Parallel Environment Runtime Edition
Version 1 Release 2

PAMI Programming Guide

SA23-2273-03

Parallel Environment Runtime Edition
Version 1 Release 2

PAMI Programming Guide

IBM

SA23-2273-03

Note

Before using this information and the product it supports, read the information in "Notices" on page 151.

This edition applies to:

- version 1, release 1, modification 0 of IBM Parallel Environment Runtime Edition for AIX (product number 5765-PER)
- version 1, release 2, modification 0 of IBM Parallel Environment Runtime Edition for Linux on Power (product number 5765-PRP)
- version 1, release 2, modification 0 of IBM Parallel Environment Runtime Edition for Linux on X-Architecture (product number 5725-G00)

and to all subsequent releases and modifications, until otherwise indicated in new editions.

Contents

About this information

Attention

The functions or features found herein may not be available on all operating systems or platforms and do not indicate the availability of these functions or features within the IBM product or future versions of the IBM product. The development, release, and timing of any future features or functionality is at IBM's sole discretion. IBM's plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. The information mentioned is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The information may not be incorporated into any contract and it should not be relied on in making a purchasing decision.

This publication contains reference information about the parallel active messaging interface (PAMI), which is a component of the Parallel Environment Runtime Edition (PE) licensed program.

Information for AIX users

This information supports IBM® Parallel Environment Runtime Edition for AIX® (5765-PER), Version 1 Release 1.

To make this information easier to read, the name *IBM Parallel Environment Runtime Edition* has been abbreviated to *IBM PE Runtime Edition*, *PE for AIX*, *Parallel Environment*, or more generally, *PE* throughout.

To use this information, you should be familiar with the AIX operating system. Where necessary, background information related to AIX is provided but, more commonly, it refers you to the appropriate documentation.

The Parallel Environment Runtime Edition for AIX information assumes that AIX Version 7.1 (or later) is installed, in one of two ways:

- Standalone
- Connected by way of an Ethernet LAN supporting IP

For information on installing the AIX operating system, see the *AIX Installation Guide and Reference*.

Information for Linux users

This information supports:

| - IBM Parallel Environment Runtime Edition for Linux on Power Version 1,
| Release 2 (5765-PRP)
- IBM Parallel Environment Runtime Edition for Linux on X-Architecture Version 1, Release 2 (5725-G00).

To make this information easier to read, the name *IBM Parallel Environment Runtime Edition* has been abbreviated to, *PE for Linux*, or more generally, *PE* throughout.

To use this information, you should be familiar with the Linux operating system. Where necessary, background information related to Linux is provided but, more commonly, it refers you to the appropriate documentation.

The Parallel Environment for Linux information assumes that one of the following Linux distributions is already installed:

- Red Hat Enterprise Linux 6, Update 2 on IBM Power Systems™ servers, IBM System x servers, and supported non-IBM x86-based servers
- SUSE LINUX Enterprise Server (SLES) 11 SP1 on IBM System x servers and supported non-IBM x86-based servers.

PE for Linux is based on its predecessor, PE for AIX, with which you might be familiar.

Who should use this information

This information is intended for programmers who write and run PAMI programs on an AIX or Linux operating system. The programmer should be experienced with UNIX-like environments, networked systems, and the C, C++, or Fortran programming language.

Conventions and terminology used in this information

Table 1 shows the conventions used in this information:

Table 1. Conventions

Convention	Usage
bold	**Bold** words or characters represent system elements that you must use literally, such as commands, flags, path names, directories, file names, values, PE component names (**poe**, for example), and selected menu options.
bold underlined	**bold underlined** keywords are defaults. These take effect if you do not specify a different keyword.
`constant width`	Examples and information that the system displays appear in `constant-width` typeface.
italic	*Italic* words or characters represent variable values that you must supply. Italics are also used for unit titles, the first use of a glossary term, and general emphasis in text.
<*key*>	Angle brackets (less-than and greater-than) enclose the name of a key on the keyboard. For example, <**Enter**> refers to the key on your terminal or workstation that is labeled with the word *Enter*.

\	In command examples, a backslash indicates that the command or coding example continues on the next line. For example:
	`mkcondition -r IBM.FileSystem -e "PercentTotUsed > 90" \`
	`-E "PercentTotUsed < 85" -m d "FileSystem space used"`

{*item*}	Braces enclose a list from which you must choose an item in format and syntax descriptions.
[*item*]	Brackets enclose optional items in format and syntax descriptions.
<**Ctrl-*x***>	The notation <**Ctrl-*x***> indicates a control character sequence. For example, <**Ctrl-c**> means that you hold down the control key while pressing <**c**>.

Table 1. Conventions (continued)

Convention	Usage
item...	Ellipses indicate that you can repeat the preceding item one or more times.
|	• In *synopsis* statements, vertical lines separate a list of choices. In other words, a vertical line means *Or*. • In the margin of the document, vertical lines indicate technical changes to the information.

In addition to the highlighting conventions, this information uses the following conventions when describing how to perform tasks.

User actions appear in uppercase boldface type. For example, if the action is to enter the **tool** command, this information presents the instruction as:

ENTER
tool

Abbreviated names

Some of the abbreviated names used in this information follow.

AIX	Advanced Interactive Executive
CSS	communication subsystem
GUI	graphical user interface
HFI	Host Fabric Interface
IP	Internet Protocol
LAPI	Low-level Application Programming Interface
MDCR	MetaCluster Distributed Checkpoint Restart
MPI	Message Passing Interface
MPICH2	Implementation of the Message Passing Interface created by Argonne National Laboratory.
PAMI	Parallel Active Messaging Interface
PDB	Parallel Debugger
PE	IBM Parallel Environment Runtime Edition
PE MPI	IBM's implementation of the MPI standard for PE
PE MPI-IO	IBM's implementation of MPI I/O for PE
PNSD	Protocol Network Services Daemon
POE	Parallel Operating Environment
PTF	Program Temporary Fix
RSCT	Reliable Scalable Cluster Technology
rsh	remote shell

SCI Scalable Communication Infrastructure

STDERR
standard error

STDIN
standard input

STDOUT
standard output

System x™
IBM System x

Prerequisite and related information

The Parallel Environment Runtime Edition for AIX and Linux library consists of:

- IBM Parallel Environment Runtime Edition: Installation, SC23-6780
- IBM Parallel Environment Runtime Edition: LAPI Programming Guide, SA23-2272
- IBM Parallel Environment Runtime Edition: Messages, SC23-6782
- IBM Parallel Environment Runtime Edition: MPI Programming Guide, SC23-6783
- IBM Parallel Environment Runtime Edition: MPI Subroutine Reference, SC23-6784
- IBM Parallel Environment Runtime Edition: NRT API Programming Guide, SC23-6785
- IBM Parallel Environment Runtime Edition: Operation and Use, SC23-6781
- IBM Parallel Environment Runtime Edition: PAMI Programming Guide, SA23-2273

To access the most recent Parallel Environment Runtime Edition documentation in PDF and HTML format, refer to the IBM Clusters Information Center (**http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp**), on the Web.

Both the current Parallel Environment Runtime Edition books and earlier versions of the library are also available in PDF format from the IBM Publication Center (**http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss**), on the Web.

It is easiest to locate a book in the IBM Publications Center by supplying the book's publication number. The publication number for each of the Parallel Environment books is listed after the book title in the preceding list.

You may also have the related product, IBM Parallel Environment Developer Edition. The PE Developer Edition contains the IBM High Performance Toolkit (HPC Toolkit), which is a collection of tools that allow you to analyze the performance of both parallel and serial applications, written in C or FORTRAN, over the AIX or Linux operating system.

After installation, documentation for the PE Developer Edition is located in the **/opt/ibmhpc/ppedev.hpct/doc** directory. You can access the PE Developer Edition documentation, as well as information about IBM's other High Performance Computing products, from the HPC Central Web site (**http://www.ibm.com/developerworks/wikis/display/hpccentral/HPC+Central**).

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have comments about this information or other PE documentation:

- Send your comments by e-mail to: mhvrcfs@us.ibm.com

  Be sure to include the name of the book, the part number of the book, the version of Parallel Environment Runtime Edition, and, if applicable, the specific location of the text you are commenting on (for example, a page number or table number).
- Fill out one of the forms at the back of this book and return it by mail, by fax, or by giving it to an IBM representative.

National language support (NLS)

For national language support (NLS), all PE components and tools display messages that are located in externalized message catalogs. English versions of the message catalogs are shipped with the PE licensed program, but your site may be using its own translated message catalogs. The PE components use the environment variable **NLSPATH** to find the appropriate message catalog. **NLSPATH** specifies a list of directories to search for message catalogs. The directories are searched, in the order listed, to locate the message catalog. In resolving the path to the message catalog, **NLSPATH** is affected by the values of the environment variables **LC_MESSAGES** and **LANG**. If you get an error saying that a message catalog is not found and you want the default message catalog, do the following.

If you are using PE for AIX:

ENTER

```
export NLSPATH=/usr/lib/nls/msg/%L/%N

export LANG=C
```

If you are using PE for Linux:

ENTER

```
export NLSPATH=/usr/share/locale/%L/%N

export LANG=en_US
```

The PE message catalogs are in English, and are located in the following directories.

If you are using PE for AIX:

/usr/lib/nls/msg/C

/usr/lib/nls/msg/En_US

/usr/lib/nls/msg/en_US

If you are using PE for Linux:

/usr/share/locale/C

/usr/share/locale/En_US

/usr/share/locale/en_US

/usr/share/locale/en_US.UTF-8

If your site is using its own translations of the message catalogs, consult your system administrator for the appropriate value of **NLSPATH** or **LANG**.

PE for AIX users can refer to *AIX: General Programming Concepts: Writing and Debugging Programs* for more information on NLS and message catalogs.

Functional restrictions for IBM PE Runtime Edition

For this release, there are functional restrictions that apply to IBM PE Runtime Edition for AIX and IBM PE Runtime Edition for Linux.

Functional restrictions for IBM PE Runtime Edition for AIX 1.1

The functional restrictions for IBM PE Runtime Edition for AIX 1.1 are:

- The InfiniBand interconnect is not supported.

Functional restrictions for IBM PE Runtime Edition for Linux 1.2

Although many of the following functions, are currently available with IBM PE Runtime Edition for AIX, they are not supported by IBM PE Runtime Edition for Linux 1.2. The functional restrictions for IBM PE Runtime Edition for Linux are:

- User Space jobs with Red Hat Enterprise Linux, when running on IBM Power Systems servers.
- Barrier synchronization register, when running on IBM POWER7 servers.

Summary of changes

Changes for PE

IBM Parallel Environment Runtime Edition contains a number of functional enhancements, including:

- The IBM PE Runtime Edition is a new product, with new installation paths and support structures. For more information, refer to *IBM Parallel Environment Runtime Edition: Installation*.
- Support for launching and managing multiple parallel dynamic subjobs using a single scheduler or resource management allocation of cluster resources.
- A generic tool called Parallel Environment shell (PESH), which resembles a distributed shell with advanced grouping and filtering. It can be used for executing commands on distributed nodes, as well as sending commands to, and collecting distributed statistics from, running jobs. PESH also provides the ability to run PNSD commands concurrently on multiple nodes.
- Support for running applications in lockless mode.
- Enhanced scalability such that PE is now architected to run one million tasks per POE job.
- Support for the Host Fabric Interface (HFI) interconnect of the IBM POWER7 server in User Space.
- Support for the HFI global counter of the IBM POWER7 server, which has replaced the global counter of the high performance switch as a time source.
- A new messaging API called the parallel active messaging interface (PAMI), which replaces the LAPI interface used in earlier versions of Parallel Environment. In addition to providing point-to-point messaging support, PAMI also provides support for collective communications. This collective support is

used by PE MPI where it is appropriate. LAPI is still supported by PE, but its use is deprecated, and users should migrate to PAMI as soon as possible.

- For PE for AIX users, a new run queue-based coscheduler, in addition to the POE priority adjustment coscheduler. The run queue-based coscheduler uses features of AIX 7.1 and the POWER7 architecture to minimize the impact of jitter on user applications.
- Compliance with all requirements of the Message Passing Interface 2.2 standard, including the revisions listed in the *Annex B Change-Log*.
- For Linux users, support for checkpointing and restarting parallel jobs.
- The ability to create multiple PAMI contexts within each of the parallel API instances defined for a job. This allows cross-talk between the communication threads of each process, thereby providing greater performance. Note that the current MPI implementation does not take advantage of the multiple PAMI contexts feature.
- Extended task affinity support for jobs that are run on System x servers.
- Support for compiling and running applications that are Intel MPI ABI (Application Binary Interface) compatible, using Intel compilers and other Intel libraries. This support also includes a method for incorporating a POE module in the user application such that a user can simply invoke the application binary without running under POE (not Intel MPI ABI compatible).

  Note: At this time, PE Runtime Edition provides an initial port of MPICH2 over PAMI, with limited function only.
- Plug-ins (extensions) for third-party tool developers who want to create their own implementations of selected PE function.

Note also that the Partition Manager daemon (PMD) is now a setuid (set-userid-on-exec) program, which is owned only by the root user.

Chapter 1. What's new in PAMI?

Major changes and additions to PAMI include:

Table 2. Changes in this edition

Change or addition	For more information, see:
PAMI is packaged with Parallel Environment Runtime Edition for Linux on Power Version 1.2, which runs on the Host Fabric Interface (HFI) interconnect adapter or the InfiniBand host channel adapter.	

Table 3. Changes in the third edition

Change or addition	For more information, see:
PAMI is packaged with Parallel Environment Runtime Edition for Linux on X-Architecture Version 1.2, which runs on the Host Fabric Interface (HFI) interconnect adapter or the InfiniBand host channel adapter.	
New environment variables for MPICH2: **PAMI_EAGER** **PAMI_RMA_PENDING** **PAMI_RVZ** **PAMI_RZV** **PAMI_SHMEM_PT2PT**	Chapter 3, "PAMI environment variables," on page 147

Chapter 2. PAMI subroutines

The parallel active messaging interface (PAMI) component of Parallel Environment Runtime Edition includes several subroutines that are available for parallel programming.

