IBM Parallel Environment for AIX

Operation and Use, Volume 1
Using the Parallel Operating Environment

Version 3 Release 2

SA22-7425-01

IBM Parallel Environment for AIX

Operation and Use, Volume 1
Using the Parallel Operating Environment

Version 3 Release 2

SA22-7425-01

Note

Before using this information and the product it supports, read the information in “Notices” on page 145.

Second Edition (December 2001)

This edition applies to version 3, release 2 of IBM Parallel Environment for AIX (product number 5765-D93) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers’ comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation

Department 55JA, Mail Station P384

2455 South Road

Poughkeepsie, NY 12601-5400

United States of America

FAX (United States & Canada): 1+845+432-9405

FAX (Other Countries):

Your International Access Code +1+845+432-9405

IBMLink (United States customers only): IBMUSM10(MHVRCFS)

Internet e-mail: mhvrcfs@us.ibm.com

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000, 2001. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Tables vii

About this book ix
Who should read this book ix
How this book is organized ix
Overview of contents ix
Conventions and terminology used in this book x
Abbreviated names x
How to send your comments xi
National language support (NLS) xi
Setting POE environment variables xii
What's new in Parallel Environment 3.2? xii
New PE Benchmarker tools xii
Improved parallel checkpointing capabilities xiii
MPI enhancements xiii
DPCL is now an open source offering xiv
Removal of pedb debugger support xiv
Removal of VT trace collection support xv
Commands no longer supported xv

Chapter 1. Introduction 1
PE Version 3.2 Migration information 4
AIX compatibility 4
Parallel tracing removed 4
Checkpoint/Restart function 5
pedb removed 5
64-bit application support 5
Fortran application support 6

Chapter 2. Executing parallel programs 7
Executing parallel programs using POE 7
Step 1: Compile the program 8
Step 2: Copy files to individual nodes 10
Step 3: Set up the execution environment 11
Step 4: Start the program marker array analysis tool 27
Step 5: Invoke the executable 28
Controlling program execution 35
Specifying develop mode 35
Making POE wait for processor nodes 36
Making POE ignore arguments 36
Managing standard input, output, and error 37
Determining which nodes will participate in parallel I/O 44
Checkpointing and restarting programs 45
POE user authorization 46
Supported security methods 46
Using DCE user authorization 47
Using AIX user authorization 47
Running POE with MALLOCDEBUG 48

Chapter 3. Managing POE jobs 49
Multi-task core file 49
Specifying the format of corefiles or suppressing corefile generation 50
Generating standard AIX corefiles 51

© Copyright IBM Corp. 2000, 2001

Writing corefile information to standard error . . . 51
Generating lightweight corefiles . . . 51
Stopping a POE job . . . 53
Cancelling and killing a POE job . . . 53
Detecting remote node failures . . . 53
Considerations for using the SP switch and SP switch 2. . . . 53
Scenarios for allocating nodes with LoadLeveler . . . 54
Submitting a batch POE job using IBM LoadLeveler . . . 57
Submitting an interactive POE job using an IBM LoadLeveler command file 59
Generating an output LoadLeveler job command file . . . 61
Running programs under the C shell . . . 61
Using MP_CSS_INTERRUPT . . . 63
Support for performance improvements . . . 65
Interrupt mode control . . . 65
Rcvncall improvements . . . 66
Parallel file copy utilities . . . 66

Chapter 4. Monitoring program execution using the Program Marker Array 69
Step 1: Call PM Array parallel utility functions . . . 70
Step 2: Compile the program. . . . 70
Step 3: Set the number of lights . . . 70
Step 4: Open the PM array window . . . 71
Step 5: Invoke the program and monitor its execution . . . 71
Displaying details of a light . . . 71
Displaying task output . . . 72
Step 6: Close the PM Array window . . . 72

Appendix A. Parallel environment commands . . . 73
mcp . . . 74
mcpgath . . . 76
mcpscat . . . 81
mpamddir . . . 85
mpcc . . . 86
mpcc_r . . . 88
mpCC . . . 90
mpCC_r . . . 92
mpiexec . . . 94
mpxlf . . . 95
mpxlf_r . . . 97
mpxlf90 . . . 100
mpxlf90_r . . . 102
mpxlf95 . . . 105
mpxlf95_r . . . 107
pmarray . . . 110
poe. . . . 112
poeckpt . . . 127
poekill. . . . 128
poerestart . . . 129

Appendix B. POE environment variables and command-line flags . . . 131

Notices . . . 145
Trademarks. . . . 146
Acknowledgments . . . 147

Glossary . . . 149

Ex.1008 / Page 6 of 186
Intel Corporation et al

Bibliography 157
Information formats 157
Finding documentation on the World Wide Web 157
Accessing PE documentation online 157
RS/6000 SP publications 158
SP planning publications 158
SP software publications 158
AIX publications 159
DCE publications 159
Red books 159
Non-IBM publications 159

Index 161

Ex.1008 / Page 7 of 186
Intel Corporation et al

Tables

1. Execution setup summary (for an SP system). 14
2. Execution environment setup summary (for pSeries or RS/6000 network cluster or a mixed system whose additional nodes are not part of the LoadLeveler cluster) 15
3. Node allocation summary. 15
4. Adapter/CPU default settings 20
5. Adapter/CPU use under LoadLeveler 21
6. LoadLeveler node allocation. 27
7. Variables and flags for partition manager control 132
8. Variables and flags for job specification 134
9. Variables and flags for I/O control 136
10. Variables and flags for diagnostic information 136
11. Variables and flags for message passing 138
12. Variables and flags for core file generation 142
13. Other variables and flags 143

© Copyright IBM Corp. 2000, 2001

About this book

This book describes the IBM® Parallel Environment (PE) for AIX® program product and its Parallel Operating Environment (POE). It shows how to use POE's facilities to compile, execute, and analyze parallel programs.

This book concentrates on the actual commands and how to use them, as opposed to the writing of parallel programs. For this reason, you should use this book in conjunction with *IBM Parallel Environment for AIX: MPI Programming Guide* and *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference*. New users should refer to *IBM Parallel Environment for AIX: Hitchhiker's Guide*, for basic and introductory information on PE.

This book assumes that AIX 5L Version 5.1, X-Windows**, and the PE software are already installed. It also assumes that you have been authorized to run the Parallel Operating Environment (POE). The PE software is designed to run on an IBM RS/6000 SP, an @server pSeries or RS/6000® network cluster, or on a mixed system where additional pSeries or RS/6000 processors supplement an SP™ system. For complete information on installing the PE software and setting up users, see *IBM Parallel Environment for AIX: Installation*. Also, see the appropriate AIX 5L Version 5.1 documentation listed in "Bibliography" on page 157.

Who should read this book

This book is designed primarily for end users and application developers. It is also intended for those who run parallel programs, and some of the information covered should interest system administrators. Readers should have knowledge of the AIX operating system and the X-Window system. Where necessary, this book provides some background information relating to these areas. More commonly, this book refers you to the appropriate documentation.

How this book is organized

Overview of contents

This book contains the following information:

- "Chapter 1. Introduction" on page 1 is a quick overview of the PE program product. It describes the various PE components, and how you might use each in developing a parallel application program.
- "Chapter 2. Executing parallel programs" on page 7 describes how to compile and execute parallel programs using the Parallel Operating Environment (POE).
- "Chapter 3. Managing POE jobs" on page 49 includes information on allocating nodes with IBM LoadLeveler®, and the environment variables to use when running your applications.
- "Chapter 4. Monitoring program execution using the Program Marker Array" on page 69 describes the Program Marker Array which allows you to monitor program execution online.
- "Appendix A. Parallel environment commands" on page 73 contains the manual pages for the PE commands discussed throughout this book.
- "Appendix B. POE environment variables and command-line flags" on page 131 describes the environment variables you can set to influence the execution of parallel programs and the operation of PE tools. This appendix also describes

© Copyright IBM Corp. 2000, 2001

the command-line flags associated with each of the environment variables. When invoking a parallel program, you can use these flags to override the value of an environment variable.

Conventions and terminology used in this book

This book uses the following typographic conventions:

Type Style	Used For
bold	**Bold** words or characters represent system elements that you must use literally, such as command names, flag names, and path names. **Bold** words also indicate the first use of a term included in the glossary.
italic	*Italic* words or characters represent variable values that you must supply. *Italics* are also used for book titles and for general emphasis in text.
`Constant width`	Examples and information that the system displays appear in `constant width` typeface.

In addition to the highlighting conventions, this manual 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 **poe** command, this manual presents the instruction as:

ENTER
poe

The system's response to entering the **poe** command would read:

The Partition Manager allocates the processor nodes of your partition.

Abbreviated names

Some of the abbreviated names used in this book follow.

Short Name	Full Name
AIX	Advanced Interactive Executive
CSS	communication subsystem
DPCL	dynamic probe class library
dsh	distributed shell
GUI	graphical user interface
HDF	Hierarchical Data Format
IP	Internet Protocol
MPI	Message Passing Interface
MPL	Message Passing Library
PE	IBM Parallel Environment for AIX
PE MPI	IBM's implementation of the MPI standard for PE
PE MPI-IO	IBM's implementation of MPI I/O for PE
PM array	program marker array

Ex.1008 / Page 12 of 186
Intel Corporation et al

Short Name	Full Name
POE	parallel operating environment
pSeries	IBM @server pSeries
PSSP	IBM Parallel System Support Programs for AIX
RISC	reduced instruction set computer
rsh	remote shell
RS/6000	IBM RS/6000
SDR	System Data Repository
SP	IBM RS/6000 SP
STDERR	standard error
STDIN	standard input
STDOUT	standard output
US	user space

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments about this book or any 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 PE, 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 located in externalized message catalogs. English versions of the message catalogs are shipped with the PE program product, but your site may be using its own translated message catalogs. The AIX environment variable **NLSPATH** is used by the various PE components 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 want the default message catalog:

ENTER

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

export LANG=C
```

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

/usr/lib/nls/msg/C
/usr/lib/nls/msg/En_US
/usr/lib/nls/msg/en_US

If your site is using its own translations of the message catalogs, consult your system administrator for the appropriate value of **NLSPATH** or **LANG**. For additional information on NLS and message catalogs, see *IBM AIX 5L Version 5.1 General Programming Concepts: Writing and Debugging Programs*.

Setting POE environment variables

Throughout this book, a number of POE environment variables are discussed. These environment variables can be set to influence the operation of certain tools and the execution of parallel programs. The POE environment variables also have associated command-line flags that can be used when invoking a parallel program or tool. The POE command-line flags temporarily override their associated environment variable. The following table shows how to set a POE environment variable or command-line flag depending on the shell you are using. The remainder of this book assumes use of the Korn Shell. For a complete listing of the POE environment variables and command-line flags, refer to “Appendix B. POE environment variables and command-line flags” on page 131.

	To set an environment variable:	**To override an environment variable when invoking a parallel program or PE tool:**
In Korn Shell:	**ENTER** **export** *VARIABLE*=*value*	**ENTER** *command program -flag value*
In C Shell	**ENTER** **setenv** *VARIABLE value*	**ENTER** *command program -flag value*

What’s new in Parallel Environment 3.2?

This release of the Parallel Environment contains a number of functional enhancements, including:

- new PE Benchmarker tools
- improved parallel checkpointing capabilities
- support for 64-bit applications
- MPI enhancements

In addition, PE 3.2 includes these changes:

- PE now supports, and requires, AIX 5L™ 5.1.
- PE Version 2, Release 2 is no longer supported.
- The Dynamic Probe Class Library (DPCL) is no longer a part of PE, though it is still shipped with PE. Instead, DPCL is now an open source offering that supports PE.
- The **pedb** debugger has been removed.
- The VT parallel tracing facility has been removed.

The following sections describe these functional enhancements and changes in more detail.

New PE Benchmarker tools

This release of PE contains a new suite of applications and utilities that you can use to analyze the performance of programs. This suite of tools is called *PE Benchmarker* and contains:

Ex.1008 / Page 14 of 186
Intel Corporation et al

- the *Performance Collection Tool*, which enables you to collect either MPI and user event data, or else hardware and operating system profiles for one or more application processes. This tool is built on dynamic instrumentation technology (the Dynamic Probe Class Library, or DPCL) which enables you to make the decision of what data to collect at run time. Probes are placed in the running executable to collect just the information you requested. This ability typically results in a more acceptable intrusion cost than you would have with more traditional styles of instrumentation.
- a set of Unified Trace Environment (UTE) utilities for converting one or more AIX trace files (as output by the Performance Collection Tool) into UTE interval files. Additional UTE utilities enable you to:
  - generate statistics tables from UTE interval files.
  - convert UTE interval files into a single SLOG file. You can analyze an SLOG file using Jumpshot (a public domain tool developed by Argonne National Laboratory).
- the *Profile Visualization Tool*, for viewing hardware and operating system profiles collected by the Performance Collection Tool.

Improved parallel checkpointing capabilities

This release of PE includes more flexible parallel checkpoint/restart capabilities. In previous releases, only POE/MPI applications submitted under LoadLeveler in batch mode could be checkpointed, and there were significant limitations. What's more, a checkpoint sequence could be initiated only by all tasks in the parallel MPI program. In this release, PE's checkpointing capabilities have been extended to allow:

- a user to initiate a checkpoint sequence explicitly by issuing the new command **poeckpt**. Another new command, **poerestart**, enables a user to restart a POE job using a checkpoint file.
- a single task in a parallel MPI or LAPI job to initiate a checkpoint sequence. It is no longer necessary for all tasks in the job to call checkpointing functions.
- a system administrator or LoadLeveler to initiate a checkpoint sequence.

Also, many significant limitations of the checkpointing capabilities have been removed, and the checkpointing compiler scripts (**mpcc_chkpt**, for example) are no longer used.

MPI enhancements

With this release, PE MPI provides all of the functions in the MPI standard, except for the functionality defined in the ″Process Creation and Management″ chapter of MPI-2.

Additional command for starting MPI jobs

While you can continue to use the **poe** command to start MPI jobs, this release of PE provides support for the **mpiexec** command described in the MPI-2 standard. This command is not meant to replace the **poe** command; instead it is provided as a portable way to start MPI programs, and should prove helpful for applications that target multiple implementations of MPI.

Support for 64–bit applications

Certain architectures that PE supports, such as POWER3 SMP High Node, POWER3 SMP Thin Node, and POWER3 Wide Node, can run applications using a 64-bit address space. Accordingly, the tools that are provided with PE, as well as the MPI threads library, have been enhanced to support 64-bit applications on 64-bit processors.

MPI-IO performance enhancements

To improve the performance of IBM's implementation of MPI-IO, a variety of optimizations have been made to this release of PE. For example, two new POE environment variable/command-line flag pairs and one new file hint have been added. The two new POE environment variable/command-line flag pairs are the:

- **MP_IO_BUFFER_SIZE** environment variable (**-io_buffer_size** flag), which enables you to set the default size of buffers used by I/O agents.
- **MP_IO_ERRLOG** environment variable (**-io_errlog** flag), which enables you to log errors that occurred at the file system level throughout the MPI-IO application run.

The new file hint, **IBM_sparse_access**, enables you to specify whether the file access requests from participating tasks are sparse or dense.

Extended collective communication

This release provides the MPI_IN_PLACE and intercommunicator semantic that MPI-2 has added to a number of MPI 1.1 collective communication subroutines. The new subroutines, MPI_ALLTOALLW and MPI_EXSCAN, are also provided. The nonblocking collective subroutines, which have a prefix of MPE_I, are not enhanced.

C++ and FORTRAN90 support

This release provides the C++ bindings described by MPI-2. C++ programmers can now use PE MPI more naturally, as they no longer need to use the C bindings.

FORTRAN programs can now use the **mpi** module in place of **mpif.h**.

The nonblocking collective subroutines, which have a prefix of MPE_I, are not enhanced for C++.

MPI-2 external interfaces support

With this release, PE now provides full support of MPI-2 external interfaces in the MPI threads library. This support enables you to layer additional functionality on top of MPI with an interface that is similar to MPI's.

Miscellaneous MPI-2 enhancements

Some of the functions included in the ″Miscellany″ chapter of MPI-2 were provided in prior releases of PE. This release provides the rest of these miscellaneous functions.

DPCL is now an open source offering

The Dynamic Probe Class Library (DPCL) is no longer a part of the IBM PE for AIX licensed program, but it is still shipped with PE for convenience. Instead, DPCL is now available as an open source offering that supports PE. For more information on the DPCL open source project, go to this World Wide Web address:

http://oss.software.ibm.com/developerworks/opensource/dpcl/

Removal of pedb debugger support

Beginning with this release, PE no longer includes the **pedb** parallel debugger. As a result, the **pedb** command is no longer available. The **pdbx** parallel debugger, however, is still supported; use it instead of **pedb** to debug your parallel applications.

Ex.1008 / Page 16 of 186
Intel Corporation et al

Removal of VT trace collection support

Beginning with this release, PE no longer supports the parallel trace collection facility formerly used by the PE Visualization Tool (which was removed from PE in release 3.1). Because the usefulness of examining these traces independent of VT is limited, the VT trace facility has been replaced by the more robust tracing capabilities of the new PE Benchmarker suite of tools.

Commands no longer supported

Beginning with this release, PE no longer supports these commands:

- **mpcc_chkpt**
- **mpCC_chkpt**
- **mpxlf_chkpt**
- **mpxlf90_chkpt**
- **mpxlf95_chkpt**
- **pedb**

Ex.1008 / Page 17 of 186
Intel Corporation et al

Chapter 1. Introduction

The IBM Parallel Environment for AIX program product (PE) is an environment designed for developing and executing parallel Fortran, C, or C++ programs. PE consists of components and tools for developing, executing, debugging, profiling, and tuning parallel programs.

The PE is a distributed memory message passing system. It runs on the pSeries or RS/6000 platform using the AIX 5L Version 5.1 operating system. Specifically, you can use the PE to execute parallel programs on:

- any configuration of the IBM RS/6000 SP as described in the *IBM Parallel System Support Programs for AIX: Installation and Migration Guide*. Essentially, an SP system is a collection of RS/6000 processors grouped into a number of *frames*.
- a networked cluster of pSeries or RS/6000 processors, including a single processor or a single workstation.
- a *mixed system*. In a mixed system, additional pSeries or RS/6000 processors supplement the processors of an SP system.

The pSeries or RS/6000 processors of your system are called *processor nodes*. If you are using a Symmetric Multiprocessor (SMP) system, it is important to know that, although an SMP node has more than one processing unit, it is still considered, and referred to as, a *processor node*.

A parallel program executes as a number of individual, but related, *parallel tasks* on a number of your system's processor nodes. The group of parallel tasks is called a *partition*. The processor nodes are connected on the same LAN, so the parallel tasks of your partition can communicate to exchange data or synchronize execution. If you are using an SP system:

- Your system may have an optional high performance switch for communication. The switch increases the speed of communication between nodes. It supports a high volume of message passing with increased bandwidth and low latency.
- Your system administrator can divide its nodes into separate pools. A LoadLeveler system pool is a subset of processor nodes and is given an identifying pool name or number.

PE supports the two basic parallel programming models – SPMD and MPMD. In the *SPMD (Single Program Multiple Data) model*, the programs running the parallel tasks of your partition are identical. The tasks, however, work on different sets of data. In the *MPMD (Multiple Program Multiple Data) model*, each node may be running a different program. A typical example of this is the master/worker MPMD program. In a master/worker program, one task – the master – coordinates the execution of all the others – the workers.

Note: While the remainder of this introduction describes each of the PE components and tools in relation to a specific phase of an application's life cycle, this does not imply that they are limited to one phase. They are ordered this way for descriptive purposes only; you will find many of the tools useful across an application's entire life cycle.

The application developer begins by creating a parallel program's source code. The application developer might create this program from scratch or could modify an existing serial program. In either case, the developer places calls to **Message**

© Copyright IBM Corp. 2000, 2001

Passing Interface (MPI) or **Low-level Application Programming Interface (LAPI)** routines so that it can run as a number of parallel tasks. This is known as *parallelizing* the application. The MPI is similar to the Message Passing Library (MPL) from an earlier version of Parallel Environment. MPI provides message passing capabilities for the current version of PE. There are two libraries for MPI:

- Signal handling - which uses UNIX® signals and signal handlers
- Threaded - which uses and supports POSIX user threads.

All tasks of a program must use either signal handling or threaded calls but not a combination of each.

MPL programs are still supported for 32–bit non-threaded applications only.

Note: Throughout this book, when referring to anything non-specific for MPI and MPL, the term *message passing* will be used. For example:

