nearly 35x to over 3,400 companies.

Innovations in the Pascal architecture, including native 16-bit floating point (FP) precision, allow GP100 to deliver great speedups for many deep learning algorithms. These algorithms do not require high levels of FP precision, but they gain large benefits from the additional computational power that FP16 affords and from the reduced storage requirements for 16-bit data types.

For more information, see the NVIDA Tesla P100 website.

7.3 Operating environment

Table 7-1 lists the operating environment specifications for the S822LC for HPC server.

Table 7-1 Operating environment for the S822LC for HPC server

Server operating environment		
Description	**Operating**	**Non-operating**
Temperature	Allowable: 5 - 40°C[a] (41 - 104°F) Recommended: 18 - 27 °C (64 - 80 °F)	1 - 60°C (34 - 140°F)
Relative humidity	8 - 80%	8 - 80%
Maximum dew point	24°C (75° F)	27°C (80°F)
Operating voltage	200 - 240 V AC	N/A
Operating frequency	50 - 60 Hz +/- 3 Hz	N/A
Power consumption	2550 watts maximum	N/A
Power source loading	2.6 kVA maximum	N/A

Server operating environment		
Description	**Operating**	**Non-operating**
Thermal output	8703 BTU/hr maximum	N/A
Maximum altitude	3050 m (10,000 ft.)	N/A
Noise level and sound power	7.6/6.7 bels operating/idling	N/A

a. Heavy workloads might see some performance degradation above 35°C if internal temperatures trigger a CPU clock reduction.

Tip: The maximum measured value is expected from a fully populated server under an intensive workload. The maximum measured value also accounts for component tolerance and operating conditions that are not ideal.

Power consumption and heat load vary greatly by server configuration and usage. Use the IBM Systems Energy Estimator to obtain a heat output estimate that is based on a specific configuration.

7.4 Physical package

Table 7-2 lists the physical dimensions of the chassis. The server is available only in a rack-mounted form factor and requires 2U (2 EIA units) of rack space.

Table 7-2 Physical dimensions for the S822LC for HPC server

Dimension	S822LC for HPC server (8335-GTB)
Width	441.5 mm (17.4 in.)
Depth	822 mm (32.4 in.)
Height	86 mm (3.4 in.)
Weight (maximum configuration)	30 kg (65 lbs.)

7.5 System architecture

The bandwidths that are provided throughout the section are theoretical maximums that are used for reference. The speeds that are shown are at an individual component level. Multiple components and application implementation are key to achieving the preferred performance. Always assess the performance sizing at the application-workload environment level and evaluate performance by using real-world performance measurements and production workloads.

The S822LC for HPC server is a two single-chip module (SCM) system. Each SCM is attached to four memory riser cards that have buffer chips for the L4 cache and four memory RDIMM slots. The server has a maximum capacity of 32 memory DIMMs when all the memory riser cards are populated, which allows for up to 1024 GB of memory.

The server has a total of three PCIe Gen3 slots; all of these slots are CAPI-capable. The system has sockets for four GPUs, each 300 Watt capable.

An integrated SATA controller is fed through a dedicated PCI bus on the main system board and allows for up to two SATA HDDs or SSDs to be installed. This bus also drives the integrated Ethernet and USB port.

Figure 7-5 shows the logical system for the S822LC for HPC server.

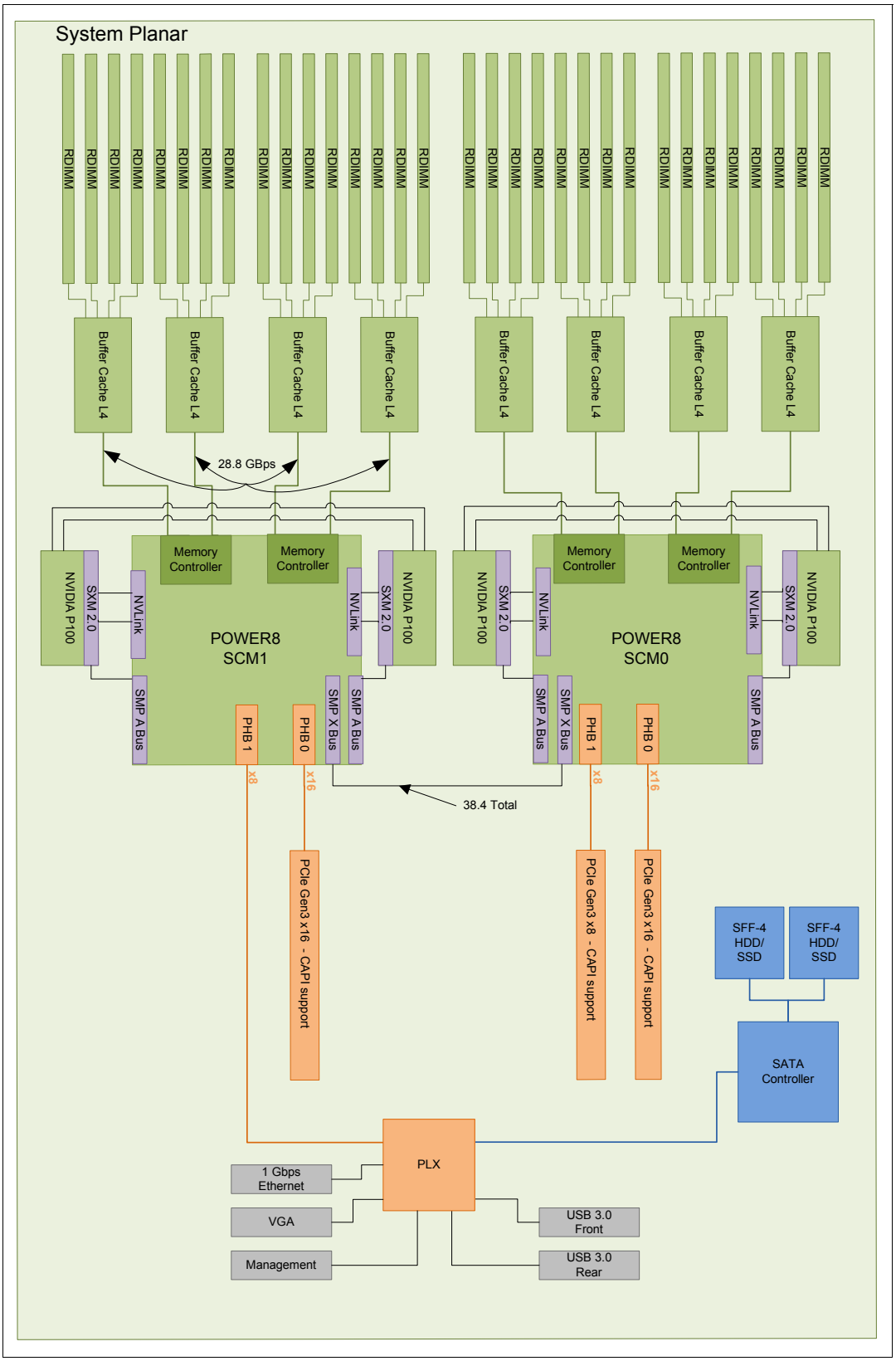


Figure 7-5 S822LC for HPC server logical system

7.6 POWER8 processor

This section introduces the latest processor in the Power Systems product family and describes its main characteristics and features in general.

The POWER8 processor in the S822LC for HPC server is unique to the 8335-GTB model. By removing the A-bus interfaces along with SMP over PCI support, space is available for the NVLink interface. The resulting chip grows slightly from 649 mm^2 to 659 mm^2. Socket-to-socket communication is provided through an SMP A-bus.

7.6.1 POWER8 processor overview

The POWER8 processor used in the 8335-GTB is manufactured by using the IBM 22 nm silicon-on-insulator (SOI) technology. Each chip is 65 mm^2 and contains over 4.2 billion transistors. The POWER8 chip can contain up to 12 cores, 2 memory controllers, PCIe Gen3 I/O controllers, and an interconnection system that connects all components within the chip. Each core has 512 KB of L2 cache, and all cores share 96 MB of L3 embedded DRAM (eDRAM). The interconnect also extends through module and system board technology to other POWER8 processors in addition to DDR4 memory and various I/O devices.

POWER8 processor-based systems use memory buffer chips to interface between the POWER8 processor and DDR4 memory. Each buffer chip also includes an L4 cache to reduce the latency of local memory accesses.

The following features augment the performance of the POWER8 processor:

- Support for DDR4 memory through memory buffer chips that offload the memory support from the POWER8 memory controller.
- An L4 cache within the memory buffer chip that reduces the memory latency for local access to memory behind the buffer chip; the operation of the L4 cache is not apparent to applications that are running on the POWER8 processor. Up to 128 MB of L4 cache can be available for each POWER8 processor.
- Hardware transactional memory.
- On-chip accelerators, including on-chip encryption, compression, and random number generation accelerators.
- CAPI, which allows accelerators that are plugged into a PCIe slot to access the processor bus by using a low-latency, high-speed protocol interface.
- Adaptive power management.

Table 7-3 lists the technology characteristics of the POWER8 processor.

Table 7-3 Summary of POWER8 processor technology

Technology	8335-GTB POWER8 processor
Die size	659 mm^2
Fabrication technology	► 22 nm lithography ► Copper interconnect ► SOI ► eDRAM
Maximum processor cores	12
Maximum execution threads core/chip	8/96

Technology	8335-GTB POWER8 processor
Maximum L2 cache core/chip	512 KB/6 MB
Maximum On-chip L3 cache core/chip	8 MB/96 MB
Maximum L4 cache per chip	128 MB
Maximum memory controllers	2
SMP design-point	16 sockets with POWER8 processors
Compatibility	Specific to the 8335-GTB

Figure 7-6 shows the areas of the processor that were modified to include the NVLink and more CAPI interface.

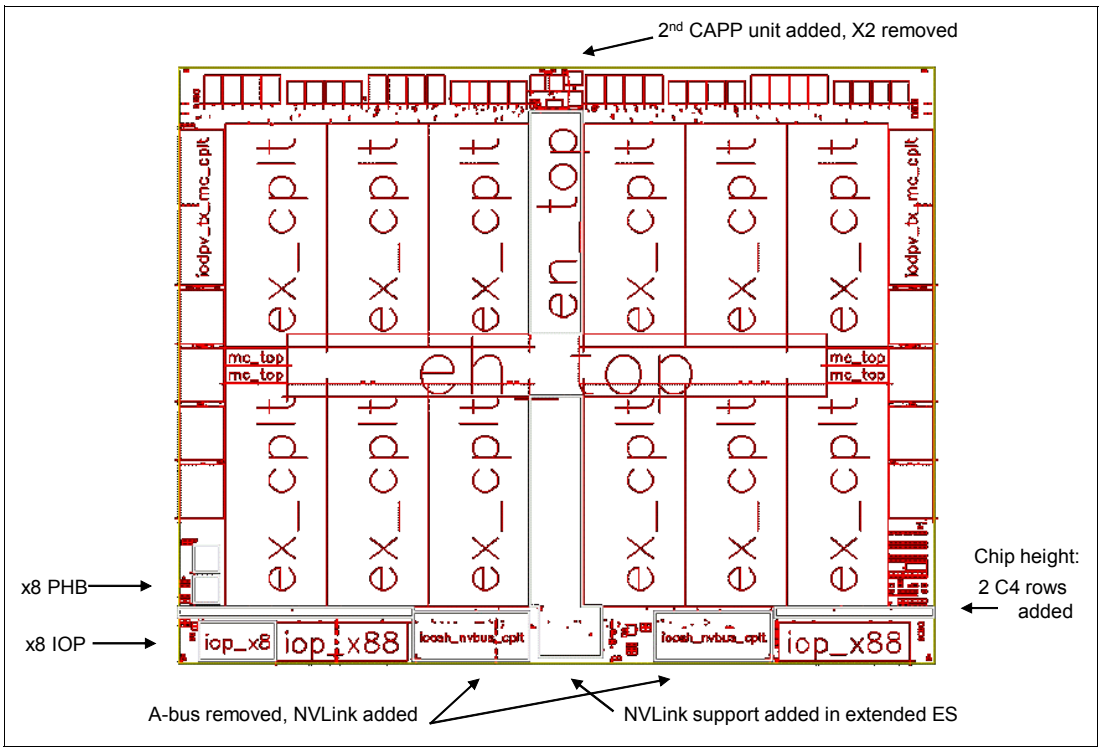


Figure 7-6 Areas modified on the POWER8 processor core

7.6.2 POWER8 processor core

The POWER8 processor core is a 64-bit implementation of the IBM Power Instruction Set Architecture (ISA) Version 2.07 and has the following features:

- Multithreaded design, which is capable of up to eight-way simultaneous multithreading (SMT)
- 32 KB, eight-way set-associative L1 instruction cache
- 64 KB, eight-way set-associative L1 data cache
- Enhanced prefetch, with instruction speculation awareness and data prefetch depth awareness
- Enhanced branch prediction, which uses local and global prediction tables with a selector table to choose the preferred predictor

- Improved out-of-order execution
- Two symmetric fixed-point execution units
- Two symmetric load/store units and two load units, all four of which can also run simple fixed-point instructions
- An integrated, multi-pipeline vector-scalar FP unit for running both scalar and SIMD-type instructions, including the Vector Multimedia eXtension (VMX) instruction set and the improved Vector Scalar eXtension (VSX) instruction set, and capable of up to eight FP operations per cycle (four double precision or eight single precision)
- In-core Advanced Encryption Standard (AES) encryption capability
- Hardware data prefetching with 16 independent data streams and software control
- Hardware decimal floating point (DFP) capability

For more information about Power ISA Version 2.07, see the POWER8 Instruction Set Architecture (ISA) OpenPOWER Profile Workgroup Specification documentation.

Figure 7-7 shows the POWER8 core with some of the functional units highlighted.