PAMI_AMCollective_dispatch_set

Purpose

Initializes the dispatch functions for a dispatch ID.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_AMCollective_dispatch_set(pami_context_t                context,
                                             pami_algorithm_t              algorithm,
                                             size_t                        dispatch,
                                             pami_dispatch_callback_function fn,
                                             void *                        cookie,
                                             pami_collective_hint_t        options
                                             );
```

Fortran synopsis

```
include 'pamif.h'

pami_amcollective_dispatch_set(context, algorithm, dispatch, fn, cookie, options, ierror)
integer context
integer algorithm
integer dispatch
integer fn
integer cookie
integer options
integer ierror
```

Parameters

Input

context	Specifies the communication context.
algorithm	Specifies the active messaging collective to use to set the dispatch.
dispatch	Specifies the dispatch identifier to initialize.
fn	Specifies the dispatch receive function.
cookie	Specifies the dispatch function cookie.
options	Specifies the dispatch registration assertions.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: local, non-collective

Use this subroutine to initialize the dispatch functions for a dispatch ID.

Restrictions

There is no communication between tasks.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: ambcast.c

Subroutines: **PAMI_Dispatch_set**

PAMI_Client_create

Purpose

Initializes the PAMI runtime library for a client program.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Client_create(const char          *name,
                                 pami_client_t       *client,
                                 pami_configuration_t configuration[],
                                 size_t               num_configs
                                );
```

Fortran synopsis

```
include 'pamif.h'

pami_client_create(name, client, configuration, num_configs, ierror)
integer name
integer client
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

name Specifies a unique PAMI client name.

configuration
Specifies objects for the client.

num_configs
Specifies the number of configuration elements.

Output

client Specifies an opaque client object.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to initialize the PAMI runtime library for a client program.

A PAMI client represents a collection of resources to enable network communication. Each PAMI client that is initialized is unque and does not communicate directly with other clients. This allows middleware to be developed independently. An application can use various middleware concurrently. Resources are allocated and assigned at client creation time.

A PAMI client program is any software that calls a PAMI function. This includes applications, libraries, and other middleware. Some sample client names include: **ARMCI**, **MPI**, and **UPC**.

Client creation can be a synchronizing event, but it is not required to be implemented as a synchonizing event. Application code must not make any assumption about synchronization during client creation, and therefore must create clients in the same order in all processes of the job.

A communication context must be created before any data transfer functions can be called.

Return values

PAMI_SUCCESS
: The client has been successfully created.

PAMI_INVAL
: The client name has been rejected by the runtime library. This happens when a job scheduler requires the client name to match what is in the job description.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, advance.c, create.c, default-send.c, default-send-1.c, eager_concurrency.c, endpoint_table.c, hello.c, immediate_send_overflow.c, init.c, lock.c, long-header.c, long-header-matrix.c, multi-advance.c, multi-create.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c, tick.c, time.c, timebase.c

Subroutines: PAMI_Client_destroy, PAMI_Client_query, PAMI_Client_update, PAMI_Context_createv

PAMI_Client_destroy

Purpose

Finalizes the PAMI runtime library for a client program.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Client_destroy(pami_client_t *client
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_client_destroy(client, ierror)
integer client
integer ierror
```

Parameters

Input

client Specifies a client handle.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to finalize the PAMI runtime library for a client program. The client handle will be changed to a non-valid value so that it is clearly destroyed.

Restrictions

Calling any PAMI subroutines after this subroutine, using the client handle from any thread, is not recommended.

Return values

PAMI_SUCCESS
The client has been successfully destroyed.

PAMI_INVAL
The client is not valid, that is, it has already been destroyed.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, advance.c, create.c, default-send-1.c, default-send.c, eager_concurrency.c, endpoint_table.c, hello.c, immediate_send_overflow.c, init.c, lock.c, long-header-matrix.c, long-header.c, multi-advance.c, multi-create.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_unexpected_func.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c

Subroutines: **PAMI_Client_create**, **PAMI_Client_query**, **PAMI_Client_update**

PAMI_Client_query

Purpose

Queries the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Client_query(pami_client_t      *client,
                                pami_configuration_t configuration[],
                                size_t              num_configs
                               );
```

Fortran synopsis

```
include 'pamif.h'

pami_client_query(client, configuration, num_configs, ierror)
integer client
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

client Specifies the PAMI client.

num_configs
Specifies the number of configuration elements.

Input/output

configuration
Specifies the configuration attribute to query.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to query the value of a configuration attribute.

Return values

PAMI_SUCCESS
The query has completed successfully.

PAMI_INVAL
The query has failed due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, bcast_subcomm2.c, default-send-1.c, default-send-nplus1.c, default-send.c, eager_concurrency.c, endpoint_table.c, immediate_send_overflow.c, init.c, long-header-hard-match.c, long-header-hard-opp.c, long-header-matrix.c, long-header.c, multi-advance.c, multi-create.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c, tick.c

Subroutines: PAMI_Client_create, PAMI_Client_destroy, PAMI_Client_update

PAMI_Client_update

Purpose

Updates the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Client_update(pami_client_t       *client,
                                 pami_configuration_t configuration[],
                                 size_t               num_configs
                                );
```

Fortran synopsis

```
include 'pamif.h'

pami_client_update(client, configuration, num_configs, ierror)
integer client
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

client
Specifies the PAMI client.

configuration
Specifies the configuration attribute to update.

num_configs
Specifies the number of configuration elements.

Output

ierror
Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to update the value of a configuration attribute.

Return values

PAMI_SUCCESS
The update has completed successfully.

PAMI_INVAL
The update has failed due to non-valid parameters. This failure occurs if the subroutine tries to update a read-only attribute, for example.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Client_create, PAMI_Client_destroy, PAMI_Client_query

PAMI_Collective

Purpose

Serves as a wrapper function for PAMI operations that are related to collective communication.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Collective(pami_context_t context,
                              pami_xfer_t   *cmd
                              );

typedef void* pami_context_t;

typedef struct pami_xfer_t
{
  pami_event_function      cb_done;
  void                    *cookie;
  pami_algorithm_t         algorithm;
  pami_collective_hint_t   options;
  pami_collective_t        cmd;
} pami_xfer_t;
```

Fortran synopsis

```
include 'pamif.h'

pami_collective(context, cmd, ierror)
integer context
integer cmd
integer ierror
```

Parameters

Input

context Specifies the communication context.

Output

cmd Specifies the command type value.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

This subroutine is used for several different operations, which are indicated by the transfer type values in the **pami_collective_t** data structure. The **pami_collective_t** data structure is defined as:

```
{
    pami_allreduce_t        xfer_allreduce;
    pami_broadcast_t        xfer_broadcast;
pami_reduce_t          xfer_reduce;
pami_allgather_t       xfer_allgather;
pami_allgatherv_t      xfer_allgatherv;
pami_allgatherv_int_t  xfer_allgatherv_int;
pami_scatter_t         xfer_scatter;
pami_scatterv_t        xfer_scatterv;
pami_scatterv_int_t    xfer_scatterv_int;
pami_gather_t          xfer_gather;
pami_gatherv_t         xfer_gatherv;
pami_gatherv_int_t     xfer_gatherv_int;
pami_alltoall_t        xfer_alltoall;
pami_alltoallv_t       xfer_alltoallv;
pami_alltoallv_int_t   xfer_alltoallv_int;
pami_ambroadcast_t     xfer_ambroadcast;
pami_amscatter_t       xfer_amscatter;
pami_amgather_t        xfer_amgather;
pami_amreduce_t        xfer_amreduce;
pami_scan_t            xfer_scan;
pami_barrier_t         xfer_barrier;
pami_reduce_scatter_t  xfer_reduce_scatter;
} pami_collective_t;
```

Though the **pami_xfer_t** structure applies only to the C version of **PAMI_Collective**, Table 4 includes the Fortran equivalents of the C datatypes.

Table 4 lists the values of the **pami_xfer_type_t** structure for C and the explicit transfer types and values for Fortran.

Table 4. PAMI_Collective structure types

Transfer type, value of transfer type (C)	Enumeration member as interpreted by PAMI_Collective (C)	Transfer types and values (Fortran)
xfer_broadcast, PAMI_XFER_BROADCAST	**pami_broadcast_t**	**integer PAMI_XFER_BROADCAST, parameter (PAMI_XFER_BROADCAST=0)**
xfer_allreduce, PAMI_XFER_ALLREDUCE	**pami_allreduce_t**	**integer PAMI_XFER_ALLREDUCE, parameter (PAMI_XFER_ALLREDUCE=1)**
xfer_reduce, PAMI_XFER_REDUCE	**pami_reduce_t**	**integer PAMI_XFER_REDUCE, parameter (PAMI_XFER_REDUCE=2)**
xfer_allgather, PAMI_XFER_ALLGATHER	**pami_allgather_t**	**integer PAMI_XFER_ALLGATHER, parameter (PAMI_XFER_ALLGATHER=3)**
xfer_allgatherv, PAMI_XFER_ALLGATHERV	**pami_allgatherv_t**	**integer PAMI_XFER_ALLGATHERV, parameter (PAMI_XFER_ALLGATHERV=4)**
xfer_allgatherv_int, PAMI_XFER_ALLGATHERV_INT	**pami_allgatherv_int_t**	**integer PAMI_XFER_ALLGATHERV_INT, parameter (PAMI_XFER_ALLGATHERV_INT=5)**
xfer_scatter, PAMI_XFER_SCATTER	**pami_scatter_t**	**integer PAMI_XFER_SCATTER, parameter (PAMI_XFER_SCATTER=6)**
xfer_scatterv, PAMI_XFER_SCATTERV	**pami_scatterv_t**	**integer PAMI_XFER_SCATTERV, parameter (PAMI_XFER_SCATTERV=7)**
xfer_scatterv_int, PAMI_XFER_SCATTERV_INT	**pami_scatterv_int_t**	**integer PAMI_XFER_SCATTERV_INT, parameter (PAMI_XFER_SCATTERV_INT=8)**
xfer_gather, PAMI_XFER_GATHER	**pami_gather_t**	**integer PAMI_XFER_GATHER, parameter (PAMI_XFER_GATHER=9)**
xfer_gatherv, PAMI_XFER_GATHERV	**pami_gatherv_t**	**integer PAMI_XFER_GATHERV, parameter (PAMI_XFER_GATHERV=10)**
xfer_gatherv_int, PAMI_XFER_GATHERV_INT	**pami_gatherv_int_t**	**integer PAMI_XFER_GATHERV_INT, parameter (PAMI_XFER_GATHERV_INT=11)**

Table 4. PAMI_Collective structure types (continued)

Transfer type, value of transfer type (C)	Enumeration member as interpreted by PAMI_Collective (C)	Transfer types and values (Fortran)
xfer_barrier, PAMI_XFER_BARRIER	**pami_barrier_t**	**integer PAMI_XFER_BARRIER, parameter (PAMI_XFER_BARRIER=12)**
xfer_fence, PAMI_XFER_FENCE	**pami_fence_t**	**integer PAMI_XFER_FENCE, parameter (PAMI_XFER_FENCE=13)**
xfer_alltoall, PAMI_XFER_ALLTOALL	**pami_alltoall_t**	**integer PAMI_XFER_ALLTOALL, parameter (PAMI_XFER_ALLTOALL=14)**
xfer_alltoall, PAMI_XFER_ALLTOALL	**pami_alltoall_t**	**integer PAMI_XFER_ALLTOALL, parameter (PAMI_XFER_ALLTOALL=14)**
xfer_alltoallv, PAMI_XFER_ALLTOALLV	**pami_alltoallv_t**	**integer PAMI_XFER_ALLTOALLV, parameter (PAMI_XFER_ALLTOALLV=15)**
xfer_alltoallv_int, PAMI_XFER_ALLTOALLV_INT	**pami_alltoallv_int_t**	**integer PAMI_XFER_ALLTOALLV_INT, parameter (PAMI_XFER_ALLTOALLV_INT=16)**
xfer_scan, PAMI_XFER_SCAN	**pami_scan_t**	**integer PAMI_XFER_SCAN, parameter (PAMI_XFER_SCAN=17)**
xfer_reduce_scatter, PAMI_XFER_REDUCE_SCATTER	**pami_reduce_scatter_t**	**integer PAMI_XFER_REDUCE_SCATTER, parameter (PAMI_XFER_REDUCE_SCATTER=18)**
xfer_ambroadcast, PAMI_XFER_AMBROADCAST	**pami_ambroadcast_t**	**integer PAMI_XFER_AMBROADCAST, parameter (PAMI_XFER_AMBROADCAST=19)**
xfer_amscatter, PAMI_XFER_AMSCATTER	**pami_amscatter_t**	**integer PAMI_XFER_AMSCATTER, parameter (PAMI_XFER_AMSCATTER=20)**
xfer_amgather, PAMI_XFER_AMGATHER	**pami_amgather_t**	**integer PAMI_XFER_AMGATHER, parameter (PAMI_XFER_AMGATHER=21)**
xfer_amreduce, PAMI_XFER_AMREDUCE	**pami_amreduce_t**	**integer PAMI_XFER_AMREDUCE, parameter (PAMI_XFER_AMREDUCE=22)**

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

pami_allreduce_t details

Use the **pami_allreduce_t** data structure to create and post a non-blocking allreduce operation.

```
typedef struct
  {
    char              *sndbuf;
    pami_type_t        stype;
    size_t             stypecount;
    char              *rcvbuf;
    pami_type_t         rtype;
    size_t              rtypecount;
    pami_data_function  op;
    void               *data_cookie;
    int                 commutative;
  } pami_allreduce_t;
```

Parameters

```
[in] cb_done     Callback to invoke when message is complete
[in] geometry    Geometry to use for this collective operation
[in] sbuffer     Source buffer
[in] stype       Source buffer type and datatype of the operation
[in] stypecount  Source buffer type count
[in] rbuffer     Receive buffer
[in] rtype       Receive buffer layout
[in] rtypecount  Receive buffer type count
[in] op          Reduce operation
```

Return codes

0 The operation has completed successfully.

pami_broadcast_t details

Use the **pami_broadcast_t** data structure to create and post a non-blocking broadcast operation.

```
typedef struct
  {
    pami_endpoint_t            root;
    char                     * buf;
    pami_type_t                type;
    size_t                     typecount;
  } pami_broadcast_t;
```

Parameters

```
[in] root      Endpoint of the node performing the broadcast
[in] buf       Source buffer to broadcast on root, destination buffer on non-root
[in] type      Data type layout (might be different on root and destinations)
[in] count     Single type replication count
```

Return codes

0 The operation has completed successfully.

pami_reduce_t details

Use the **pami_reduce_t** data structure to create and post a non-blocking reduce operation.

```
typedef struct
  {
    pami_endpoint_t            root;
    char                     * sndbuf;
    pami_type_t                stype;
    size_t                     stypecount;
    char                     * rcvbuf;
    pami_type_t                rtype;
    size_t                     rtypecount;
    pami_data_function         op;
    void                     * data_cookie;
    int                        commutative;
  } pami_reduce_t;
```

Parameters

```
[in]  root       Endpoint of the reduce root
[in]  sndbuf     Source buffer
[in]  stype      Source buffer type and datatype of the operation
[in]  stypecount Source buffer type count
[in]  rcvbuf     Receive buffer
[in]  rtype      Receive buffer layout
[in]  rtypecount Receive buffer type count
[in]  op         Reduce operation
```

Return codes

0 The operation has completed successfully.

pami_allgather_t details

Use the **pami_allgather_t** data structure to create and post a non-blocking allgather operation.

```
typedef struct
  {
    char                        * sndbuf;
    pami_type_t                   stype;
    size_t                        stypecount;
    char                        * rcvbuf;
    pami_type_t                   rtype;
    size_t                        rtypecount;
  } pami_allgather_t;
```

Parameters

```
[in]  cb_done    Callback to invoke when message is complete
[in]  geometry   Geometry to use for this collective operation
[in]  src        Source buffer to send
[in]  stype      Data layout of send buffer
[in]  stypecount Replication count of the type
[in]  rcv        Source buffer to receive the data
[in]  rtype      Data layout of each receive buffer
[in]  rtypecount Replication count of the type
```

Return codes

0 The operation has completed successfully.

pami_allgatherv_t details

Use the **pami_allgatherv_t** data structure to create and post a non-blocking allgatherv operation.

