

KSR-1 $\mathbb{P}$ fortran Users Guide

a $\mathbb{P}$ arallel extension of Fortran

Terry W. Clark
University of Houston

Release 1.4.1

October 11, 1994

Contents

1	Introduction	2
2	Using the IPfortran Translator	3
2.1	Overview of <i>pfc</i>	3
2.2	Include files	3
2.3	Scratch memory and variables	4
2.4	Intrinsic names	4
2.5	The KSR-1 and IPfortran variable sizes	4
3	Compiling and loading	6
3.1	IPfortran KSR-1 Libraries	6
3.2	Compiling programs for SHMSG	6
4	Executing SHMSG based programs	7
4.1	Specifying processor configurations for SHMSG based programs . . .	7
4.2	Running SHMSG based programs	7
4.3	Stopping SHMSG based programs	8
4.4	Things to look out for	8
5	Final usage notes	9
5.1	Site specific locations	9

1 Introduction

The IPfortran programming approach eases the task of writing parallel programs for massively parallel computers by providing constructs to coordinate non-local memory accesses. At IPfortran's center is an operator that specifies the location of off-processor variables and an operator for aggregate operations. Portability and ease of programming are achieved by unifying off-processor accesses with the guarded-memory model [2]. Because the programmer writes low-level *node code* with IPfortran, the loss in performance compared to using the lowest-level primitives is typically negligible.

This document discusses running IPfortran on the KSR. For details about programming in IPfortran, see the IPfortran *Reference Manual* [1]. (The complete IPfortran *User's Guide* discusses IPfortran on a wider set of architectures [3]).

2 Using the IPfortran Translator

2.1 Overview of *pfc*

IPfortran translates the input *.pf*, producing output into a *.f* file. Code not involving IPfortran extensions is output “as is,” including comments and blank lines. Code generated from translating statements involving `{}` or `@` adheres to the rules of FORTRAN77.

IPfortran can be used on one, or all, of the modules composing a program with the exception of the entry point to the Fortran program: *The main program (entry point to a program) must be processed by the IPfortran processor.*

The IPfortran translator **pfc** can be invoked with

```
pfc  -ksr  [switches]  f1.pf  [ f2.pf [ ... ] ]
```

or alternatively

```
pfc  -ksr  [switches]  *.pf.
```

Note, the *-ksr* switch is optional.

The optional switches are

- I** Search this directory for include files.
- scratch <n>** Number of bytes for IPfortran buffer (default 100000).
- T string** Use *string* as variable name for THISNODE (default).
- i4** Default integer size to 4 bytes.
- r8** Default double size to 8 bytes.

2.2 Include files

Optional include file search directories can be used with the syntax:

```
$pfc -Id1 -Id2 ... -Idn myprog.pf
```

for directories *d*₁ ... *d*_{*n*}.

Up to ten directories can be specified with one directory per I flag. The directories are searched in order of occurrence; the current working directory is always searched

and it is searched last unless `-I.` is the first include search directory. The maximum nesting depth of include files is ten.

2.3 Scratch memory and variables

Temporary memory for arrays uses a block of storage declared with the array *pf_ib*. The default size of this storage can be modified using the `-scratch` switch. This storage is re-typed as needed through the use of equivalence statements.

IPfortran generates scalar variables for buffering or naming quantities. These variables are named from the base string *o10000* with successive variables incrementing the suffix *10000*. Clashes can occur if these variable names are used as *implicitly* typed variables in your program.

Labels are also generated by IPfortran in generating FORTRAN77 loops. These are guaranteed to be unique to the program section.

2.4 Intrinsic names

In addition to the intrinsic functions used by the target processor, IPfortran uses the following variables to regularize the parallel program.

THISNODE Processor number of the processor running the code.

NUMNODE Total number of processors.

See the complete IPfortran User's Guide for a comprehensive list of reserved variables [3].

2.5 The KSR-1 and IPfortran variable sizes

IPfortran interprets variable sizes according to the specific architecture. The size of integers, reals, double precision can be controlled with compiler flags, overriding the IPfortran defaults. We recommended that variable sizes are consistent throughout a program, that is, all reals the same size, all double precision the same size, and so

on. With experience, however, it's possible to use almost any combination of sizes permitted by the native compiler.

The following table is useful for determining how to size variables in your program. It shows the default IPfortran variable sizes for the KSR-1, and the effect of IPfortran switches. The 3rd and 4th columns show the effect of the *i4* and *r8* IPfortran flags, respectively. Note that a dash indicates no effect.

IPfortran variable sizes for the KSR-1			
type	default	-i4	-r8
integer	8	4	–
real	8	–	–
double	16	–	8
complex	8	–	–

The IPfortran default variable sizes, and *i4* and *r8* effect, shown above, are identical to the KSR-1 Fortran compiler's.

3 Compiling and loading

3.1 IPfortran KSR-1 Libraries

The SHMSG system is a message-passing library for shared memory architectures. Two libraries are required for running a SHMSG-based program on the KSR: the standard IPfortran library and the message-passing library. The message passing library is called *shmsg.a*. The standard IPfortran libraries are *pflibshm4.f* and *pflibshm8.f*, corresponding to 4-byte and 8-byte integers; it's the programmer's responsibility to ensure that variable sizes, compiler flags, and libraries are compatible. The 8-byte and 16-byte double-precision modes are compatible with either *pflibshm*. (Also see Section 2.5.)