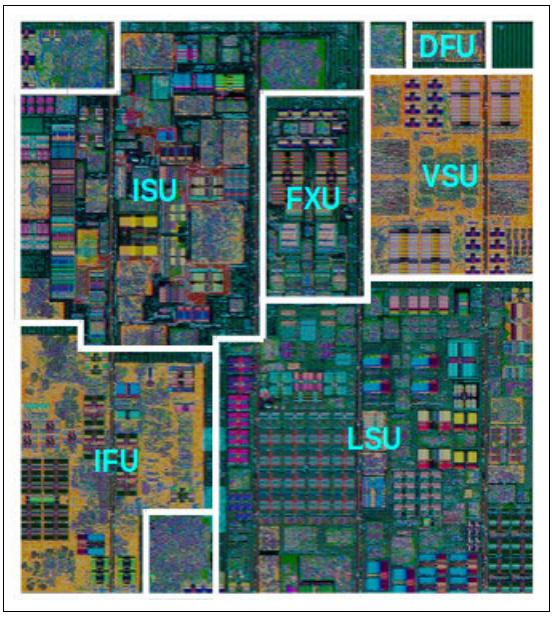


Figure 7-7 POWER8 processor core

7.6.3 Simultaneous multithreading

Simultaneous multithreading (SMT) allows a single physical processor core to dispatch simultaneously instructions from more than one hardware thread context. With SMT, each POWER8 core can present eight hardware threads. Because there are multiple hardware threads per physical processor core, more instructions can run at the same time.

SMT is primarily beneficial in commercial environments where the speed of an individual transaction is not as critical as the total number of transactions that are performed. SMT typically increases the throughput of workloads with large or frequently changing working sets, such as database servers and web servers.

Table 7-4 shows a comparison between the different IBM POWER processors' options for an S822LC for HPC server and the number of threads that are supported by each SMT mode.

Table 7-4 SMT levels supported by an S822LC for HPC server

Cores per system	SMT mode	Hardware threads per system
20	Single thread (ST)	20
20	SMT2	40
20	SMT4	80
20	SMT8	160
24	ST	24
24	SMT2	48
24	SMT4	96
24	SMT8	192

The architecture of the POWER8 processor, with its larger caches, larger cache bandwidth, and faster memory, allows threads to have faster access to memory resources, which results into a more efficient use of threads. Therefore, POWER8 allows more threads per core to run concurrently, which increases the total throughput of the processor and system.

7.6.4 Memory access

On the S822LC for HPC server, each POWER8 module has two memory controllers, each connected to two memory channels. Each memory channel operates at 1600 MHz and connects to a memory riser card. Each memory riser card has a memory buffer that is responsible for many functions that were previously on the memory controller, such as scheduling logic and energy management.

The memory buffer also has 16 MB of L4 cache. The memory riser card also houses four industry-standard RDIMMs.

Each memory channel can address up to 128 GB. Therefore, the server can address up to 1024 GB (1 TB) of total memory.

Figure 7-8 shows a POWER8 processor that is connected to four memory riser cards and their components.

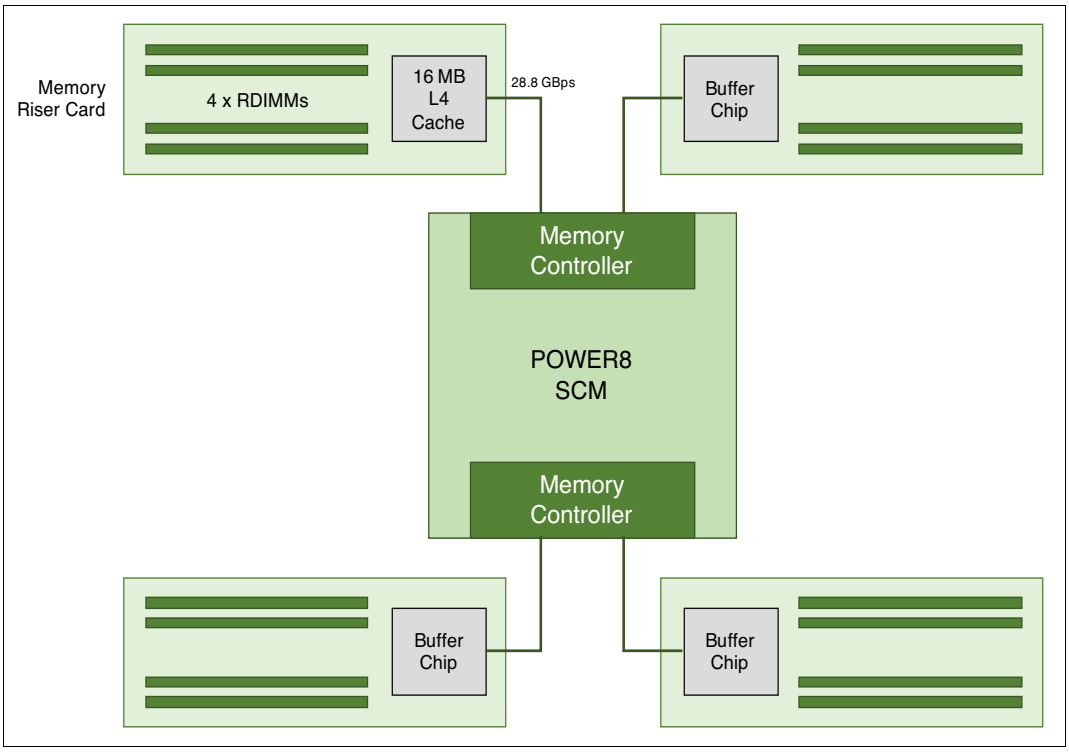


Figure 7-8 POWER8 processor that is connected to four memory riser cards

7.6.5 On-chip L3 cache innovation and intelligent cache

The POWER8 processor uses a unique material engineering and microprocessor fabrication to implement the L3 cache in eDRAM and place it on the processor die. L3 cache is critical to a balanced design. Also critical to a balanced design is the ability to provide good signaling between the L3 cache and other elements of the hierarchy, such as the L2 cache or SMP interconnect.

The on-chip L3 cache is organized into separate areas with differing latency characteristics. Each processor core is associated with a fast 8 MB local region of L3 cache (FLR-L3), but also has access to other L3 cache regions as shared L3 cache. Additionally, each core can negotiate to use the FLR-L3 cache that is associated with another core, depending on reference patterns. Data can also be cloned to be stored in more than one core's FLR-L3 cache, depending on reference patterns. This intelligent cache management enables the POWER8 processor to optimize the access to L3 cache lines and minimize overall cache latencies.

The innovation of using eDRAM on the POWER8 processor die is significant for the following reasons:

- Latency improvement

  A six-to-one latency improvement occurs by moving the L3 cache on-chip compared to L3 accesses on an external (on-ceramic) Application Specific Integrated Circuit (ASIC).

- Bandwidth improvement

  A 2x bandwidth improvement occurs with on-chip interconnect. Frequency and bus sizes are increased to and from each core.

- No off-chip driver or receivers

  Removing drivers or receivers from the L3 access path lowers interface requirements, conserves energy, and lowers latency.

- Small physical footprint

  The performance of eDRAM when implemented on-chip is similar to conventional SRAM but requires less physical space. IBM on-chip eDRAM uses only a third of the components that conventional SRAM uses, which has a minimum of six transistors to implement a 1-bit memory cell.

- Low energy consumption

  The on-chip eDRAM uses only 20% of the standby power of SRAM.

7.6.6 L4 cache and memory buffer

POWER8 processor-based systems introduce another level in memory hierarchy. The L4 cache is implemented with the memory buffer in the memory riser cards. Each memory buffer contains 16 MB of L4 cache. On an S822LC for HPC server, you can have up to 128 MB of L4 cache by using all the eight memory riser cards.

Figure 7-9 shows the memory buffer and highlights the 16 MB L4 cache and processor links and memory interfaces.

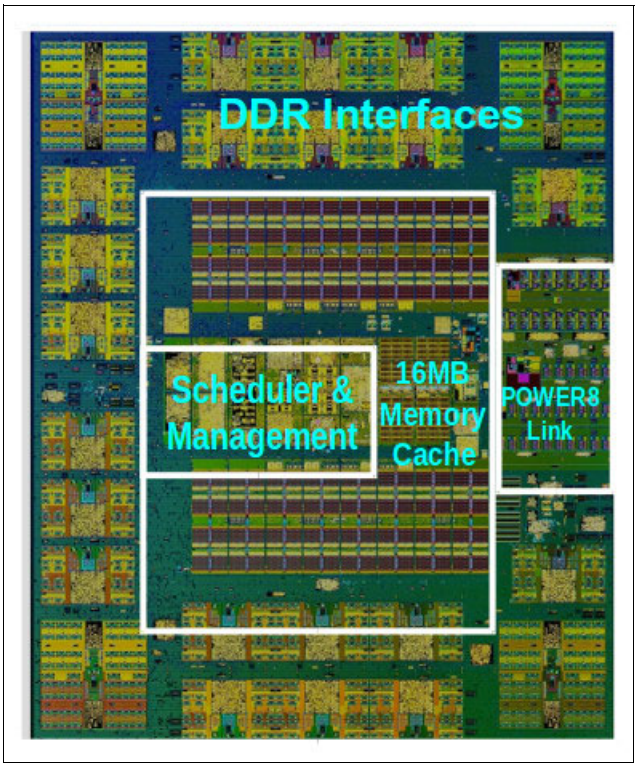


Figure 7-9 Memory buffer chip

7.6.7 Hardware transactional memory

Transactional memory is an alternative to lock-based synchronization. It attempts to simplify parallel programming by grouping read and write operations and running them as a single operation. Transactional memory is similar to database transactions, where all shared memory accesses and their effects are committed together or discarded as a group.

All threads can enter the critical region simultaneously. If there are conflicts in accessing the shared memory data, threads try accessing the shared memory data again or are stopped without updating the shared memory data. Therefore, transactional memory is also called a *lock-free synchronization*. Transactional memory can be a competitive alternative to lock-based synchronization.

Transactional memory provides a programming model that simplifies parallel programming. A programmer delimits regions of code that access shared data and the hardware runs these regions atomically and in isolation, buffering the results of individual instructions, and trying execution again if isolation is violated. Generally, transactional memory allows programs to use a programming style that is close to coarse-grained locking to achieve performance that is close to fine-grained locking.

Most implementations of transactional memory are based on software. The POWER8 processor-based systems provide a hardware-based implementation of transactional memory that is more efficient than the software implementations and requires no interaction with the processor core. This configuration allows the system to operate in maximum performance.

7.7 Memory subsystem

The S822LC for HPC server is a two-socket system that supports two POWER8 SCM processor modules. The server supports a maximum of 32 DDR4 RDIMMs slots that are housed in eight memory riser cards.

Memory features equate to a riser card with four memory DIMMs. The following memory feature codes are supported:

- 16 GB
- 32 GB
- 64 GB
- 128 GB

The memory feature codes run at speeds of 1600 MHz, which allows for a maximum system memory of 1024 GB.

7.7.1 Memory riser cards

Memory riser cards are designed to house up to four industry-standard DRAM memory DIMMs and include the following set of components that allow for higher bandwidth and lower latency communications:

- Memory scheduler
- Memory management; reliability, availability, and serviceability (RAS) decisions and energy management
- Buffer cache

By adopting this architecture, several decisions and processes regarding memory optimizations are run outside the processor, which saves bandwidth and allows for faster processor to memory communications. It also allows for more robust RAS.

Figure 7-10 shows the memory riser card that is available for the S822LC for HPC server and its location on the server.

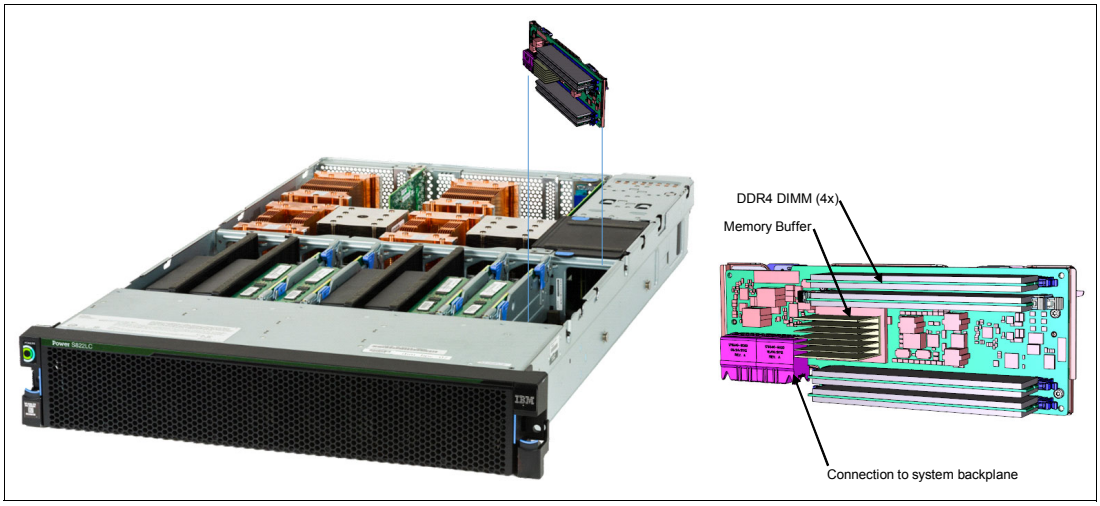


Figure 7-10 Memory riser card components and server location

The buffer cache is an L4 cache and is built on eDRAM technology (same as the L3 cache), which has lower latency than regular SRAM. Each memory riser card has a buffer chip with 16 MB of L4 cache; a fully populated server (two processors and eight memory riser cards) has 128 MB of L4 cache. The L4 cache performs the following functions that have a direct effect on performance and realize several benefits for the server:

- Reduces energy consumption by reducing the number of memory requests.
- Increases memory write performance by acting as a cache and by grouping several random writes into larger transactions.
- *Gathers* partial write operations that target the same cache block within the L4 cache before written to memory, becoming a single write operation.
- Reduces latency on memory access. Memory access for cached blocks has up to 55% lower latency than noncached blocks.

7.7.2 Memory placement rules

Each feature code equates to a riser card with four memory DIMMs. You can order the following memory feature codes:

- 16 GB DDR4: A riser card with four 4 GB 1600 MHz DDR4 DRAMs (#EM55)
- 32 GB DDR4: A riser card with four 8 GB 1600 MHz DDR4 DRAMs (#EM56)
- 64 GB DDR4: A riser card with four 16 GB 1600 MHz DDR4 DRAMs (#EM57)
- 128 GB DDR4: A riser card with four 32 GB 1600 MHz DDR4 DRAMs (#EM58)

The supported maximum memory is 1024 GB by installing a quantity of eight #EM58 components. For the S822LC for HPC server (8335-GTB), the following requirements apply:

- All the memory modules must be populated
- Memory features cannot be mixed

- The base memory is 128 GB with eight 16 GB, 1600 MHz DDR4 memory modules (#EM55)
- Memory upgrades are not supported

Table 7-5 lists the supported quantities for each memory feature code.