```
typedef struct
  {
    char                        * sndbuf;
    pami_type_t                   stype;
    size_t                        stypecount;
    char                        * rcvbuf;
    pami_type_t                   rtype;
    size_t                      * rtypecounts;
    size_t                      * rdispls;
  } pami_allgatherv_t;
```

Parameters

```
[in]  cb_done     Callback to invoke when message is complete
[in]  geometry    Geometry to use for this collective operation
[in]  sndbuf      The base address of the buffers containing data to be sent
```

```
[in]  stype       A single type datatype
[in]  stypecount  Type replication count
[out]   rcvbuf      The base address of the buffer for data reception
[in]  rtype       A single type datatype
[in]  rtypecounts Array of type replication counts (size of geometry length)
[in]  rdispls     Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_allgatherv_int_t details

Use the **pami_allgatherv_int_t** data structure to create and post a non-blocking allgatherv operation.

```
typedef struct
  {
    char                        * sndbuf;
    pami_type_t                   stype;
    int                           stypecount;
    char                        * rcvbuf;
    pami_type_t                   rtype;
    int                         * rtypecounts;
    int                         * rdispls;
  } pami_allgatherv_int_t;
```

Parameters

```
[in]  cb_done     Callback to invoke when message is complete
[in]  geometry    Geometry to use for this collective operation
[in]  sndbuf      The base address of the buffers containing data to be sent
[in]  stype       A single type datatype
[in]  stypecount  Type replication count
[out]   rcvbuf      The base address of the buffer for data reception
[in]  rtype       A single type datatype
[in]  rtypecounts Array of type replication counts (size of geometry length)
[in]  rdispls     Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_scatter_t details

Use the **pami_scatter_t** data structure to create and post a non-blocking scatter operation.

```
typedef struct
  {
    pami_endpoint_t               root;
    char                        * sndbuf;
    pami_type_t                   stype;
    size_t                        stypecount;
    char                        * rcvbuf;
    pami_type_t                   rtype;
    size_t                        rtypecount;
  } pami_scatter_t;
```

Parameters

```
[in]  root       Endopint of the reduce root node
[in]  sndbuf     Source buffer
[in]  stype      Source buffer type
[in]  stypecount Source buffer type count
```

```
[in]  rcvbuf      Receive buffer
[in]  rtype       Receive buffer layout
[in]  rtypecount  Receive buffer type count
```

Return codes

0 The operation has completed successfully.

pami_scatterv_t details

Use the **pami_scatterv_t** data structure to create and post a non-blocking scatterv operation.

```
  typedef struct
  {
    pami_endpoint_t             root;
    char                      * sndbuf;
    pami_type_t                 stype;
    size_t                    * stypecounts;
    size_t                    * sdispls;
    char                      * rcvbuf;
    pami_type_t                 rtype;
    size_t                      rtypecount;
  } pami_scatterv_t;
```

Parameters

```
[in]  root         Endopint of the scatterv root node
[in]  sndbuf       The base address of the buffers containing data to be sent
[in]  stype        A single type datatype
[in]  stypecounts  An array of type replication counts (size of geometry length)
[in]  sdispls      Array of offsets into the sndbuf (size of geometry length)
[in]  rbuffer      Receive buffer
[in]  rtype        A single type datatype
[in]  rtypecount   Receive buffer type replication count
```

Return codes

0 The operation has completed successfully.

pami_scatterv_int_t details

Use the **pami_scatterv_int_t** data structure to create and post a non-blocking scatterv operation.

```
typedef struct
  {
    pami_endpoint_t             root;
    char                      * sndbuf;
    pami_type_t                 stype;
    int                       * stypecounts;
    int                       * sdispls;
    char                      * rcvbuf;
    pami_type_t                 rtype;
    int                         rtypecount;
  } pami_scatterv_int_t;
```

Parameters

```
[in]  root         Endopint of the scatterv_int root node
[in]  sndbuf       The base address of the buffers containing data to be sent
[in]  stype        A single type datatype
[in]  stypecounts  An array of type replication counts (size of geometry length)
[in]  sdispls      Array of offsets into the sndbuf (size of geometry length)
```

```
[in]  rbuffer      Receive buffer
[in]  rtype        A single type datatype
[in]  rtypecount   Receive buffer type replication count
```

Return codes

0 The operation has completed successfully.

pami_gather_t details

Use the **pami_gather_t** data structure to create and post a non-blocking gather operation.

```
typedef struct
  {
    pami_endpoint_t            root;
    char                     * sndbuf;
    pami_type_t                stype;
    size_t                     stypecount;
    char                     * rcvbuf;
    pami_type_t                rtype;
    size_t                     rtypecount;
  } pami_gather_t;
```

Parameters

```
[in]  root        The root endpoint of the gather operation
[in]  sndbuf      Source buffer to send
[in]  stype       Data layout of send buffer
[in]  stypecount  Replication count of the type
[in]  rcvbuf      Source buffer to receive the data
[in]  rtype       Data layout of each receive buffer
[in]  rtypecount  Replication count of the type
```

Return codes

0 The operation has completed successfully.

pami_gatherv_t details

Use the **pami_gatherv_t** data structure to create and post a non-blocking gatherv operation.

```
typedef struct
  {
    pami_endpoint_t            root;
    char                     * sndbuf;
    pami_type_t                stype;
    size_t                     stypecount;
    char                     * rcvbuf;
    pami_type_t                rtype;
    size_t                   * rtypecounts;
    size_t                   * rdispls;
  } pami_gatherv_t;
```

Parameters

```
[in]  root          The root endpoint for the gatherv operation
[in]  sndbuf        The base address of the buffers containing data to be sent
[in]  stype         A single type datatype
[in]  stypecount    Type replication count
[out]   rcvbuf        The base address of the buffer for data reception
[in]  rtype         A single type datatype
[in]  rtypecounts   Array of type replication counts (size of geometry length)
[in]  rdispls       Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_gatherv_int_t details

Use the **pami_gatherv_int_t** data structure to create and post a non-blocking gatherv operation.

```
typedef struct
  {
    pami_endpoint_t            root;
    char                     * sndbuf;
    pami_type_t                stype;
    int                        stypecount;
    char                     * rcvbuf;
    pami_type_t                rtype;
    int                      * rtypecounts;
    int                      * rdispls;
  } pami_gatherv_int_t;
```

Parameters

```
[in]  root        The root endpoint for the gatherv operation
[in]  sndbuf      The base address of the buffers containing data to be sent
[in]  stype       A single type datatype
[in]  stypecount  Type replication count
[out]   rcvbuf      The base address of the buffer for data reception
[in]  rtype       A single type datatype
[in]  rtypecounts  Array of type replication counts (size of geometry length)
[in]  rdispls     Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_alltoall_t details

Use the **pami_alltoall_t** data structure to create and post a non-blocking alltoall operation.

```
typedef struct
  {
    char                     * sndbuf;
    pami_type_t                stype;
    size_t                     stypecount;
    char                     * rcvbuf;
    pami_type_t                rtype;
    size_t                     rtypecount;
  } pami_alltoall_t;
```

Parameters

```
[in]  sndbuf      The base address of the buffers containing data to be sent
[in]  stype       Single datatype of the send buffer
[in]  stypecount  Single type replication count
[out]   rbuf        The base address of the buffer for data reception
[in]  rtype       Single datatype of the receive buffer
[in]  rtypecount  Single type replication count
```

Return codes

0 The operation has completed successfully.

pami_alltoallv_t details

Use the **pami_alltoallv_t** data structure to create and post a non-blocking alltoall vector operation.

```
typedef struct
  {
    char                      * sndbuf;
    pami_type_t                 stype;
    size_t                    * stypecounts;
    size_t                    * sdispls;
    char                      * rcvbuf;
    pami_type_t                 rtype;
    size_t                    * rtypecounts;
    size_t                    * rdispls;
  } pami_alltoallv_t;
```

Parameters

```
[in]  sndbuf       The base address of the buffers containing data to be sent
[in]  stype        A single type datatype
[in]  stypecounts  An array of type replication counts (size of geometry length)
[in]  sdispls      Array of offsets into the sndbuf (size of geometry length)
[out]   rcvbuf       The base address of the buffer for data reception
[in]  rtype        A single type datatype
[in]  rtypecounts  Array of type replication counts (size of geometry length)
[in]  rdispls      Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_alltoallv_int_t details

Use the **pami_alltoallv_int_t** data structure to create and post a non-blocking alltoall vector operation.

```
typedef struct
  {
    char                      * sndbuf;
    pami_type_t                 stype;
    int                       * stypecounts;
    int                       * sdispls;
    char                      * rcvbuf;
    pami_type_t                 rtype;
    int                       * rtypecounts;
    int                       * rdispls;
  } pami_alltoallv_int_t;
```

Parameters

```
[in]  sndbuf       The base address of the buffers containing data to be sent
[in]  stype        A single type datatype
[in]  stypecounts  An array of type replication counts (size of geometry length)
[in]  sdispls      Array of offsets into the sndbuf (size of geometry length)
[out]   rcvbuf       The base address of the buffer for data reception
[in]  rtype        A single type datatype
[in]  rtypecounts  Array of type replication counts (size of geometry length)
[in]  rdispls      Array of offsets into the rcvbuf (size of geometry length)
```

Return codes

0 The operation has completed successfully.

pami_ambroadcast_t details

Use the **pami_ambroadcast_t** data structure to create and post a non-blocking active message broadcast operation. This operation differs from **LAPI_Amsend** in only one way: it takes geometry/team as an argument. The semantics are as follows: the included header and data are broadcast to every place in the team. The completion handler is called on the sending side as soon as send buffers can be reused. On the receiving side, the usual two-phase reception protocol is run: a header handler determines the address to which to deposit the data and sets the address of a receive completion hander to be called once the data has arrived.

```
typedef struct
  {
    size_t                    dispatch;
    void                    * user_header;
    size_t                    headerlen;
    void                    * sndbuf;
    pami_type_t               stype;
    size_t                    stypecount;
  } pami_ambroadcast_t;
```

Parameters

```
[in]  dispatch     Registered dispatch ID to use
[in]  user_header  Single metadata to send to destination in the header
[in]  headerlen    Length of the metadata (can be 0)
[in]  src          Base source buffer to broadcast
[in]  stype        Datatype of the send buffer
[in]  stypecount   Replication count of the send buffer datatype
```

Return codes

0 The operation has completed successfully.

pami_amscatter_t details

Use the **pami_amscatter_t** data structure to create and post a non-blocking active message scatter operation. This operation is somewhat more complex than an active message broadcast operation, because it allows different headers and data buffers to be sent to everyone in the team.

```
typedef struct
  {
    size_t                    dispatch;
    void                    * headers;
    size_t                    headerlen;
    void                    * sndbuf;
    pami_type_t               stype;
    size_t                    stypecount;
  } pami_amscatter_t;
```

Parameters

```
[in]  dispatch      Registered dispatch ID to use
[in]  headers       Array of metadata to send to destination
[in]  headerlength  Length of every header in the headers array
[in]  src           Base source buffer to scatter (size of geometry)
[in]  stype         Single datatype of the send buffer
[in]  stypecount    Replication count of the send buffer data type
```

Return codes

0 The operation has completed successfully.

pami_amgather_t details

Use the **pami_amgather_t** data structure to create and post a non-blocking active message gather operation. This operation is the reverse of an active message scatter operation. It works as follows: only the header is broadcast to the team. No data is broadcast to the team. Each place in the team runs the header handler and points to a data buffer in local space. A reverse transfer then takes place, in which the buffer is sent from the receiver back to the sender, and deposited in one of the buffers provided as part of the original call (the "data" parameter).

```
typedef struct
  {
    size_t                      dispatch;
    void                      * headers;
    size_t                      headerlen;
    void                      * rcvbuf;
    pami_type_t                 rtype;
    size_t                      rtypecount;
  } pami_amgather_t;
```

Parameters

```
[in]  dispatch    Registered dispatch ID to use
[in]  headers     Array of metadata to send to destination
[in]  headerlen   Length of every header in headers array
[in]  rcvbuf      Target buffer of the gather operation (size of geometry)
[in]  rtype       Data layout of the incoming gather
[in]  rtypecount  Replication count of the incoming gather
[in]  cb_info     Data done callback to call on completion
```

Return codes

0	The operation has completed successfully.

pami_amreduce_t details

Use the **pami_amreduce_t** data structure to create and post a non-blocking active message reduce operation. This operation is fairly straightforward compared to an active message gather operation. Instead of collecting the data without processing it, all buffers are reduced using the operation and data type provided by the sender. The final reduced data is deposited in the original buffer provided by the initiator. On the receiving side, the algorithm can change the buffers provided by the header handler, which might avoid having the implementor allocate more memory for internal buffering.

```
typedef struct
  {
    size_t                      dispatch;
    void                      * user_header;
    size_t                      headerlen;
    void                      * rcvbuf;
    pami_type_t                 rtype;
    size_t                      rtypecount;
    pami_data_function          op;
    void                      * data_cookie;
    int                         commutative;
  } pami_amreduce_t;
```

Parameters

```
[in]  dispatch    Registered dispatch ID to use
[in]    geometry    Geometry to use for this collective operation
                    (Null indicates the global geometry)
[in]  headers     Metadata to send to destinations in the header
```

```
[in]  rcvbuf      Target buffer of the reduce operation (size of geometry)
[in]  rtype       Data layout of the incoming reduce
[in]  rtypecount  Replication count of the incoming reduce
[in]  op          Operation type
```

Return codes

0 The operation has completed successfully.

pami_scan_t details

Use the **pami_scan_t** data structure to create and post a non-blocking scan operation.

```
typedef struct
  {
    char                       * sndbuf;
    pami_type_t                  stype;
    size_t                       stypecount;
    char                       * rcvbuf;
    pami_type_t                  rtype;
    size_t                       rtypecount;
    pami_data_function           op;
    void                       * data_cookie;
    int                          exclusive;
  } pami_scan_t;
```

Parameters

```
[in]  sndbuf      Source buffer
[in]  stype       Source buffer type and datatype of the operation
[in]  stypecount  Source buffer type count
[in]  rcvbuf      Receive buffer
[in]  rtype       Receive buffer layout
[in]  rtypecount  Receive buffer type count
[in]  op          Reduce operation
[in]  exclusive   The scan operation is exclusive of the current node
```

Return codes

0 The operation has completed successfully.

pami_barrier_t details

Use the **pami_barrier_t** data structure to create and post a non-blocking barrier operation.

```
typedef struct
  {
  } pami_barrier_t;
```

Parameters

```
        geometry  Geometry to use for this collective operation
[in]    cb_done   Callback to invoke when message is complete
```

Return codes

0 The operation has completed successfully.

pami_reduce_scatter_t details

Use the **pami_reduce_scatter_t** data structure to create and post a non-blocking reduce scatter operation.

```
typedef struct
  {
    char                       * sndbuf;
    pami_type_t                  stype;
    size_t                       stypecount;
    char                       * rcvbuf;
    pami_type_t                  rtype;
    size_t                     * rcounts;
    pami_data_function           op;
    void                       * data_cookie;
    int                          commutative;
  } pami_reduce_scatter_t;
```

Parameters

```
[in]  sndbuf      Source buffer
[in]  stype       Source buffer type and datatype of the operation
[in]  stypecount  Source buffer type count
[in]  rcvbuf      Receive buffer
[in]  rtype       Receive buffer layout
[in]  rtypecount  Receive buffer type count
[in]  rcounts     Number of elements to receive from the destinations (common on all nodes)
[in]  op          Reduce operation
```

Return codes

0	The operation has completed successfully.

PAMI_Context_advance

Purpose

Advances the progress engine for a single communication context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_advance(pami_context_t context,
                                    size_t         maximum
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_advance (pami_context_t context, maximum, ierror)
integer context
integer maximum
integer ierror
```

Parameters

Input

context Specifies the communication context.

maximum
Specifies the maximum number of internal poll iterations.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: polling advance

Use this subroutine to advance the progress engine for a single communication context.

This subroutine can complete zero, one, or more outbound transfers. It can call dispatch handlers for incoming transfers. It can also call work event callbacks that were previously posted to a communication context.

This subroutine will return after the first poll iteration that results in a processed event, or, if no events are processed, after polling for the maximum number of iterations.

Restrictions

This subroutine is not thread-safe. The application must make sure that only one thread advances a context at any time.

Return values

PAMI_SUCCESS
: An event has occurred and has been processed.

PAMI_EAGAIN
: No event has occurred.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: advance.c, ambcast.c, bcast_subcomm.c, default-send-1.c, default-send.c, immediate_send_overflow.c, init_coll.c, long-header-matrix.c, long-header.c, multi-advance.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple-send-immediate.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c

Subroutines: PAMI_Context_lock, PAMI_Context_trylock

PAMI_Context_advancev

Purpose

Advances the progress engine for multiple communication contexts.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_advancev(pami_context_t context[],
                                    size_t         count,
                                    size_t         maximum
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_advancev(pami_context_t context, count, maximum, ierror)
integer context
integer maximum
integer ierror
```

Parameters

Input

context Specifies an array of communication contexts.

count Specifies the number of communication contexts.

maximum
Specifies the maximum number of internal poll iterations on each context.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: polling advance

Use this subroutine to advance the progress engine for multiple communication contexts.