```
message passing program

message passing routine

message passing call
```

The message passing calls enable the parallel tasks of your partition to communicate data and coordinate their execution. The message passing routines, in turn, call communication subsystem library routines which handle communication among the processor nodes. There are two separate implementations of the communication subsystem library – the Internet Protocol (IP) Communication Subsystem and the User Space (US) Communication Subsystem. While the message passing application interface remains the same, the communication subsystem libraries use different protocols for communication among processor nodes. The IP communication subsystem uses Internet Protocol, while the US communication subsystem is designed for the SP system's high performance switch feature. The communication subsystem library implementations are dynamically loaded when you invoke the program. For more information on the message passing subroutine calls, refer to *IBM Parallel Environment for AIX: MPI Subroutine Reference*, *IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference* , and *IBM Parallel Environment for AIX: Hitchhiker's Guide*.

In addition to message passing communication, the Parallel Environment supports a separate communication protocol known as the **Low-level Application Programming Interface (LAPI)**. LAPI differs from MPI in that it is based on an "active message style" mechanism that provides a one-sided communications model. That is, one process initiates an operation and the completion of that operation does not require any other process to take a complementary action.

LAPI is designed to run on the SP system's high performance communication adapter only. Thus, it only runs with the US Communication Subsystem. A pSeries or RS/6000 workstation cluster does not support LAPI.

Although LAPI is used for data communication in conjunction with PE, it is actually part of the communication subsystem for IBM's Parallel System Support Programs (PSSP). For more information on LAPI, see *IBM Parallel System Support Programs for AIX: Administration Guide* , and *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.

Ex.1008 / Page 20 of 186
Intel Corporation et al

After writing the parallel program, the application developer then begins a cycle of modification and testing. The application developer now compiles and runs his program from his **home node** using the **Parallel Operating Environment (POE)**. The home node is any workstation on the LAN. POE is an execution environment designed to hide, or at least smooth, the differences between serial and parallel execution.

To assist with node allocation for job management, IBM LoadLeveler provides resource management function both on and off the SP system. You can run parallel programs on a cluster of processor nodes running LoadLeveler, or on a mixed system of LoadLeveler processor nodes that supplement an SP system. LoadLeveler not only provides SP node allocation for jobs using the US communication subsystem, but also provides management for non-SP nodes, or for SP nodes being used for jobs other than user space. LoadLeveler can also be used for POE batch jobs. See the IBM LoadLeveler documentation for more information on this job management system.

In general, with POE, you invoke a parallel program from your home node and run its parallel tasks on a number of **remote nodes**. When you invoke a program on your home node, POE starts your **Partition Manager** which allocates the nodes of your partition and initializes the local environment. Depending on your hardware and configuration, the Partition Manager uses a **host list file**, LoadLeveler, or both a host list file and LoadLeveler to allocate nodes. A host list file contains an explicit list of node requests, while LoadLeveler can allocate nodes from one or more system pools implicitly based on their availability.

When using LoadLeveler, POE provides an option to enable you to specify whether your program will use MPI, LAPI, or both. Using this option, POE ensures that each API initializes properly and informs LoadLeveler which APIs are used so each node is set up completely.

For Single Program Multiple Data (SPMD) applications the Partition Manager executes the same program on all nodes. For Multiple Program Multiple Data (MPMD) applications, the Partition Manager prompts you for the name of the program to load on each node. The Partition Manager also connects standard I/O to each remote node so the parallel tasks can communicate with the home node. Although you are running tasks on remote nodes, POE allows you to continue using the standard UNIX and AIX execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs, or use shell tools. The POE includes:

- A number of **parallel compiler scripts**. These are shell scripts that call the C, C++, or Fortran compilers while also linking in an interface library to enable communication between your home node and the parallel tasks running on the remote nodes. You dynamically link in a communication subsystem implementation when you invoke the executable.
- A number of **POE Environment Variables** you can use to set up your execution environment. These are AIX environment variables you can set to influence the operation of POE. These environment variables control such things as how processor nodes are allocated, what programming model you are using, and how standard I/O between the home node and the parallel tasks should be handled. Most of the POE environment variables also have associated command-line flags that enable you to temporarily override the environment variable value when invoking POE and your parallel program.
- The **Program Marker Array**. This is a programmable array of small boxes, or *lights*, which are associated with parallel tasks. Under program control, these

lights can change color to provide you with immediate visual feedback as your program executes. See “Chapter 4. Monitoring program execution using the Program Marker Array” on page 69 for a complete description of this tool.

The following tools are discussed in *IBM Parallel Environment for AIX: Operation and Use, Volume 2* and allow you to debug and tune parallel programs.

The **parallel debugging facility** is **pdbx** – a line-oriented debugger based on the **dbx** debugger.

Once the parallel program is debugged, you now want to tune the program for optimal performance. To do this, you turn to the PE parallel profiling capability to analyze the program.

The **parallel profiling capability** enables you to use the PE Xprofiler graphical user interface, as well as the AIX commands **prof** and **gprof** on parallel programs. Xprofiler is a tool that helps you analyze your parallel application’s performance quickly and easily. It uses procedure profiling information to construct a graphical display of the functions within your application. Xprofiler provides quick access to the profiled data, which lets you identify the functions that are the most CPU-intensive. The graphical user interface also lets you manipulate the display in order to focus on the application’s critical areas.

Note: Once the parallel program is tuned to your satisfaction, you might prefer to execute it using a job management system such as IBM LoadLeveler. If you do use a job management system, consult its documentation for information on its use.

PE Version 3.2 Migration information

This section is intended for customers migrating from earlier releases of PE to PE 3.2. It contains specific information on some differences between earlier releases that you need to consider prior to installing and using PE 3.2. To find out which release of PE you currently have installed, use the command:

lslpp -ha *ppe.poe*

AIX compatibility

PE 3.2 commands and applications are compatible with AIX 5L Version 5.1 or later only, and not with earlier versions of AIX.

Parallel tracing removed

Parallel tracing function from previous releases of PE has been completely removed in PE 3.2. The Parallel tracing symbols and header file remain in PE 3.2 to allow binary compatibility of existing applications, however, these functions are no longer executed. The following POE options and environment variables that supported Parallel tracing are no longer available, and customers should remove them from execution scripts and command files:

-samplefreq or **-sfreq**
: **MP_SAMPLEFREQ**

-tbuffsize or **-tbsize**
: **MP_TBUFFSIZE**

-tbuffwrap or **-tbwrap**
: **MP_TBUFFWRAP**

-tmpdir
MP_TMPDIR
-tracedir or **-tdir**
MP_TRACEDIR
-tracefile or **-tfile**
MP_TRACEFILE
-tracelevel or **-tlevel**
MP_TRACELEVEL
-ttempsize or **-ttsize**
MP_TTEMPSIZE

Checkpoint/Restart function

The parallel Checkpoint/Restart function has been completely replaced in PE 3.2. Applications developed with the previous checkpointing function will likely need to be rewritten, and at a minimum will need to be recompiled to use with PE 3.2.

As a result, the previous POE checkpoint compiler scripts are no longer used and have been removed from PE 3.2:

- **mpcc_chkpt**
- **mpCC_chkpt**
- **mpxlf_chkpt**
- **mpxlf90_chkpt**
- **mpxlf95_chkpt**

Also, the **MP_CHECKDIR** and **MP_CHECKFILE** environment variables from previous releases are no longer used.

In addition, checkpoint files from previous versions are not compatible with PE 3.2 and must not be used to restart a job.

pedb removed

The Motif based X-Windows debugger, **pedb**, has been removed from PE 3.2. Previous versions of the *ppe.pedb* fileset should not be used with PE 3.2 filesets, and are no longer supported with PE 3.2. For this reason, the POE option **MP_DEBUG_LOG**, that supported **pedb**, is no longer available, and customers should remove it from execution scripts and command files:

Also, the **pedb** command should not be used if it is installed from previous PE versions.

64-bit application support

PE 3.2 now fully supports both 64-bit and 32-bit threaded applications, using the **-q64** compiler option. Please see the threaded POE compiler script descriptions (like **mpcc_r**) for more details. Non-threaded applications still must run as 32-bit applications.

The 64-bit and 32-bit libraries are installed together. Jobs running 32-bit and 64-bit may coexist on the same node. An MPI job must consist of all 32-bit or all 64-bit tasks. LAPI jobs can consist of a mixture of 32-bit and 64-bit tasks. 64-bit applications must be threaded and must use AIX 5L Version 5.1 and PSSP 3.4, if run on an SP system.

Fortran application support

High Performance Fortran (HPF) is no longer supported with AIX 5L Version 5.1. Therefore, the POE Fortran compiler scripts (**mpxlf**, **mpxlf90**, and **mpxlf95**) no longer use the **MP_HPF** environment variable, or check for HPF compatibility with other Fortran compilers. See *IBM Parallel Environment for AIX: Installation* for more information on Fortran application limitations.

Chapter 2. Executing parallel programs

This chapter describes the Parallel Operating Environment (POE). POE is a simple and friendly environment designed to ease the transition from serial to parallel application development and execution. POE lets you develop and run parallel programs using many of the same methods and mechanisms as you would for serial jobs. POE allows you to continue to use the standard UNIX and AIX application development and execution techniques with which you are already familiar. For example, you can redirect input and output, pipe the output of programs into **more** or **grep**, write shell scripts to invoke parallel programs, and use shell tools such as **history**. You do all these in just the same way you would for serial programs. So while the concepts and approach to writing parallel programs must necessarily be different, POE makes your working environment as familiar as possible.

This chapter describes the steps involved in compiling and executing your parallel C, C++, or Fortran programs using either an IBM RS/6000 SP, a pSeries or RS/6000 network cluster, or a mixed system.

Executing parallel programs using POE

This section discusses how to compile and execute your parallel C, C++, or Fortran programs. It leaves out the first step in any application's life cycle which is actually writing the program. For information on writing parallel programs, refer to *IBM Parallel Environment for AIX: MPI Programming Guide IBM Parallel Environment for AIX: MPL Programming and Subroutine Reference IBM Parallel Environment for AIX: Hitchhiker's Guide* , and *IBM Parallel System Support Programs for AIX: Command and Technical Reference*.

Note: If you are using POE for the first time, check that you have authorized access. See *IBM Parallel Environment for AIX: Installation* for information on setting up users.

In order to execute an MPI, MPL, or LAPI parallel program, you need to:

1. Compile and link the program using shell scripts or makefiles which call the C, C++, or Fortran compilers while linking in the Partition Manager interface and message passing subroutines.
2. Copy your executable to the individual nodes in your partition if it is not accessible to the remote nodes.
3. Set up your execution environment. This includes setting the number of tasks, and determining the method of node allocation.
4. Optionally start the Program Marker Array analysis tool. For more information about this tool, refer to "Chapter 4. Monitoring program execution using the Program Marker Array" on page 69.
5. Load and execute the parallel program on the processor nodes of your partition. You can:
   - load a copy of the same executable on all nodes of your partition. This is the normal procedure for SPMD programs.
   - individually load the nodes of your partition with separate executables. This is the normal procedure for MPMD programs.
   - load and execute a series of SPMD or MPMD programs, in job step fashion, on all nodes of your partition.

© Copyright IBM Corp. 2000, 2001

Step 1: Compile the program

As with a serial application, you must compile a parallel C, C++, or Fortran program before you can run it. Instead of using the **cc**, **xlC**, or **xlf** commands, however, you use commands that not only compile your program, but also link in the Partition Manager and message passing interface libraries. When you later invoke the program, the subroutines in these libraries enable the home node Partition Manager to communicate with the parallel tasks, and the tasks with each other.

For each of the standard compilers supported in PE (**cc**, **xlC**, **xlf**, and so on), there are two separate MPI compiler commands — one for standard PE programs, and one for threaded programs.

- To compile standard POE programs, you use the **mpcc** (c compiler), **mpCC** (C++ compiler), **mpxlf** (Fortran compiler), **mpxlf90** (Fortran 90 compiler), or **mpxlf95** (Fortran 95 compiler) command.
- To compile programs that use the threaded MPI, the compiler commands are basically the same, except that they are appended by ″**_r**″ and are **mpcc_r**, **mpCC_r**, **mpxlf_r**, **mpxlf90_r**, or **mpxlf95_r**. You can also use these commands to compile non-threaded programs with the threaded libraries.

  Threaded programs can also utilize functions to checkpoint and later restart a parallel program. For more information on checkpointing, refer to “Checkpointing and restarting programs” on page 45.

These compiler commands are actually shell scripts which call the appropriate compiler. You can use any of the **cc**, **xlC**, or **xlf** flags on these commands.

The following table shows what to enter to compile a program depending on the language in which it is written, and whether or not you want to compile with the threaded MPI libraries. For more information on these commands, see “Appendix A. Parallel environment commands” on page 73.

To compile:	With threaded MPI?	With the checkpoint/restart capability?	ENTER
a C program	NO	NO	**mpcc** *program.c* **-o** *program*
	YES	YES	**mpcc_r** *program.c* **-o** *program*
a C++ program	NO	NO	**mpCC** *program.C* **-o** *program*
	YES	YES	**mpCC_r** *program.C* **-o** *program*
a Fortran program	NO	NO	**mpxlf** *program.f* **-o** *program*
	YES	YES	**mpxlf_r** *program.f* **-o** *program*
a Fortran 90 program	NO	NO	**mpxlf90** *program.f* **-o** *program*
	YES	YES	**mpxlf90_r** *program.f* **-o** *program*
a Fortran 95 program	NO	NO	**mpxlf95** *program.f* **-o** *program*
	YES	YES	**mpxlf95_r** *program.f* **-o** *program*

Notes:

1. The checkpoint/restart capability is only supported with the threaded MPI libraries.
2. Be sure to specify the **-g** flag when compiling a program for use with the parallel debugger. The **-g** flag is a standard compiler flag that produces an object file with symbol table references. These symbol table references are needed by the

Ex.1008 / Page 26 of 186
Intel Corporation et al

debugger. For more information on the **-g** option, refer to its use on the **cc** command as described in *IBM AIX 5L Version 5.1 Commands Reference*.

3. For 32-bit only, application programs compiled for use with POE are limited to eight (8) data segments. The **-bmaxdata** option cannot specify any more than 0x80000000. The actual amount available may be less, depending on whether shared memory is being used by MPI and/or LAPI.
4. The POE compiler scripts will evaluate a dollar sign ($) in a file name as if it were a shell variable, which may not produce the desired result in resolving the file name to be compiled. If your program file names contain the dollar sign, you will need to prevent the compiler scripts from evaluating it as a shell variable.

   For example, if your file name is *$foo.f*, you need to invoke the compiler script as:

   ```
   mpxlf "\\\$foo.f"
   ```

   or

   ```
   mpxlf "*foo.f"
   ```

Creating a static executable

Note: IBM discourages you from creating statically bound executables with POE. If service is ever applied that affects any of the Parallel Environment libraries, you will need to recompile your application to create a new executable that will work with the new libraries. This could lead to a lot of work and may expose you to potential problems, which would be avoided if dynamic libraries are used.

In general, to create a static executable, do the following:

1. Create an object file of your program using **cc**, **xlf**, or **xlC**. For your threaded program, use **cc_r** or **xlC_r**. Include the path **/usr/lpp/ppe.poe/include** to the message passing include files in your compilation statement. For example:

   ```
   cc -c myprog.c -I/usr/lpp/ppe.poe/include
   ```

2. Using **ld**, create the executable.

The following table shows how you create a C, C++, or Fortran static executable for IP or US.

Note: When you see **ld**, **l** represents a lower case **L**. When you see **-bl**, **l** represents an upper case **i**.

To:	For IP, Enter:	For US (SP only), Enter:
Create a C or C++ static executable	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -lmpci -lmpi -lvtd -lc -lppe -bl :/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip -lodm -lcfg**	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -bl:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpci -lmpi -lvtd -lc -lppe -bl:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us -lodm -lcfg -lhal**
Create a Fortran static executable	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -lmpci -lmpi -lvtd -lxlf90 -lxlf -lc -lppe -bl:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip -lodm -lcfg**	**ld -o** *myprog myprog.o* **/lib/crt0.o -binitfini:poe_remote_main -bnso -bl:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpci -lmpi -lvtd -lxlf90 -lxlf -lc -lppe -bl:/lib/syscalls.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us -lodm -lcfg -lhal**

Ex.1008 / Page 27 of 186
Intel Corporation et al

To:	For IP, Enter:	For US (SP only), Enter:
Create a C or C++ static executable which uses threaded MPI	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -lmpi_r -lvtd_r -lppe_r -lpthreads -lmpci_r /usr/lib/libc.a -bI:/lib/threads.exp -bI:/lib/syscalls.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip -lodm -lcfg**	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpi_r -lvtd_r -lppe_r -lpthreads -lmpci_r /usr/lib/libc.a -bI:/lib/threads.exp -bI:/lib/syscalls.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us -lodm -lcfg -lhal_r**
Create a Fortran static executable which uses threaded MPI	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -lmpci_r -lmpi_r -lvtd_r -lxlf90_r -lc_r -lppe_r -lc -lpthreads /usr/lib/libc.a -bI:/lib/syscalls.exp -bI:/lib/threads.exp -L/usr/lib/threads -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/ip -lodm -lcfg**	**ld -o** *myprog myprog.o* **/lib/crt0_r.o -binitfini:poe_remote_main -bnso -bI:/usr/lpp/ssp/css/libus/fs_ext.exp -lmpci_r -lmpi_r -lvtd_r -lxlf90_r -lc_r -lppe_r -lc -lpthreads /usr/lib/libc.a -bI:/lib/syscalls.exp -bI:/lib/threads.exp -L/usr/lpp/ppe.poe/lib -L/usr/lpp/ppe.poe/lib/us -lodm -lcfg -lhal_r**