Table 7-5 Supported quantity of feature codes for model 8335-GTB

	Total installed memory			
Memory features	**128 GB**	**256 GB**	**512 GB**	**1024 GB**
16 GB (#EM55)	8			
32 GB (#EM56)		8		
64 GB (#EM57)			8	
128 GB (#EM58)				8

7.7.3 Memory bandwidth

The POWER8 processor has exceptional cache, memory, and interconnect bandwidths. Table 7-6 lists the maximum bandwidth estimates for a single core on the server.

Table 7-6 The S822LC for HPC server single-core bandwidth estimates

Single core	**8335-GTB**	
	2.860 GHz	**3.259 GHz**
L1 (data) cache	137.28 GBps	156.43 GBps
L2 cache	137.28 GBps	156.43 GBps
L3 cache	183.04 GBps	208.57 GBps

The bandwidth figures for the caches are calculated as follows:

- L1 cache: In one clock cycle, two 16-byte load operations and one 16-byte store operation can be accomplished. The value varies depending on the clock of the core, and the following formulas are used:
  - 2.860 GHz Core: $(2 \times 16\ B + 1 \times 16\ B) \times 2.860\ GHz = 137.28\ GBps$
  - 3.259 GHz Core: $(2 \times 16\ B + 1 \times 16\ B) \times 3.259\ GHz = 156.43\ GBps$
- L2 cache: In one clock cycle, one 32-byte load operation and one 16-byte store operation can be accomplished. The value varies depending on the clock of the core, and the following formulas are used:
  - 2.860 GHz Core: $(1 \times 32\ B + 1 \times 16\ B) \times 2.860\ GHz = 137.28\ GBps$
  - 3.259 GHz Core: $(1 \times 32\ B + 1 \times 16\ B) \times 3.259\ GHz = 156.43\ GBps$
- L3 cache: One 32-byte load operation and one 32-byte store operation can be accomplished at half-clock speed. The following formulas are used:
  - 2.860 GHz Core: $(1 \times 32\ B + 1 \times 32\ B) \times 2.860\ GHz = 183.04\ GBps$
  - 3.259 GHz Core: $(1 \times 32\ B + 1 \times 32\ B) \times 3.259\ GHz = 208.57\ GBps$

Table 7-7 lists the overall bandwidths for the entire S822LC for HPC server populated with the two processor modules.

Table 7-7 S822LC for HPC server total bandwidth estimates

Total bandwidths	8335-GTB	
	20 cores @ 2.860 GHz	16 cores @ 3.259 GHz
L1 (data) cache	2746 GBps	2503 GBps
L2 cache	2746 GBps	2503 GBps
L3 cache	3661 GBps	3337 GBps
Total memory	230 GBps	230 GBps
PCIe interconnect	128 GBps	128 GBps

Consider the following points:

- Total memory bandwidth

  Each POWER8 processor has four memory channels that are running at 9.6 GBps, which is capable of reading 2 bytes and writing 1 byte at a time. The bandwidth is calculated by using the following formula:

  4 channels × 9.6 GBps × 3 bytes = 115.2 GBps per processor module

- SMP interconnect

  The POWER8 processors are connected by using an A-bus. The bandwidth is calculated by using the following formula:

  1 A-bus × 8 bytes × 4.8 GHz = 38.4 GBps

- PCIe interconnect: Each POWER8 processor has 32 PCIe lanes that are running at 8 Gbps full-duplex. The bandwidth is calculated by using the following formula:

  32 lanes × 2 processors × 8 Gbps × 2 = 128 GBps

7.8 POWERAccel

POWERAccel is an emerging term for a family of technologies that provides high-bandwidth connections between the processor, memory, and I/O. PCI Express in combination with CAPI and NVLink provide the foundation for POWERAccel.

For more information about POWERAccel, see the following resources:

- *How Power Systems and OpenPOWER enable acceleration*
- IBM PowerAccel blog

7.8.1 PCIe

PCIe uses a serial interface and allows for point-to-point interconnections between devices by using a directly wired interface between these connection points. A single PCIe serial link is a dual-simplex connection that uses two pairs of wires (one pair for transmit and one pair for receive) and can transmit only one bit per cycle. These two pairs of wires are called a *lane*. A PCIe link can consist of multiple lanes. In these configurations, the connection is labeled as x1, x2, x8, x12, x16, or x32, where the number is effectively the number of lanes.

The PCIe interfaces that are supported on this server are PCIe Gen3, which are capable of 16 GBps simplex (32 GBps duplex) on a single x16 interface. PCIe Gen3 slots also support previous generation (Gen2 and Gen1) adapters, which operate at lower speeds according to the following rules:

- Place x1, x4, x8, and x16 speed adapters in the same size connector slots first before mixing adapter speed with connector slot size.
- Adapters with lower speeds are allowed in larger sized PCIe connectors, but larger speed adapters are not compatible in smaller connector sizes (that is, a x16 adapter cannot be installed in an x8 PCIe slot connector).

PCIe adapters use a different type of slot than PCI adapters. If you attempt to force an adapter into the wrong type of slot, you might damage the adapter or the slot.

POWER8-based servers can support the following form factors of PCIe adapters:

- PCIe low profile (LP) cards, which are used with the S822LC for HPC server.
- PCIe full-height and full-high cards, which are designed for the 4 EIA scale-out servers, such as the Power S824L server.

Before adding or rearranging adapters, use the System Planning Tool to validate the new adapter configuration. For more information, see the IBM System Planning Tool for POWER processor-based systems website.

If you are installing a new feature, ensure that you have the software that is required to support the new feature and determine whether there are update prerequisites to install. For more information, see the IBM prerequisite website.

The following sections describe the supported adapters and provide tables of feature code numbers that you can use to order.

7.8.2 CAPI

CAPI defines a coherent accelerator interface structure for attaching special processing devices to the POWER8 processor bus. The CAPI can attach accelerators that have coherent shared memory access with the processors in the server and share full virtual address translation with these processors, which use a standard PCIe Gen3 bus.

Applications can have customized functions in FPGAs and enqueue work requests directly in shared memory queues to the FPGA. Applications can also have customized functions by using the same effective addresses (pointers) they use for any threads that are running on a host processor. From a practical perspective, CAPI allows a specialized hardware accelerator to be seen as another processor in the system with access to the main system memory and coherent communication with other processors in the system.

The benefits of using CAPI include the ability to access shared memory blocks directly from the accelerator, the ability to perform memory transfers directly between the accelerator and processor cache, and the ability to reduce the code path length between the adapter and the processors. This reduction in the code path length might occur because the adapter is not operating as a traditional I/O device, and there is no device driver layer to perform processing. CAPI also presents a simpler programming model.

Figure 7-11 shows a high-level view of how an accelerator communicates with the POWER8 processor through CAPI. The POWER8 processor provides a Coherent Attached Processor Proxy (CAPP), which is responsible for extending the coherence in the processor communications to an external device. The coherency protocol is tunneled over standard PCIe Gen3, effectively making the accelerator part of the coherency domain.

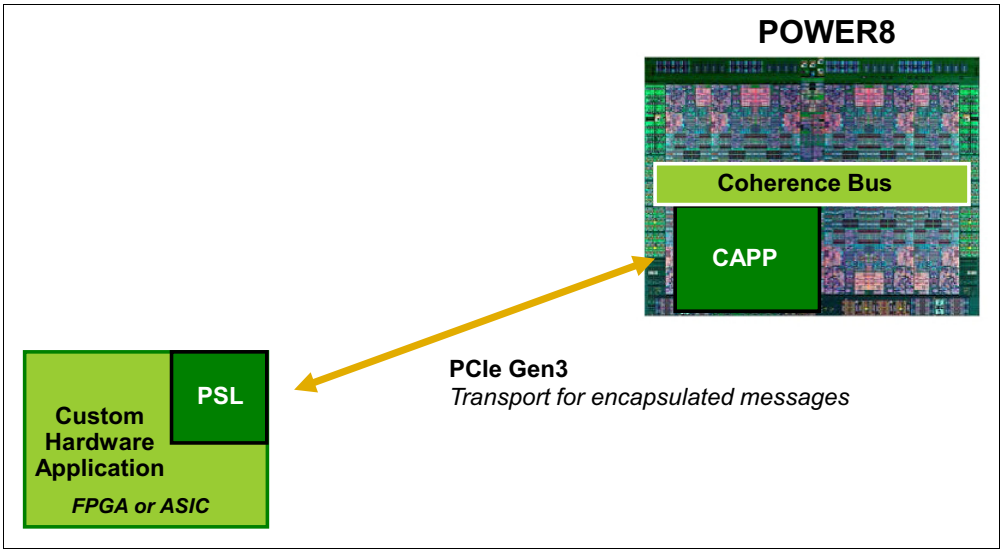


Figure 7-11 CAPI accelerator attached to the POWER8 processor

The accelerator adapter implements the Power Service Layer (PSL), which provides address translation and system memory cache for the accelerator functions. The custom processors on the system board, which consist of an FPGA or an ASIC, use this layer to access shared memory regions, and cache areas as though they were a processor in the system. This ability enhances the performance of the data access for the device and simplifies the programming effort to use the device.

Instead of treating the hardware accelerator as an I/O device, it is treated as a processor, which eliminates the requirement of a device driver to perform communication tasks. It also eliminates the need for direct memory access that requires system calls to the operating system kernel. By removing these layers, the data transfer operation requires fewer clock cycles in the processor, which improves the I/O performance.

The implementation of CAPI on the POWER8 processor allows hardware companies to develop solutions for specific application demands. Companies use the performance of the POWER8 processor for general applications and the custom acceleration of specific functions. They do so by using a hardware accelerator with a simplified programming model and efficient communication with the processor and memory resources.

For more information about supported CAPI adapters, see 7.10.4, “CAPI-enabled InfiniBand adapters” on page 345.

7.8.3 NVLink

NVLink is NVIDIA's high-speed interconnect technology for GPU-accelerated computing. Supported on SXM2-based Tesla P100 accelerator boards, NVLink significantly increases performance for GPU-to-GPU communications and for GPU access to system memory.

Multiple GPUs are common in workstations, as are the nodes of high-performance computing clusters and deep-learning training systems. A powerful interconnect is extremely valuable in multiprocessing systems. NVLink creates an interconnect for GPUs that offer higher bandwidth than PCI Express Gen3 (PCIe) and are compatible with the GPU ISA to support shared memory multiprocessing workloads.

Support for the GPU ISA allows programs that are running on NVLink-connected GPUs to run directly on data in the memory of another GPU and on local memory. GPUs can also perform atomic memory operations on remote GPU memory addresses, which enables much tighter data sharing and improved application scaling.

NVLink uses NVIDIA's new High-Speed Signaling interconnect (NVHS). NVHS transmits data over a differential pair that is running at up to 20 Gbps. Eight of these differential connections form a *sublink* that sends data in one direction, and two sublinks (one for each direction) form a *link* that connects two processors (GPU-to-GPU or GPU-to-CPU). A single link supports up to 40 GBps of bidirectional bandwidth between the endpoints. Multiple links can be combined to form *gangs* for even higher-bandwidth connectivity between processors. The NVLink implementation in Tesla P100 supports up to four links, which allows for a gang with an aggregate maximum theoretical bandwidth of 160 GBps bidirectional bandwidth.

Although NVLink primarily focuses on connecting multiple NVIDIA Tesla P100s, it can also connect Tesla P100 GPUs with IBM Power CPUs with NVLink support. Figure 7-12 shows how the CPUs are connected with NVLink in the S822LC for HPC server. In this configuration, each GPU has 180 GBps bidirectional bandwidth to the other connected GPU and 80 GBps bidirectional bandwidth to the connected CPU.

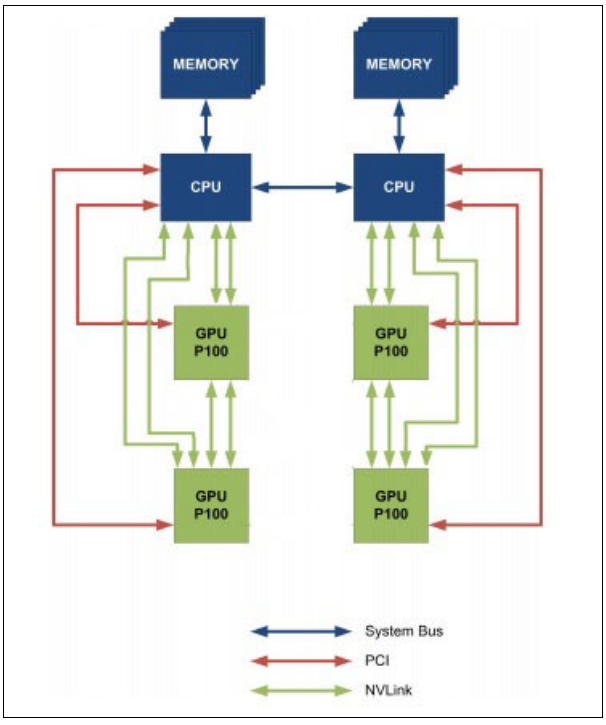


Figure 7-12 CPU to GPU and GPU to GPU interconnect using NVLink

7.9 System bus

This section provides more information about the internal buses of the S822LC for HPC server.

The S822LC for HPC server has internal I/O connectivity through PCIe Gen3 slots. The internal I/O subsystem on the systems is connected to the PCIe controllers on a POWER8 processor in the system. Each POWER8 processor has a bus that has 32 PCIe lanes that are running at 9.6 Gbps full-duplex and provides 64 GBps of I/O connectivity to the PCIe slots, SAS internal adapters, and USB ports.

Some PCIe devices are connected directly to the PCIe Gen3 buses on the processors. Other devices are connected to these buses through PCIe Gen3 switches. The PCIe Gen3 switches are high-speed devices (512 - 768 GBps each) that allow for the optimal use of the processors PCIe Gen3 x16 buses. The switches do so by grouping slower x8 or x4 devices that might plug into a x8 slot and not use its full bandwidth. For more information about which slots are connected directly to the processor and which slots are attached to PCIe Gen3 switches (referred to as *PLX*), see 7.6, “POWER8 processor” on page 329.

Figure 7-13 shows the server buses and logical architecture.