This subroutine complete zero, one, or more outbound transfers. It can call dispatch handlers for incoming transfers. It can also call work event callbacks that were previously posted to a communication context.

This subroutine will return after the first poll iteration that results in a processed event on any context, or, if no events are processed, after polling for the maximum number of iterations.

It is possible to define a set of communication contexts that are always advanced together by any PAMI client thread. It is the responsibility of the PAMI client to atomically lock the context set, perhaps by using the **PAMI_Context_lock()** subroutine on a designated leader context, and to manage the PAMI client threads to make sure that only one thread ever advances the set of contexts.

Restrictions

This subroutine is not thread-safe. The application must make sure that only one thread advances the contexts at any time.

Return values

PAMI_SUCCESS
: An event has occurred and has been processed.

PAMI_EAGAIN
: No event has occurred.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, endpoint_table.c

PAMI_Context_createv

Purpose

Creates new, independent communication contexts for a client.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_createv(pami_client_t        client,
                                    pami_configuration_t configuration[],
                                    size_t               num_configs,
                                    pami_context_t      *context,
                                    size_t               ncontexts
                                   );
```

Fortran synopsis

```
include 'pamif.h'
pami_context_createv(client, configuration, num_configs, context, ncontexts, ierror)
integer client
integer configuration
integer num_configs
integer context
integer ncontexts
integer ierror
```

Parameters

Input

client Specifies the client handle.

configuration
Specifies a list of configurable attributes and values.

num_configs
Specifies the number of configurations. The value can be **0**.

ncontexts
Specifies the number of contexts to be created.

Output

context Specifies an array of communication contexts to initialize.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: local

Use this subroutine to create new, independent communication contexts for a client.

Contexts are local "threading points" that an application may use to optimize concurrent communcation operations. A context handle is an opaque object type that the application must not directly read or write the value of the object. Communication contexts have these features:

- Each context is a partition of the local resources assigned to the client object for each task.
- Every context within a client has equivalent functionality and semantics.
- Communcation operations initiated by the local task will use the opaque context object to identify the specific threading point that will be used to issue the communication independent of communication occuring in other contexts.
- All local event callbacks associated with a communication operation will be invoked by the thread which advances the context that was used to initiate the operation.
- A context is a local object and is not used to directly address a communication destination.
- Progress is driven independently among contexts.
- Progress may be driven concurrently among contexts, by using multiple threads, as desired by the application.
- All contexts created by a client must be advanced by the application to prevent deadlocks. This is the "all advance" rule.
- The rationale for the "all-advance" rule is that for a point-to-point send or a collective operation, a communication is posted to a context, and delivered to a context on a remote task. The internals of the messaging layer could implement "horizontal" parallelism by injecting data or processing across multiple contexts associated with the client. Consequently, data can be received across multiple contexts. To guarantee progress of a single operation, every context must be advanced by the user.
- The user application/client of pami may have more knowledge about the communication patterns and the "all advance" rule can be relaxed. To do this the user can specify special hints to disable "horizontal", or cross context parallelism. See **pami_send_hint_t** and the "multicontext" option. This option must be switched "off" to disable parallelization and the "all advance rule".
- The task based geometry constructor implies all contexts are included in the geometry, with a single participant per task. All contexts must be advanced during a collective operation. However, the user can specify special hints to disable "horizontal", or cross context parallelsm. See **pami_collective_hint_t** and the "multicontext" option. This option must be switched "off" to disable parallelization and the "all advance rule".

The context configuration attributes can include such context optimizations as shared memory and collective acceleration.

Context creation does not involve communication or syncronization with other tasks.

Thread considerations

Applications map, or "apply", threading resources to contexts. Operations on contexts are critical sections and are not thread-safe. The application must make sure that critical sections are protected from re-entrant use. PAMI provides mechanisms for controlling access to critical sections.

Return values

PAMI_SUCCESS
Contexts have been created.

PAMI_INVAL
: The configuration could not be satisified or there were errors in other parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C Examples adi.c, advance.c, create.c, default-send-1.c, default-send.c, eager_concurrency.c, endpoint_table.c, immediate_send_overflow.c, init.c, lock.c, long-header-matrix.c, long-header.c, multi-advance.c, multi-create.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c, tick.c

PAMI_Context_destroyv

Purpose

Destroys the communication context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_destroyv (pami_context_t *contexts,
                                     size_t          ncontexts
                                     );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_destroyv (contexts, ncontexts, ierror)
integer contexts
integer ncontexts
integer ierror
```

Parameters

Input

ncontexts
Specifies the number of contexts in the list.

Input/output

contexts
Specifies the communication context list.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to destroy the communication context. The context handles will be changed to an non-valid value so that they are clearly destroyed.

Restrictions

This subroutine is not thread-safe. The application must make sure that only one thread destroys the communication contexts for a client.

The PAMI_Context_lock(), PAMI_Context_trylock(), and PAMI_Context_unlock() subroutines must not be used to ensure thread-safe access to the context destroy function, as the lock associated with each context will be destroyed.

Calling any PAMI subroutines after the context is destroyed, using the communication context from any thread, is not recommended.

Return values

PAMI_SUCCESS
: The contexts have been destroyed.

PAMI_INVAL
: Some context is non-valid, that is, it has already been destroyed.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: advance.c, create.c, default-send-1.c, default-send.c, endpoint_table.c, immediate_send_overflow.c, init.c, lock.c, long-header-matrix.c, long-header.c, multi-advance.c, multi-create.c, post-multithreaded-perf.c, post-multithreaded.c, post.c, rdma-matrix.c, send_to_self.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c

PAMI_Context_lock

Purpose

Acquires an atomic lock on a communication context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_lock(pami_context_t context
                                );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_lock(context, ierror)
integer context
integer ierror
```

Parameters

Input

context	Specifies the communication context.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to acquire an atomic lock on a communication context.

Restrictions

The lock cannot be assumed to be recursive or non-recursive.

This subroutine will block until the lock is acquired.

Return values

PAMI_SUCCESS
The lock has been acquired.

PAMI_INVAL
The parameter is not valid.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: default-send-1.c, endpoint_table.c, lock.c, post.c, post-multithreaded.c, post-multithreaded-perf.c

PAMI_Context_post

Purpose

Posts work to a context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_post(pami_context_t      context,
                                pami_work_t        *work,
                                pami_work_function  fn,
                                void               *cookie
                               );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_post(context, work, fn, cookie, ierror)
integer context
integer work
integer fn
integer cookie
integer ierror
```

Parameters

Input

context	Specifies the communication context.
work	Specifies the opaque storage for the work (used internally).
fn	Specifies the event work function to call on the context.
cookie	Specifies the opaque data pointer to pass to the work function.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to post work to a context. This subroutine is thread safe.

It is not required that the target context is locked, or otherwise reserved, by an external atomic operation to ensure thread safety. The PAMI runtime library performs any necessary atomic operations in order to post the work to the context.

The work function will be invoked in the thread that advances the target context. There is no explicit completion notification provided to the posting thread when a thread advancing the target context returns from the work function. If the posting thread desires a completion notification it must explicitly program such notifications, via the PAMI_Context_post() interface or other mechanism.

If this subroutine returns PAMI_SUCCESS, it may dispose of the pami_work_t object (including re-posting) if it so chooses, provided arrangements were made to pass the address of the pami_work_t object into the work function (using the cookie, for example).

If this subroutine returns PAMI_EAGAIN, it must not have altered the pami_work_t object in any way.

Return values

PAMI_SUCCESS
: The work has been posted.

PAMI_INVAL
: The post operation was rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, post.c, post-multithreaded.c, post-multithreaded-perf.c

PAMI_Context_query

Purpose

Queries the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_query(pami_context_t       context,
                                 pami_configuration_t configuration[],
                                 size_t               num_configs
                                );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_query(context, configuration, num_configs, ierror)
integer context
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

context Specifies the PAMI context.

num_configs
Specifies the number of configuration elements.

Input/output

configuration
Specifies the configuration attribute to query.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to query the value of an attribute.

Return values

PAMI_SUCCESS
The query has completed successfully.

PAMI_INVAL
The query has failed due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

PAMI_Context_trylock

Purpose

Tries to acquire an atomic lock on a communication context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_trylock(pami_context_t context
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_trylock(context, ierror)
integer context
integer ierror
```

Parameters

Input

context	Specifies the communication context.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to try to acquire an atomic lock on a communication context.

Restrictions

The lock cannot be assumed to be recursive or non-recursive.

Return values

PAMI_SUCCESS
: The lock has been acquired.

PAMI_EAGAIN
: The lock has not been acquired. Try again later.

PAMI_INVAL
: The context is not valid.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

PAMI_Context_trylock_advancev

Purpose

Advances the progress engine for multiple communication contexts.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_trylock_advancev(pami_context_t context[],
                                             size_t         count,
                                             size_t         maximum
                                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_trylock_advancev(context, count, maximum, ierror)
integer context
integer count
integer maximum
integer ierror
```

Parameters

Input

context Specifies an array of communication contexts.

count Specifies the number of communication contexts.

maximum
Specifies the maximum umber of internal poll iterations. Users cannot assume that events processed by other threads will cause this thread to return before "maximum" loop iterations.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: polling advance

Use this thread-safe subroutine to advance the progress engine for multiple communication contexts.

This subroutine can complete zero, one, or more outbound transfers. It can call dispatch handlers for incoming transfers. It can also call work event callbacks previously posted to a communication context.

This subroutine will return after the first poll iteration that results in a processed event on any context, or if, no events are processed, after polling for the maximum number of iterations.

Restrictions

This subroutine uses context locks for mutual exclusion. If you are using a different system, this will not be thread-safe.

Return values

PAMI_SUCCESS
An event has occurred and been processed.

PAMI_EAGAIN
No event has occurred.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Context_lock, PAMI_Context_trylock

PAMI_Context_unlock

Purpose

Releases an atomic lock on a communication context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_unlock(pami_context_t context
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_unlock(context, ierror)
integer context
integer ierror
```

Parameters

Input

context	Specifies the communication context.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to release an atomic lock on a communication context.

Restrictions

The lock cannot be assumed to be recursive or non-recursive.

Return values

PAMI_SUCCESS
The lock has been released.

PAMI_INVAL
The parameter is not valid.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: default-send-1.c, endpoint_table.c, lock.c, post-multithreaded.c, post-multithreaded-perf.c

PAMI_Context_update

Purpose

Updates the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Context_update(pami_context_t      *context,
                                  pami_configuration_t configuration[],
                                  size_t               num_configs
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_context_update(context, configuration, num_configs, ierror)
integer context
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

context Specifies the PAMI context.

configuration
Specifies the configuration attribute to update.

num_configs
Specifies the number of configuration elements.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to update the value of an attribute.

Return values

PAMI_SUCCESS
The update has completed successfully.

PAMI_INVAL
The update has failed due to non-valid parameters. This failure occurs if the subroutine tries to update a read-only attribute, for example.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

PAMI_Dispatch_query

Purpose

Queries the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Dispatch_query(pami_context_t       context,
                                  size_t               dispatch,
                                  pami_configuration_t configuration[],
                                  size_t               num_configs
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_dispatch_query(context, dispatch, configuration, num_configs, ierror)
integer context
integer dispatch
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

dispatch
Specifies the PAMI dispatch.

num_configs
Specifies the number of configuration elements.

Input/output

configuration
Specifies the configuration attribute to query.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to query the value of an attribute.

Return values

PAMI_SUCCESS
The query has completed successfully.

PAMI_INVAL
The query has failed due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: immediate_send_overflow.c, send_to_self_immed.c

Subroutines: PAMI_Dispatch_set, PAMI_Dispatch_update

PAMI_Dispatch_set

Purpose

Initializes the dispatch function for a dispatch identifier.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Dispatch_set(pami_context_t                 context,
                                size_t                         dispatch,
                                pami_dispatch_callback_function fn,
                                void *                         cookie,
                                pami_dispatch_hint_t           options
                               );
```

Fortran synopsis

```
include 'pamif.h'

pami_dispatch_set(context, dispatch, fn, cookie, options, ierror)
integer context
integer dispatch
integer fn
integer cookie
integer options
integer ierror
```

Parameters

Input

context Specifies the communication context.

dispatch
Specifies the dispatch identifier to initialize.

fn Specifies the dispatch receive function.

cookie Specifies the dispatch function cookie.

options Specifies the dispatch registration assertions.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: local, non-collective

Use this subroutine to initialize the dispatch function for a dispatch identifier. The maximum allowed dispatch identifier attribute, **PAMI_CONTEXT_DISPATCH_ID_MAX**, can be queried with the configuration interface.

Restrictions

There is no communication between tasks.

The user must not specify different hint assertions for the same client, context offset, and dispatch identifier on different tasks. However, there is no specific error check that will prevent specifying different hint assertions. The result of a communication operation using mismatched hint assertions is undefined.

Related information

C examples: adi.c, default-send.c, default-send-1.c, endpoint_table.c, immediate_send_overflow.c, long-header.c, long-header-matrix.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c, simple-send-immediate.c

Subroutines: PAMI_AMCollective_dispatch_set, PAMI_Context_query, PAMI_Dispatch_query, PAMI_Dispatch_update

PAMI_Dispatch_update

Purpose

Updates the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

 pami_result_t PAMI_Dispatch_update(pami_context_t       context,
                                    size_t               dispatch,
                                    pami_configuration_t configuration[],
                                    size_t               num_configs
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_dispatch_update(context, dispatch, configuration, num_configs, ierror)
integer context
integer dispatch
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

context Specifies the communication context.

dispatch
Specifies the PAMI dispatch.

configuration
Specifies the configuration attribute to update.

num_configs
Specifies the number of configuration elements.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to update the value of an attribute.

Return values

PAMI_SUCCESS
The update has completed successfully.