Notes:

1. POE compile scripts utilize the **-binitfini** binder option. As a result, POE programs have a priority default of zero. If other user applications are using the **initfini** binder option, they should only specify a priority in the range of 1 to 2,147,483,647.
2. If you try to create a US static executable on the SP control workstation, and the **ld** command fails because it cannot find the **mpci** library file, it is possible that a link needs to be set by your system administrator. Refer to *IBM Parallel Environment for AIX: Installation* for instructions on installing PE on the SP control workstation.
3. On a cluster, you can create an IP static executable only. The US libraries are only shipped with an SP system.
4. When creating a Fortran static executable, include the **xlf90** and **xlf** libraries in the **ld** command after the **-lvtd** statement.
5. To create a static executable of a program which uses LAPI subroutines, see “Understanding and Using the Communications Low-Level Application Programming Interface (LAPI)”, in *IBM Parallel System Support Programs for AIX: Administration Guide*

Step 2: Copy files to individual nodes

Note: You only need to perform this step if your executable, your data files, and (if you plan to use **pdbx**) your source code files are not in a commonly accessed, or *shared*, file system. If running on an SP system, you can also skip this step if the needed files are part of a file collection which is distributed automatically. For information on using file collections, see *IBM Parallel System Support Programs for AIX: Administration Guide* For more information on the parallel debuggers, see *IBM Parallel Environment for AIX: Operation and Use, Volume 2*.

If the program you are running is in a shared file system, the Partition Manager loads a copy of your executable in each processor node in your partition when you invoke a program. If your executable is in a private file system, however, you must copy it to the nodes in your partition. If you plan to use the parallel debugger **pdbx**, you must copy your source files to all nodes as well.

Ex.1008 / Page 28 of 186
Intel Corporation et al

You can copy your executable to each node with the **mcp** command. **mcp** uses the message passing facilities of the Parallel Environment to copy a file from a file system on the home node to a remote node file system. For example, assume that your executable program is on a mounted file system (*/u/edgar/somedir/myexecutable*), and you want to make a private copy in */tmp* on each node in *host.list*.

ENTER

mcp */u/edgar/somedir/myexecutable /tmp/myexecutable* **-procs** *n*

For more information on the **mcp** command, refer to "mcp" on page 74. You can also copy files to individual nodes of your partition using the PSSP commands **dsh** and **pcp**. For more information on these commands refer to *IBM Parallel System Support Programs for AIX: Administration Guide*.

Note: If you load your executable from a mounted file system, you may experience an initial delay while the program is being initialized on all nodes. You may experience this delay even after the program begins executing, because individual pages of the program are brought in on demand. This is particularly apparent during initialization of the message passing interface; since individual nodes are synchronized, there are simultaneous demands on the network file transfer system. You can minimize this delay by copying the executable to a local file system on each node, using the **mcp** message passing file copy program.

Step 3: Set up the execution environment

This step contains the following sections:

- "Step 3a: Set the MP_PROCS environment variable" on page 16
- "Step 3b: Create a host list file" on page 17
- "Step 3c: Set the MP_HOSTFILE environment variable" on page 22
- "Step 3d: Set the MP_RESD environment variable" on page 23
- "Step 3e: Set the MP_EUILIB environment variable" on page 23
- "Step 3f: Set the MP_EUIDEVICE environment variable" on page 25
- "Step 3g: Set the MP_MSG_API environment variable" on page 26
- "Step 3h: Set the MP_RMPOOL environment variable" on page 26

Before invoking your program, you need to set up your execution environment. There are a number of POE environment variables discussed throughout this book and summarized in "Appendix B. POE environment variables and command-line flags" on page 131. Any of these environment variables can be set at this time to later influence the execution of parallel programs. If your system is configured for DCE (Kerberos 5) authentication, you should, before setting any POE environment variables, obtain your DCE user credentials using the **dce_login** command. You should do this before setting any POE environment variables, because the **dce_login** command creates a new shell; any environment variables set prior to creating the new shell will not be in effect in the new shell.

This step covers those environment variables most important to successfully invoke a parallel program. When you invoke a parallel program, your home node Partition Manager checks these environment variables to determine:

- the number of tasks in your program as specified by the **MP_PROCS** environment variable.

Ex.1008 / Page 29 of 186
Intel Corporation et al

- how to allocate processor nodes for these tasks. There are two basic methods of node allocation – specific and non-specific.

  For *specific node allocation*, the Partition Manager reads an explicit list of nodes contained in a *host list* file you create. If you do not have LoadLeveler, or if you are using nodes that are not part of the LoadLeveler cluster, you must use this method of node allocation.

  For *non-specific node allocation*, you give the Partition Manager the name or number of a LoadLeveler pool. A pool name or number may also be provided in a host list file. The Partition Manager then connects to LoadLeveler, which allocates nodes from the specified pool(s) for you. For more information on LoadLeveler and LoadLeveler pools, refer to "Scenarios for allocating nodes with LoadLeveler" on page 54.

Note: If you are using an SP system, and plan to use its high performance switch adapter for communication, note that each node has been configured by your system administrator for communication using the IP communication subsystem, the US communication subsystem, or both. Any node you request through specific or non-specific node allocation must be configured for the appropriate communication subsystem library implementation. Check with your system administrator to learn which nodes were initialized for the US communication subsystem, which were initialized for the IP communication subsystem, or which nodes allow either.

The maximum number of nodes is 512. The maximum number of tasks is 4096 for User Space and 2048 for the IP communication subsystem.

There are five separate environment variables that, collectively, determine how nodes are allocated by the Partition Manager. The following description of these environment variables assumes that you are **not** submitting a job using a LoadLeveler job command file as described in "Submitting an interactive POE job using an IBM LoadLeveler command file" on page 59. If you do intend to use a LoadLeveler job command file, be aware that, in order to avoid conflicting allocation specifications made via POE environment variables/command-line flags, LoadLeveler job command file statements, and POE host list file entries, certain settings will be ignored or will cause errors. The following information, therefore, assumes that you are not using a LoadLeveler job command file. Also keep in mind that, while the following environment variables are the only ones you must set to allocate nodes, there are many other environment variables you can set. These are summarized in "Appendix B. POE environment variables and command-line flags" on page 131, and control such things as standard I/O handling and message passing information. The environment variables for node allocation are:

MP_HOSTFILE
which specifies the name of a host list file to use for node allocation. If set to an empty string (*" "*) or to the word "*NULL*", this environment variable specifies that no host list file should be used. If **MP_HOSTFILE** is not set, POE looks for a file *host.list* in the current directory. You need to create a host list file if you want specific node allocation.

MP_RESD
which specifies whether or not the Partition Manager should connect to LoadLeveler to allocate nodes.

Note: When running POE from a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the

external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).

MP_EUILIB
which specifies the communication subsystem library implementation to use – either the IP communication subsystem implementation or the User Space (US) communication subsystem implementation. The IP communication subsystem library uses Internet Protocol for communication among processor nodes, while the US communication subsystem library lets you drive an SP system's high performance switch directly from your parallel tasks, without going through the kernel or operating system. For US communication on an SP system, you must have the high performance switch feature.

MP_EUIDEVICE
which specifies the adapter set you want to use for IP communication among processor nodes. The Partition Manager only checks this if you are using the IP communication subsystem implementation with LoadLeveler. It does not check this if you are using a network cluster. If **MP_RESD**=*no*, the value of **MP_EUIDEVICE** is ignored.

MP_RMPOOL
which specifies the name or number of a LoadLeveler pool. The Partition Manager only checks this if you are using LoadLeveler without a host list file. You can use the **llstatus** command to return information about LoadLeveler pools. To use **llstatus** on a workstation that is external to the LoadLeveler system, the *LoadL.so* fileset must be installed on the external node. For more information, see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation*.

The remainder of this step consists of sub-steps describing how to set each of these environment variables, and how to create a host list file. Depending on the hardware and message passing library you are using, and the method of node allocation you want, some of the sub-steps that follow may not apply to you. For this reason, pay close attention to the task variant tables at the beginning of many of the sub-steps. They will tell you whether or not you need to perform the sub-step.

For further clarification, the following tables summarize the procedure for determining how nodes are allocated. The tables describe the possible methods of node allocation available to you, to what each environment variable must be set, and whether or not you need to create a host list file. As already stated, this section assumes that you are **not** using a LoadLeveler job command file and, therefore, the **MP_LLFILE** environment variable (or its associated command-line flag **-llfile**) is **not** set. To allocate nodes using a LoadLeveler job command file, refer to "Submitting an interactive POE job using an IBM LoadLeveler command file" on page 59 or the manual *Using and Administering LoadLeveler*. To make the procedure of setting up the execution environment easier and less prone to error, you may eventually wish to create a shell script which automates some of the environment variable settings. To allocate nodes of an SP system, see Table 1 on page 14. If you are using a pSeries or RS/6000 network cluster, or a mixed system and want to allocate some nodes that are not part of the LoadLeveler cluster, see Table 2 on page 15.

Table 1. Execution setup summary (for an SP system).

	If you want to use the US communication subsystem library for communication among parallel tasks and...		**If you want to use the IP communication subsystem library for communication among parallel tasks and...**	
	you want specific node allocation:	**you want non-specific node allocation from a single LoadLeveler pool:**	**you want specific node allocation:**	**you want non-specific node allocation from a single LoadLeveler pool:**
a host list file is...	required	not required. If used, however, all entries must specify the same LoadLeveler pool.	required	not required. If used, however, all entries must specify the same LoadLeveler pool.
MP_HOSTFILE	Should be set to the name of your host list file. If not set, the host list file is assumed to be *host.list* in the current directory.	No host list file is required. If none is used, **MP_HOSTFILE** should be set to an empty string (″″) or the word ″NULL″.	Should be set to the name of your host list file. If not set, the host list file is assumed to be *host.list* in the current directory.	No host list file is required. If none is used, **MP_HOSTFILE** should be set to an empty string (″″) or the word ″NULL″.
MP_RESD	Should be set to *yes*. If set to an empty string (″″), or if not set, the Partition Manager assumes the value of **MP_RESD** is *yes*.	Should be set to *yes*. If set to an empty string (″″), or if not set, the Partition Manager assumes the value of **MP_RESD** is *yes*.	Should be set to *yes*. If set to an empty string (″″), or if not set, the Partition Manager assumes the value of **MP_RESD** is *no*.	Should be set to *yes*. If set to an empty string (″″), or if not set, the Partition Manager assumes the value of **MP_RESD** is *yes*.
MP_EUILIB	*us*	*us*	*ip*	*ip*
MP_EUIDEVICE	*css0* (the high performance switch). However, the actual value is ignored when **MP_EUILIB** is set to *us*.	*css0* (the high performance switch). However, the actual value is ignored when **MP_EUILIB** is set to *us*.	should specify the adapter type. A valid, case-sensitive, value is *css0* (the high performance switch). Note that the **MP_EUIDEVICE** value is only used when the value of **MP_EUILIB** is *ip*.	should specify the adapter type. A valid, case-sensitive, value is *css0* (the high performance switch). Note that the **MP_EUIDEVICE** value is only used when the value of **MP_EUILIB** is *ip*.
MP_RMPOOL	is ignored because you are using a host list file.	if you are not using a host list file, **MP_RMPOOL** should be set to the name or number of a LoadLeveler pool. If you are using a host list file, **MP_RMPOOL** is ignored; you must specify the pool in the host list file.	is ignored because you are using a host list file.	if you are not using a host list file, **MP_RMPOOL** should be set to the name or number of a LoadLeveler pool. If you are using a host list file, **MP_RMPOOL** is ignored; you must specify the pool in the host list file.

Note: This preceding table assumes that the **MP_LLFILE** environment variable is not set, and the **-llfile** flag is not used. If the **MP_LLFILE** environment variable (or its associated command-line flag) is used, indicating that a LoadLeveler job command file should participate in node allocation, be aware that some of the environment variables shown in this table will be

ignored. The reason they will be ignored is to avoid conflicting allocation specifications made via POE environment variables/command-line flags, POE host list file entries, and LoadLeveler job command file statements. For more information on the POE environment variables that will be ignored when a LoadLeveler job command file is used, refer to “Submitting an interactive POE job using an IBM LoadLeveler command file” on page 59.

Table 2. Execution environment setup summary (for pSeries or RS/6000 network cluster or a mixed system whose additional nodes are not part of the LoadLeveler cluster)

A host list file...	is used.
MP_HOSTFILE	should be set to the name of a host list file. If not defined, the host list file is assumed to be *host.list* in the current directory.
MP_RESD	should be set to *no*.
MP_EUILIB	should be set to *ip*.
MP_RMPOOL	is not used because you are using a host list file.

The following table shows how nodes will be allocated depending on the value of the environment variables discussed in this step. It is provided here for additional illustration. Refer to it in situations when the environment variables are set in patterns other than those suggested in Table 1 on page 14, and Table 2. When reading the following table, be aware that, if a LoadLeveler job command file is specified (using the **MP_LLFILE** environment variable or the **-llfile** flag), the value of **MP_RESD** will be *yes*.

Table 3. Node allocation summary

If			**Then**		
The value of MP_EUILIB is:	**The value of MP_RESD is:**	**Your Host List file contains a list of:**	**The allocation mode will be:**	**The communication subsystem library implementation used will be:**	**The message passing address used will be:**
ip	-	nodes	Node_List	IP	Nodes
		pools	LL_List	IP	MP_EUIDEVICE
		NULL	LL	IP	MP_EUIDEVICE
	yes	nodes	LL_List	IP	MP_EUIDEVICE
		pools	LL_List	IP	MP_EUIDEVICE
		NULL	LL	IP	MP_EUIDEVICE
	no	nodes	Node_List	IP	Nodes
		pools	Error	-	-
		NULL	Error	-	-

Table 3. Node allocation summary (continued)

If			Then		
us	-	nodes	LL_List	US	N/A
		pools	LL_List	US	N/A
		NULL	LL	US	N/A
	yes	nodes	LL_List	US	N/A
		pools	LL_List	US	N/A
		NULL	LL	US	N/A
	no	nodes	Error	-	-
		pools	Error	-	-
		NULL	Error	-	-
-	-	nodes	Node_List	IP	Nodes
		pools	LL_List	IP	MP_EUIDEVICE
		NULL	LL	US	N/A
	yes	nodes	LL_List	US	N/A
		pools	LL_List	US	N/A
		NULL	LL	US	N/A
	no	nodes	Node_List	IP	Nodes
		pools	Error	-	-
		NULL	Error	-	-

Note:

Node_List means that the host list file is used to create the partition.

LL_List means that the host list file is used to create the partition, but the nodes are requested from LoadLeveler.

LL means that the partition is created by requesting nodes in **MP_RMPOOL** from LoadLeveler.

Nodes indicates that the external IP address of the processor node is used for communication.

MP_EUIDEVICE indicates that the IP adapter address indicated by **MP_EUIDEVICE** is used for communication.

Step 3a: Set the MP_PROCS environment variable

Before you execute a program, you need to set the size of the partition. To do this, use the **MP_PROCS** environment variable or its associated command-line flag **-procs**. For example, say you want to specify the number of task processes as 6. You could:

Set the MP_PROCS environment variable:	Use the -procs flag when invoking the program:
ENTER **export MP_PROCS=6**	**ENTER** **poe** *program* **-procs 6**

If you do not set **MP_PROCS**, the default number of task processes is 1 unless you have set the **MP_RMPOOL** environment variable (or **-rmpool** command-line flag) for non-specific node allocation from a single LoadLeveler pool, and have set both the **MP_NODES** and **MP_TASKS_PER_NODE** environment variables (or their associated command-line flags) to further specify how LoadLeveler should allocate nodes within the pool. In such cases, if **MP_PROCS** is not set, the parallel job will

Ex.1008 / Page 34 of 186
Intel Corporation et al

consist of **MP_TASKS_PER_NODE** multiplied by **MP_NODES** tasks. See “Step 3h: Set the MP_RMPOOL environment variable” on page 26 for more details.

Step 3b: Create a host list file

You need to create a host list file if:	You do not need to create a host list file if:
• you are using a pSeries or RS/6000 network cluster. • you are using a mixed system which consists of some nodes not part of the LoadLeveler cluster.	you are using an SP system (or a mixed system whose additional processor nodes are part of the LoadLeveler cluster) and want non-specific node allocation.

A host list file specifies the processor nodes on which the individual tasks of your program should run. When you invoke a parallel program, your Partition Manager checks to see if you have specified a host list file. If you have, it reads the file to allocate processor nodes. The procedure for creating a host list file differs depending on whether you are using a pSeries or RS/6000 network cluster, a LoadLeveler cluster, an SP system, or a mixed system. If you are using a pSeries or RS/6000 network cluster, see “Creating a host list file to allocate nodes of a cluster”. If you are using a LoadLeveler cluster, an SP system, or a mixed system, see “Creating a host list file to allocate nodes of an SP system” on page 18.

Creating a host list file to allocate nodes of a cluster: If you are using a pSeries or RS/6000 cluster, a host list file simply lists a series of host names – one per line. These must be the names of remote nodes accessible from the Home Node. Lines beginning with an exclamation point (!) or asterisk (*) are comments. The Partition Manager ignores blank lines and comments. The host list file can list more names than are required by the number of program tasks. The additional names are ignored.

To understand how the Partition Manager uses a host list file to determine the nodes on which your program should run, consider the following example host list file:

```
! Host list file for allocating 6 tasks

* An asterisk may also be used to indicate a comment



host1_name

host2_name

host3_name

host4_name

host5_name

host6_name
```

The Partition Manager ignores the first two lines because they are comments, and the third line because it is blank. It then allocates *host1_name* to run task 0, *host2_name* to run task 1, *host3_name* to run task 2, and so on. If any of the processor nodes listed in the host list file are unavailable when you invoke your program, the Partition Manager returns a message stating this and does not run your program.

You can also have multiple tasks of a program share the same node by simply listing the same node multiple times in your host list file. For example, say your host list file contains the following:

```
host1_name

host2_name

host3_name

host1_name

host2_name

host3_name
```

Tasks 0 and 3 will run on *host1_name*, tasks 1 and 4 will run on *host2_name*, and tasks 2 and 5 will run on *host3_name*.

Creating a host list file to allocate nodes of an SP system: If you are using an SP system (or a mixed system whose additional nodes are part of the LoadLeveler cluster), you can use a host list file for either:

- non-specific node allocation from one system pool only.
- specific node allocation. If you are using a mixed system whose additional nodes are not part of the LoadLeveler cluster, you must use specific node allocation.

In either case, the host list file can contain a number of records – one per line. For specific node allocation, each record indicates a processor node. For non-specific node allocation you can have one system pool only. Your host list file cannot contain a mixture of node and pool requests, so you must use one method or the other. The host list file can contain more records than required by the number of program tasks. The additional records are ignored.

For specific node allocation: Each record is either a host name or IP adapter address of a specific processor node of the SP system. If you are using a mixed system and want to allocate nodes that are not part of the LoadLeveler cluster, you must request them by host name. Lines beginning with an exclamation point (!) or asterisk (*) are comments. The Partition Manager ignores blank lines and comments.

To understand how the Partition Manager uses a host list file to determine the SP system nodes on which your program should run, consider the following representation of a host list file.

```
! Host list file for allocating 6 tasks

host1_name
host2_name
host3_name
9.117.8.53
9.117.8.53
9.117.8.53
```

The Partition Manager ignores the first line because it is a comment, and the second because it is blank. It then allocates *host1_name* to run task 0, *host2_name* to run task 1, *host3_name* to run task 2, and so on. The last three nodes are requested by adapter IP address using dot decimal notation.

Note: If any of the processor nodes listed in the host list file are unavailable when you invoke your program, the Partition Manager returns a message stating this and does not run your program.

For non-specific node allocation from a LoadLeveler pool: After installation of a LoadLeveler cluster, your system administrator divides its processor nodes into a number of pools. With LoadLeveler, each pool has an identifying pool name or number. Using LoadLeveler for non-specific node allocation, you need to supply the appropriate pool name or number. When specifying pools in a host list file, each entry must be for the same pool.

If you require information about LoadLeveler pools, use the command **llstatus**. To use **llstatus** on a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* for more information).

ENTER

llstatus -l (lower case L)

LoadLeveler lists status information including the pools in the LoadLeveler cluster.

For more information on the **llstatus** command and LoadLeveler pools, refer to *Using and Administering LoadLeveler*.

When specifying LoadLeveler pools in a host list file, each entry must refer to the same pool (by name or number), and should be preceded by an at symbol (@). Lines beginning with an exclamation point (!) and asterisk (*) are comments. The Partition Manager ignores blank lines and comments.

To understand how the Partition Manager uses a host list file for non-specific node allocation, consider the following example host list file:

```
! Host list file for allocating 3 tasks with LoadLeveler

@6
@6
@6
```

The Partition Manager ignores the first line because it is a comment, and the second line because it is blank. The at (@) symbols tell the Partition Manager that these are pool requests. It connects to LoadLeveler to request three nodes from pool 6.

Notes:

1. If there are insufficient nodes available in a requested pool when you invoke your program, the Partition Manager returns a message stating this, and does not run your program.
2. If the number of program tasks is greater than the number of records in the host list file, the last record in the file is used for the remaining requests.

Specifying how a node's resources are used: When requesting nodes of an SP system using specific node allocation, you can optionally request how each node's resources – its adapter and CPU – should be used. You can specify:

- Whether the node's adapter is to be *dedicated* or *shared*. If dedicated, only a single program task can use it for the same protocol. If *shared*, a number of tasks on that node can use it. (see Table 4 on page 20).

Ex.1008 / Page 37 of 186
Intel Corporation et al

- Whether the node's CPU usage should be *unique* or *multiple*. If *unique*, only your program's tasks can use the CPU. If *multiple*, your program may share the node with other users.

When using LoadLeveler for non-specific node allocation, any usage specification in the host list file will be ignored. Instead, you can request how nodes are used with the **MP_CPU_USE** and/or **MP_ADAPTER_USE** environment variables (or their associated command line options) or you can specify this information in a LoadLeveler Job Command File.

Using the environment variables **MP_ADAPTER_USE** and **MP_CPU_USE**, or the associated command line options (**-adapter_use** and **-cpu_use**) to make either or both of these specifications will affect the resource usage for each node allocated from the pool specified using **MP_RMPOOL** or **-rmpool**. For example, if you wanted nodes from pool 5, and you wanted your program to have exclusive use of both the adapter and CPU, the following command line could be used:

poe [*program*] *-rmpool 5 -adapter_use d*[*edicated*]

-cpu_use u[*nique*] [*more_poe_options*]

Associated environment variables (**MP_RMPOOL**, **MP_ADAPTER_USE**, **MP_CPU_USE**) could also be used to specify any or all of the options in this example.

Note: You can also use a LoadLeveler job command file to specify how a node's resources are used. If you use a LoadLeveler job command file, the **MP_RMPOOL**, **MP_ADAPTER_USE**, and **MP_CPU_USE** environment variables will be validated but ignored. For more information about LoadLeveler job command files, refer to *IBM LoadLeveler for AIX: Using and Administering*.

The following tables illustrate how node resources are used. Table 4 shows the default settings for adapter and CPU use, while Table 5 on page 21 outlines how the two separate specifications determine how the allocated node's resources are used.

Table 4. Adapter/CPU default settings

	Adapter	**CPU**
If host list file contains non-specific pool requests:	Dedicated	Unique
If host list file requests specific nodes:	Shared [1]	Multiple
If host list file is not used:	Dedicated[2]	Unique[3]
Note: [1] For US jobs, adapter is dedicated. [2] For IP jobs, adapter is shared. [3] For IP jobs, CPU is multiple.		

Ex.1008 / Page 38 of 186
Intel Corporation et al

Table 5. Adapter/CPU use under LoadLeveler

	If the Node's CPU is "Unique":	**If the Node's CPU is "Multiple":**
If the adapter use is "Dedicated":	Intended for production runs of high performance applications. Only the tasks of that parallel job use the adapter and CPU.	The adapter you specified with **MP_EUIDEVICE** is dedicated to the tasks of your parallel job. However, you and other users still have access to the CPU through another adapter.
If the adapter use is "Shared":	Only your program tasks have access to the node's CPU, but other program's tasks can share the adapter.	Both the adapter and CPU can be used by a number of your program's tasks and other users.

Notes:

1. When using LoadLeveler, the US communication subsystem library does not require dedicated use of the SP switch on the node. Adapter use will be defaulted, as in Table 4 on page 20, but shared usage may be specified.
2. Adapter/CPU usage specification is only enforced for jobs using LoadLeveler for node allocation.

Generating an output host list file: When running parallel programs using LoadLeveler, you can generate an output host list file of the nodes that LoadLeveler allocated. When you have LoadLeveler perform non-specific node allocation from SP system pools, this enables you to learn which nodes were allocated. This information is vital if you want to perform some postmortem analysis or file cleanup on those nodes, or if you want to rerun the program using the same nodes. To generate a host list file, set the **MP_SAVEHOSTFILE** environment variable to a file name. You can specify this using a relative or full path name. As with most POE environment variables, you can temporarily override the value of **MP_SAVEHOSTFILE** using its associated command-line flag **-savehostfile**. For example, to save LoadLeveler's node allocation into a file called */u/hinkle/myhosts*, you could:

Set the MP_SAVEHOSTFILE environment variable:	**Use the -savehostfile flag when invoking the program:**
ENTER **export MP_SAVEHOSTFILE=/u/hinkle/myhosts**	**ENTER** **poe** *program* **-savehostfile /u/hinkle/myhosts**

Each record in the output host list file will be the original non-specific pool request. Following each record will be comments indicating the specific node that was allocated. The specific node is identified by:

- hostname
- external IP address
- adapter IP address (which may be the same as the external IP address)

For example, say the input host list file contains the following records:

```
@mypool
@mypool
@mypool
```

The following is a representation of the output hostlist file.

Ex.1008 / Page 39 of 186
Intel Corporation et al

```
host1_name
! 9.117.11.47                  9.117.8.53

!@mypool
host1_name
! 9.117.11.47                  9.117.8.53

!@mypool
host1_name
! 9.117.11.47                  9.117.8.53

!@mypool
```

Note: The name of your output host list file can be the same as your input host list file. If a file of the same name already exists, it is overwritten by the output host list file.

Step 3c: Set the MP_HOSTFILE environment variable

You need to set the MP_HOSTFILE environment variable if:	You do not need to set the MP_HOSTFILE environment variable if:
• you are using a host list file other than the default *./host.list* • you are requesting non-specific node allocation without a host list file.	If your host list file is the default *./host.list*

The default host list file used by the Partition Manager to allocate nodes is called *host.list* and is located in your current directory. You can specify a file other than *host.list* by setting the environment variable **MP_HOSTFILE** to the name of a host list file, or by using either the **-hostfile** or **-hfile** flag when invoking the program. In either case, you can specify the file using its relative or full path name. For example, say you want to use the host list file *myhosts* located in the directory */u/hinkle*. You could:

Set the MP_HOSTFILE environment variable:	Use the -hostfile flag when invoking the program:
ENTER **export MP_HOSTFILE=/u/hinkle/myhosts**	**ENTER** **poe** *program* **-hostfile /u/hinkle/myhosts** or **poe** *program* **-hfile /u/hinkle/myhosts**

If you are using LoadLeveler for non-specific node allocation from a single pool specified by **MP_RMPOOL**, and a host list file exists in the current directory, you must set **MP_HOSTFILE** to an empty string or to the word "*NULL*". Otherwise the Partition Manager uses the host list file. You can either:

Set the MP_HOSTFILE environment variable:	Use the -hostfile flag when invoking the program:
ENTER **export MP_HOSTFILE=** or **export MP_HOSTFILE=***""* or **export MP_HOSTFILE=***NULL*	**ENTER** **poe** *program* **-hostfile** *""* or **poe** *program* **-hostfile** *NULL*

Step 3d: Set the MP_RESD environment variable

To indicate whether or not LoadLeveler should be used to allocate nodes, you set the **MP_RESD** environment variable to *yes* or *no*. As specified in Table 1 on page 14 and Table 2 on page 15, **MP_RESD** controls whether or not the Partition Manager connects to LoadLeveler to allocate processor nodes.

If you are allocating nodes that are **not** part of a LoadLeveler cluster, **MP_RESD** should be set to *no*. If **MP_RESD** is set to *yes*, only nodes within the LoadLeveler cluster can be allocated.

If you are allocating nodes of a pSeries or RS/6000 network cluster, you do not have LoadLeveler and therefore should set **MP_RESD** to *no*. If you are using a mixed system, you may set **MP_RESD** to *yes*. However, LoadLeveler only has knowledge of nodes that are part of the LoadLeveler cluster. If the additional pSeries or RS/6000 processors are not part of the LoadLeveler cluster, you must also use a host list file to allocate them, and cannot set **MP_RESD** to *yes* in that case.

As with most POE environment variables, you can temporarily override the value of **MP_RESD** using its associated command-line flag **-resd**. For example, to specify that you want the Partition Manager to connect LoadLeveler to allocate nodes, you could:

Set the MP_RESD environment variable:	Use the -resd flag when invoking the program:
ENTER **export MP_RESD=***yes*	**ENTER** **poe** *program* **-resd** *yes*

You can also set **MP_RESD** to an empty string. If set to an empty string, or if not set, the default value of **MP_RESD** is interpreted as *yes* or *no* depending on the context. Specifically, the value of **MP_RESD** will be determined by the value of **MP_EUILIB** and whether or not you are using a host list file. The following table shows how the context determines the value of **MP_RESD**.

MP_EUILIB setting	and you are using a host list file:	and you are not using a host list file:
If MP_EUILIB is set to ip, an empty string, the word ″NULL″, or if not set:	**MP_RESD** is interpreted as *no* by default, unless: • the host list file includes pool requests, or • the **MP_LLFILE** environment variable is set (or the **-llfile** command-line flag is used).	**MP_RESD** is interpreted as *yes* by default.
If MP_EUILIB is set to us:	**MP_RESD** is interpreted as *yes* by default.	**MP_RESD** is interpreted as *yes* by default.

Note: When running POE from a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).

Step 3e: Set the MP_EUILIB environment variable

During execution, the tasks of your program can communicate via calls to message passing routines. The message passing routines in turn call communication subsystem library routines which enable the processor nodes to exchange the

Ex.1008 / Page 41 of 186
Intel Corporation et al

message data. Before you invoke your program, you need to decide which communication subsystem library implementation you wish to use – the Internet Protocol (IP) communication subsystem or the User Space (US) communication subsystem.

- *The IP communication subsystem library implementation* uses the Internet Protocol for communication among processor nodes. If you do not have the high performance switch feature, you must use the IP communication subsystem.
- *The US communication subsystem library implementation* uses the User Space protocol for dedicated use of a high performance communication adapter. Programs that use LAPI must set **MP_EUILIB** (or **-euilib**) to *us*. It allows you to drive the switch adapter directly from your parallel tasks. You can only use the US communication subsystem when running on an SP system configured with the high performance switch feature.

The **MP_EUILIB** environment variable, or its associated command-line flag **-euilib**, is used to indicate which communication subsystem library implementation you are using. POE needs to know which communication subsystem implementation to dynamically link in as part of your executable when you invoke it. If you want the IP communication subsystem, **MP_EUILIB** or **-euilib** should specify *ip*. If you want the US communication subsystem, **MP_EUILIB** or **-euilib** should specify *us*. In either case, the specification is case-sensitive.

For example, say you want to dynamically link in the communication subsystem library at execution time. You could:

Set the MP_EUILIB environment variable:	**Use the -euilib flag when invoking the program:**
ENTER **export MP_EUILIB=ip or us**	**ENTER** **poe** *program* **-euilib ip or us**

Note: When you invoke a parallel program, your Partition Manager checks the value of **MP_EUILIB** and then looks to the directory */usr/lpp/ppe.poe/lib* for the message passing interface and the communication subsystem library implementation. If you are running on a pSeries or RS/6000 network cluster, this is the actual location of the message passing interface. If you are running on an SP system, */usr/lpp/ppe.poe/lib* contains symbolic links to the actual location. Consult your system administrator for the actual location of the message passing library if necessary.

You can make POE look to a directory other than */usr/lpp/ppe.poe/lib* by setting the **MP_EUILIBPATH** environment variable or its associated command-line flag **-euilibpath**. For example, say the communication subsystem library implementations were moved to */usr/altlib*. To instruct the Partition Manager to look there, you could:

Set the MP_EUILIBPATH environment variable:	**Use the -euilibpath flag when invoking the program:**
ENTER **export MP_EUILIBPATH=/usr/altlib**	**ENTER** **poe** *program* **-euilibpath /usr/altlib**

The expected library for loading the communication subsystem library implementation is in directory */usr/lpp/ppe.poe/lib/***$MP_EUILIB**. Setting the **MP_EUILIBPATH** environment variable causes POE to try to load the communication subsystem library from the directory **$MP_EUILIBPATH/$MP_EUILIB**. If the communication subsystem library (*libmpci.a*)

Ex.1008 / Page 42 of 186
Intel Corporation et al

is not in the requested path, it will be loaded from the library path for the IP communication subsystem library implementation used when the program was compiled – **$MP_PREFIX**/*ppe.poe/lib/ip*. **MP_PREFIX** can be set by the user, but is normally */usr/lpp*. Thus the default library path is normally */usr/lpp/ppe.poe/lib/ip*, provided the library is not specified by the **MP_EUILIB** and/or **MP_EUILIBPATH** environment variables.

Step 3f: Set the MP_EUIDEVICE environment variable

You need to set the MP_EUIDEVICE environment variable if:	You do not need to set the MP_EUIDEVICE environment variable if:
you have set the **MP_EUILIB** environment variable to *ip*, and are using LoadLeveler for node allocation.	you have set the **MP_EUILIB** environment variable to *us*. The Partition Manager assumes that **MP_EUIDEVICE** is *css0* – the high performance communication adapter.

If you are using LoadLeveler and the IP communication subsystem library implementation for communication among parallel tasks on an SP system, you can specify which adapter set to use for message passing – either Ethernet, FDDI, token-ring, or the high performance switch. The **MP_EUIDEVICE** environment variable and its associated command-line flag **-euidevice** are used to select an alternate adapter set for communication among processor nodes. If neither **MP_EUIDEVICE** device nor the **-euidevice** flag is set, the communication subsystem library uses the external IP address of each remote node. The following table shows the possible, case-sensitive, settings for **MP_EUIDEVICE**.

Setting the MP_EUIDEVICE environment variable to:	Selects:
en0	The Ethernet adapter
fi0	The FDDI adapter
tr0	The token-ring adapter
css0	The high performance switch adapter
csss	The SP switch 2 adapter

For example, say you want to use IP over the high performance switch. The nodes have been initialized for IP as described in *IBM Parallel System Support Programs for AIX: Installation and Migration Guide* , and you have already set the **MP_EUILIB** environment variable to *ip*. To specify the high performance switch, you could:

Set the MP_EUIDEVICE environment variable:	Use the -euidevice flag when invoking the program:
ENTER **export MP_EUIDEVICE=css0**	**ENTER** **poe** *program* **-euidevice css0**

Notes:

1. If you do not set the **MP_EUIDEVICE** environment variable, the default is the adapter set used as the external network address.
2. If using LoadLeveler for node allocation, the adapters must be configured in LoadLeveler. See *Using and Administering LoadLeveler* for more information.
3. When running parallel jobs with the User Space protocol specified by the **MP_EUILIB/-euilib** combination, if **MP_EUIDEVICE/-euidevice** is set to *en0*, *fi0*, or *tr0*, the value is ignored and the default of *csss* will be used.

   Existing User Space applications that set **MP_EUIDEVICE/-euidevice** to *css0* on systems using two SP switch 2 adapters will not benefit from the

Ex.1008 / Page 43 of 186
Intel Corporation et al

performance improvements provided by using the *csss* value. In this case, you may want to change the **MP_EUIDEVICE/-euidevice** settings for such applications.

Step 3g: Set the MP_MSG_API environment variable

The **MP_MSG_API** environment variable, or its associated command line option, is used to indicate to POE which message passing API is being used by the parallel tasks.

You need to set the MP_MSG_API environment variable if:	You do not need to set the MP_MSG_API environment variable if:
A parallel task is using LAPI alone or in conjunction with MPI.	A parallel task is using MPI only.

Step 3h: Set the MP_RMPOOL environment variable

You need to set the MP_RMPOOL environment variable if:	You do not need to set the MP_RMPOOL environment variable if:
You are allocating nodes using LoadLeveler and want non-specific node allocation from a single pool.	You are allocating nodes using a host list file.

After installation of a LoadLeveler cluster or SP system, your system administrator divides its processor nodes into a number of pools. Each pool has an identifying pool name or number. When using LoadLeveler, and you want non-specific node allocation from a single pool, you need to set the **MP_RMPOOL** environment variable to the name or number of that pool. If the value of the **MP_RMPOOL** environment variable is numeric, that pool number must be configured in LoadLeveler. If the value of **MP_RMPOOL** contains any non-numeric characters, that pool name must be configured as a *feature* in LoadLeveler.

If you need information about available LoadLeveler pools, use the command **llstatus**. To use **llstatus** on a workstation that is external to the LoadLeveler cluster, the *LoadL.so* fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).

ENTER

llstatus -l (lower case L)

LoadLeveler lists information about all LoadLeveler pools and/or features.

For more information on the **llstatus** command and on LoadLeveler pools, refer to *Using and Administering LoadLeveler*.

As with most POE environment variables, you can temporarily override the value of **MP_RMPOOL** using its associated command-line flag **-rmpool**. To specify pool 6, for example, you could:

Set the MP_RMPOOL environment variable:	Use the -rmpool flag when invoking the program:
ENTER **export MP_RMPOOL=6**	**ENTER** **poe** *program* **-rmpool 6**

For additional control over how LoadLeveler allocates nodes within the pool specified by **MP_RMPOOL** or **-rmpool**, you can use the **MP_NODES** or **MP_TASKS_PER_NODE** environment variables or their associated command line options.

- The **MP_NODES** and **MP_TASKS_PER_NODE** settings are ignored unless **MP_RMPOOL** is set and no hostfile is used. A restarted job may actually use these previously ignored settings if **MP_RMPOOL** is used when restarting. See the **poerestart** man page in Appendix A for more information.
- **MP_NODES** or **-nodes** specifies the number of physical nodes on which to run the parallel tasks. You may use it alone or in conjunction with **-tasks_per_node** and/or **-procs**, as described in Table 6 below.
- **MP_TASKS_PER_NODE** or **-tasks_per_node** specifies the number of tasks to be run on each of the physical nodes. You may use it in conjunction with **-nodes** and/or **-procs**, as described in Table 6 below, but may not use it alone.
- The maximum number of nodes is 512. The maximum number of tasks is 4096 for User Space, and 2048 for the IP communication subsystem.

Table 6. LoadLeveler node allocation

MP_PROCS set?	MP_TASKS_PER_NODE set?	MP_NODES set?	Conditions and Results
Yes	Yes	Yes	**MP_TASKS_PER_NODE** multiplied by **MP_NODES** must equal **MP_PROCS**, otherwise an error occurs.
Yes	Yes	No	**MP_TASKS_PER_NODE** must divide evenly into **MP_PROCS**, otherwise an error occurs.
Yes	No	Yes	**MP_NODES** (*n*) must be less than or equal to **MP_PROCS** (*p*). If less than, LoadLeveler will allocate one task to each node, from 0 to *n* - 1, and will then allocate a second task to each of the nodes from 0 to *n* - 1, etc., until there are *p* tasks allocated. For example, if *n* = 3 and *p* = 5, 2 tasks will run on node 0, 2 tasks will run on node 1, and 1 task will run on node 2.
Yes	No	No	The parallel job will run with the indicated number of **MP_PROCS** (*p*) on *p* nodes.
No	Yes	Yes	The parallel job will consist of **MP_TASKS_PER_NODE** multiplied by **MP_NODES** tasks.
No	Yes	No	An error occurs. **MP_NODES** or **MP_PROCS** *must* be specified with **MP_TASKS_PER_NODE**.
No	No	Yes	One parallel task will be run on each of *n* nodes.
No	No	No	One parallel task will be run on one node.
Note: The examples in this table use the environment variable setting to illustrate each of the three options. The associated command line options may also be used.			

Step 4: Start the program marker array analysis tool

To use the Program Marker Array analysis tool, start it before invoking the executable. For more information on this tool and how to start it, see “Chapter 4. Monitoring program execution using the Program Marker Array” on page 69.

Step 5: Invoke the executable

Note:

In order to perform this step, you need to have a user account on, and be able to remotely login to, each of the processor nodes. The requirements for running on the remote nodes depends on whether the nodes are configured for DCE (Kerberos 5) authentication or AIX-based authentication.

- If the system is configured to use DCE (Kerberos 5) authentication, you are required to have a *.k5login* file set up in your home directory on each of the processor nodes. Unless your DCE authentication server grants unauthenticated access, you should, before invoking your executable, first login to DCE from the home node using the **dce_login** command. For information on the *.k5login* file format, refer to the *IBM AIX 5L Version 5.1 System User's Guide: Communications and Networks*
- If the system is configured to use AIX-based authentication, you are required to have an *.rhosts* file set up in your home directory on each of the remote processor nodes. Alternatively, your user id on the home node can be authorized in the */etc/host.equiv* file on each remote node. For information on the TCP/IP *.rhosts* file format, see *IBM General Concepts and Procedures for RS/6000* and *IBM AIX 5L Version 5.1 Files Reference*.

Be aware that your system administrator may also have configured your system to support both DCE and AIX-based authentication. In such situations, POE will attempt DCE-based authentication first, and, if such authentication fails, will attempt AIX-based authentication. To determine the SP security method that has been set by your system administrator, you can use the **lsauthts** command as described in the *IBM Parallel System Support Programs for AIX: Command and Technical Reference, Volume 1*. For additional information about POE security, refer to "POE user authorization" on page 46.

The **poe** command enables you to load and execute programs on remote nodes. You can use it to:

- load and execute an SPMD program onto all nodes of your partition. For more information, see "Invoking an SPMD program" on page 29.
- individually load the nodes of your partition. This capability is intended for MPMD programs. For more information, see "Invoking an MPMD program" on page 30.
- load and execute a series of SPMD or MPMD programs, in individual job steps, on the same partition. For more information, see "Loading a series of programs as job steps" on page 32.
- run non-parallel programs on remote nodes. For more information, see "Invoking a non-parallel program on remote nodes" on page 34.

When you invoke **poe**, the Partition Manager allocates processor nodes for each task and initializes the local environment. It then loads your program, and reproduces your local environment, on each processor node. The Partition Manager also passes the option list to each remote node. If your program is in a shared file system, the Partition Manager loads a copy of it on each node. If your program is in a private file system, you will have already manually copied your executable to the nodes using the **mcp** command (or the PSSP commands **dsh** or **pcp**) as described in "Step 2: Copy files to individual nodes" on page 10. If you are using the dynamic message passing interface, the appropriate communication subsystem library implementation (IP or US) is automatically loaded at this time.

Since the Partition Manager attempts to reproduce your local environment on each remote node, your current directory is important. When you invoke **poe**, the Partition Manager will, immediately before running your executable, issue the **cd** command to your current working directory on each remote node. If you are in a local directory that does not exist on remote nodes, you will get an error as the Partition Manager attempts to change to that directory on remote nodes. Typically, this will happen when you invoke **poe** from a directory under */tmp*. We suggest that you invoke **poe** from a file system that is mounted across the system. If it is important that the current directory be under */tmp*, make sure that directory exists on all the remote nodes. If you are running in the C shell, see "Running programs under the C shell" on page 61.

Note: The Parallel Environment opens several file descriptors before passing control to the user. The Parallel Environment will not assign *specific* file descriptors other than standard in, standard out, and standard error.

Before using the **poe** command, you can first specify which programming model you are using by setting the **MP_PGMMODEL** environment variable to either *spmd* or *mpmd*. As with most POE environment variables, you can temporarily override the value of **MP_PGMMODEL** using its associated command-line flag **-pgmmodel**. For example, if you want to run an MPMD program, you could:

Set the MP_PGMMODEL environment variable:	Use the -pgmmodel flag when invoking the program:
ENTER **export MP_PGMMODEL=***mpmd*	**ENTER** **poe** *program* **-pgmmodel** *mpmd*

If you do not set the **MP_PGMMODEL** environment variable or **-pgmmodel** flag, the default programming model is SPMD.

Note: If you load your executable from a mounted file system, you may experience an initial delay while the program is being initialized on all nodes. You may experience this delay even after the program begins executing, because individual pages of the program are brought in on demand. This is particularly apparent during initialization of the message passing interface; since individual nodes are synchronized, there are simultaneous demands on the network file transfer system. You can minimize this delay by copying the executable to a local file system on each node, using the **mcp** command.

Invoking an SPMD program

If you have an SPMD program, you want to load it as a separate task on each node of your partition. To do this, follow the **poe** command with the program name and any options. The options can be program options or any of the POE command-line flags shown in "Appendix B. POE environment variables and command-line flags" on page 131. You can also invoke an SPMD program by entering the program name and any options:

ENTER

poe *program [options]*

or

program [options]

You can also enter **poe** without a program name:

ENTER

poe *[options]*

Once your partition is established, a prompt appears.

ENTER
the name of the program you want to load. You can follow the program name with any program options or a subset of the POE flags.

Note: For National Language Support, POE displays messages located in an externalized message catalog. POE checks the **LANG** and **NLSPATH** environment variables, and if either is not set, it will set up the following defaults:

- **LANG=C**
- **NLSPATH=/usr/lib/nls/msg/%L/%N**

For more information about the message catalog, see “National language support (NLS)” on page xi.

Invoking an MPMD program

Note: You must set the **MP_PGMMODEL** environment variable or **-pgmmodel** flag to invoke an MPMD program.

With an SPMD application, the name of the same executable is sent to, and runs on, each of the processor nodes of your partition. If you are invoking an MPMD application, you are dealing with more than one program and need to individually load the nodes of your partition.

For example, say you have two programs – *master* and *workers* – designed to run together and communicate via calls to message passing subroutines. The program *master* is designed to run on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The *master* program will coordinate and synchronize the execution of all the worker tasks. Neither program can run without the other, as *master* only does sends and the *workers* tasks only do receives.

You can establish a partition and load each node individually using:
- standard input (from the keyboard or redirected)
- a POE commands file

Loading nodes individually from standard input: To establish a partition and load each node individually using STDIN:

ENTER
poe *[options]*

The Partition Manager allocates the processor nodes of your partition. Once your partition is established, a prompt containing both the logical node identifier 0 and the actual host name it maps to, appears.

ENTER
the name of the program you want to load on node 0. You can follow the program name with any program options or a subset of the POE flags.

A prompt for the next node in the partition displays.

ENTER
the name of the program you want to load on each processor node as you are prompted.

When you have specified the program to run on the last node of your partition, the message “Partition loaded...” displays and execution begins.

Ex.1008 / Page 48 of 186
Intel Corporation et al

For additional illustration, the following shows the command prompts that would appear, as well as the program names you would enter, to load the example *master* and *workers* programs. This example assumes that the **MP_PROCS** environment variable is set to 5.