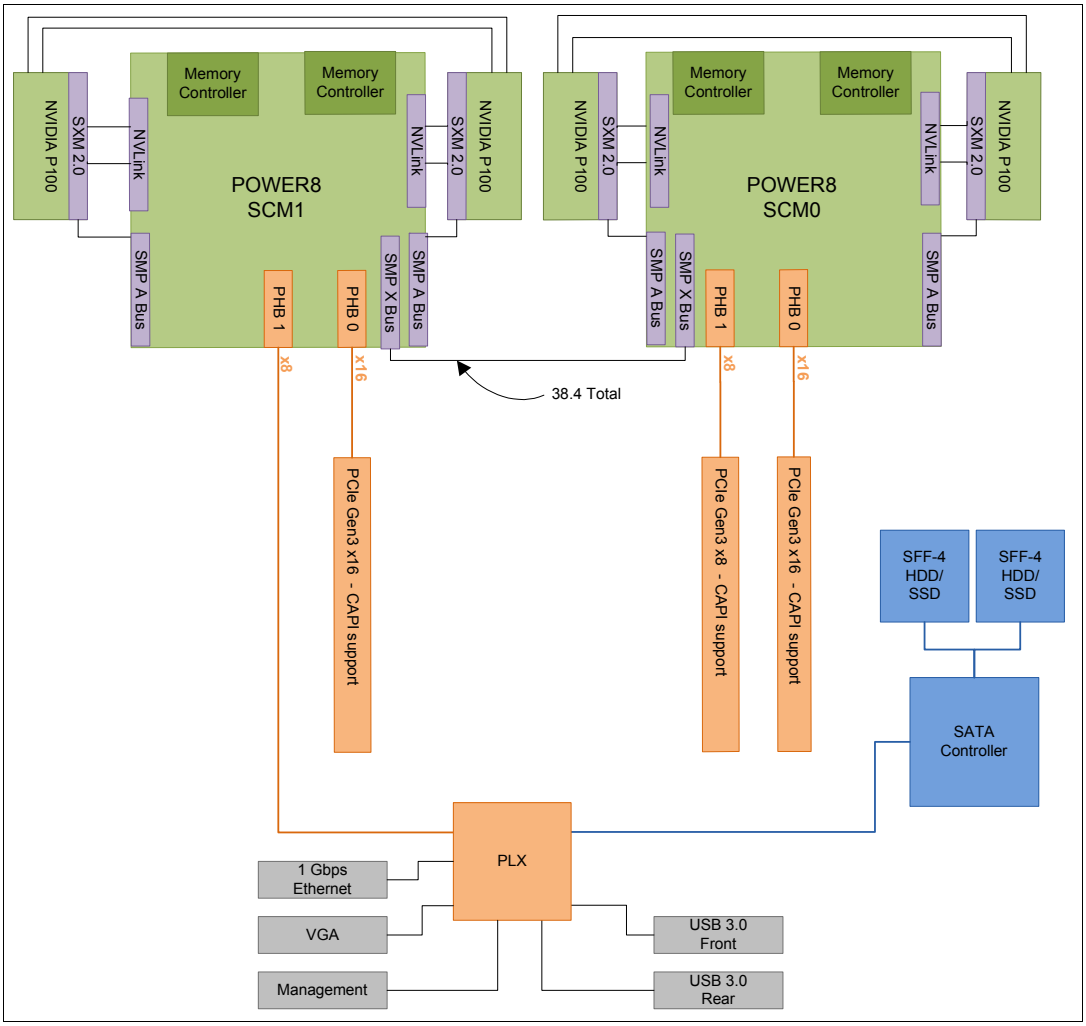


Figure 7-13 S822LC for HPC server buses and logical architecture

Each processor has 32 PCIe lanes split into the following channels:

- Two PCIe Gen3 x8 channels
- One PCIe Gen 3 x16 channel

The PCIe channels are connected to the PCIe slots, which can support GPUs and other high-performance adapters, such as InfiniBand.

Table 7-8 lists the total I/O bandwidth of an S822LC for HPC server.

Table 7-8 I/O bandwidth

I/O	I/O bandwidth (maximum theoretical)
Total I/O bandwidth	► 64 GBps simplex ► 128 GBps duplex

For the PCIe Interconnect, each POWER8 processor has 32 PCIe lanes that are running at 9.6 Gbps full-duplex. The bandwidth formula is calculated as follows:

`Thirty-two lanes × 2 processors × 9.6 Gbps × 2 = 128 GBps`

7.10 PCI adapters

This section describes the types and functions of the PCI adapters that are supported by the S822LC for HPC server.

Note: PCIe adapters on the S822LC for HPC server are not hot-pluggable.

7.10.1 Slot configuration

The S822LC for HPC server has three PCIe Gen3 slots. Figure 7-14 shows a rear-view of the PCIe slots.

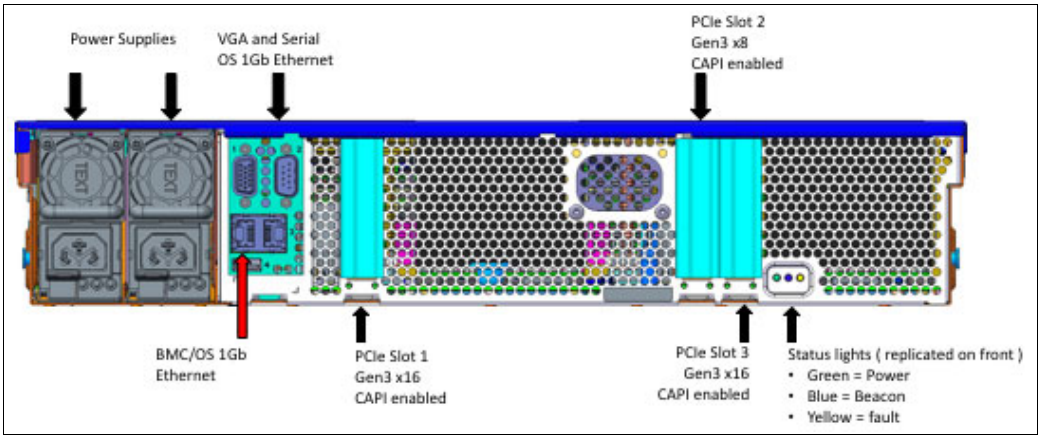


Figure 7-14 Rear-view PCIe slots and connectors

Table 7-9 provides the PCIe Gen3 slot configuration.

Table 7-9 S822LC for HPC server PCIe slot properties

Slot	Description	Card size	CAPI capable	Power limit
Slot 1	PCIe Gen3 x16	Half height, half length	Yes	75 W
Slot 2	PCIe Gen3 x8	Half height, half length	Yes	50 W
Slot 3	PCIe Gen3 x16	Full height, half length	Yes	75 W

Only LP adapters can be placed in LP slots. A x8 adapter can be placed in a x16 slot, but a x16 adapter cannot be placed in a x8 slot. One LP slot must be used for a required Ethernet adapter (#5260, #EL3Z, or #EN0T).

7.10.2 LAN adapters

To connect the S822LC for HPC server to a local area network (LAN), you can use the LAN adapters that are supported in the PCIe slots of the system unit. Table 7-10 lists the supported LAN adapters for the server.

Table 7-10 Supported LAN adapters

Feature code	Description	Max	OS support
EC3A	PCIe3 LP 2-Port 40 GbE NIC RoCE QSFP+ Adapter	2	Linux
EL3Z	PCIe2 LP 2-port 10/1 GbE BaseT RJ45 Adapter	3	Linux
EL4M	PCIe2 x4 LP 4-port (UTP) 1 GbE Adapter	3	Linux
EN0T	PCIe2 LP 4-Port (10 Gb + 1 GbE) SR+RJ45 Adapter	3	Linux
EN0v	PCIe2 LP 4-port (10 Gb + 1 GbE) Copper SFP+RJ45 Adapter	3	Linux

7.10.3 Fibre Channel adapters

The S822LC for HPC server supports direct or SAN connection to devices that use Fibre Channel adapters. Table 7-11 lists the available Fibre Channel adapter that includes LC connectors.

Table 7-11 Supported Fibre Channel adapters

Feature code	Description	Max	OS support
EL43	PCIe3 LP 16 Gb 2-port Fibre Channel adapter	2	Linux

If you are attaching a device or switch with an SC-type fiber connector, an LC-SC 50-micron fibre converter cable (#2456) or an LC-SC 62.5-micron fibre converter cable (#2459) is required.

7.10.4 CAPI-enabled InfiniBand adapters

Table 7-12 lists the available CAPI adapters.

Table 7-12 Available CAPI adapters

Feature code	Description	Maximum	OS support
EC3E	PCIe3 LP 2-port 100 Gb EDR InfiniBand Adapter x16	2	Linux
EC3T	PCIe3 LP 1-port 100 Gb EDR InfiniBand Adapter x16	2	Linux

7.10.5 Compute intensive accelerator

Compute intensive accelerators are GPUs that are developed by NVIDIA. With NVIDIA GPUs, the server can offload processor-intensive operations to a GPU accelerator and boost performance. The S822LC for HPC server aims to deliver a new class of technology that maximizes performance and efficiency for all types of scientific, engineering, Java, big data analytics, and other technical computing workloads.

Table 7-13 lists the available compute-intensive accelerators.

Table 7-13 Supported graphics processing units adapters

Feature code	Description	Max	OS support
EC4C	Two air-cooled NVIDIA Tesla P100 GPUs (for first pair)	2	Linux
EC4D	Two air-cooled NVIDIA Tesla P100 GPUs (for second pair)	2	Linux
EC4F	Four water-cooled NVIDIA Tesla P100 GPUs	4	Linux

7.10.6 Flash storage adapters

The available flash storage adapters are listed in Table 7-14.

Table 7-14 Available flash storage adapters

Feature code	CCIN	Description	Max	OS support
EC54	58CB	PCIe3 1.6 TB NVMe Flash Adapter	7	Linux
EC56	58CC	PCIe3 3.2 TB NVMe Flash Adapter	7	Linux

7.11 System ports

The system board has one 1 Gbps Ethernet port, one Intelligent Platform Management Interface (IPMI) port, and a VGA port, as shown in Figure 7-14 on page 343.

The integrated system ports are supported for modem and asynchronous terminal connections with Linux. Any other application that uses serial ports requires a serial port adapter to be installed in a PCI slot. The integrated system ports do not support IBM PowerHA® SystemMirror® configurations. The VGA port does not support cable lengths that exceed 3 meters.

7.12 Internal storage

The internal storage on the S822LC for HPC server includes the following features, as listed in Table 7-15:

- A storage backplane for two 2.5-inch SFF Gen4 SATA HDDs or SDDs.

  Limitation: The disks use an SFF-4 carrier. Disks that are used in other Power Systems servers usually have an SFF-3 or SFF-2 carrier and are not compatible with this system.

- One integrated SATA disk controller without redundant array of independent disks (RAID) capability.
- The storage split backplane feature is not supported.

Table 7-15 Integrated SATA disk controller features

Feature	Integrated SATA disk controller
Supported RAID types	JBOD
Disk bays	Two SFF Gen4 (HDDs/SDDs)
SATA controllers	Single
IBM Easy Tier® capable controllers	No
External SAS ports	No
Split backplane	No

The 2.5 inch or SFF SAS bays can contain SATA drives (HDD or SSD) that are mounted on a Gen4 tray or carrier (also knows as SFF-4). SFF-2 or SFF-3 drives do not fit in an SFF-4 bay. All SFF-4 bays support concurrent maintenance or hot-plug capability.

Figure 7-15 shows the server front view with the standard backplane.

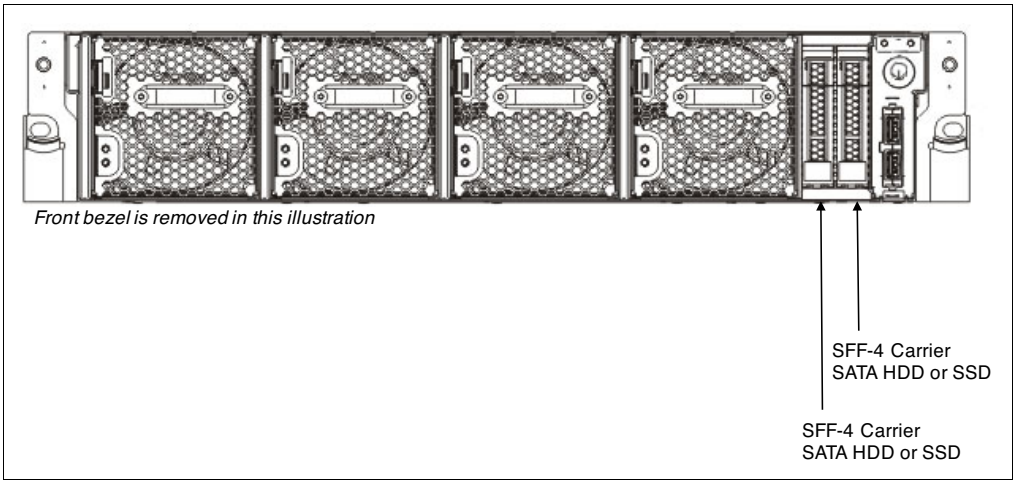


Figure 7-15 Server front view with SFF-4 locations

Figure 7-16 shows the logical connections of the integrated SATA disk controller.

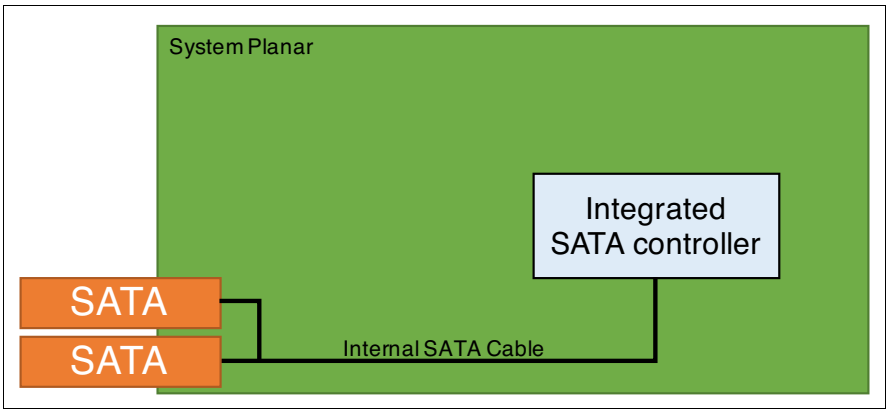


Figure 7-16 Logical diagram for integrated SATA disk controller

7.12.1 Disk and media features

The server supports the attachment of up to two SATA storage devices, as listed in Table 7-16.