PAMI_INVAL
The update has failed due to non-valid parameters. This failure occurs if the subroutine tries to update a read-only attribute, for example.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Dispatch_query, PAMI_Dispatch_set

PAMI_Endpoint_create

Purpose

Constructs an endpoint to address communication destinations.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Endpoint_create(pami_client_t    client,
                                   pami_task_t      task,
                                   size_t           offset,
                                   pami_endpoint_t *endpoint
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_endpoint_create(client,task, offset, endpoint, ierror)
integer client
integer task
integer offset
integer endpoint
integer ierror
```

Parameters

Input

client Specifies an opaque destination client object.

task Specifies an opaque destination task object.

Input/output

offset Specifies the destination context offset.

Output

endpoint
Specifies an opaque endpoint object to initialize.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to construct an endpoint to address communication destinations.

Endpoints are opaque objects that are used to address a destination in a client and are constructed from a client, task, and context offset:

- The client is required to disambiguate the task and context offset identifiers, as these identifiers may be the same for multiple clients.
- The task is required to construct an endpoint to address the specific process that contains the destination context.

- The context offset is required to identify the specific context on the destination task. A context identifies a specific threading point on a task. The context offset identifies which threading point will process the communication operation.

Point-to-point communication operations, such as send, put, and get, address a destination with the opaque endpoint object. Collective communication operations are addressed by an opaque geometry object.

Applications can write an endpoint table in shared memory to save storage in an environment where multiple tasks of a client have access to the same shared memory area. It is the application's responsibility to allocate this shared memory area and coordinate the initialization and access of any shared data structures. This includes any opaque endpoint objects that can be created by one task and read by another task.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, bcast.c, bcast_subcomm.c, default-send-1.c, default-send.c, endpoint_table.c, immediate_send_overflow.c, long-header-matrix.c, long-header.c, rdma-matrix.c, scatter.c, scatterv.c, send_flood_perf.c, send_to_self.c, send_to_self_immed.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c, simple-send-immediate.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c

Subroutines: PAMI_Endpoint_createv, PAMI_Endpoint_query

PAMI_Endpoint_query

Purpose

Retrieves the task and context offset that are associated with an endpoint.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Endpoint_query(pami_endpoint_t  endpoint,
                                  pami_task_t     *task,
                                  size_t          *offset
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_endpoint_query(endpoint, task, offset, ierror)
integer endpoint
integer task
integer offset
integer ierror
```

Parameters

Input

endpoint
Specifies an opaque endpoint object.

Output

task Specifies an opaque destination task object.

offset Specifies the destination context offset.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to retrieve the task and context offset that are associated with an endpoint. The endpoint must already have been initialized.

If needed to optimize performance, this subroutine can be replaced with a generated macro that is specific to the installation platform.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: ambcast.c, endpoint_table.c

Subroutines: PAMI_Endpoint_create

PAMI_Error_text

Purpose

Provides a detailed description of the most recent PAMI result.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

size_t PAMI_Error_text(char  *string,
                       size_t length
                      );
```

Fortran synopsis

```
include 'pamif.h'

pami_error_text(string, length, ierror)
integer string
integer length
integer ierror
```

Parameters

Input

string	Specifies the character array to write the descriptive text.
length	Specifies the length of the character array.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to provide a detailed description of the most recent PAMI result. This result is specific to each thread. This subroutine returns the number of characters written into the array.

PAMI implementations can provide translated (**i18n**) text.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

PAMI_Extension_close

Purpose

Closes an extension.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Extension_close(pami_extension_t extension
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_extension_close(extension, ierror)
integer extension
integer ierror
```

Parameters

Input

extension
: Specifies the extension handle.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to close an extension.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Extension_open**, **PAMI_Extension_symbol**

PAMI_Extension_open

Purpose

Opens an extension for use by a client.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Extension_open(pami_client_t     client,
                                  const char       *name,
                                  pami_extension_t *extension
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_extension_open(client, name, extension, ierror)
integer client
integer name
integer extension
integer ierror
```

Parameters

Input

client Specifies the client handle.

name Specifies the unique extension name.

Output

extension
Specifies the extension handle.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to open an extension for use by a client. The extension can also be queried during compilation pre-processing using an **#ifdef** statement of the form **__pami_extension_***name***__**. For example:

```
#ifdef __pami_extension_1234__

// Use the "1234" extension
#endif
```

Return values

PAMI_SUCCESS
The specified extension is available and is implemented by the PAMI runtime library.

PAMI_ERROR
The specified extension was not initialized by the PAMI runtime library.

PAMI_UNIMPL
The specified extension is not implemented by the PAMI runtime library.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Extension_close**, **PAMI_Extension_symbol**

PAMI_Extension_symbol

Purpose

Queries an extension symbol.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

void* PAMI_Extension_symbol(pami_extension_t extension,
                            const char      *fn
                           );
```

Fortran synopsis

```
include 'pamif.h'

pami_extension_symbol (extension, fn, ierror)
integer extension
integer fn
integer ierror
```

Parameters

Input

extension
: Specifies the extension handle.

fn
: Specifies the extension symbol name.

Output

ierror
: Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to query an extension symbol. If the requested extension is available and implemented by the PAMI runtime library, this subroutine returns a pointer to the extension symbol. This can be a function pointer which can be used to call an extension function or a pointer to an extension variable. If the requested extension is not available, this subroutine returns a value of Null (in C) or PAMI_ADDR_NULL (in Fortran).

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

C example

```
typedef void (*pami_extension_1234_foo_fn) (pami_context_t context, size_t foo);
typedef void (*pami_extension_1234_bar_fn) (pami_context_t context, struct iovec ** iov);
typedef void pami_extension_1234_var_t;
pami_extension_1234_foo_fn pami_1234_foo =
   (pami_extension_1234_foo_fn) PAMI_Extension_symbol ("pami_extension_1234", "foo");
pami_extension_1234_bar_fn pami_1234_bar =
   (pami_extension_1234_bar_fn) PAMI_Extension_symbol ("pami_extension_1234", "bar");
pami_extension_1234_var_t * pami_1234_var =
   (pami_extension_1234_var_t *) PAMI_Extension_symbol ("pami_extension_1234", "var");
pami_context_t context = ...;
pami_extension_1234_foo (context, 0);
struct iovec iov[1024];
pami_extension_1234_bar (context, &iov);
*var = 1234;
```

Related information

Subroutines: **PAMI_Extension_close**, **PAMI_Extension_open**

PAMI_Fence_all

Purpose

Synchronizes all transfers between all endpoints on a context.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Fence_all(pami_context_t      context,
                             pami_event_function done_fn,
                             void               *cookie
                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_fence_all(context, done_fn, cookie, ierror)
integer context
integer done_fn
integer cookie
integer ierror
```

Parameters

Input

context Specifies the communication context.

done_fn
Specifies the event callback to call when the fence operation is complete.

cookie Specifies the event callback argument.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to synchronize all transfers between all endpoints on a context.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Fence_begin**, **PAMI_Fence_end**, **PAMI_Fence_endpoint**

PAMI_Fence_begin

Purpose

Begins a memory synchronization region.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Fence_begin(pami_context_t context
                               );
```

Fortran synopsis

```
include 'pamif.h'

pami_fence_begin(context, ierror)
integer context
integer ierror
```

Parameters

Input

context	Specifies the PAMI communication context.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to begin a memory synchronization region. A fence region is an area of program control on the local task that is bounded by the **PAMI_Fence_begin()** and **PAMI_Fence_end()** subroutines.

Restrictions

Do not call a fence operation outside of a fence region.

Do not begin a fence region inside an existing fence region. Fence regions cannot be nested.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Fence_all**, **PAMI_Fence_end**, **PAMI_Fence_endpoint**

PAMI_Fence_end

Purpose

Ends a memory synchronization region.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Fence_end(pami_context_t context
                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_fence_end(context, ierror)
integer context
integer ierror
```

Parameters

Input

context	Specifies the communication context.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to end a memory synchronization region. A fence region is an area of program control on the local task that is bounded by the **PAMI_Fence_begin()** and **PAMI_Fence_end()** subroutines.

Restrictions

Do not call a fence operation outside of a fence region.

Do not end a fence region outside of an existing fence region.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Fence_all**, **PAMI_Fence_begin**, **PAMI_Fence_endpoint**

PAMI_Fence_endpoint

Purpose

Synchronizes all transfers to an endpoint.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Fence_endpoint(pami_context_t      context,
                                  pami_event_function done_fn,
                                  void               *cookie,
                                  pami_endpoint_t     target
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_fence_endpoint(context, done_fn, cookie, target, ierror)
integer context
integer done_fn
integer cookie
integer target
integer ierror
```

Parameters

Input

context Specifies the communication context.

done_fn
Specifies the event callback to call when the fence operation is complete.

cookie Specifies the event callback argument.

target Specifies the endpoint to synchronize.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to synchronize all transfers to an endpoint.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Fence_all**, **PAMI_Fence_begin**, **PAMI_Fence_end**

PAMI_Geometry_algorithms_num

Purpose

Determines the number of algorithms available for a given operation in the "always work" list and the "under certain conditions" list.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_algorithms_num(pami_geometry_t  geometry,
                                           pami_xfer_type_t coll_type,
                                           size_t          *lists_lengths
                                          );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_algorithms_num (geometry, coll_type, lists_lengths, ierror)
integer geometry
integer coll_type
integer lists_lengths
integer ierror
```

Parameters

Input

geometry
Specifies an input geometry to be analyzed.

coll_type
Specifies the type of collective operation.

Input/output

lists_lengths
Specifies an array of two numbers representing all valid algorithms and optimized algorithms.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to determine the number of algorithms available for a given operation in the "always work" list and the "under certain conditions" list.

Return values

PAMI_SUCCESS
The number of algorithms is determined.

PAMI_INVAL
There is an error with the input parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: bcast_subcomm2.c, init_coll.c

Subroutines: PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_create_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_algorithms_query

Purpose

Queries the protocols and attributes for a set of algorithms.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_algorithms_query(pami_geometry_t  geometry,
                                             pami_xfer_type_t coll_type,
                                             pami_algorithm_t *algs0,
                                             pami_metadata_t  *mdata0,
                                             size_t            num0,
                                             pami_algorithm_t *algs1,
                                             pami_metadata_t  *mdata1,
                                             size_t            num1
                                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_algorithms_query(geometry, coll_type, algs0, mdata0,
                               num0, algs1, mdata1, num1, ierror)
integer geometry
integer coll_type
integer algs0
integer mdata0
integer num0
integer algs1
integer mdata1
integer num1
integer ierror
```

Parameters

Input

coll_type
Specifies the type of collective operation.

num0 Specifies the number of algorithms to fill in.

num1 Specifies the number of algorithms to fill in.

Input/output

algs0 Specifies an array of algorithms to query.

mdata0 Specifies a metadata array to be filled in if the algorithms are applicable. The value can be Null (in C) or PAMI_ADDR_NULL (in Fortran).

algs1 Specifies an array of algorithms to query.

mdata1 Specifies a metadata array to be filled in if the algorithms are applicable. The value can be Null (in C) or PAMI_ADDR_NULL (in Fortran).

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to fill in the protocols and attributes for a set of algorithms. The first set of lists are used to populate collectives that work under any condidtion. The second set of lists are used to populate collectives that the metadata must be checked before use.

Return values

PAMI_SUCCESS
The algorithm is applicable to the geometry.

PAMI_INVAL
There is an error in the input arguments or the algorithm is not applicable to the geometry.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: bcast_subcomm2.c, init_coll.c

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_create_tasklist, PAMI_Geometry_create_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_create_tasklist

Purpose

Initializes the geometry.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_create_tasklist (pami_client_t       client,
                                             size_t              context_offset,
                                             pami_configuration_t configuration[],
                                             size_t              num_configs,
                                             pami_geometry_t    *geometry,
                                             pami_geometry_t     parent,
                                             unsigned            id,
                                             pami_task_t        *tasks,
                                             size_t              task_count,
                                             pami_context_t      context,
                                             pami_event_function fn,
                                             void               *cookie
                                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_create_tasklist (client, context_offset, configuration, num_configs, geometry,
                               parent, id, tasks, task_count, context, fn, cookie, ierror)
integer client
integer context_offset
integer configuration
integer num_configs
integer geometry
integer parent
integer id
integer tasks
integer task_count
integer context
integer fn
integer cookie
integer ierror
```

Parameters

Input

client Specifies the PAMI client.

context_offset
Specifies the offset of the context to use for task-based collectives.

configuration
Specifies the list of configurable attributes and values.

num_configs
Specifies the number of configuration elements.

parent Specifies the parent geometry that contains all of the nodes in the task list.

id Specifies the identifier for this geometry that uniquely represents this geometry (if tasks overlap).

tasks Specifies an array of tasks to build the geometry list. The user must keep the task list in memory for the duration of the geometry's existence.

task_count
Specifies the number of tasks that are participating in the geometry.

context Specifies the context to which to deliver the asynchronous callback.

fn Specifies the event function to call when the geometry has been created.

cookie Specifies the user cookie to deliver with the callback.

Output

geometry
Specifies the opaque geometry object to initialize.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to initialize the geometry. A synchronizing operation takes place during **geometry_initialize** on the parent geometry. If the output geometry's *geometry* pointer has a value of Null (in C) or PAMI_ADDR_NULL (in Fortran), no geometry is created. However, all nodes in the parent must participate in the **geometry_initialize** operation, even if they do not create a geometry.

If a geometry is created without a parent geometry (the parent is set to **PAMI_NULL_GEOMETRY**), an "immediate" geometry is created. In this case, the new geometry is created and synchronized. However, the new geometry cannot take advantage of optimized collectives from the parent in the creation of the new geometry. This kind of geometry creation might not be as optimal as when a parent has been provided.

A unique geometry ID is required to give each task described by a geometry a unique communication context/channel to the other tasks in that geometry. Many higher-level APIs have code to manage the creation of unique geometry IDs. PAMI defers the unique geometry id assignment to these higher-level APIs.

This subroutine takes a context_offset into the array of contexts created at PAMI_Context_createv. There may be more than one context created per task, so this offset specifies a set of contexts that will be participating in the geometry. All tasks in the create API must use the same context_offset. This effectively selects a single endpoint (associated with this "primary" context) per task that will be a participant in the collective communication.

Note that this does not effect the advance rules, which state that the user must advance all contexts unless the multicontext hint is disabled for the collective operation.

Restrictions

It is an error to post a collective via PAMI_Collective() to a context that is not specified by the client/context_offset pair.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_create_taskrange

Purpose

Initializes the geometry.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_create_taskrange (pami_client_t         client,
                                               size_t                context_offset,
                                               pami_configuration_t  configuration[],
                                               size_t                num_configs,
                                               pami_geometry_t      *geometry,
                                               pami_geometry_t       parent,
                                               unsigned              id,
                                               pami_geometry_range_t *task_slices,
                                               size_t                slice_count,
                                               pami_context_t        context,
                                               pami_event_function   fn,
                                               void                 *cookie
                                              );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_create_taskrange (client, context_offset, configuration, num_configs, geometry,
                                parent, id, task_slices, slice_count, context, fn, cookie,
                                ierror)
integer client
integer context_offset
integer configuration
integer num_configs
integer geometry
integer parent
integer id
integer task_slices
integer slice_count
integer context
integer fn
integer cookie
integer ierror
```

Parameters

Input

client Specifies the PAMI client.

context_offset
Specifies the offset of the context to use for task-based collectives.

configuration
Specifies the list of configurable attributes and values.

num_configs
Specifies the number of configuration elements.

parent Specifies the parent geometry that contains all of the nodes in the task list.

id Specifies the identifier for this geometry that uniquely represents this geometry (if tasks overlap).

task_slices
Specifies an array of node slices that are participating in the geometry. The user must keep the array of slices in memory for the duration of the geometry's existence.

slice_count
Specifies the number of task slices that are participating in the geometry.

context Specifies the context to which to deliver the asynchronous callback.

fn Specifies the event function to call when the geometry has been created.

cookie Specifies the user cookie to deliver with the callback.

Output

geometry
Specifies the opaque geometry object to initialize.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to initialize the geometry. A synchronizing operation takes place during **geometry_initialize** on the parent geometry. If the output geometry's *geometry* pointer has a value of Null (in C) or PAMI_ADDR_NULL (in Fortran), no geometry is created. However, all nodes in the parent must participate in the **geometry_initialize** operation, even if they do not create a geometry.

If a geometry is created without a parent geometry (the parent is set to **PAMI_NULL_GEOMETRY**), an "immediate" geometry is created. In this case, the new geometry is created and synchronized. However, the new geometry cannot take advantage of optimized collectives from the parent in the creation of the new geometry. This kind of geometry creation might not be as optimal as when a parent has been provided.