3.2 Compiling programs for SHMSG

A sample Makefile for a program using 4-byte integers follows. Notice the integer switch supplied to *pfc* is the same integer switch supplied to the KSR-1 Fortran compiler; generally, one should follow this strategy with variable sizing.

```
LIBDIR = /home1/tclark/lib
F77 = f77
CC = cc
INC = .
PFC = /home1/tclark/bin/pfc
SHMSG = $(LIBDIR)/shmsg.a
LIBS = $(SHMSG) -para

# 4-byte integers
PFFLAGS = -ksr -I$(INC) -scratch 10000 -i4
FFLAGS = -i4 -noopt
PFL = $(LIBDIR)/pflibshm4.o
# suffix rules
#
.SUFFIXES: .pf .f .pc .c .s .o
.f.o:; $(F77) -I$(INC) $(PFFLAGS) -c $.f
.pf.f:; $(PFC) $(PFFLAGS) $.pf
.pf.o:; $(PFC) $(PFFLAGS) $.pf
        $(F77) -I$(INC) $(PFFLAGS) -c $.f

test4: test4.o
        $(F77) -o test4 test4.o $(PFL) $(LIBS)
```

4 Executing SHMSG based programs

4.1 Specifying processor configurations for SHMSG based programs

A configuration consists of the number of processes (P) and the number of cells (C) on which the processes will execute. Processes will be bound to some cell. If $P > C$, processes will be cell mapped many-to-one.

A third parameter here is the number of cells in your pset (`actual_C`), or the “machine” that your application sees. If $C > \text{actual_C}$, SHMSG will adjust and inform you that intra-application pset timesharing will occur.

Intra-application timesharing is not recommended at this time.

Applications can have problems with intra-application pset timesharing, especially when $P > 2C$. We have found that either it works, or it doesn't. We discourage this usage.

The configuration, then, consists of two integers contained in a file called `Config` (upper case 'C'). This file must be in the directory where program executes. The values of P and C are integers on the first line of this file, e.g., a first record of “8 8” specifies 8 processes to be executed on 8 processors.

Processor configurations should be powers of two for Pfortran compatibility. That limitation will disappear with an expanded Pfortran communication library. Non-power of two processor operations will work with SHMSG in any case, and with Pfortran provided global exchanges are not used there.

4.2 Running SHMSG based programs

Type the program name in the directory where the `Config` file resides. Input and output behave as usual.

4.3 Stopping SHMSG based programs

So that your application releases system resources, a cleanup routine must be invoked prior to normal termination. The Pfortran compiler will translate a Fortran **stop** to a *pfstop* which in turn invokes the routines to release resources. A Fortran program should terminate with a Fortran **stop**.

To stop interactive processes, use `ctl-c`. If the process is not attached to a terminal, send it the SIGINT signal, i.e. `kill -2 <pid>`. If all else fails, use `kill -9`, with the disadvantage that the processes cannot release system resources when hit with the SIGKILL signal.

4.4 Things to look out for

Hung program: If your program hangs, or abnormally terminates, take a look at system *ipc resources* using the command “`ipcs`.” If this seems like too much trouble, just run `bubba:: ~tclark/bin/rmem`. This will remove the shared-memory segments that the message-passing system used, and did not release, due to drastic termination, e.g. `kill -9`.

Intra-application pset timesharing: When an application’s processes timeshare cells, a less efficient algorithm is used for coordinating access to shared data in the message-passing system. This stems from a bug in a hardware mechanism used for enforcing mutual exclusion.

5 Final usage notes

KSR IPfortran programs are efficient and portable. There are, however, some limitations that the user should be aware of.

- Global sum algorithms require power of two processors, e.g. 1,2,4, Using other than powers of two processors will result in erroneous results for IPfortran global sum operations. Workaround: use powers of two total number of processors, or do not use the IPfortran intrinsic global sum operations.
- The KSR debugger is not operational for IPfortran programs. Workaround: Use print statements or debug on an Intel hypercube.

5.1 Site specific locations

At the TCAMC, the following directories contain up-to-date libraries, etc.:

pflibshm4.f, pflibshm8.f:	/home1/tclark/lib
shmsg.a:	/home1/tclark/lib
pfc:	/home1/tclark/bin
example 1:	/home1/tclark/pf/pfshmsg/4
example 2:	/home1/tclark/pf/pfshmsg/8

The example-program directories each contain a complete, working example of a test program, and its Makefile. The test program reports success or failure while testing IPfortran communication primitives.

References

- [1] B. Bagheri, T. W. Clark, and L. R. Scott. *IPfortran* (a parallel extension of Fortran) Reference Manual. Research Report UH/MD-119, Dept. of Mathematics, University of Houston, 1991.
- [2] Babak Bagheri. *Parallel Programming with Guarded Objects*. PhD thesis, Pennsylvania State University, 1994.
- [3] T. W. Clark, Rick Kufrin, and Edward Robinson. *IPfortran (a Parallel extension of Fortran) User's Guide*. University of Houston, 1994.