Table 7-16 Supported storage devices

Feature code	Description	Max	OS support
ELD0	1 TB 7.2k RPM SATA SFF-4 disk drive	2	Linux
ES6A	2 TB 7.2k RPM 5xx SATA SFF-4 disk drive	2	Linux

The S822LC for HPC server is designed for network installation or USB media installation. It does not support an internal DVD drive.

7.13 External I/O subsystems

The S822LC for HPC server does not support external PCIe Gen3 I/O expansion drawers or EXP24S SFF Gen2-bay drawers.

7.13.1 BMC

The IBM Power System S822LC (8335-GTB) server features a baseboard management controller (BMC) for out-of-band system management. Some functions are also available by way of in-band methods. This system management differs from previous lines of IBM Power System servers, which traditionally featured a flexible service processor (FSP) for that purpose, with increased focus on RAS functions.

The BMC offers functions that are more focused on scale-out environments, such as HPC clusters and cloud platform infrastructure, where factors (such as automation and simplicity for deployment and maintenance) play an important role. For example, the BMC provides the following functions:

- Power management
- Console (or terminal) sessions
- Network and boot device configuration
- Sensors information (for example, temperature, fan speed, and voltage)
- Firmware and vital product data (VPD) information (for example, firmware components version, serial number, and machine-type model)
- Virtual hard disk and optical drives (for example, for installing operating systems)
- System firmware upgrade

The BMC functions are available by way of several methods, for example:

- IPMI (in-band and out-of-band)
- Advanced System Management Interface (ASMI)
- Secure Shell (SSH)

The BMC provides the following ports:

- Ethernet port, for out-of-band IPMI, ASMI (web), and SSH access
- Video Graphics Adapter (VGA) port, for graphics display (functional from Petitboot onward)
- Serial port
- Internal (in-chassis) serial port

7.14 Mellanox InfiniBand

The IBM Power System S822LC (8335-GTB) server can include one-port or two-port high bandwidth and low latency Mellanox InfiniBand host channel adapters (HCAs). These HCAs connect to the bus through PCIe 3.0 x16, which can deliver 100 Gbps of data at each EDR port.

InfiniBand provides the following significant features, among others:

- Delivers more than 100 Gb per second overall throughput
- Implements Virtual Protocol Interconnect (VPI)
- Takes over transport operations to offload the CPU
- Supports noncontinuous memory transfers
- Supported by IBM Spectrum MPI

Device drivers and support tools are available with the Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux.

7.15 IBM System Storage

The IBM System Storage® disk systems products and offerings provide compelling storage solutions with value for all levels of business, from entry-level to high-end storage systems. For more information about the offerings, see the Disk storage page of the IBM Storage website.

The following section highlights a few of the offerings.

7.15.1 IBM Storwize family

The IBM Storwize® family is the ideal solution to optimize the data architecture for business flexibility and data storage efficiency. Different models, such as the IBM Storwize V3700, IBM Storwize V5000, and IBM Storwize V7000, offer storage virtualization, IBM Real-time Compression™, Easy Tier, and many more functions. For more information, see the IBM Storwize family page of the IBM Storage website.

7.15.2 IBM FlashSystem family

The IBM FlashSystem® family delivers extreme performance to derive measurable economic value across the data architecture (servers, software, applications, and storage). IBM offers a comprehensive flash portfolio with the IBM FlashSystem family. For more information, see the IBM Flash Storage is the Future page of the IBM Storage website.

7.15.3 IBM XIV Storage System

The IBM XIV® Storage System is a high-end disk storage system that helps thousands of enterprises meet the challenge of data growth with hot spot-free performance and ease of use. Simple scaling, high service levels for dynamic, heterogeneous workloads, and tight integration with hypervisors and the OpenStack platform enable optimal storage agility for cloud environments.

XIV Storage Systems extend ease of use with integrated management for large and multi-site XIV deployments, reducing operational complexity, and enhancing capacity planning. For more information, see the IBM XIV Storage System page of the IBM IT infrastructure website.

7.15.4 IBM Elastic Storage Server

The IBM Elastic Storage™ Server is software-defined storage that combines IBM Spectrum Scale, which provides the clustered file systems, and the CPU and I/O capability of the IBM POWER8 architecture.

The building block-based solution of the Elastic Storage Server delivers high performance, high available, and scalable IBM Spectrum Scale functions to today's high performance and business analytics clusters. For more information, see the IBM Elastic Storage Server page of the IBM IT infrastructure website.

8

Software stack

This chapter describes the software stack that is used in the implementation of an IBM High Performance Computing (HPC) solution on an IBM POWER8 with the IBM Power System S822LC (8335-GTB) HPC server.

This chapter includes the following topics:

8.1 System management

The baseboard management controller (BMC) provides the system management functions by way of the following methods:

- The Advanced System Management Interface (ASMI)
- The Secure Shell (SSH) protocol
- The Intelligent Platform Management Interface (IPMI) protocol

You can access each method by way of one or more BMC IP addresses with the following software:

- ASMI: Any standards-compliant web browser, by way of both (Secure) Hypertext Transfer Protocols (HTTP and HTTPS)
- SSH: Any standards-compliant SSH client
- IPMI: The IPMItool utility, version 1.8.17 and later (requirement for some functions)

  The IPMI method is available out-of-band (from other systems, by way of network), and in-band (from the current system, by way of internal communication).

  For more information about build instructions for IPMItool, see 5.4.8, “IPMI authentication credentials” on page 224.

  For more information about the IPMItool, see the IPMtool page of the Sourceforge website.

You also can configure access credentials for each method on the ASMI.

8.2 OPAL firmware

The Open Power Abstraction Layer (OPAL) firmware is available on the IBM Power System 8335-GTB server and several other IBM Power Systems servers with POWER8 processors. The OPAL firmware supports running Linux in non-virtualized (or bare-metal) mode and virtualized mode (guest or virtual machine) with kernel-based virtual machine (KVM) acceleration.

With the OPAL firmware, several of the system management functions are performed by way of IPMI rather than Hardware Management Console (HMC), which was used in previous generations of IBM Power Systems servers. This configuration makes systems with OPAL firmware more suited for a wider range of environments and system management tools, which allows for more choice and integration between software and hardware components.

For more information about the OPAL firmware, see the OPAL firmware open source project page.

8.3 xCAT

The Extreme Cluster/Cloud Administration Toolkit (xCAT) performs the roles of deployment and management of the software stack. Among other tasks, it controls the node discovery process, power management, console sessions, operating system provisioning, and software stack installation and configuration.

xCAT is open source software, and relatively recently moved toward more openness, adopting significant changes to its development process and documentation pages. The development process is now hosted on GitHub (with public milestones and schedules, issue reporting and tracking, and source-code pull requests), and the documentation pages are refreshed and hosted on Read The Docs (a GitHub service).

xCAT also provides community support, and support options are available from IBM.

xCAT 2.12 release introduces support for the IBM Power System 8335-GTB server, which is accompanied with Linux distributions, installation modes, and the following features:

- Red Hat Enterprise Linux (RHEL) Server 7.3 for PowerPC 64-bit Little-Endian (ppc64le) in non-virtualized (or bare-metal) mode
- CUDA Toolkit for NVIDIA graphics processing units (GPUs)
- Mellanox OpenFabrics Enterprise Distribution (OFED) for Linux
- IBM HPC software support with xCAT kits
- System management: Hardware discovery, hardware control, and console sessions
- Diskful and diskless installation

For more information, see the following resources:

- xCAT project page
- xCAT documentation page
- GitHub xCAT 2.12 Release Notes page

8.4 RHEL server

RHEL is the world's leading enterprise Linux platform,[1] which runs on highly scalable, multi-core systems that support the most demanding workloads. Collaboration between Red Hat and engineers from major hardware vendors ensures that the operating system uses the newest hardware innovations that are available in chip design, system architecture, and device drivers to improve performance and reduce power utilization.

For more information, see the RHEL documentation at the Product Documentation for Red Hat Enterprise Linux page of the Red Hat website.

8.5 NVIDIA CUDA Toolkit

The Compute Unified Device Architecture (CUDA) is a programming model and application programming interface (API) that uses the NVIDIA General Purpose Graphics Processing Unit (GPGPU), which provides a highly scalable parallel computing platform.

[1] According to the Red Hat Enterprise Linux data sheet published by Red Hat, Inc. (#12182617_V1_0514).

The toolkit offers a multi-platform apparatus to develop, compile, debug, profile, and run CUDA programs on GPUs. In addition to APIs and runtime libraries, the following main resources are bundled:

- nvcc: A CUDA compiler (nvcc)
- cuda-gdb: A command-line program debugger
- cuda-memcheck: Suite of tools for dynamic error detection
- profiler: A command-line program profiling tool (nvprof) and GUI-oriented (Visual Profiler)
- Binary utilities
- Development libraries
- Nsight Eclipse Edition: An Eclipse-based integrated development environment (IDE)

Note: At the time of this writing, the latest CUDA Toolkit version 8.0 is fully supported on the IBM Power 8335-GTB System with RHEL version 7.3 for Linux on POWER8 Little-endian.

In many sections of this book, references are made to some aspects of the CUDA Toolkit usage in the context of HPC on IBM Power Systems. For more information about the toolkit, see NVIDIA's CUDA Toolkit Documentation v8.0.61 website.

8.6 Mellanox OFED for Linux

Mellanox OFED for Linux is a version of the OFED distribution from the OpenFabrics Alliance that is tested and packaged by Mellanox. It supports Remote Direct Memory Access (RDMA) and kernel bypass APIs (OFED verbs) over InfiniBand and Ethernet.

The Mellanox OFED for Linux includes the following components:

- Drivers for InfiniBand, RDMA over Converged Ethernet (RoCE), L2 network interface controller (NIC)
- Access Layers and common verbs interface
- Virtual Protocol Interconnect (VPI)
- IP-over-IB
- Subnet Manager (OpenSM)
- Installation, administration, and diagnostics tools
- Performance test suites

For more information, see the Mellanox website.

8.7 IBM XL compilers, GCC, and Advance Toolchain

This section describes some of the compiler options that are available for the software stack, such as the IBM XL compilers, GNU Compiler Collection (GCC), and IBM Advance Toolchain (more recent GCC and libraries than in the Linux distribution).

8.7.1 XL compilers

IBM XL compilers enhancements help to increase application performance and developer productivity. XL Fortran v15.1.4 and v15.1.5 and XLC/C++ v13.1.2 and v13.1.5 compilers support the latest Linux distributions, including RHEL 7.3, and all of the new features of the POWER8 processor, including the latest built-in vector intrinsics.

The XL Fortran compiler improves support of the following features from release to release:

- Intrinsic procedures, which help to increase utilization of POWER8 processors
- New compiler options
- Fortran 2008 features
- Language interoperability, with which developers write programs that contain parts that are written in Fortran and parts that are written in the C language
- OpenMP (OpenMP Application Program Interface Version 3.1 specification and partially supports the OpenMP Application Program Interface Version 4.5 specification)

The following changes are included in the latest releases of the XL C/C++ compilers:

- Support of new built-in functions for POWER8 processors
- More compiler options
- Increase support of the following C/C++ standards:
  - C++14
  - C++11
  - C11
- Partial support of OpenMP 4.0 (Version 13.1.2 fully supports only the OpenMP Application Program Interface Version 3.1 specification).

For more information about XL Fortran support for POWER8 processor, see the following resources:

- IBM XL Fortran for Linux, V15.1.4 (little endian distributions) documentation
- IBM XL Fortran for Linux, V15.1.5 (little endian distributions) documentation

For more information about XL C/C++ compilers, see the following websites:

- IBM XL C/C++ for Linux, V13.1.4 (little endian distributions) documentation
- IBM XL C/C++ for Linux, V13.1.5 (little endian distributions) documentation

8.7.2 GCC and Advance Toolchain

GCC 4.8.5 is included in the RHEL 7.3 distribution. It includes features and optimizations in the common parts of the compiler that improve support for the POWER8 processor architecture.

The IBM Advance Toolchain for Linux on Power is a set of open source development tools and runtime libraries that allows users to use the latest IBM POWER8 hardware features on Linux. It supports big endian (ppc64) and little endian (ppc64le).

The latest release includes current versions of the following packages, which can help with porting and tuning applications for POWER8:

- GNU Compiler Collection (gcc, g++, gfortran), including individually optimized gcc runtime libraries for supported POWER8 processor
- GNU C library (glibc), individually optimized for supported POWER8 processor
- GNU Binary Utilities (binutils)
- AUXV Library (libauxv)
- GNU Debugger (gdb)
- Performance analysis tools (oprofile, valgrind, and itrace)
- Multi-core exploitation libraries (Intel TBB, Userspace RCU, and SPHDE)
- Plus several support libraries (libhugetlbfs, Boost, zlib, and more)

Note: For some specific workloads, GCC 5.2 provided with the IBM Advance Toolchain and GCC 4.8.5 provided with the RHEL 7.3 distribution have different performance results.

For more information about GCC support on POWER8 for RHEL 7.3, see the GCC 4.8 Release Series website.

For more information about the IBM Advance Toolchain, see the IBM Advance Toolchain for PowerLinux Documentation website.

8.8 IBM Spectrum MPI

IBM Spectrum MPI is a production-quality implementation of the Message Passing Interface (MPI) that supports the broadest range of industry standard platforms, interconnects, and operating systems to help ensure that parallel applications can run on any platform. It provides a familiar interface that is easily portable, and incorporates advanced CPU affinity features, dynamic selection of interface libraries, and superior workload manager integrations that lead to improved performance.

IBM Spectrum MPI (V10) is supported on Linux on x86 and POWER8 (Little Endian). This new version of Spectrum MPI is based on the open source Open MPI distribution.

For more information about IBM Spectrum MPI, see the IBM Spectrum MPI website.