A unique geometry ID is required to give each task described by a geometry a unique communication context/channel to the other tasks in that geometry. Many higher-level APIs have code to manage the creation of unique geometry IDs. PAMI defers the unique geometry id assignment to these higher-level APIs.

This subroutine takes a context_offset into the array of contexts created at PAMI_Context_createv. There may be more than one context created per task, so this offset specifies a set of contexts that will be participating in the geometry. All tasks in the create API must use the same context_offset. This effectively selects a single endpoint (associated with this "primary" context) per task that will be a participant in the collective communication.

Note that this does not effect the advance rules, which state that the user must advance all contexts unless the multicontext hint is disabled for the collective operation.

Restrictions

It is an error to post a collective via PAMI_Collective() to a context that is not specified by the client/context_offset pair.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: bcast_subcomm2.c

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_destroy

Purpose

Frees any memory allocated inside of a geometry.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_destroy(pami_client_t       client,
                                     pami_geometry_t    *geometry,
                                     pami_context_t      context,
                                     pami_event_function fn,
                                     void               *cookie
                                    );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_destroy(client, geometry, context, fn, cookie, ierror)
integer client
integer geometry
integer context
integer fn
integer cookie
integer ierror
```

Parameters

Input

client Specifies the PAMI client.

geometry
Specifies the geometry object to free.

context Specifies the context to which to deliver the asynchronous callback.

fn Specifies the event function to call when the geometry has been destroyed.

cookie Specifies the user cookie to deliver with the callback.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: non-blocking

Use this subroutine to free any memory allocated inside of a geometry.

PAMI_Geometry_destroy frees any internal resources that are allocated during a geometry create routine. This could involve synchronization with other tasks in the geometry to free the resource, which would require that a callback be run before completion. Synchronization with other tasks can not be assumed, however, as the operation to free the geometry might be local only, and dependent on the hardware and networks used for communication.

After the callback function has been called, the user is free to reuse the geometry ID of the geometry that is being freed, subject to the geometry ID creation rules. It is valid to assume the geometry has been destroyed in the callback function.

A valid client, geometry pointer, and context should be provided to this subroutine. The event function can be Null, but the user cannot re-use the geometry ID (provided to the geometry create routine) for the lifecycle of the client. If the event function is Null, there is no way for the user to determine when the resources can be reused.

The geometry handle will be changed to a non-valid value so that it is clearly destroyed.

Restrictions

It is not valid to destroy geometry 0 (the world geometry).

Return values

PAMI_SUCCESS
: The memory was freed.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_taskrange, PAMI_Geometry_query, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_query

Purpose

Queries the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_query(pami_geometry_t      geometry,
                                  pami_configuration_t configuration[],
                                  size_t               num_configs
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_query(geometry, configuration, num_configs, ierror)
integer geometry
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

geometry
Specifies the PAMI geometry.

num_configs
Specifies the number of configuration elements.

Input/output

configuration
Specifies the configuration attribute to query.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to query the value of an attribute.

Return values

PAMI_SUCCESS
The query has completed successfully.

PAMI_INVAL
The query has failed due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_update, PAMI_Geometry_world

PAMI_Geometry_update

Purpose

Updates the value of an attribute.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_update(pami_geometry_t      geometry,
                                   pami_configuration_t configuration[],
                                   size_t               num_configs,
                                   pami_context_t       context,
                                   pami_event_function  fn,
                                   void                *cookie
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_update(geometry, configuration, num_configs, context, fn, cookie, ierror)
integer geometry
integer configuration
integer num_configs
integer context
integer fn
integer cookie
integer ierror
```

Parameters

Input

geometry
Specifies the PAMI geometry.

configuration
Specifies the configuration attribute to update.

num_configs
Specifies the number of configuration elements.

context Specifies the context to which to deliver the asynchronous callback.

fn Specifies the event function to call when the geometry has been created.

cookie Specifies the user cookie to deliver with the callback.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to update the value of an attribute. The configuration variable must be set collectively.

Restrictions

Changing a geometry configuration attribute could fundamentally alter the geometry. Any saved knowledge (algorithm lists, for example) must be discarded and re-queried after a call to this subroutine.

Return values

PAMI_SUCCESS
The update has completed successfully.

PAMI_INVAL
The update has failed due to non-valid parameters. This failure occurs if the subroutine tries to update a read-only attribute, for example.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_world

PAMI_Geometry_world

Purpose

Returns a pointer to the world geometry.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Geometry_world (pami_client_t    client,
                                   pami_geometry_t *world_geometry
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_geometry_world (client, world_geometry, ierror)
integer client
integer world_geometry
integer ierror
```

Parameters

Input

client Specifies the PAMI client.

world_geometry
Specifies the world geometry object.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: collective communication

Use this subroutine to return a pointer to the "world geometry". The world geometry is created at client creation time. It contains a representation of all the tasks associated with the client.

The world geometry has a geometry ID of 0, which means that the PAMI user should not create any new geometries using an ID of 0. This geometry represents context offset 0 on all tasks in the current job.

The world geometry cannot be destroyed by the user, and is cleaned up at PAMI_Client_destroy time.

The world geometry is scoped to the client object, meaning each client geometry will have a world geometry associated with it.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: bcast_subcomm2.c, init_coll.c

Subroutines: PAMI_Geometry_algorithms_num, PAMI_Geometry_algorithms_query, PAMI_Geometry_create_tasklist, PAMI_Geometry_taskrange, PAMI_Geometry_destroy, PAMI_Geometry_query, PAMI_Geometry_update

PAMI_Get

Purpose

Performs a one-sided get operation for simple contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Get(pami_context_t     context,
                        pami_get_simple_t *parameters
                       );
```

Fortran synopsis

```
include 'pamif.h'

pami_get(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the simple get input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a one-sided get operation for a simple contiguous data transfer.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_get_func.c

Subroutines: **PAMI_Get_typed**, **PAMI_Put**, **PAMI_Put_typed**

PAMI_Get_typed

Purpose

Performs a one-sided get operation for typed non-contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Get_typed(pami_context_t    context,
                             pami_get_typed_t *parameters
                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_get_typed(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the PAMI communication context.

parameters
Specifies the typed get input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine perform a one-sided get operation for a typed non-contiguous data transfer.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Get**, **PAMI_Put**, **PAMI_Put_typed**

PAMI_Memregion_create

Purpose

Creates a local memory region for one-sided operations.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Memregion_create(pami_context_t    context,
                                    void             *address,
                                    size_t            bytes_in,
                                    size_t           *bytes_out,
                                    pami_memregion_t *memregion
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_memregion_create(context, address, bytes_in, bytes_out, memregion, ierror)
integer context
integer address
integer bytes_in
integer bytes_out
integer memregion
integer ierror
```

Parameters

Input

context Specifies the communication context.

address Specifies the base virtual address of the memory region.

bytes_in
Specifies the number of bytes requested.

Output

bytes_out
Specifies the number of bytes granted.

memregion
Specifies the memory region object to initialize.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to create a local memory region for one-sided operations. Using a send message, the local memory region can be transferred to a remote task, so the remote task can perform one-sided operations with this local task. The actual number of bytes pinned from the start of the buffer is returned in the *bytes_out* parameter. The memory region must be freed using the **PAMI_Memregion_destroy()** subroutine.

Memory regions can overlap. When one of the overlapping regions is destroyed, any remaining overlapping memory regions are still usable.

If this subroutine fails, the memory region does not need to be freed using the **PAMI_Memregion_destroy()** subroutine.

Return values

PAMI_SUCCESS
: The entire memory region, or a portion of the memory region, was pinned.

PAMI_EAGAIN
: The memory region was not pinned due to an unavailable resource.

PAMI_ERROR
: The memory region was not pinned.

PAMI_INVAL
: A non-valid parameter value was specified.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_rget_func.c, simple_rput_func.c

Subroutines: **PAMI_Memregion_destroy**, **PAMI_Rget**, **PAMI_Rget_typed**, **PAMI_Rput**, **PAMI_Rput_typed**

PAMI_Memregion_destroy

Purpose

Destroys a local memory region for one-sided operations.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Memregion_destroy(pami_context_t    context,
                                     pami_memregion_t *memregion
                                    );
```

Fortran synopsis

```
include 'pamif.h'

pami_memregion_destroy(context, memregion, ierror)
integer context
integer memregion
integer ierror
```

Parameters

Input

context Specifies the communication context.

memregion
Specifies the memory region object to destroy.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to destroy a local memory region for one-sided operations. The memory region object will be changed to an non-valid value so that it is clearly destroyed.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_rget_func.c, simple_rput_func.c

Subroutines: **PAMI_Memregion_create**, **PAMI_Rget**, **PAMI_Rget_typed**, **PAMI_Rput**, **PAMI_Rput_typed**

PAMI_Purge

Purpose

Cleans up local resources to an endpoint in preparation for task shutdown or checkpoint.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Purge(pami_context_t   context,
                         pami_endpoint_t *dest,
                         size_t           count
                        );
```

Fortran synopsis

```
include 'pamif.h'

pami_purge(context, dest, count, ierror)
integer context
integer dest
integer count
integer ierror
```

Parameters

Input

context	Specifies the communication context.
dest	Specifies an array of destination endpoints to which to close connections.
count	Specifies the number of endpoints in the *dest* array.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to clean up local resources to an endpoint in preparation for task shutdown or checkpoint. It is the user's responsibility to make sure that all communication has been quiesced to and from the destination using a fence call and synchronization.

Restrictions

It is *not* valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Resume**

PAMI_Put

Purpose

Performs a one-sided put operation for simple contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Put(pami_context_t     context,
                        pami_put_simple_t *parameters
                       );
```

Fortran synopsis

```
include 'pamif.h'

pami_put(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the simple put input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a one-sided put operation for a simple contiguous data transfer.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_put_func.c

Subroutines: **PAMI_Get**, **PAMI_Get_typed**, **PAMI_Put_typed**

PAMI_Put_typed

Purpose

Performs a one-sided put operation for typed non-contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Put_typed(pami_context_t    context,
                             pami_put_typed_t *parameters
                            );
```

Fortran synopsis

```
include 'pamif.h'

pami_put_typed(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the typed put input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a one-sided put operation for a typed non-contiguous data transfer.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Get**, **PAMI_Get_typed**, **PAMI_Put**

PAMI_Resume

Purpose

Sets up local resources to an endpoint in preparation for task restart or creation.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Resume(pami_context_t   context,
                          pami_endpoint_t *dest,
                          size_t           count
                         );
```

Fortran synopsis

```
include 'pamif.h'

pami_resume(context, dest, count, ierror)
integer context
integer dest
integer count
integer ierror
```

Parameters

Input

context	Specifies the communication context.
dest	Specifies an array of destination endpoints to which to resume connections.
count	Specifies the number of endpoints in the *dest* array.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to set up local resources to an endpoint in preparation for task restart or creation.

Restrictions

It is *not* valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Purge**

PAMI_Rget

Purpose

Performs a simple get operation for one-sided contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Rget(pami_context_t      context,
                        pami_rget_simple_t *parameters
                       );
```

Fortran synopsis

```
include 'pamif.h'

pami_rget(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the structure of the input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a simple get operation for one-sided contiguous data transfers.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_rget_func.c

Subroutines: **PAMI_Memregion_create**, **PAMI_Memregion_destroy**, **PAMI_Rget_typed**, **PAMI_Rput**, **PAMI_Rput_typed**

PAMI_Rget_typed

Purpose

Performs a get operation for datatype-specific one-sided data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Rget_typed(pami_context_t     context,
                              pami_rget_typed_t *parameters
                             );
```

Fortran synopsis

```
include 'pamif.h'

pami_rget_typed(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the structure of the input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a get operation for datatype-specific one-sided data transfers.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Memregion_create**, **PAMI_Memregion_destroy**, **PAMI_Rget**, **PAMI_Rput**, **PAMI_Rput_typed**

PAMI_Rmw

Purpose

Performs an atomic read-modify-write operation at a remote memory location.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Rmw(pami_context_t context,
                        pami_rmw_t    *parameters
                       );

typedef struct
{
  pami_endpoint_t     dest;      /* Destination endpoint */
  pami_send_hint_t    hints;     /* Hints for sending the message */
  void               *cookie;    /* Argument to all event callbacks */
  pami_event_function done_fn;   /* Atomic operation completion event */
  void               *local;     /* Local (fetch) transfer virtual address */
  void               *remote;    /* Remote (source) transfer virtual address */
  void               *value;     /* Operation input local data value location */
  void               *test;      /* Operation input local test value location */
  pami_atomic_t       operation; /* Read-modify-write operation */
  pami_type_t         type;
} pami_rmw_t;
```

Fortran synopsis

```
include 'pamif.h'

pami_rmw(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the read-modify-write input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication (non-blocking)

Use this subroutine to perform an atomic read-modify-write operation at a remote memory location.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

The following **pami_type_t**`types are valid for read-modify-write operations:
- PAMI_TYPE_SIGNED_INT
- PAMI_TYPE_UNSIGNED_INT
- PAMI_TYPE_SIGNED_LONG
- PAMI_TYPE_UNSIGNED_LONG
- PAMI_TYPE_SIGNED_LONG_LONG (only supported on 64-bit platforms)
- PAMI_TYPE_UNSIGNED_LONG_LONG (only supported on 64-bit platforms)

The **pami_atomic_t**`data structure follows:

```
typedef enum
{
  PAMI_ATOMIC_FETCH    = (0x1 << 0),
  PAMI_ATOMIC_COMPARE  = (0x1 << 1),
  PAMI_ATOMIC_SET      = (0x1 << 2),
  PAMI_ATOMIC_ADD      = (0x2 << 2),
  PAMI_ATOMIC_OR       = (0x3 << 2),
  PAMI_ATOMIC_AND      = (0x4 << 2),
  PAMI_ATOMIC_XOR      = (0x5 << 2),
  PAMI_ATOMIC_FETCH_SET = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_SET),
  PAMI_ATOMIC_FETCH_ADD = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_ADD),
  PAMI_ATOMIC_FETCH_OR  = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_OR),
  PAMI_ATOMIC_FETCH_AND = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_AND),
  PAMI_ATOMIC_FETCH_XOR = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_XOR),
  PAMI_ATOMIC_COMPARE_SET = (PAMI_ATOMIC_COMPARE | PAMI_ATOMIC_SET),
  PAMI_ATOMIC_COMPARE_ADD = (PAMI_ATOMIC_COMPARE | PAMI_ATOMIC_ADD),
  PAMI_ATOMIC_COMPARE_OR  = (PAMI_ATOMIC_COMPARE | PAMI_ATOMIC_OR),
  PAMI_ATOMIC_COMPARE_AND = (PAMI_ATOMIC_COMPARE | PAMI_ATOMIC_AND),
  PAMI_ATOMIC_COMPARE_XOR = (PAMI_ATOMIC_COMPARE | PAMI_ATOMIC_XOR),
  PAMI_ATOMIC_FETCH_COMPARE_SET = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_COMPARE_SET),
  PAMI_ATOMIC_FETCH_COMPARE_ADD = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_COMPARE_ADD),
  PAMI_ATOMIC_FETCH_COMPARE_OR  = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_COMPARE_OR),
  PAMI_ATOMIC_FETCH_COMPARE_AND = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_COMPARE_AND),
  PAMI_ATOMIC_FETCH_COMPARE_XOR = (PAMI_ATOMIC_FETCH | PAMI_ATOMIC_COMPARE_XOR),
} pami_atomic_t;
```

Restrictions

All read-modify-write operations are *unordered* relative to all other data transfer operations, including other read-modify-write operations.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

C example