```
% poe

0:host1_name> master [options]

1:host2_name> workers [options]

2:host3_name> workers [options]

3:host4_name> workers [options]

4:host5_name> workers [options]

Partition loaded...
```

Note: You can use the following POE command-line flags on individual program names, but not those that are used to set up the partition.

- **-infolevel** or **-ilevel**
- **-euidevelop**

Loading nodes individually using a POE commands file: The **MP_CMDFILE** environment variable, and its associated command-line flag **-cmdfile**, let you specify the name of a POE commands file. You can use such a file when individually loading a partition – thus freeing STDIN. The POE commands file simply lists the individual programs you want to load and run on the nodes of your partition. The programs are loaded in task order. For example, say you have a typical master/workers MPMD program that you want to run as 5 tasks. Your POE commands file would contain:

```
master [options]

workers [options]

workers [options]

workers [options]

workers [options]
```

Once you have created a POE commands file, you can specify it using a relative or full path name on the **MP_CMDFILE** environment variable or **-cmdfile** flag. For example, if your POE commands file is */u/hinkle/mpmdprog*, you could:

Set the MP_CMDFILE environment variable:	Use the -cmdfile flag on the poe command:
ENTER **export MP_CMDFILE=/u/hinkle/mpmdprog**	**ENTER** **poe -cmdfile /u/hinkle/mpmdprog**

Once you have set the **MP_CMDFILE** environment variable to the name of the POE commands file, you can individually load the nodes of your partition. To do this:

ENTER
poe *[options]*

The Partition Manager allocates the processor nodes of your partition. The programs listed in your POE commands file are run on the nodes of your partition.

Loading a series of programs as job steps

By default, the Partition Manager releases your partition when your program completes its run. However, you can set the environment variable **MP_NEWJOB**, or its associated command-line flag **-newjob**, to specify that the Partition Manager should maintain your partition for multiple job steps.

For example, say you have three separate SPMD programs. The first one sets up a particular computation by adding some files to */tmp* on each of the processor nodes on the partition. The second program does the actual computation. The third program does some postmortem analysis and file cleanup. These three parallel programs must run as job steps on the same processor nodes in order to work correctly. While specific node allocation using a host list file might work, the requested nodes might not be available when you invoke each program. The better solution is to instruct the Partition Manager to maintain your partition after execution of each program completes. You can then read multiple job steps from:

- standard input
- a POE commands file using the **MP_CMDFILE** environment variable.

In either case, you must first specify that you want the Partition Manager to maintain your partition for multiple job steps. To do this, you could:

Set the MP_NEWJOB environment variable:	Use the -newjob flag on the poe command:
ENTER **export MP_NEWJOB=***yes*	**ENTER** **poe -newjob** *yes*

Notes:

1. You can only load a series of programs as job steps using the **poe** command. You cannot do this with the **pdbx** parallel debugger command.
2. **poe** is its own shell. Whether successive steps run after a step completes is a function of the exit code, as described in *IBM Parallel Environment for AIX: MPI Programming Guide*

Reading job steps from standard input: Say you want to run three SPMD programs – *setup*, *computation*, and *cleanup* – as job steps on the same partition. Assuming STDIN is keyboard entry, **MP_PGMMODEL** is set to *spmd*, and **MP_NEWJOB** is set to *yes*, you would:

ENTER

poe [*poe-options*]

The Partition Manager allocates the processor nodes of your partition, and the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER

setup [*program-options*]

The program *setup* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER

computation [*program-options*]

The program *computation* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER

cleanup [*program-options*]

The program *cleanup* executes on all nodes of your partition. When execution completes, the following prompt displays:

```
0031-503 Enter program name (or quit):
```

ENTER

quit

or

<**Ctrl-d**>

The Partition Manager releases the nodes of your partition.

Notes:

1. You can also run a series of MPMD programs in job step fashion from STDIN. If **MP_PGMMODEL** is set to *mpmd*, the Partition Manager will, after each step completes, prompt you to individually reload the partition as described in "Loading nodes individually from standard input" on page 30.
2. When **MP_NEWJOB** is *yes*, the Partition Manager, by default, looks to STDIN for job steps. However, if the environment variable **MP_CMDFILE** is set to the name of a POE commands file as described in "Reading job steps from a POE commands file", the Partition Manger will look to the commands file instead. To ensure that job steps are read from STDIN, check that the **MP_CMDFILE** environment variable is unspecified.

Multi-step STDIN for newjob mode: POE's STDIN processing model allows redirected STDIN to be passed to all steps of a newjob sequence, when the redirection is from a file. If redirection is from a pipe, POE does not distribute the input to each step, only to the first step.

Reading job steps from a POE commands file: The **MP_CMDFILE** environment variable, and its associated command-line flag **-cmdfile**, lets you specify the name of a POE commands file. If **MP_NEWJOB** is *yes*, you can have the Partition Manager read job steps from a POE commands file. The commands file in this case simply lists the programs you want to run as job steps. For example, say you want to run the three SPMD programs *setup*, *computation*, and *cleanup* as job steps on the same partition. Your POE commands file would contain the following three lines:

```
setup [program-options]

computation [program-options]

cleanup [program-options]
```

`Program-options` represent the actual values you need to specify.

If you are loading a series of MPMD programs, the POE commands file is also responsible for individually loading the partition. For example, say you had three master/worker MPMD job steps that you wanted to run as 4 tasks on the same partition. The following is a representation of what your POE commands file would contain. `Options` represent the actual values you need to specify.