8.8.1 IBM Parallel Performance Toolkit for POWER

IBM Parallel Performance Toolkit for POWER is the new name for IBM Parallel Environment Developer Edition for Linux on Power. Version 2.3 is supported running on the IBM Power System S822LC (8335-GTB) for High Performance Computing (HPC) server with NVIDIA Tesla P100 with NVLink Graphics processor units (GPUs) running Red Hat Enterprise Linux (RHEL) 7.3 in little-endian mode. Version 2.2 and 2.3 are supported running on the IBM Power System S822LC (8335-GTA) server with NVIDIA K80 GPUs running RHEL 7.2 in little-endian mode. It includes the following features:

- Improved scaling that uses MRNet, a software overlay network that provides efficient multicast and reduction communications for parallel and distributed tools and systems (V2.3)
- Support for OpenMP Tools, an API for performance analysis (V2.3)
- Visualization improvements of trace and profile data (V2.3)
- Enhancements to event counters and performance metrics for profiling and tracing (V2.2)
- Support for NVIDIA K80 and NVIDIA Tesla P100 GPUs and PCIe InfiniBand EDR adapters interconnected that use EDR InfiniBand switches

For more information about the toolkit, see *IBM Parallel Performance Toolkit for POWER Installation and Users Guide*, SC23-7287, which is available at the Parallel Performance Toolkit page of the IBM Knowledge Center website.

8.9 IBM Engineering and Scientific Subroutine Library and IBM Parallel ESSL

The Engineering and Scientific Subroutine Library (ESSL) family of products is a state-of-the-art collection of mathematical subroutines. Running on IBM POWER8 servers and clusters, the ESSL family provides a wide range of high-performance mathematical functions for various scientific and engineering applications.

The collection features the following types of libraries:

- ESSL 5.5, which contains over 600 high-performance serial and symmetric multiprocessing (SMP) mathematical subroutines that are tuned for POWER8.
- Parallel ESSL 5.3, which contains over 125 high-performance single program, multiple data (SPMD) mathematical subroutines. These subroutines are designed to use the full power of clusters of POWER8 servers that are connected with a high-performance interconnect.

IBM ESSL includes an IBM implementation of BLAS/LAPACK and IBM Parallel ESSL includes an IBM implementation of ScaLAPACK, which are the industry standards for linear algebra subroutines. If the application uses BLAS/LAPACK/ScaLAPACK functions, you recompile your application on IBM POWER8 and link it with IBM ESSL to optimize performance.

Additionally, IBM ESSL implements some linear algebra routines that are not included in BLAS/LAPACK' for example, `_GETMI` (General Matrix Transpose [In-Place]) and `_GETMO` (General Matrix Transpose [Out-of-Place]).

For POWER8 servers and clusters (which have NVIDIA GPUs within), performance can be significantly improved by using environment variables, which enables GPU usage inside ESSL routines. The following options are available:

- GPU-only mode: All computations are performed on GPU.
- Hybrid mode: All computations are distributed between GPU and CPU to increase system use.

ESSL also provides support for the following libraries:

- Fastest Fourier Transform in the West (FFTW)
- CBLAS

All routines from the ESSL family are callable from Fortran, C, and C++.

For more information about the IBM ESSL, see the Engineering and Scientific Subroutine Library page of the IBM Knowledge Center website.

For more information about the IBM Parallel ESSL, see the Parallel Engineering and Scientific Subroutine Library page of the IBM Knowledge Center website.

8.10 IBM Spectrum Scale (formerly IBM GPFS)

IBM Spectrum Scale is a distributed, high-performance, massively scalable enterprise file system solution that addresses the most challenging demands in high-performance computing. It is a proven solution that is used to store the data for thousands of mission-critical commercial installations worldwide.

The IBM Spectrum Scale is software-defined storage for high-performance, large-scale workloads on-premises or in the cloud. This scale-out storage solution provides file, object, and integrated data analytics for the following items:

- Compute clusters (technical computing, and high-performance computing)
- Big data and analytics
- Hadoop Distributed File System (HDFS)
- Private cloud
- Content repositories
- File Placement Optimizer (FPO)

For more information, see the IBM Spectrum Scale website.

8.11 IBM Spectrum LSF (formerly IBM Platform LSF)

The IBM Spectrum LSF is a workload management that coordinates shared access and optimized use of computing resources of an HPC cluster. It provides the following features:

- Policy-driven work scheduling and load balancing
- Compute resources allocation
- Cluster resources administration
- Cluster monitoring
- Supports heterogeneous resources and multi-cluster
- Fault tolerance
- Security

At the time this writing, the latest version (10.1.0) supports the IBM Power System 8335-GTB server with RHEL version 7.3. Spectrum LSF is also fully integrated with the IBM Parallel Environment.

Cluster users and administrators can interact with Spectrum LSF by way of the command-line tools, web interface (provided by the IBM Platform Center), or application programming interface (API).

For more information, see the *IBM High Performance Computing software supports IBM Power Systems 8335-GTB servers running Red Hat Enterprise Linux (RHEL) 7.3 in little-endian mode* announcement.

For more information about Spectrum LSF in the context of the HPC solution that is described in this book, see 3.4, “Using the IBM Spectrum LSF” on page 112 and Chapter 6, “Cluster monitoring and health checking” on page 289.

For more information about IBM Spectrum LSF, see the IBM Spectrum LSF V10.1 documentation website.

A

ISV Applications

Although the target audience for this IBM Redbooks publication is system administrators and application developers, this appendix covers several topics that are mostly relevant to the application users of the IBM POWER8 high-performance computing solution.

Application software

This section describes examples of software packages from the following application domains:

- Bioinformatics
- Computational fluid dynamics
- Molecular dynamics

This section also provides basic guidance about the compilation and execution of several of these applications.

Bioinformatics

Personal healthcare is a rapidly growing area. Driven by the high demand for low-cost nucleotide sequencing, several new genome sequencing methods were developed recently. These methods are commonly known as next-generation sequencing (NGS) methods. In recent years, these methods were implemented in commercial sequencer apparatus, and sequencing of genetic material is now a routine procedure.

The NGS technology produces vast amounts of raw genome sequence data. As a result, researchers and clinicians need solutions that can solve the problem of large volume NGS data management, processing, and analysis. The underlying computing resources are equipped ideally with multiple fast processors, a large amount of RAM, and an efficient storage system. POWER8 machines that run the Linux operating system are good candidates for the role of NGS data machines.

Trinity

Trinity is a popular tool for the processing and analysis of genomic sequencing data. For more information about compilation options and an evaluation of the performance of POWER8 processor-based systems in NGS analysis, see the *Performance of Trinity RNA-seq de novo assembly* paper that is available at the IBM PartnerWorld® website.

BioBuilds suite

Major genomics applications on POWER8, including Trinity, are available for download as part of the BioBuilds suite. The suite is a collection of open source bioinformatics tools and is distributed at no extra charge. The package is pre-built and optimized for the IBM Linux on Power platform. It also includes supporting libraries for the tools. Table A-1 lists the set of available tools as of the BioBuilds 2015.11 release.

Table A-1 Bioinformatics tools available in BioBuilds 2015.11 release

Bioinformatics tools from the BioBuilds suite (2015.11 release)			
ALLPATHS-LG	ClustalW	iSAAC	SOAP3-DP
BAMtools	Cufflinks	Mothur	SOAPaligner
Barracuda	EBSEQ	NCBI	SOAPbuilder
bedtools	EMBOSS	Oases/Velvet	SOAPdenovo2
Bfast	FASTA	Picard	STAR
Bioconductor	FastQC	PLINK	tabix
BioPython	HMMER	Pysam	TMAP

Bowtie	HTSeq	RSEM	TopHat
Bowtie2	htslib	Samtools	Trinity
BWA	IGV	SHRiMP	variant_tools

Because genome sequencing is a compute-intensive task, the sequencing can gain a large performance benefit by using an accelerator. The Barracuda and SOAP3-DP tools that are listed in Table A-1 on page 362 are examples of open source bioinformatics applications that can offload computations to a GPU.

For more information, see the BioBuilds page of the IBM Global Solutions Directory website.

BALSA

BALSA is another example of an application that uses the computational power of the GPU for the secondary analysis of next generation sequencing data. With two GPUs installed, the tool can analyze two samples in parallel.

For more information about BALSA, see the BALSA page of the SourceForge website.

OpenFOAM

The Open Field Operation and Manipulation (OpenFOAM) Computational Fluid Dynamics (CFD) toolbox is an open source CFD software package that is available at no extra charge. It has a large user base across most areas of engineering and science, from commercial and academic organizations.

OpenFOAM includes an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence, and heat transfer, to solid dynamics and electromagnetic. It includes tools for meshing, notably `snappyHexMesh`, a parallelized mesher for complex computer-aided engineering (CAE) geometries, and for pre- and post-processing. Almost everything (including meshing, and pre- and post-processing) runs in parallel as standard, which enables users to take full advantage of computer hardware at their disposal.

Several versions of OpenFOAM are available. This section provides an example of OpenFOAM 2.4.0, focusing on how to install and run it, and how to get the most out of the POWER8 architecture with this application.

Preparation before installing OpenFOAM

This example uses the GNU compiler and the OpenMPI for MPI parallelization, as shown in Example A-1.

Example A-1 Preparation before installation of OpenFOAM

```
$ export MP_COMPILER=gnu
```

Installing OpenFOAM

This section described how to download and install the OpenFOAM 2.4.0 package. Complete the following steps:

1. Download and decompress the source codes of OpenFOAM and the third-party toolkit, as shown in Example A-2.

Example A-2 Preparing the required sources of OpenFOAM and third-party pack

```
$ mkdir -p $HOME/OpenFOAM
$ cd $HOME/OpenFOAM
$ wget http://jaist.dl.sourceforge.net/project/foam/foam/2.4.0/OpenFOAM-2.4.0.tgz
$ wget
http://jaist.dl.sourceforge.net/project/foam/foam/2.4.0/ThirdParty-2.4.0.tgz
$ tar zxvf OpenFOAM-2.4.0.tgz
$ tar zxvf ThirdParty-2.4.0.tgz
```

2. For OpenFOAM 2.4.0, obtain a patch file from the OpenFOAM Issue Tracking website.

   From this site, download the `enable_ppc64el_patch.patch` file to the `$HOME/OpenFOAM` directory and apply it, as shown in Example A-3.

Example A-3 Applying the required patch to OpenFOAM

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0
$ patch -p1 < ../enable_ppc64el_patch.patch
```

3. Set the environment variables that are required for OpenFOAM and start the shell script `Allwmake`, as shown in Example A-4.

Example A-4 Starting OpenFOAM building

```
$ export FOAM_INST_DIR=$HOME/OpenFOAM
$ source $FOAM_INST_DIR/OpenFOAM-2.4.0/etc/bashrc
$ cd $FOAM_INST_DIR/OpenFOAM-2.4.0
$ ./Allwmake
```

Running OpenFOAM

To run OpenFOAM, a series of input data is needed that is a combination of data sets, such as boundary conditions, initial conditions, various physical parameters, and the selections from many solvers and methods prepared in OpenFOAM. This input data defines the physical simulation that the user wants to solve.

OpenFOAM includes examples of these data sets that are called *tutorials*. This example shows a tutorial that is named “motorbike”. The motorbike tutorial simulates a typical CFD that calculates the steady flow around a motorcycle and rider and is one of the major tutorials for the performance benchmark.

Figure A-1 shows the images of the motorcycle and rider. These images are rendered by using the ParaView tool. The ParaView tool binary for Windows 64-bit, Windows 32-bit, Linux 64-bit, and Mac OS X is available from the ParaView website. (The source code of the ParaView tool is also included in this toolkit.)

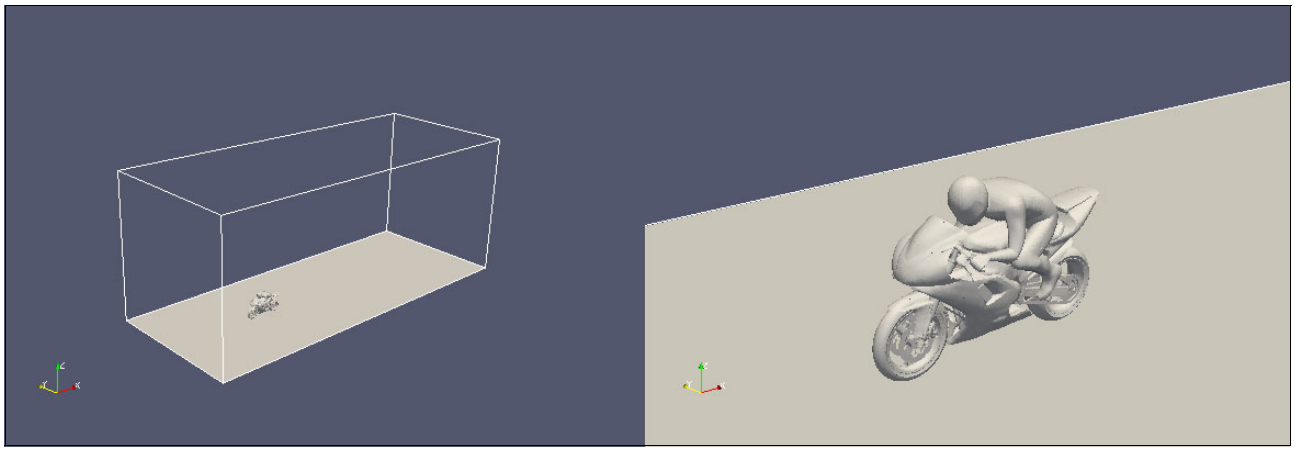


Figure A-1 Motorbike input for OpenFOAM simulation

There are nine programs that can be used for completing the motorbike simulation, as shown in Example A-5. These programs are implemented in the `Allrun` script file that is in the following directory:

`$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/`

Example A-5 Nine OpenFOAM programs executed in the motorbike simulation

```
surfaceFeatureExtract
blockMesh
decomposePar
snappyHexMesh
patchSummary
potentialFoam
simpleFoam
reconstructParMesh
reconstructPar
```

These nine programs are executed one by one in the `Allrun` script. Among these nine programs, four programs (`snappyHexMesh`, `patchSummary`, `potentialFoam`, and `simpleFoam`) are executed with message passing interface (MPI) parallelization. By default, they are executed with six MPI processes. The other five programs (`surfaceFeatureExtract`, `blockMesh`, `decomposePar`, `reconstructParMesh`, and `reconstructPar`) are not MPI implemented and executed serially by using one CPU core.

Among these programs, `simpleFoam` is the main solver for this motorbike simulation, which solves the velocity and pressure by iterative calculation by using the Semi-Implicit Method for Pressure-Linked Equation (SIMPLE) method. In general, `simpleFoam` takes much time to complete and its elapsed time accounts for most of the elapsed time of all nine programs.

Simulating a large-size problem with many MPI processes

The POWER8 architecture has high memory bandwidth. Even if you increase the number of grids that are used in the problem and the number of MPI processes, you can run your jobs comfortably without seeing the performance degradation that is caused by memory performance bottleneck. Therefore, you can increase the problem size and the number of MPI processes to use of the POWER8 architecture.

The following section shows how to increase the problem size (the number of grids) and the number of MPI processes by using the motorbike case as an example.

Increasing the problem size

This section describes how to increase the problem size and the number of processes by referring to the previous motorbike case (see Figure A-1 on page 365).

To change the problem size of the simulation, modify the parameter settings in the `blockMeshDict` file. This file is in the following directory:

```
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/constant/polyMesh/
```

The default is 1280 grids (20 x 8 x 8 grids), as shown in Figure A-2.

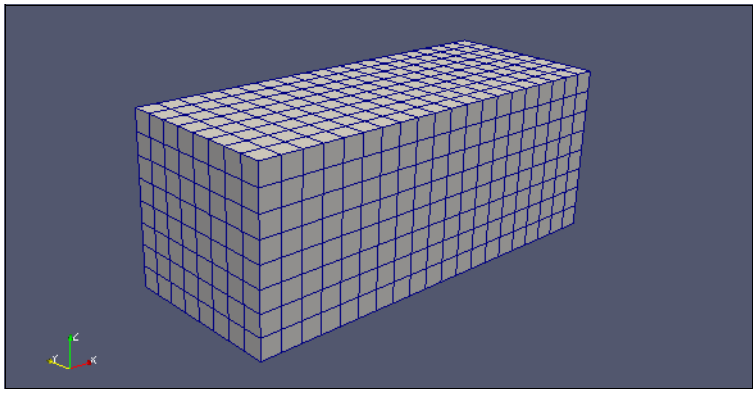


Figure A-2 20 x 8 x 8 grids (default)

For example, if you want to change the default grid into a 40 × 16 × 16 grids (10240 grids), as shown in Figure A-3, you must modify `blockMeshDict`, as shown in Example A-6.

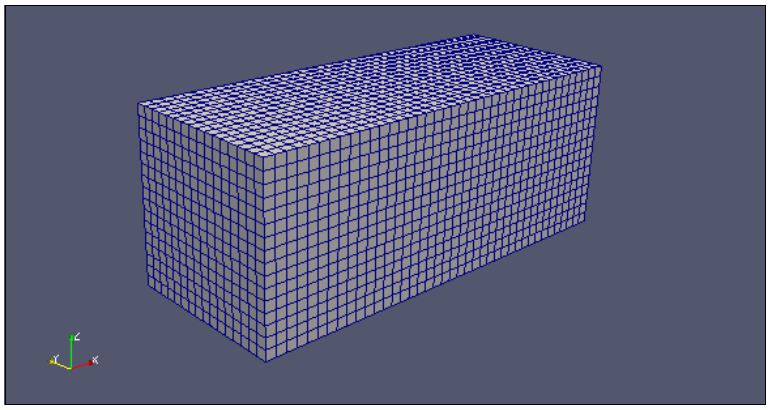


Figure A-3 40 x 16 x 16 grids

Example A-6 shows modifying the `blockMeshDict` file.

Example A-6 Modification of blockMeshDict file

```
$ diff blockMeshDict.org blockMeshDict
34c34
<     hex (0 1 2 3 4 5 6 7) (20 8 8) simpleGrading (1 1 1)
---
>     hex (0 1 2 3 4 5 6 7) (40 16 16) simpleGrading (1 1 1)
```

For more information about `blockMeshDict`, see *4.3 Mesh generation with the blockMesh utility*, which is available at the OpenFOAM website.

Increasing the number of MPI processes

The method of parallel computing that is used by OpenFOAM is known as *domain decomposition*. In this method, the geometry and associated fields are broken into pieces and allocated to each CPU core for computation. The flow of parallel computation involves the decomposition of mesh and fields, running the application in parallel, and post-processing the decomposed case, as described in the following sections. The parallel running uses OpenMPI for the standard MPI.

To change the number of MPI processes, the modifications that are described next must be made.

Selecting the method of the domain decomposition

You can select the scotch method rather than default hierarchical method for the domain decomposition. The scotch decomposition requires no geometric input from the user and attempts to minimize the number of processor boundaries.

This section shows the difference between hierarchical and scotch by using simple figures of 1280 grids (20 x 8 x 8 grids).

Figure A-4 shows the hierarchical method. The domain is decomposed as (X-direction, Y-direction, Z-direction) = (3, 2, 1), based on the parameter settings that are stated in the `decomposeParDict` file.

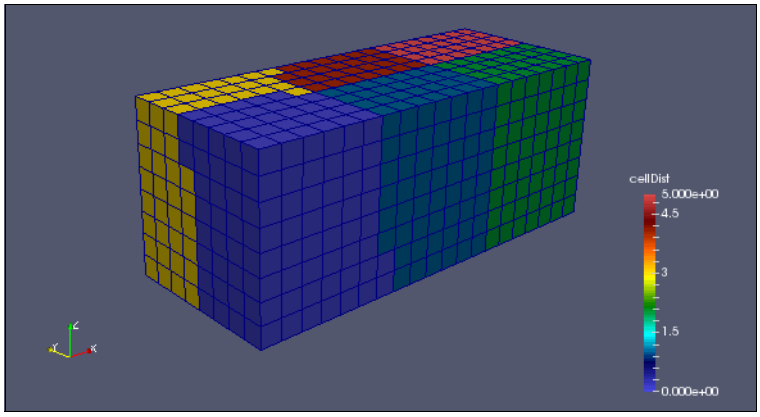


Figure A-4 Domain decomposition by hierarchical method

Figure A-5 shows the scotch method. In this method, the scotch library automatically decides the optimal domain decomposition, so you do not need to set the number of decomposition for each direction of X, Y, and Z.

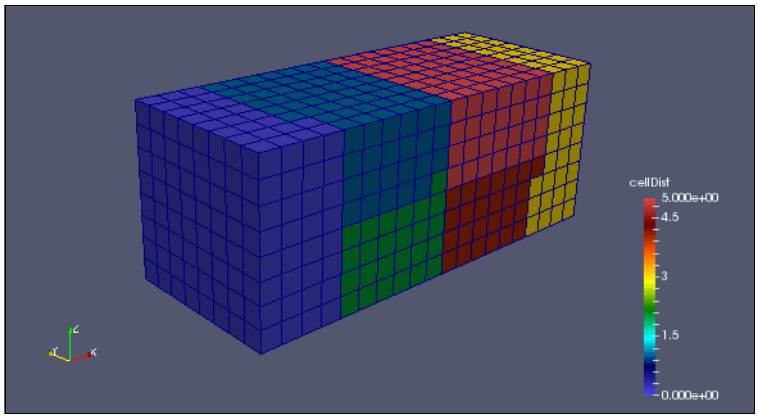


Figure A-5 Domain decomposition by scotch method

To select the scotch method instead of the hierarchical method, modify the parameter settings of the `decomposeParDict` file, as shown in Example A-7.

Example A-7 Changing the parameter settings in the decomposeParDict file

```
$ cd
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/system
/
$ diff decomposeParDict.org decomposeParDict
20c20
< method          hierarchical;
---
> method          scotch;
```

Changing the number of subdomains

To change the number of subdomains from 6 to 40 (as shown in Figure A-6), you must modify the parameter settings of the `decomposeParDict` file, as shown in Example A-8.

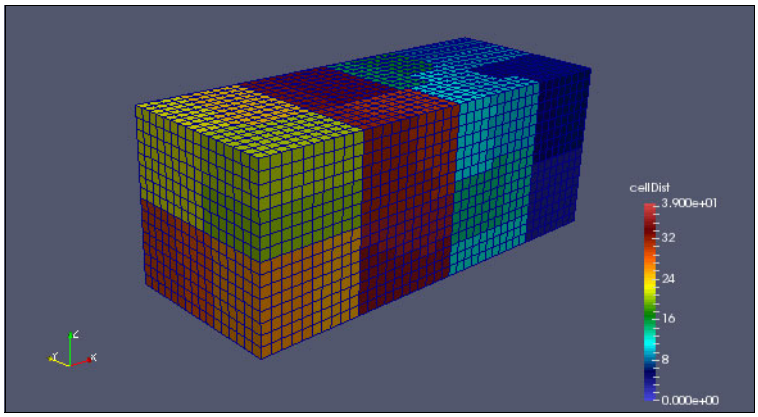


Figure A-6 Forty subdomains for forty MPI processes

Example A-8 shows how to modify the parameters of the `ecomposeParDict` file.

Example A-8 Changing the parameter settings in decomposeParDict file

```
$ cd
$HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/system
/
$ diff decomposeParDict.org decomposeParDict
18c18
< numberOfSubdomains 6;
---
> numberOfSubdomains 40;
```

Changing the number of MPI processes

Change the number of MPI processes in the `Allrun` file, as shown in Example A-9. The number of MPI processes often must be the same as the number of subdomains as described in “Changing the number of subdomains” on page 369.

Example A-9 Changing the parameter settings of Allrun

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/
$ diff Allrun.org Allrun
14c14
< runParallel snappyHexMesh 6 -overwrite
---
> runParallel snappyHexMesh 40 -overwrite
23,25c23,25
< runParallel patchSummary 6
< runParallel potentialFoam 6
< runParallel $(getApplication) 6
---
> runParallel patchSummary 40
> runParallel potentialFoam 40
> runParallel $(getApplication) 40
```

Note: To check what the domain decomposition looks like, use the following command:

```
[home/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike]$
diff Allrun.org Allrun
13c13
< runApplication decomposePar
---
> runApplication decomposePar -cellDist
```

Running the motorbike simulation

After completing these preparations, run the motorbike simulation by running the `Allrun` script, as shown in Example A-10. After the `Allrun` script starts, the binary files that are listed in Example A-5 on page 365 are automatically run individually.

Example A-10 Kick Allrun script

```
$ cd $HOME/OpenFOAM/OpenFOAM-2.4.0/tutorials/incompressible/simpleFoam/motorBike/
$ time ./Allrun
```

After the series of binary files is run, a directory named `500` is created. This new directory includes some simulated values, such as `U` (Magnitude), and `p` (Pressure), and so on. These values show the physical state after 500 iterative calculations.

Figure A-7 shows the images of these simulated values of `U` (Magnitude) and `p` (Pressure) around the motorbike visualized by the ParaView tool.

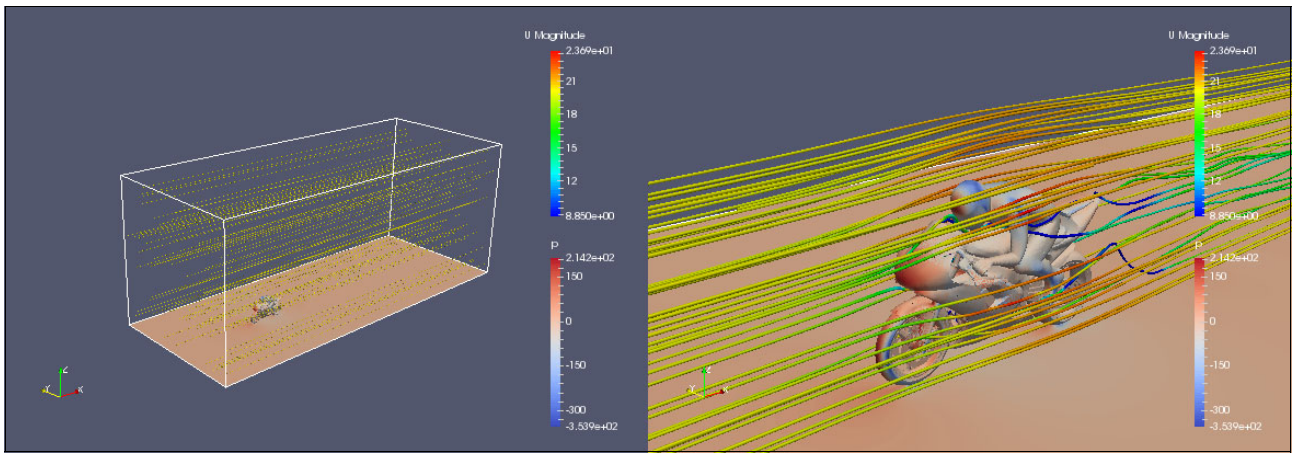


Figure A-7 Result of motorbike simulation

Tuning techniques

This section describes the following tuning techniques:

- CPU binding
- SMT settings
- Linking tcmalloc

CPU binding

To get higher and more stable performance, each MPI process must be bound to each CPU core. In OpenMPI, the use of the **`mpirun`** command automatically binds processes at the start of the v1.8 series.

For more information, see the mpirun(1) man page (version 1.8.8) page of the Open MPI website.

SMT settings

CAE applications, such as OpenFOAM, often have a difficult time making use of hardware threads. However, in some cases, the use of SMT on POWER8 and assigning some MPI processes into a single CPU core can improve the total throughput. Therefore, trying some combination of SMT modes and the number of MPI processes assigned to a single core is valuable for benchmarking and performance tuning.

Linking tcmalloc

In some cases, linking a `thread-caching malloc (tcmalloc)` library into the OpenFOAM binary files improves their performance. For the nine example OpenFOAM binary files that are listed in Example A-5 on page 365, the performance of `surfaceFeatureExtract`, `blockMesh`, `decomposePar`, `snappyHexMesh`, `patchSummary`, `reconstructParMesh`, and `reconstructPar` were improved. However, the performance of `potentialFoam` and `simpleFoam` were slightly degraded by linking to a `tcmalloc` library.

Therefore, link the `tcmalloc` library to only the binary files that are expected to be improved by using the shell-script, as shown in Example A-11.

Example A-11 Apply tcmalloc

```
$ cd $HOME/OpenFOAM
$ cat @apply_tcmalloc
#!/bin/sh
for i in surfaceFeatureExtract blockMesh decomposePar snappyHexMesh patchSummary
reconstructParMesh reconstructPar
do
  rm ./OpenFOAM-2.4.0/platforms/linuxPPC64leGccDPOpt/bin/$i
  cd `find ./OpenFOAM-2.4.0/applications/ -name $i -type d`
  wmake |tr '\\\n' ' ' > out.wmake
  echo `cat out.wmake` -L/opt/at9.0/lib64/ -ltcmalloc |sh -x
  rm out.wmake
  cd -
done
$ ./@apply_tcmalloc
```

For more information about tcmalloc, see the TCMalloc : Thread-Caching Malloc website.

NAMD program

Nanoscale Molecular Dynamics (NAMD) is a freeware molecular dynamics simulation package that is written by using the Charm++ parallel programming model.