```
rmw.type=PAMI_TYPE_SIGNED_INT; rmw.operation=PAMI_ATOMIC_FETCH_AND
    "32-bit signed integer fetch-then-and operation"

     int *local, *remote, *value, *test;
     *local = *remote; *remote &= *value;

rmw.type=PAMI_TYPE_UNSIGNED_LONG; rmw.operation=PAMI_ATOMIC_COMPARE_XOR
    "native word sized signed integer compare-and-xor operation"

     unsigned long *local, *remote, *value, *test;
     (*remote == test) ? *remote ^= *value;

rmw.type=PAMI_TYPE_SIGNED_LONG_LONG; rmw.operation=PAMI_ATOMIC_FETCH_COMPARE_OR
    "64-bit signed integer fetch-then-compare-and-or operation"

     long long *local, *remote, *value, *test;
     *local = *remote; (*remote == *test) ? *remote |= *value;
```

PAMI_Rput

Purpose

Performs a simple put operation for one-sided contiguous data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Rput(pami_context_t      context,
                        pami_rput_simple_t *parameters
                       );
```

Fortran synopsis

```
include 'pamif.h'

pami_rput(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the structure of the input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a simple put operation for one-sided contiguous data transfers.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: simple_rput_func.c

Subroutines: **PAMI_Memregion_create**, **PAMI_Memregion_destroy**, **PAMI_Rget**, **PAMI_Rget_typed**, **PAMI_Rput_typed**

PAMI_Rput_typed

Purpose

Performs a put operation for datatype-specific one-sided data transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Rput_typed(pami_context_t     context,
                               pami_rput_typed_t *parameters
                              );
```

Fortran synopsis

```
include 'pamif.h'

pami_rput_typed(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the structure of the input parameters.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication

Use this subroutine to perform a put operation for a datatype-specific one-sided data transfer.

It is valid to specify the destination endpoint associated with the communication context used to issue the operation.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Memregion_create**, **PAMI_Memregion_destroy**, **PAMI_Rget**, **PAMI_Rget_typed**, **PAMI_Rput**

PAMI_Send

Purpose

Performs an active message send operation for contiguous data.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Send(pami_context_t context,
                         pami_send_t   *parameters
                        );
```

Fortran synopsis

```
include 'pamif.h'

pami_send(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the send simple parameter structure.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication (non-blocking)

Use this subroutine to perform an active message send operation for contiguous data. A low-latency send operation can be enhanced by using a dispatch ID, which is set with the **pami_dispatch_hint_t::recv_immediate** hint bit enabled. This hint asserts that all receive operations with a dispatch ID will not exceed a certain limit. The **PAMI_DISPATCH_RECV_IMMEDIATE_MAX** implementation configuration attribute defines the maximum size of data buffers that can be completely received with a single dispatch callback. Typically, this limit is associated with a network resource attribute, such as a packet size.

The **pami_send_immediate_t::dispatch** identifier must be registered on the sending context, using **PAMI_Dispatch_set()**, prior to the send operation.

It is valid to specify the endpoint associated with the communication context used to issue the operation as the destination for the transfer.

It is safe for the application to deallocate, or otherwise alter, the **pami_send_t** parameter structure after this function returns.

Restrictions

It is not safe for the application to deallocate, or otherwise alter, the memory locations specified by **pami_send_immediate_t::header** and **pami_send_immediate_t::data** until **pami_send_event_t::local_fn** is called.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, default-send.c, default-send-1.c, endpoint_table.c, long-header.c, long-header-matrix.c, rdma-matrix.c, send_flood_perf.c, send_to_self.c, send_to_self_perf.c, send_unexpected_func.c, shmem-matrix.c

Datatypes: **pami_send_hint_t**

Subroutines: **PAMI_Dispatch_query**, **PAMI_Send_immediate**, **PAMI_Send_typed**

PAMI_Send_immediate

Purpose

Performs an immediate active message send operation for small contiguous data.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Send_immediate(pami_context_t        context,
                                  pami_send_immediate_t *parameters
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_send_immediate(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the send parameter structure.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication (blocking)

Use this subroutine to perform an immediate active message send operation for small contiguous data. A blocking send operation is only valid for small data buffers. The PAMI_DISPATCH_SEND_IMMEDIATE_MAX implementation configuration attribute defines the upper bounds for the size of data buffers, including header data, that can be sent with this subroutine. This subroutine returns an error if a data buffer larger than PAMI_DISPATCH_SEND_IMMEDIATE_MAX is attempted.

This subroutine provides a low-latency send operation that can be optimized by the specific PAMI implementation. If network resources are immediately available, the send data will be injected directly into the network. If resources are not available, the specific PAMI implementation can internally buffer the send parameters and data until network resources are available to complete the transfer. In either case, the send operation will return immediately, no "done" callback is called, and the operation is considered complete.

The low-latency send operation can be further enhanced by using a dispatch ID, which is set with the **pami_dispatch_hint_t::recv_immediate** hint bit enabled. This hint asserts that all receive operations with a dispatch ID will not exceed a certain limit.

The PAMI_DISPATCH_SEND_IMMEDIATE_MAX implementation configuration attribute defines the maximum size of data buffers that can be completely received with a single dispatch callback. Typically, this limit is associated with a network resource attribute, such as a packet size.

The **pami_send_immediate_t::dispatch** identifier must be registered on the sending context, using **PAMI_Dispatch_set()**, prior to the send operation.

It is valid to specify the endpoint associated with the communication context used to issue the operation as the destination for the transfer.

It is safe for the application to deallocate, or otherwise alter, the the send parameter structure and the memory locations specified by **pami_send_immediate_t::header** and **pami_send_immediate_t::data** after this function returns.

Return values

PAMI_SUCCESS
: The request has been accepted.

PAMI_EAGAIN
: The request could not be satisfied due to unavailable network resources and the request data could not be queued for later processing due to the value of the **pami_dispatch_hint_t::queue_immediate** hint for this dispatch identifier.

PAMI_INVAL
: The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: adi.c, immediate_send_overflow.c, send_flood_perf.c, send_to_self_immed.c, simple_get_func.c, simple_put_func.c, simple_rget_func.c, simple_rput_func.c, simple-send-immediate.c,

Datatypes: **pami_send_hint_t**

Subroutines: **PAMI_Dispatch_query**, **PAMI_Send**, **PAMI_Send_typed**

PAMI_Send_typed

Purpose

Performs an active message send operation for non-contiguous typed data.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Send_typed(pami_context_t     context,
                              pami_send_typed_t *parameters
                             );
```

Fortran synopsis

```
include 'pamif.h'

pami_send_typed(context, parameters, ierror)
integer context
integer parameters
integer ierror
```

Parameters

Input

context Specifies the communication context.

parameters
Specifies the send typed parameter structure.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Type of call: point-to-point communication (non-blocking)

Use this subroutine to perform an active message send operation for non-contiguous typed data. This subroutine transfers data according to a predefined data memory layout, or type, to the remote task. The data is transferred as a byte stream that the remote task can receive in a different format, such as a contiguous buffer, the same predefined type, or a different predefined type.

Return values

PAMI_SUCCESS
The request has been accepted.

PAMI_INVAL
The request has been rejected due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: **PAMI_Send**, **PAMI_Send_immediate**

PAMI_Type_add_simple

Purpose

Appends simple contiguous buffers to an existing type identifier.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_add_simple(pami_type_t type,
                                   size_t      bytes,
                                   size_t      offset,
                                   size_t      count,
                                   size_t      stride
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_add_simple(type, bytes, offset, count, stride, ierror)
integer type
integer bytes
integer offset
integer count
integer stride
integer ierror
```

Parameters

Input

bytes Specifies the number of bytes of each contiguous buffer.

offset Specifies the offset from the cursor to place the buffers.

count Specifies the number of buffers.

stride Specifies the stride between buffers.

Input/output

type Specifies the type identifier to be modified.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to append simple contiguous buffers to an existing type identifier. A cursor, starting from 0, tracks the placement of buffers in a type. This pseudocode places simple buffers:

```
cursor += offset;
while (count--) {
  Put a contiguous buffer of bytes at the cursor;
   cursor += stride;
}
```

If *count* is 0, this function simply moves the cursor. It is valid to move the cursor forward or backward. It is also valid to place overlapping buffers in a type, but the overlapping buffers hold undefined data when such a type is used in data manipulation.

Return values

PAMI_SUCCESS
The buffers are added to the type.

PAMI_ENOMEM
Out of memory.

PAMI_INVAL
A completed type cannot be modified.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_add_typed

Purpose

Appends typed buffers to an existing type identifier.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_add_typed(pami_type_t type,
                                  pami_type_t subtype,
                                  size_t      offset,
                                  size_t      count,
                                  size_t      stride
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_add_typed(type, subtype, offset, count, stride, ierror)
integer type
integer subtype
integer offset
integer count
integer stride
integer ierror
```

Parameters

Input

subtype
Specifies the type of each typed buffer.

offset Specifies the offset from the cursor to place the buffers.

count Specifies the number of buffers.

stride Specifies the stride between buffers.

Input/output

type Specifies the type identifier to be modified.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to append typed buffers to an existing type identifier. A cursor, starting from 0, tracks the placement of buffers in a type. This pseudocode places typed buffers:

```
cursor += offset;
while (count--) {
  Put a typed buffer of subtype at the cursor;
  cursor += stride;
}
```

The cursor movement in *subtype* has no impact to the cursor of *type*. If *count* is 0, this function simply moves the cursor. It is valid to move the cursor forward or backward. It is also valid to place overlapping buffers in a type but the overlapping buffers hold undefined data when such a type is used in data manipulation.

Restrictions

Do not append an incomplete type to another type.

Return values

PAMI_SUCCESS
The buffers are added to the type.

PAMI_ENOMEM
Out of memory.

PAMI_INVAL
A completed type cannot be modified or an incomplete subtype cannot be added.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_complete

Purpose

Completes the type identifier.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_complete(pami_type_t type,
                                  size_t      atom_size
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_complete(type, atom_size, ierror)
integer type
integer atom_size
integer ierror
```

Parameters

Input

type Specifies the type identifier to be completed.

atom_size
Specifies the atom size of the type.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to complete the type identifier. The atom size of a type must divide the size of any contiguous buffer that is described by the type. An atom size of 1 is valid for any type.

Restrictions

Do not modify a type layout after it has been completed.

Return values

PAMI_SUCCESS
The type is complete.

PAMI_INVAL
The atom size is not valid.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_create

Purpose

Creates a new type for noncontiguous transfers.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_create(pami_type_t *type
                               );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_create(type, ierror)
integer type
integer ierror
```

Parameters

Input

Output

type Specifies the type identifier to be created.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to create a new type for a noncontiguous transfer.

Return values

PAMI_SUCCESS
The type is created.

PAMI_ENOMEM
Out of memory.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_deserialize

Purpose

Reconstructs a new type from a serialized type object.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_deserialize(pami_type_t *type,
                                    void        *address,
                                    size_t       size
                                   );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_deserialize(type, address, size, ierror)
integer type
integer address
integer size
integer ierror
```

Parameters

Input

address Specifies the address of the serialized type object.

size Specifies the size of the serialized type object.

Output

type Specifies the type identifier to be created.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to reconstruct a new type from a serialized type object. Successful reconstruction completes the new type. The new type does not depend on the memory of the serialized type object. A reconstructed type can be destroyed using the **PAMI_Type_destroy** subroutine.

Return values

PAMI_SUCCESS
The reconstruction is successful.

PAMI_ENOMEM
Out of memory.

PAMI_INVAL
The serialized type object is corrupted.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_destroy

Purpose

Destroys a type.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_destroy(pami_type_t *type
                                );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_destroy(type, ierror)
integer type
integer ierror
```

Parameters

Input

type	Specifies the type identifier to be destroyed.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to destroy a type. The type handle is changed to a non-valid value so that it is clearly destroyed.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_query, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_query

Purpose

Queries the attributes of a type.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_query(pami_type_t          *type,
                              pami_configuration_t  configuration[],
                              size_t                num_configs
                             );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_query(type, configuration, num_configs, ierror)
integer type
integer configuration
integer num_configs
integer ierror
```

Parameters

Input

type Specifies the type to query.

configuration
Specifies the configuration attributes to query.

num_configs
Specifies the number of configuration elements.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to query the attributes of a type. The type being queried must have completed.

Return values

PAMI_SUCCESS
The update has completed successfully.

PAMI_INVAL
The update has failed due to non-valid parameters.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_serialize, PAMI_Type_transform_data

PAMI_Type_serialize

Purpose

Serializes a type.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_serialize(pami_type_t *type,
                                  void      **address,
                                  size_t       size
                                  );
```

Fortran synopsis

```
include 'pamif.h'

pami_type_serialize(type, address, size, ierror)
integer type
integer address
integer size
integer ierror
```

Parameters

Input

type Specifies the type identifier to be serialized.

Output

address Specifies the address of the serialized type object.

size Specifies the size of the serialized type object.

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to serialize a type and retrieve the address and the size of a serialized type object, which can be copied or transferred like normal data. A serialized type object can be reconstructed into a type using the **PAMI_Type_deserialize** subroutine. The serialization is internal to PAMI and not into user-allocated memory. Serializing an already-serialized type retrieves the address and the size of the serialized type object. A PAMI implementation can choose to keep the internal representation of a type always serialized. Otherwise, it needs to handle serialization while a type is in use.

Return values

PAMI_SUCCESS
The serialization is successful.

PAMI_ENOMEM
Out of memory.

PAMI_INVAL
The type is not valid.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_transform_data

PAMI_Type_transform_data

Purpose

Transforms typed data between buffers in the same address space.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

pami_result_t PAMI_Type_transform_data (void              *src_addr,
                                         pami_type_t        src_type,
                                         size_t             src_offset,
                                         void              *dst_addr,
                                         pami_type_t        dst_type,
                                         size_t             dst_offset,
                                         size_t             size,
                                         pami_data_function data_fn,
                                         void              *cookie
                                        );
```

Fortran synopsis

```
include 'pamif.h'

 pami_result_t PAMI_Type_transform_data (src_addr, src_type, src_offset, dst_addr, dst_type,
                                          dst_offset, size, data_fn, cookie, ierror)
integer src_addr
integer src_type
integer src_offset
integer dst_addr
integer dst_type
integer dst_offset
integer size
integer data_fn
integer cookie
integer ierror
```

Parameters

Input

src_addr
: Specifies the source buffer address.

src_type
: Specifies the source datatype.

src_offset
: Specifies the starting offset of the source datatype.

dst_addr
: Specifies the destination buffer address.

dst_type
: Specifies the destination datatype.

dst_offset
: Specifies the starting offset of the destination datatype.

size Specifies the amount of data to transform.

data_fn
Specifies the function to transform the data.

cookie Specifies the argument to data function.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to transform typed data between buffers in the same address space.

Return values

PAMI_SUCCESS
The operation has completed successfully.

PAMI_INVAL
The operation has failed due to non-valid parameters, that is, incomplete types.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

Subroutines: PAMI_Type_add_simple, PAMI_Type_add_typed, PAMI_Type_complete, PAMI_Type_create, PAMI_Type_deserialize, PAMI_Type_destroy, PAMI_Type_query, PAMI_Type_serialize

PAMI_Wtime

Purpose

Returns an elapsed time on the calling processor.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

double PAMI_Wtime(pami_client_t client
                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_wtime(client, ierror)
integer client
integer ierror
```

Parameters

Input

client	Specifies the client handle.