```
master1 [options]

workers1 [options]

workers1 [options]

workers1 [options]

master2 [options]

workers2 [options]

workers2 [options]

workers2 [options]

master3 [options]

workers3 [options]

workers3 [options]

workers3 [options]
```

While you could also redirect STDIN to read job steps from a file, a POE commands file gives you more flexibility by not tying up STDIN. You can specify a POE commands file using its relative or full path name. Say your POE commands file is called */u/hinkle/jobsteps*. To specify that the Partition Manager should read job steps from this file rather than STDIN, you could:

Set the MP_CMDFILE environment variable:	Use the -cmdfile flag on the poe command:
ENTER **export MP_CMDFILE=/u/hinkle/jobsteps**	**ENTER** **poe -cmdfile /u/hinkle/jobsteps**

Once **MP_NEWJOB** is set to *yes*, and **MP_CMDFILE** is set to the name of your POE commands file, you would:

ENTER

poe [*poe-options*]

The Partition Manager allocates the processor nodes of your partition, and reads job steps from your POE commands file. The Partition Manager does not release your partition until it reaches the end of your commands file.

Invoking a non-parallel program on remote nodes

You can also use POE to run non-parallel programs on the remote nodes of your partition. Any executable (binary file, shell script, UNIX utility) is suitable, and it does not need to have been compiled with **mpcc**, **mpCC**, or **mpxlf**. For example, if you wanted to check the process status (using the AIX command **ps**) for all remote nodes in your partition, you would:

ENTER

poe ps

The process status for each remote node is written to standard out (STDOUT) at your home node. How STDOUT from all the remote nodes is handled at your home node depends on the output mode. See “Managing standard output (STDOUT)” on page 40 for more information.

Controlling program execution

This section describes a number of additional POE environment variables for monitoring and controlling program execution. It describes how to use the:

- **MP_EUIDEVELOP** environment variable to specify that you want to run your program in message passing develop mode. In this mode, more detailed checking of your program is performed.
- **MP_RETRY** environment variable to make POE wait for processor nodes to become available.
- **MP_RETRYCOUNT** environment variable to specify the number of times the Partition Manager should request nodes before returning.
- **MP_NOARGLIST** and **MP_FENCE** environment variable to make POE ignore arguments.
- **MP_STDINMODE** and **MP_HOLD_STDIN** environment variables to manage standard input.
- **MP_STDOUTMODE** environment variable to manage standard output.
- **MP_LABELIO** environment variable to label message output with task identifiers.
- **MP_INFOLEVEL** environment variable to specify the level of messages you want reported to standard error.
- **MP_PMDLOG** environment variable to generate a diagnostic log on remote nodes.
- **MP_IONODEFILE** environment variable to specify an I/O node file that indicates which nodes should participate in parallel I/O.
- **MP_CKPTFILE** environment variable to define the base name of the checkpoint file when checkpointing a program. See “Checkpointing and restarting programs” on page 45 for more information.
- **MP_CKPTDIR** environment variable to define the directory where the checkpoint file will reside when checkpointing a program. See “Checkpointing and restarting programs” on page 45 for more information.

For a complete listing of all POE environment variables, see “Appendix B. POE environment variables and command-line flags” on page 131.

Specifying develop mode

You can run programs in one of two modes – *develop mode* or *run mode*. In develop mode, intended for developing applications, the message passing interface performs more detailed checking during execution. Because of the additional checking it performs, develop mode can significantly slow program performance. In run mode, intended for completed applications, only minimal checking is done. While run mode is the default, you can use the **MP_EUIDEVELOP** environment variable to specify message passing develop mode. As with most POE environment variables, **MP_EUIDEVELOP** has an associated command-line flag **-euidevelop**. To specify MPI develop mode, you could:

Set the MP_EUIDEVELOP environment variable:	Use the -euidevelop flag when invoking the program:
ENTER **export MP_EUIDEVELOP=***yes*	**ENTER** **poe** *program* **-euidevelop** *yes*

To later go back to run mode, set **MP_EUIDEVELOP** to *no*.

Ex.1008 / Page 53 of 186
Intel Corporation et al

Set the MP_EUIDEVELOP environment variable:	Use the -euidevelop flag when invoking the program:
ENTER **export MP_EUIDEVELOP=***DEB*	**ENTER** **poe** *program* **-euidevelop** *DEB*

To stop parameter checking, set **MP_EUIDEVELOP** to *min*, for “minimum”.

Making POE wait for processor nodes

If you are using an SP system, and there are not enough available nodes to run your program, the Partition Manager, by default, returns immediately with an error. Your program does not run. Using the **MP_RETRY** and **MP_RETRYCOUNT** environment variables, however, you can instruct the Partition Manager to repeat the node request a set number of times at set intervals. Each time the Partition Manager repeats the node request, it displays the following message:

```
Retry allocation      ......press control-C to terminate
```

The **MP_RETRY** environment variable, and its associated command-line flag **-retry**, specifies the interval (in seconds) to wait before repeating the node request. The **MP_RETRYCOUNT** environment variable, and its associated command-line flag **-retrycount**, specifies the number of times the Partition Manager should make the request before returning. For example, if you wanted to retry the node request five times at five minute (300 second) intervals, you could:

Set the MP_RETRY and MP_RETRYCOUNT environment variables:	Use the -retry and -retrycount flags when invoking the program:
ENTER **export MP_RETRY=***300* **export MP_RETRYCOUNT=***5*	**ENTER** **poe** *program* **-retry** *300* **-retrycount** *5*

Note: If the **MP_RETRYCOUNT** environment variable or the **-retrycount** command-line flag is used, the **MP_RETRY** environment variable or the **-retry** command-line flag must be set to at least one second.

Making POE ignore arguments

When you invoke a parallel executable, you can specify an argument list consisting of a number of program options and POE command-line flags. The argument list is parsed by POE – the POE command-line flags are removed and the remainder of the list is passed on to the program. If any of your program arguments are identical to POE command-line flags, however, this can cause problems. For example, say you have a program that takes the argument **-retry**. You invoke the program with the **-retry** option, but it does not execute correctly. This is because there is also a POE command-line flag **-retry**. POE parses the argument list and so the **-retry** option is never passed on to your program. There are two ways to correct this sort of problem. You can:

- make POE ignore the entire argument list using the **MP_NOARGLIST** environment variable.
- make POE ignore a portion of the argument list using the **MP_FENCE** environment variable.

Making POE ignore the entire argument list

When you invoke a parallel executable, POE, by default, parses the argument list and removes all POE command-line flags before passing the rest of the list on to the program. Using the environment variable **MP_NOARGLIST**, you can prevent POE from parsing the argument list. To do this:

ENTER

export MP_NOARGLIST=*yes*

When the **MP_NOARGLIST** environment variable is set to *yes*, POE does not examine the argument list at all. It simply passes the entire list on to the program. For this reason, you can not use any POE command-line flags, but must use the POE environment variables exclusively. While most POE environment variables have associated command-line flags, **MP_NOARGLIST**, for obvious reasons, does not. To specify that POE should again examine argument lists, either set **MP_NOARGLIST** to *no*, or unset it.

ENTER

export MP_NOARGLIST=*no*

or

unset MP_NOARGLIST

Making POE ignore a portion of the argument list

When you invoke a parallel executable, POE, by default, parses the entire argument list and removes all POE command-line flags before passing the rest of the list on to the program. You can use a fence, however, to prevent POE from parsing the remainder of the argument list. A *fence* is simply a character string you define using the **MP_FENCE** environment variable. Once defined, you can use the fence to separate those arguments you want parsed by POE from those you do not. For example, say you have a program that takes the argument **-retry**. Because there is also a POE command-line flag **-retry**, you need to put this argument after a fence. To do this, you could:

ENTER

export MP_FENCE=*Q*

poe *program* **-procs** *26* **-infolevel** *2 Q -retry RGB*

While this example defines Q as the fence, keep in mind that the fence can be any character string. Any arguments placed after the fence are passed by POE, unexamined, to the program. While most POE environment variables have associated command-line flags, **MP_FENCE** does not.

Managing standard input, output, and error

POE lets you control standard input (STDIN), standard output (STDOUT), and standard error (STDERR) in several ways. You can continue using the traditional I/O manipulation techniques such as redirection and piping, and can also:

- determine whether a single task or all parallel tasks should receive data from STDIN.
- determine whether a single task or all parallel tasks should write to STDOUT. If all tasks are writing to STDOUT, you can further define whether or not the messages are ordered by task id.
- specify the level of messages that will be reported to STDERR during program execution.
- specify that messages to STDOUT and STDERR should be labeled by task id.

Ex.1008 / Page 55 of 186
Intel Corporation et al

Managing standard input (STDIN)

STDIN is the primary source of data going into a command. Usually, STDIN refers to keyboard input. If you use redirection or piping, however, STDIN could refer to a file or the output from another command (see “Using MP_HOLD_STDIN”). How you manage STDIN for a parallel application depends on whether or not its parallel tasks require the same input data. Using the environment variable **MP_STDINMODE** or the command-line flag **-stdinmode**, you can specify that:

- all tasks should receive the same input data from STDIN. This is *multiple input mode*.
- STDIN should be sent to a single task of your partition. This is *single input mode*.
- no task should receive input data from STDIN.

Multiple Input Mode: Setting **MP_STDINMODE** to *all* indicates that all tasks should receive the same input data from STDIN. The home node Partition Manager sends STDIN to each task as it is read.

To specify multiple input mode so all tasks receive the same input data from STDIN, you could:

Set the MP_STDINMODE environment variable:	Use the -stdinmode flag when invoking the program:
ENTER **export MP_STDINMODE=all**	**ENTER** **poe** *program* **-stdinmode all**

Note: If you do not set the **MP_STDINMODE** environment variable or use the **-stdinmode** command-line flag, multiple input mode is the default.

Single Input Mode: There are times when you only want a single task to read from STDIN. To do this, you set **MP_STDINMODE** to the appropriate task id. For example, say you have an MPMD application consisting of two programs – *master* and *workers*. The program *master* is designed to run as a single task on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The *master* program handles all I/O, so only its task needs to read STDIN. If *master* is running as task 0, you need to specify that only task 0 should receive STDIN. To do this, you could:

Set the MP_STDINMODE environment variable:	Use the -stdinmode flag when invoking the program:
ENTER **export MP_STDINMODE=0**	**ENTER** **poe** *program* **-stdinmode 0**

Using MP_HOLD_STDIN

The environment variable **MP_HOLD_STDIN** is used to defer sending of STDIN from the home node to the remote node(s) until the message passing library has been initialized. The variable must be set to “yes” when using POE to invoke a program which: (1) has been compiled with **mpcc**, **mpxlf**, or **mpCC** and their **_r** equivalents for the threaded environment, *and* (2) will be reading STDIN from other than the keyboard (redirection or piping). Failing to export this environment variable when running these programs could likely result in the user program hanging.

In addition, if a program invoked using POE has not been compiled with **mpcc**, **mpxlf**, or **mpCC**, the environment variable must not be set (or set to “no”) to ensure that STDIN is delivered to the remote node(s).

Ex.1008 / Page 56 of 186
Intel Corporation et al

To set **MP_HOLD_STDIN** correctly, you need to know the relative order of your program's use of stdin data and initialization of the message passing library.

The discussion immediately below applies to the signal handling message passing library (MPI/MPL), which is initialized before the user's executable gets control.

The subsequent section addresses the question for the threaded MPI library.

Using redirected STDIN with the non-threaded (signal) library

Note: Wherever the following description refers to a POE environment variable (starting with **MP_**), the use of the associated command line option produces the same effect, with the exception of **MP_HOLD_STDIN**, which has no associated command line option.

A POE process can use its STDIN in two ways. First, if the program name is not supplied on the command line and no command file (**MP_CMDFILE**) is specified, POE uses STDIN to resolve the names of the programs to be run as the remote tasks. Second, any "remaining" STDIN is then distributed to the remote tasks as indicated by the **MP_STDINMODE** and **MP_HOLD_STDIN** settings. In this dual STDIN model, redirected STDIN can then pose two problems:

1. If using job steps (**MP_NEWJOB=yes**), the "remaining" STDIN is always consumed by the remote tasks during the first job step.
2. If POE attempts program name resolution on the redirected STDIN, program behavior can vary when using job steps, depending on the type of redirection used and the size of the redirected STDIN.

The first problem is addressed in POE by performing a rewind of STDIN between job steps (only if STDIN is redirected from a file, for reasons beyond the scope of this document). The second problem is addressed by providing an additional setting for **MP_STDINMODE** of "none", which tells POE to only use STDIN for program name resolution. As far as STDIN is concerned, "none" ever gets delivered to the remote tasks. This provides an additional method of reliably specifying the program name to POE, by redirecting STDIN from a file or pipe, or by using the shell's here-document syntax in conjunction with the "none" setting. If **MP_STDINMODE** is not set to "none" when POE attempts program name resolution on redirected STDIN, program behavior is undefined.

The following scenarios describe in more detail the effects of using (or not using) an **MP_STDINMODE** of "none" when redirecting (or not redirecting) STDIN, as shown in the example:

```
                                  Is STDIN Redirected?

                                       Yes    No


                                 Yes    A     B

Is MP_STDINMODE set to "none"?