This section introduces how to install and run NAMD 2.11 in an IBM POWER8 server by using NVIDIA graphics programming units (GPUs) with CUDA 7.5.

Installing NAMD

Download the source code `NAMD_2.11_Source.tar.gz` from the Theoretical and Computational Biophysics Group's Software Downloads page of the University of Illinois at Urbana-Champaign website.

Before downloading the source code, complete the registration process, which requires your name and email address, and that you answer some questions and accept the license agreement.

Download the `NAMD_2.11_Source.tar.gz` file to your preferred directory; for example, `$HOME/NAMD`. Then, build the prerequisite package, as shown in Example A-12.

Example A-12 Building prerequisite package for NAMD

```
$ cd $HOME/NAMD
$ tar zxvf NAMD_2.11_Source.tar.gz
$ cd NAMD_2.11_Source
$ tar xvf charm-6.7.0.tar
$ cp -rp charm-6.7.0/src/arch/mpi-linux-ppc charm-6.7.0/src/arch/mpi-linux-ppc64le
$ cd charm-6.7.0
$ ./build charm++ mpi-linux-ppc64le -O -DCMK_OPTIMIZE=1
$ cd ../arch
$ ls -l | grep Linux-POWER
-rw-r----- 1 kame IBM1  190  6? 18  2014 Linux-POWER-g++.arch
-rw-r----- 1 kame IBM1  495  2?  7 2011 Linux-POWER-xlC.arch
-rw-r----- 1 kame IBM1    0  2?  7 2011 Linux-POWER.base
-rw-r----- 1 kame IBM1  467 12?  2 12:52 Linux-POWER.cuda
-rw-r----- 1 kame IBM1  138  2?  7 2011 Linux-POWER.fftw
-rw-r----- 1 kame IBM1  191  6? 18  2014 Linux-POWER.tcl
```

Before starting to build NAMD, prepare the FFTW package, as shown in Example A-13.

Example A-13 Preparing FFTW package

```
[/work/NAMD]$ wget ftp://ftp.fftw.org/pub/fftw/fftw-3.3.4.tar.gz
[/work/NAMD]$ tar zxvf fftw-3.3.4.tar.gz
[/work/NAMD]$ cd fftw-3.3.4
[/work/NAMD/fftw-3.3.4]$ ./configure --enable-float --prefix=`pwd`
[/work/NAMD/fftw-3.3.4]$ make
[/work/NAMD/fftw-3.3.4]$ make install
[/work/NAMD/fftw-3.3.4]$ ls lib
libfftw3f.a  libfftw3f.la pkgconfig
```

Prepare the TCL package, as shown in Example A-14.

Example A-14 Preparing TCL package

```
[/work/NAMD]$ wget
http://www.ks.uiuc.edu/Research/namd/libraries/tcl8.5.9-linux-ppc64le-threaded.tar
.gz
[/work/NAMD]$ tar zxvf tcl8.5.9-linux-ppc64le-threaded.tar.gz
[/work/NAMD]$ ls tcl8.5.9-linux-ppc64le-threaded/lib
libtcl8.5.a  libtclstub8.5.a  tcl8 tcl8.5  tclConfig.sh
```

Prepare some files for the parameter settings, as shown in Example A-15.

Example A-15 Preparing files for parameter settings

```
[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER-xlC_MPI.arch
NAMD_ARCH = Linux-POWER
CHARMARCH = mpi-linux-ppc64le
CXX = mpCC -w
CXXOPTS = -O3 -q64 -qnohot -qstrict -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8
-qtune=pwr8
CXXNOALIASOPTS = -O4 -q64 -qaggrcopy=nooverlap -qalias=noallptrs -qarch=pwr8
-qtune=pwr8
CXXTHREADOPTS = -O3 -q64 -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8 -qtune=pwr8
CC = xlc -w
COPTS = -O4 -q64 -qarch=pwr8 -qtune=pwr8

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.cuda
CUDADIR=/usr/local/cuda-7.5
CUDAINCL=-I$(CUDADIR)/include
CUDALIB=-L$(CUDADIR)/lib64 -lcudart_static -lrt -ldl
CUDASODIR=$(CUDADIR)/lib64
LIBCUDARTSO=
CUDAFLAGS=-DNAMD_CUDA
CUDAOBJS=$(CUDAOBJSRAW)
CUDA=$(CUDAFLAGS) -I. $(CUDAINCL)
CUDACC=$(CUDADIR)/bin/nvcc -O3 --maxrregcount 32 $(CUDAGENCODE) $(CUDA)
CUDAGENCODE=-gencode arch=compute_20,code=sm_20 -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode
arch=compute_37,code=sm_37 -gencode arch=compute_50,code=sm_50 -gencode
arch=compute_52,code=sm_52 -gencode arch=compute_52,code=compute_52

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.fftw3
FFTDIR=/work/NAMD/fftw-3.3.4/
FFTINCL=-I$(FFTDIR)/include
FFTLIB=-L$(FFTDIR)/lib -lfftw3f
FFTFLAGS=-DNAMD_FFTW -DNAMD_FFTW_3
FFT=$(FFTINCL) $(FFTFLAGS)

[/work/NAMD/NAMD_2.11_Source/arch]$ cat Linux-POWER.tcl
TCLDIR=/work/NAMD/tcl8.5.9-linux-ppc64le-threaded/
TCLINCL=-I$(TCLDIR)/include
# TCLLIB=-L$(TCLDIR)/lib -ltcl8.5 -ldl
TCLLIB=-L$(TCLDIR)/lib -ltcl8.5 -ldl -lpthread
TCLFLAGS=-DNAMD_TCL
TCL=$(TCLINCL) $(TCLFLAGS)
```

Modify the `config` file, as shown in Example A-16, to avoid errors during configuration.

Example A-16 Modifying the config file

```
[/work/NAMD/NAMD_2.11_Source]$ diff config.org config
383,390c383,390
<     if ( $charm_arch_mpi || ! $charm_arch_smp ) then
<       echo ''
<       echo "ERROR: $ERRTYPE builds require non-MPI SMP or multicore Charm++ arch
for reasonable performance."
<       echo ''
<       echo "Consider ibverbs-smp or verbs-smp (InfiniBand), gni-smp (Cray), or
multicore (single node)."
<       echo ''
<       exit 1
<     endif
---
> #    if ( $charm_arch_mpi || ! $charm_arch_smp ) then
> #      echo ''
> #      echo "ERROR: $ERRTYPE builds require non-MPI SMP or multicore Charm++
arch for reasonable performance."
> #      echo ''
> #      echo "Consider ibverbs-smp or verbs-smp (InfiniBand), gni-smp (Cray), or
multicore (single node)."
> #      echo ''
> #      exit 1
> #    endif
```

Configure NAMD, as shown in Example A-17.

Example A-17 Configuring NAMD

```
[/work/NAMD/NAMD_2.11_Source]$ ./config Linux-POWER-xlC_MPI --with-fftw3
--with-tcl --with-cuda

Selected arch file arch/Linux-POWER-xlC_MPI.arch contains:

NAMD_ARCH = Linux-POWER
CHARMARCH = mpi-linux-ppc64le
CXX = mpCC_r -w
CXXOPTS = -O3 -q64 -qnohot -qstrict -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8
-qtune=pwr8
CXXNOALIASOPTS = -O4 -q64 -qaggrcopy=nooverlap -qalias=noallptrs -qarch=pwr8
-qtune=pwr8
CXXTHREADOPTS = -O3 -q64 -qaggrcopy=nooverlap -qalias=ansi -qarch=pwr8 -qtune=pwr8
CC = xlc_r -w
COPTS = -O4 -q64 -qarch=pwr8 -qtune=pwr8
Creating directory: Linux-POWER-xlC_MPI
Creating link: .. to .rootdir
Writing build options to Linux-POWER-xlC_MPI/Make.config
Using Charm++ 6.7.0 build found in main build directory
Linking Makefile
Linking Make.depends
Linking src directory
Linking plugins directory
Linking psfgen directory
```

```
Generated Linux-POWER-xlC_MPI/Make.config contains the following:

CHARMBASE = .rootdir/charm-6.7.0
include .rootdir/arch/Linux-POWER-xlC_MPI.arch
CHARM = $(CHARMBASE)/$(CHARMARCH)
NAMD_PLATFORM = $(NAMD_ARCH)-MPI-CUDA
include .rootdir/arch/$(NAMD_ARCH).base
include .rootdir/arch/$(NAMD_ARCH).tcl
include .rootdir/arch/$(NAMD_ARCH).fftw3
include .rootdir/arch/$(NAMD_ARCH).cuda

You are ready to run make in directory Linux-POWER-xlC_MPI now.
```

Then, run the **make** command to build NAMD, as shown in Example A-18.

Example A-18 Building NAMD

```
[/work/NAMD/NAMD_2.11_Source]$ cd Linux-POWER-xlC_MPI
[/work/NAMD/NAMD_2.11_Source/Linux-POWER-xlC_MPI]$ make

******

xlc -w -Isrc
-I/vol/xcae1/b5p218za/work/NAMD/tcl8.5.9-linux-ppc64le-threaded//include
-DNAMD_TCL -O4 -q64 -qarch=pwr8 -qtune=pwr8 -DNAMD_VERSION=\"2.11\"
-DNAMD_PLATFORM=\"Linux-POWER-MPI-CUDA\"  -DREMOVE_PROXYRESULTMSG_EXTRACOPY
-DNODEAWARE_PROXY_SPANNINGTREE -DUSE_NODEPATCHMGR  -o flipbinpdb src/flipbinpdb.c
|| \
echo "#!/bin/sh\necho unavailable on this platform" > flipbinpdb; \
chmod +x flipbinpdb
cp .rootdir/charm-6.7.0/mpi-linux-ppc64le/bin/charmrun charmrun
```

If you succeed in running the **make** command, you can find the execution binary file that is named `namd2` in the `Linux-POWER-xlC_MPI` directory.

Running NAMD

Sample simulations of NAMD are available from the Theoretical and Computational Biophysics Group's NAMD Utilities page of the University of Illinois at Urbana-Champaign website.

For this example, select **ApoA1**, which has been the standard NAMD cross-platform benchmark for years, as shown in Figure A-8. The official name of this gene is *apolipoprotein A-I*.

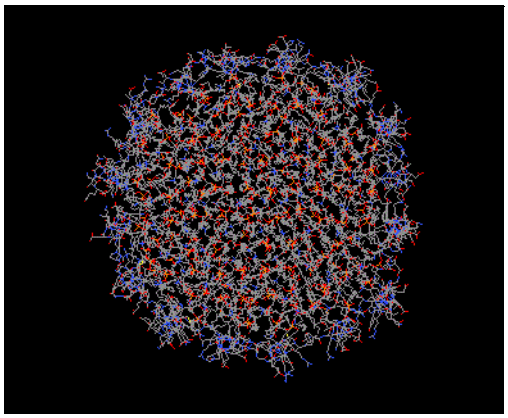


Figure A-8 Visualization of ApoA1 protein

The APOA1 gene provides instructions for making a protein that is called *apolipoprotein A-I* (apoA-I). ApoA-I is a component of high-density lipoprotein (HDL). HDL is a molecule that transports cholesterol and certain fats called phospholipids through the bloodstream from the body's tissues to the liver. After the molecules are in the liver, cholesterol and phospholipids are redistributed to other tissues or removed from the body. Figure A-8 shows a visualization of this protein.

For more information about ApoA1, see the APOA1 gene page of the US National Library of Medicine's website.

After preparing the data directory of ApoA1 that is named `apoa1` on the same directory as `namd2`, run this sample model by using 40 MPI processes, as shown in Example A-19.

Example A-19 Running NAMD with each example

```
$ MP_RESD=poe MP_HOSTFILE=./hf MP_PROCS=40 MP_SHARED_MEMORY=yes
MP_EAGER_LIMIT=65536 MEMORY_AFFINITY=MCM MP_INFOLEVEL=4 MP_BINDPROC=yes
MP_PE_AFFINITY=yes MP_BIND_MODE=spread MP_TASK_AFFINITY=cpu time poe ./namd2
./apoa1/apoa1.namd
```

For more information about the environment variables for the **poe** command that is shown in Example A-19, see the poe page of the IBM Knowledge Center website.

Note: NAMD scales best by using simultaneous multithreading (SMT) of no more than two threads per core. For example, if you have 20 physical cores on a Power System S822LC, running NAMD with 40 threads can result in the best performance.

B

Additional material

This book refers to additional material that can be downloaded from the Internet, as described in the following sections.

Locating the Web material

The Web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG248371`

Alternatively, you can go to the IBM Redbooks website at:

`ibm.com/redbooks`

Select **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248371.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
smn_md.cfg.zip	kickstarter file

System requirements for downloading the Web material

The Web material requires the following system configuration:

Hard disk space: 100 MB minimum
Operating System: Windows, Linux or macOS
Processor: i3 or higher
Memory: 1024 MB or higher

Downloading and extracting the Web material

Create a subdirectory (folder) on your workstation, and extract the contents of the Web material `.zip` file into this folder.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Note that some publications that are referenced in this list might be available in softcopy only:

- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263
- *Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8*, SG24-8171
- *Implementing an IBM High-Performance Computing Solution on IBM POWER8*, SG24-8263

You can search for, view, download, or order these documents and other Redbooks, Redpapers, Web Docs, draft, and other materials at the following website:

ibm.com/redbooks

Other publications

These publication *Optimization and Programming Guide - XL C/C++ for Linux, V13.1.5, for little endian distributions,* SC27-6560, also is relevant as a further information source.

Online resources

The following websites are also relevant as further information sources:

- IBM Systems Hardware and Software for high performance computing:

  http://www.ibm.com/systems/power/hardware/hpc.html
- xCAT (Extreme Cloud/Cluster Administration Toolkit):

  http://xcat.org
- IBM Spectrum LSF:

  http://www.ibm.com/systems/spectrum-computing/products/lsf/
- NVIDA Tesla P100:

  https://devblogs.nvidia.com/parallelforall/inside-pascal/
- POWERAccel:
  - https://ibm.biz/BdiSn2
  - https://www.ibm.com/blogs/systems/tag/poweraccel/

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

IBM POWER8 High-Performance Computing Guide: IBM Power System S822LC (8335-GTB) Edition

SG24-8371-00

ISBN 0738442550

IBM

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

SG24-8371-00

ISBN 0738442550

Printed in U.S.A.