Output

ierror	Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to return an elapsed time on the calling processor. This subroutine has the same definition as the **MPI_Wtime** subroutine. This subroutine returns the time in seconds since an arbitrary time in the past.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: time.c

Subroutines: **MPI_Wtime**, **PAMI_Wtimebase**

PAMI_Wtimebase

Purpose

Returns the number of "cycles" that have elapsed on the calling processor.

Library

PAMI Runtime Library (**libpami_r.a** - AIX, **libpami.so** - Linux, **libpamiudp.so** - Linux on UDP/IP)

C synopsis

```
#include <pami.h>

unsigned long long PAMI_Wtimebase(pami_client_t client
                                 );
```

Fortran synopsis

```
include 'pamif.h'

pami_wtimebase(client, ierror)
integer client
integer ierror
```

Parameters

Input

client Specifies the client handle.

Output

ierror Specifies a Fortran return code. This is always the last parameter.

Description

Use this subroutine to return the number of "cycles" that have elapsed on the calling processor. This subroutine returns the number of "cycles" since an arbitrary time in the past. "Cycles" could be any quickly and continuously increasing counter if true cycles are unavailable.

Location

/usr/lib/libpami_r.a (AIX)

/usr/lib/libpami.so (32-bit Linux)

/usr/lib/libpamiudp.so (32-bit Linux, UDP/IP)

/usr/lib64/libpami.so (64-bit, Linux)

/usr/lib64/libpamiudp.so (64-bit Linux, UDP/IP)

Related information

C examples: post-multithreaded-perf.c, send_flood_perf.c, send_to_self_immed.c, send_to_self_perf.c, timebase.c

Chapter 3. PAMI environment variables

This appendix includes PE environment variables that apply to PAMI.

Table 5. Environment variables for MPICH2

Environment variable	**Set:**	**Possible values**	**Default value**
PAMI_EAGER	The cutoff for the switch to the rendezvous protocol. This environment variable switches from the eager protocol to the rendezvous protocol for point-to-point messaging. Increasing the limit could help larger partitions, if most of the communication is with the closest neighbor.	*nnnnn* *nn*K (where: K = 1024 bytes)	**4096**
PAMI_RMA_PENDING	The maximum number of outstanding remote memory access (RMA) requests. This environment variable limits the number of **PAMI_Request** objects that are allocated by MPI one-sided operations.	Any positive integer	**1000**
PAMI_RVZ	The cutoff for the switch to the rendezvous protocol. This is a value, in bytes, to switch from the eager protocol to the rendezvous protocol for point-to-point messaging. Increasing the limit could help for larger partitions and if most of the communication is with the closest neighbor.	*nnnnn* *nn*K (where: K = 1024 bytes)	**4096**
PAMI_RZV	The cutoff for the switch to the rendezvous protocol. This is a value, in bytes, to switch from the eager protocol to the rendezvous protocol for point-to-point messaging. Increasing the limit could help for larger partitions and if most of the communication is with the closest neighbor.	*nnnnn* *nn*K (where: K = 1024 bytes)	**4096**
PAMI_SHMEM_PT2PT	To determine whether intranode point-to-point communication will use the optimized shared memory protocols.	**no** **yes**	**yes**

Accessibility features for PE

Accessibility features help users who have a disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in PE:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are discernible by touch but do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

The **IBM Cluster information center**, and its related publications, are enabled for accessibility. This information center's accessibility features are described under Accessibility (http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.addinfo.doc/access.html).

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

IBM and accessibility

See the IBM Human Ability and Accessibility Center (http://www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A.

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
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

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 states 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web

All implemented function in the PE MPI product is designed to comply with the requirements of the Message Passing Interface Forum, MPI: A Message-Passing Interface Standard. The standard is documented in two volumes, Version 1.1, University of Tennessee, Knoxville, Tennessee, June 6, 1995 and *MPI-2: Extensions to the Message-Passing Interface*, University of Tennessee, Knoxville, Tennessee, July 18, 1997. The second volume includes a section identified as MPI 1.2 with clarifications and limited enhancements to MPI 1.1. It also contains the extensions identified as MPI 2.0. The three sections, MPI 1.1, MPI 1.2 and MPI 2.0 taken together constitute the current standard for MPI.

PE MPI provides full support for all of MPI 2.2.

If you believe that PE MPI does not comply with the MPI standard for the portions that are implemented, please contact IBM Service.

Trademarks

Glossary

This glossary defines technical terms used in the IBM Parallel Environment documentation. If you do not find the term you are looking for, refer to the IBM Terminology site on the World Wide Web (**http://www.ibm.com/software/globalization/terminology/index.html**).

A

address
A unique code or identifier for a register, device, workstation, system, or storage location.

API
application programming interface (API): An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

application
One or more computer programs or software components that provide a function in direct support of a specific business process or processes.

argument
A value passed to or returned from a function or procedure at run time.

authentication
The process of validating the identity of a user or server.

authorization
The process of obtaining permission to perform specific actions.

B

bandwidth
A measure of frequency range, typically measured in hertz. Bandwidth also is commonly used to refer to data transmission rates as measured in bits or bytes per second.

blocking operation
An operation that has not completed until the operation either succeeds or fails. For example, a blocking receive will not return until a message is received or until the channel is closed and no further messages can be received.

breakpoint
A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broadcast
The simultaneous transmission of data to more than one destination.

C

C
A programming language designed by Bell Labs in 1972 for use as the systems language for the UNIX operating system.

C++
An enhancement of the C language that adds features supporting object-oriented programming.

client A software program or computer that requests services from a server.

cluster
A group of processors interconnected through a high-speed network that can be used for high-performance computing.

collective communication
A communication operation that involves more than two processes or tasks. Broadcasts and reductions are examples of collective communication operations. All tasks in a communicator must participate.

communicator
A Message Passing Interface (MPI) object that describes the communication context and an associated group of processes.

compile
translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language.

condition
One of a set of specified values that a data item can assume.

core dump
A process by which the current state of a program is preserved in a file. Core dumps are usually associated with programs that have encountered an unexpected, system-detected fault, such as a segmentation fault or a severe user error. A programmer can use the core dump to diagnose and correct the problem.

core file
A file that preserves the state of a program, usually just before a program is terminated because of an unexpected error. See also *core dump.*

D

data parallelism
A situation in which parallel tasks perform the same computation on different sets of data.

debugger
A tool used to detect and trace errors in computer programs.

E

environment variable
(1) A variable that defines an aspect of the operating environment for a process. For example, environment variables can define the home directory, the command search path, the terminal in use, or the current time zone. (2) A variable that is included in the current software environment and is therefore available to any called program that requests it.

Ethernet
A packet-based networking technology for local area networks (LANs) that supports multiple access and handles contention by using Carrier Sense Multiple Access with Collision Detection (CSMA/CD) as the access method. Ethernet is standardized in the IEEE 802.3 specification.

executable program
A program that can be run as a self-contained procedure. It consists of a main program and, optionally, one or more subprograms.

execution
The process of carrying out an instruction or instructions of a computer program by a computer.

F

fairness
A policy in which tasks, threads, or processes must eventually gain access to a resource for which they are competing. For example, if multiple threads are simultaneously seeking a lock, no set of circumstances can cause any thread to wait indefinitely for access to the lock.

Fiber Distributed Data Interface (FDDI)
An American National Standards Institute (ANSI) standard for a 100-Mbps LAN using fiber optic cables.

file system
The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

fileset (1) An individually-installable option or update. Options provide specific function, and updates correct an error in, or enhance, a previously installed program. (2) One or more separately-installable, logically-grouped units in an installation package. See also *licensed program* and *package.*

FORTRAN
A high-level programming language used primarily for scientific, engineering, and mathematical applications.

G

GDB An open-source portable debugger supporting Ada, C, C++, and FORTRAN. GDB is a useful tool for determining why a program crashes and where, in the program, the problem occurs.

global max
The maximum value across all processors for a given variable. It is global in the sense that it is global to the available processors.

global variable
A symbol defined in one program module that is used in other program modules that are independently compiled.

graphical user interface (GUI)
A type of computer interface that presents a visual metaphor of a real-world scene, often of a desktop, by combining high-resolution graphics, pointing devices, menu bars and other menus, overlapping windows, icons and the object-action relationship.

GUI See graphical user interface.

H

high performance switch
A high-performance message-passing network that connects all processor nodes.

home node
The node from which an application developer compiles and runs a program. The home node can be any workstation on the LAN.

host A computer that is connected to a network and provides an access point to that network. The host can be a client, a server, or both a client and server simultaneously.

host list file
A file that contains a list of host names, and possibly other information. The host list file is defined by the application that reads it.

host name
The name used to uniquely identify any computer on a network.

I

installation image
A copy of the software, in backup format, that the user is installing, as well as copies of other files the system needs to install the software product.

Internet
The collection of worldwide networks and gateways that function as a single, cooperative virtual network.

Internet Protocol (IP)
A protocol that routes data through a network or interconnected networks. This protocol acts as an intermediary between the higher protocol layers and the physical network.

IP Internet Protocol.

K

kernel The part of an operating system that contains programs for such tasks as input/output, management and control of hardware, and the scheduling of user tasks.

L

latency
The time from the initiation of an operation until something actually starts happening (for example, data transmission begins).

licensed program
A separately priced program and its associated materials that bear a copyright and are offered to customers under the terms and conditions of a licensing agreement.

lightweight core files
An alternative to standard AIX core files. Core files produced in the Standardized Lightweight Corefile Format provide simple process stack traces (listings of function calls that led to the error) and consume fewer system resources than traditional core files.

LoadLeveler pool
A group of resources with similar characteristics and attributes.

local variable
A symbol defined in one program module or procedure that can only be used within that program module or procedure.

M

management domain
A set of nodes that are configured for management by Cluster Systems

Management. Such a domain has a management server that is used to administer a number of managed nodes. Only management servers have knowledge of the domain. Managed nodes only know about the servers managing them.

menu A displayed list of items from which a user can make a selection.

message catalog
An indexed table of messages. Two or more catalogs can contain the same index values. The index value in each table refers to a different language version of the same message.

message passing
The process by which parallel tasks explicitly exchange program data.

Message Passing Interface (MPI)
A library specification for message passing. MPI is a standard application programming interface (API) that can be used by parallel applications.

MIMD
multiple instruction stream, multiple data stream.

multiple instruction stream, multiple data stream (MIMD)
A parallel programming model in which different processors perform different instructions on different sets of data.

MPMD
Multiple program, multiple data.

Multiple program, multiple data (MPMD)
A parallel programming model in which different, but related, programs are run on different sets of data.

N

network
In data communication, a configuration in which two or more locations are physically connected for the purpose of exchanging data.

network information services (NIS)
A set of network services (for example, a distributed service for retrieving information about the users, groups, network addresses, and gateways in a network) that resolve naming and addressing differences among computers in a network.

NIS See *network information services*.

node ID
A string of unique characters that identifies the node on a network.

nonblocking operation
An operation, such as sending or receiving a message, that returns immediately whether or not the operation has completed. For example, a nonblocking receive does not wait until a message arrives. A nonblocking receive must be completed by a later test or wait.

O

object code
Machine-executable instructions, usually generated by a compiler from source code written in a higher level language. Object code might itself be executable or it might require linking with other object code files.

optimization
The process of achieving improved run-time performance or reduced code size of an application. Optimization can be performed by a compiler, by a preprocessor, or through hand tuning of source code.

option flag
Arguments or any other additional information that a user specifies with a program name. Also referred to as parameters or command line options.

P

package
1) In AIX, a number of filesets that have been collected into a single installable image of licensed programs. See also fileset and licensed program. 2) In Linux, a collection of files, usually used to install a piece of software. The equivalent AIX term is fileset.

parallelism
The degree to which parts of a program may be concurrently executed.

parallelize
To convert a serial program for parallel execution.

parameter
A value or reference passed to a function, command, or program that serves as input or controls actions. The value is supplied by a user or by another program or process.

peer domain
A set of nodes configured for high availability. Such a domain has no distinguished or master node. All nodes are aware of all other nodes, and administrative commands can be issued from any node in the domain. All nodes also have a consistent view of the domain membership. Contrast with *management domain*.

point-to-point communication
A communication operation that involves exactly two processes or tasks. One process initiates the communication through a send operation. The partner process issues a receive operation to accept the data being sent.

procedure
In a programming language, a block, with or without formal parameters, that is initiated by means of a procedure call. (2) A set of related control statements that cause one or more programs to be performed.

process
A program or command that is actually running the computer. A process consists of a loaded version of the executable file, its data, its stack, and its kernel data structures that represent the process's state within a multitasking environment. The executable file contains the machine instructions (and any calls to shared objects) that will be executed by the hardware. A process can contain multiple threads of execution.

The process is created with a **fork()** system call and ends using an **exit()** system call. Between **fork** and **exit**, the process is known to the system by a unique process identifier (PID).

Each process has its own virtual memory space and cannot access another process's memory directly. Communication methods across processes include pipes, sockets, shared memory, and message passing.

profiling
A performance analysis process that is based on statistics for the resources that are used by a program or application.

pthread
A shortened name for the i5/OS threads API set that is based on a subset of the POSIX standard.

R

reduction operation
An operation, usually mathematical, that reduces a collection of data by one or more dimensions. For example, an operation that reduces an array to a scalar value.

remote host
Any host on a network except the host at which a particular operator is working.

remote shell (rsh)
A variant of the remote login (rlogin) command that invokes a command interpreter on a remote UNIX machine and passes the command-line arguments to the command interpreter, omitting the login step completely.

RSCT peer domain
See *peer domain*.

S

secure shell (ssh)
A Unix-based command interface and protocol for securely accessing a remote computer.

shell script
A program, or script, that is interpreted by the shell of an operating system.

segmentation fault
A system-detected error, usually caused by a reference to a memory address that is not valid.

server A software program or a computer that provides services to other software programs or other computers.

single program, multiple data (SPMD)
A parallel programming model in which different processors run the same program on different sets of data.

source code
A computer program in a format that is readable by people. Source code is converted into binary code that can be used by a computer.

source line
A line of source code.

SPMD
single program, multiple data.

standard error (STDERR)
The output stream to which error messages or diagnostic messages are sent.

standard input (STDIN)
An input stream from which data is retrieved. Standard input is normally associated with the keyboard, but if redirection or piping is used, the standard input can be a file or the output from a command.

standard output (STDOUT)
The output stream to which data is directed. Standard output is normally associated with the console, but if redirection or piping is used, the standard output can be a file or the input to a command.

STDERR
standard error.

STDIN
standard input.

STDOUT
standard output.

subroutine
A sequence of instructions within a larger program that performs a particular task. A subroutine can be accessed repeatedly, can be used in more than one program, and can be called at more than one point in a program.

synchronization
The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

system administrator
The person who controls and manages a computer system.

T

task In a parallel job, there are two or more concurrent tasks working together through message passing. Though it is common to allocate one task per processor, the terms *task* and *processor* are not interchangeable.

thread A stream of computer instructions. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

trace A record of the processing of a computer program or transaction. The information collected from a trace can be used to assess problems and performance.

U

user (1) An individual who uses license-enabled software products. (2) Any individual, organization, process, device, program, protocol, or system that uses the services of a computing system.

User Space
A version of the message passing library that is optimized for direct access to the high performance switch (PE for AIX) or communication adapter (PE for Linux). User Space maximizes performance by not involving the kernel in sending or receiving a message.

utility program
A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program.

utility routine
: A routine in general support of the processes of a computer; for example, an input routine.

V

variable
: A representation of a changeable value.

X

X Window System
: A software system, developed by the Massachusetts Institute of Technology, that enables the user of a display to concurrently use multiple application programs through different windows of the display. The application programs can execute on different computers.

Index

Product Number: 5725-G00
5765-PER
5765-PRP

Printed in USA

SA23-2273-03