                                 No     C     D
```

Scenario A

POE will use the redirected STDIN for program name resolution, only if no program name is supplied on the command line (**MP_CMDFILE** is ignored when **MP_STDINMODE=none**). No STDIN is distributed to the remote tasks. No rewind of STDIN is performed when **MP_STDINMODE=none**. If **MP_HOLD_STDIN** is set to “yes”, this is ignored because no STDIN is being distributed.

Scenario B

POE will use the keyboard STDIN for program name resolution, only if no program name is supplied on the command line (**MP_CMDFILE** is ignored when **MP_STDINMODE=none**). No STDIN is distributed to the remote tasks. No rewind of STDIN is performed when **MP_STDINMODE=none** (also, STDIN is not from a file). If **MP_HOLD_STDIN** is set to “yes”, this is ignored because no STDIN is being distributed.

Scenario C

POE will use the redirected STDIN for program name resolution, if required, and will distribute “remaining” STDIN to the remote tasks. If STDIN is intended to be used for program name resolution, program behavior is *undefined* in this case, since POE was not informed of this by setting **STDINMODE** to “none” (see Problem 2 above). If STDIN is redirected from a file, POE will rewind STDIN between each job step. If **MP_HOLD_STDIN** is set to “yes”, this feature will behave accordingly.

Scenario D

POE will use the keyboard STDIN for program name resolution, if required. Any “remaining” STDIN is distributed to the remote tasks. No rewind of STDIN is performed since STDIN is not from a file. If **MP_HOLD_STDIN** is set to “yes”, it is ignored because STDIN is not redirected.

Using redirected STDIN with the threaded MPI library

If the user’s executable is compiled with the threaded MPI library, message passing initialization occurs when MPI_Init is called, not before POE gives the user program control. If MPI_Init is called before any STDIN data is read, the discussions of the previous section apply. If, however, all STDIN is read before MPI_Init is called, then **MP_HOLD_STDIN** should be set to “no”, to allow the STDIN data to be sent to the user’s executable by POE.

Managing standard output (STDOUT)

STDOUT is where the data coming from the command will eventually go. Usually, STDOUT refers to the display. If you use redirection or piping, however, STDOUT could refer to a file or another command. How you manage STDOUT for a parallel application depends on whether you want output data from one task or all tasks. If all tasks are writing to STDOUT, you can also specify whether or not output is ordered by task id. Using the environment variable **MP_STDOUTMODE**, you can specify that:

- all tasks should write output data to STDOUT asynchronously. This is *unordered output mode*.
- output data from each parallel task should be written to its own buffer, and later all buffers should be flushed, in task order, to STDOUT. This is *ordered output mode*.
- a single task of your partition should write to STDOUT. This is *single output mode*.

Unordered output mode: Setting **MP_STDOUTMODE** to *unordered* specifies that all tasks should write output data to STDOUT asynchronously. To specify unordered output mode, you could:

Ex.1008 / Page 58 of 186
Intel Corporation et al

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***unordered*	**ENTER** **poe** *program* **-stdoutmode** *unordered*

Notes:

1. If you do not set the **MP_STDOUTMODE** environment variable or use the **-stdoutmode** command-line flag, unordered output mode is the default.
2. If you are using unordered output mode, you will probably want the messages labeled by task id. Otherwise it will be difficult to know which task sent which message. See "Labeling message output" on page 42 for more information.
3. You can also specify unordered output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.
4. Although the above environment variable and Parallel Utility Function are both described as "MP_STDOUTMODE", they are each used independently for their specific purposes.

Ordered output mode: Setting **MP_STDOUTMODE** to *ordered* specifies ordered output mode. In this mode, each task writes output data to its own buffer. Later, all the task buffers are flushed, in order of task id, to STDOUT. The buffers are flushed when:

- any one of the individual task buffers fills
- execution of the program completes.
- all tasks explicitly flush the buffers by calling the MP_FLUSH or mpc_flush Parallel Utility Function.
- tasks change output mode using calls to Parallel Utility Functions. For more information on Parallel Utility Functions, refer to *IBM Parallel Environment for AIX: MPI Programming Guide*

Note: When running the parallel application under **pdbx** with **MP_STDOUTMODE** set to ordered, there will be a difference in the ordering from when the application is run directly under **poe**. The buffer size available for the application's STDOUT is smaller because **pdbx** uses some of the buffer, so the task buffers fill up more often.

To specify ordered output mode, you could:

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***ordered*	**ENTER** **poe** *program* **-stdoutmode** *ordered*

Note: You can also specify ordered output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

Single output mode: You can specify that only one task should write its output data to STDOUT. To do this, you set **MP_STDOUTMODE** to the appropriate task id. For example, say you have an SPMD application in which all the parallel tasks are

Ex.1008 / Page 59 of 186
Intel Corporation et al

sending the exact same output messages. For easier readability, you would prefer output from only one task – task 0. To specify this, you could:

Set the MP_STDOUTMODE environment variable:	Use the -stdoutmode flag when invoking the program:
ENTER **export MP_STDOUTMODE=***0*	**ENTER** **poe** *program* **-stdoutmode** *0*

Note: You can also specify single output mode from your program by calling the MP_STDOUTMODE or mpc_stdoutmode Parallel Utility Function. Refer to *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.

Labeling message output

You can set the environment variable **MP_LABELIO**, or use the **-labelio** flag when invoking a program, so that output from the parallel tasks of your program are labeled by task id. While not necessary when output is being generated in *single* mode, this ability can be useful in *ordered* and *unordered* modes. For example, say the output mode is *unordered*. You are executing a program and receiving asynchronous output messages from all the tasks. This output is not labeled, so you do not know which task has sent which message. It would be clearer if the unordered output was labeled. For example:

```
 7: Hello World

 0: Hello World

 3: Hello World

23: Hello World

14: Hello World

 9: Hello World
```

To have the messages labeled with the appropriate task id, you could:

Set the MP_LABELIO environment variable:	Use the -labelio flag when invoking the program:
ENTER **export MP_LABELIO=***yes*	**ENTER** **poe** *program* **-labelio** *yes*

To no longer have message output labeled, set the **MP_LABELIO** environment variable to *no*.

Setting the message reporting level for standard error (STDERR)

You can set the environment variable **MP_INFOLEVEL** to specify the level of messages you want from POE. You can set the value of **MP_INFOLEVEL** to one of the integers shown in the following table. The integers *0*, *1*, and *2* give you different levels of informational, warning, and error messages. The integers *3* through *6* indicate debug levels that provide additional debugging and diagnostic information. Should you require help from the IBM Support Center in resolving a PE-related problem, you will probably be asked to run with one of the debug levels. As with most POE environment variables, you can override **MP_INFOLEVEL** when you invoke a program. This is done using either the **-infolevel** or **-ilevel** flag followed by the appropriate integer.

This integer:	Indicates this level of message reporting:	In other words:
0	Error	Only error messages from POE are written to STDERR.
1	Normal	Warning and error messages from POE are written to STDERR. This level of message reporting is the default.
2	Verbose	Informational, warning, and error messages from POE are written to STDERR.
3	Debug Level 1	Informational, warning, and error messages from POE are written to STDERR. Also written is some high-level debugging and diagnostic information.
4	Debug Level 2	Informational, warning, and error messages from POE are written to STDERR. Also written is some high- and low-level debugging and diagnostic information.
5	Debug Level 3	Debug level 2 messages plus some additional loop detail.
6	Debug Level 4	Debug level 3 messages plus other informational error messages for the greatest amount of diagnostic information.

Let us say you want the POE message level set to verbose. The following table shows the two ways to do this. You could:

Set the MP_INFOLEVEL environment variable:	Use the -infolevel flag when invoking the program:
ENTER **export MP_INFOLEVEL**=*2*	**ENTER** **poe** *program* **-infolevel** *2* or **poe** *program* **-ilevel** *2*

As with most POE command-line flags, the **-infolevel** or **-ilevel** flag temporarily override their associated environment variable.

Generating a diagnostic log on remote nodes

Using the **MP_PMDLOG** environment variable, you can also specify that diagnostic messages should be logged to a file in */tmp* on each of the remote nodes of your partition. The log file is named *mplog.jobid.n* where *jobid* is a unique job identifier. The *jobid* will be the same for all remote nodes. Should you require help from the IBM Support Center in resolving a PE-related problem, you will probably be asked to generate these diagnostic logs.

The ability to generate diagnostic logs on each node is particularly useful for isolating the cause of abnormal termination, especially when the connection between the remote node and the home node Partition Manager has been broken. As with most POE environment variables, you can temporarily override the value of **MP_PMDLOG** using its associated command-line flag **-pmdlog**. For example, to generate a **pmd** log file, you could:

Set the MP_PMDLOG environment variable:	Use the -pmdlog flag when invoking the program:
ENTER **export MP_PMDLOG=***yes*	**ENTER** **poe** *program* **-pmdlog** *yes*

Ex.1008 / Page 61 of 186
Intel Corporation et al

Note: By default, **MP_PMDLOG** is set to *no*. No diagnostic logs are generated. You should not run **MP_PMDLOG** routinely, because this will greatly impact performance and fill up your file system space.

Determining which nodes will participate in parallel I/O

MPI has a number of subroutines that enable your application program to perform efficient parallel input-output operations. These subroutines (collectively referred to as ″MPI-IO″) allow efficient file I/O on a data structure which is distributed across several tasks for computation, but organized in a unified way in a single underlying file. MPI-IO presupposes a single parallel file system underlying all the tasks in the parallel job; PE's implementation of it is intended for use with the IBM Generalized Parallel File System (GPFS).

If your application program uses MPI-IO subroutines, all tasks in your MPI job will, by default, participate in parallel I/O. You can, however, specify that only tasks on a subset of the nodes in your job should handle parallel I/O. You might want to do this to ensure that all I/O operations are performed on the same node. To specify the nodes that should participate in parallel I/O, you:

- create an *I/O node file* (a text file that lists the nodes that should handle parallel I/O) and
- set the **MP_IONODEFILE** environment variable to the name of the I/O node file. As with most POE environment variables, **MP_IONODEFILE** has an associated command-line flag **-ionodefile**.

For example, say your job will be run with the following host list file dictating the nodes on which you program should run.

```
host1_name
host2_name
host3_name
host4_name
host5_name
host6_name
```

Say, however, that you want parallel I/O handled by only two of these nodes — *host5_name* and *host6_name*. to specify this, you would create an I/O node file that lists just the two host names.

```
host5_name
host6_name
```

One situation in which **MP_IONODEFILE** becomes useful is when running on a cluster of workstations which will not have a true parallel file system across multiple machines. By selecting one workstation to do the actual IO, you can reliably use JFS, NFS, and AFS® files with MPI-IO across multiple machines. (The file systems currently used, like NFS and AFS, to make a set of files available to multiple workstations are not parallel file systems in the way that GPFS is.) With respect to MPI-IO, an SP without GPFS is similar to a workstation cluster.

There should be no comments or blank lines in the I/O node file, there should be only one node name per line. Node names may be in any form recognizable to name service on the machine. Names which are not recognizable or which appear more that once yield advisory messages. Names which are valid but which do not represent nodes in the job are ignored. If **MP_IONODEFILE** is used and no node listed in the file is involved in the job, the job will abort. **MP_IONODEFILE** is most useful when used in conjunction with a host list file.

To indicate that the Partition Manager should use a particular I/O node file to determine which nodes handle parallel I/O, you must set the **MP_IONODEFILE** environment variable (or use the **-ionodefile** command-line flag to specify) the name of the file. You can specify the file using its relative or full path name. For example, say you have created an I/O node file *ionodes* in the directory */u/dlecker*. You could:

Set the MP_IONODEFILE environment variable:	Use the -ionodefile flag when invoking the program:
ENTER export MP_IONODEFILE=/u/dlecker/ionodes	**ENTER poe** *program* **-ionodefile /u/dlecker/ionodes**

Checkpointing and restarting programs

POE in Parallel Environment Version 3.2 provides enhanced capabilities to checkpoint and later restart the entire set of programs that make up a parallel application, including the checkpoint and restart of POE itself. A number of previous restrictions for checkpointing have been removed as well.

Checkpointing is a method of periodically saving the state of job so that, if for some reason the job does not complete, it can be restarted from the saved state. At checkpoint time, checkpoint files are created on the executing machines. The checkpoint file of POE contains all information required to restart the job from the checkpoint files of the parallel applications.

Earlier versions of Parallel Environment's checkpoint/restart capability were based on user level checkpointing, with significant limitations. You can now checkpoint both batch and interactive jobs using LoadLeveler or PE in a system-initiated mode (external to the task) or in a user-initiated mode (internal to the task).

With system-initiated checkpointing, you can use the PE **poeckpt** command to checkpoint a non-LoadLeveler POE job. LoadLeveler also provides commands for checkpointing jobs being run under LoadLeveler (for more information, see *Using and Administering LoadLeveler*). The PE **poerestart** command can be used to restart any interactive checkpointed jobs.

With user-initiated checkpointing, your application can use the POE checkpointing function call **mpc_init_ckpt**. In either mode, **mpc_set_ckpt_callbacks** and **mpc_unset_ckpt_callbacks** calls can be made from within your parallel program. The *IBM Parallel Environment for AIX: MPI Programming Guide* contains the specific information on these functions.

Using the settings of the **MP_CKPTDIR** and **MP_CKPTFILE** POE environment variables, the checkpoint data files are saved during the checkpointing phase, and the job is restarted by reading data from the checkpoint files during the restart phase. The **MP_CHECKDIR** and **MP_CHECKFILE** environment variables from previous releases are no longer used by POE.

When a checkpoint is taken, a set of checkpoint files is generated which consists of a POE checkpoint file and checkpoint files from each task of the parallel application. Each parallel task is checkpointed separately, and any processes created by a parallel task make up a checkpoint/restart group. The task checkpoint file contains information for all processes in the checkpoint/restart group. The checkpoint directory name is derived from the **MP_CKPTFILE** value (if it contains a full path name), the **MP_CKPTDIR** value, or the initial working directory. Tasks that change directories internally will not impact the place where the checkpoint file is written.

Ex.1008 / Page 63 of 186
Intel Corporation et al

Note: When running a parallel program under LoadLeveler, the **MP_CKPTDIR** and **MP_CKPTFILE** environment variables are set by LoadLeveler. If the value for the checkpoint file name or directory is specified in the job command file, those values will override the current settings.

When the checkpointing files are created, tags are added to the names to differentiate between earlier versions of the files.

There are certain limitations associated with checkpointing an application. For example, the **CHECKPOINT** environment variable must be set to *yes* when POE is invoked for it and any of the parallel tasks to be checkpointable. Refer to *IBM Parallel Environment for AIX: MPI Programming Guide* for specific details.

Checkpointing file management

The ability to checkpoint or restart programs is controlled by the definition and availability of the checkpoint files, as specified by the **MP_CKPTFILE** environment variable.

The checkpoint files may be defined on the local file system (JFS) of the node on which the instance of the program is running, or they may be defined in some shared file system (such as NFS, AFS, DFS™, GPFS, etc.). When the files are in a local file system, then in order to perform process migration, the checkpoint files will have to be moved to the new system on which the process is to be restarted. If the old system crashed and is unavailable, it may not be possible to restart the program. It may be necessary, therefore, to use some kind of file management to avoid such a problem. If migration is not desired, it is sufficient to place checkpoint files in the local JFS file system.

The program checkpoint files can be large, and numerous. There is the potential need for significant amounts of available disk space to maintain the files. If possible, you should avoid using NFS, AFS, or DFS to manage checkpoint files. The nature of these systems is such that it takes a very long time to write and read large files. Instead, use GPFS or JFS.

If a local JFS file system is used, the checkpoint file must be written to each remote task's local file system during checkpointing. Consequently, during a restart, each remote task's local file system must be able to access the checkpoint file from the previously checkpointed program from the directory where the checkpoint file was written when the chekcpoint occurred. This is of special concern when opting to restart a program on a different set of nodes from which it was checkpointed. The local checkpoint file may need to be relocated to any new nodes. For these reasons, it is suggested that GPFS be the file system best suited for checkpoint and restart file management.

POE user authorization

POE uses the PSSP Security services functions for its user authorization and authentication. The system administrator is responsible for defining and maintaining the security methods. See the appropriate sections of *IBM Parallel Environment for AIX: Installation* for specific details.

Supported security methods

POE supports the following user authentication methods, based on the SP Security services methods, set with the PSSP **chauthts** command:

Ex.1008 / Page 64 of 186
Intel Corporation et al

compatibility
AIX authentication will be used, based on entries in the */etc/hosts.equiv* or *.rhosts* files. This is the default mechanism.

dce DCE authentication will be used, for which you need to have:

- a valid DCE Id and principal, with which you perform a **dce_login**.
- the system administrator must have set up the PMD service principal.

dce and compatibility
where DCE authentication is attempted first, and, if that is unsuccessful, AIX authentication is tried.

none no security methods are enabled. POE defaults to use AIX authentication, as it does with compatibility.

Notes:

1. If you are using LoadLeveler to submit POE jobs, which includes all user space applications, be aware that LoadLeveler is responsible for the security authentication. The security function in POE is not invoked when POE is run under LoadLeveler.
2. When POE is used in a standalone RISC/6000 workstation environment without the *ssp.clients* fileset installed, only AIX authentication can be used.
3. The **lsauthts** command can be used to check the authentication method in use. For more information on the **chauthts** and **lsauthts** commands, see the *IBM Parallel System Support Programs for AIX: Command and Technical Reference, Volume 1*.
4. DCE credential forwarding is not supported by PE.

Using DCE user authorization

When DCE authentication is enabled as the SP Security method of choice, POE will expect a valid set of DCE credentials in order to submit parallel jobs.

When both DCE and ″compatibility″ methods are enabled, POE will first try DCE authentication. If DCE authentication is unsuccessful, POE will then use AIX authentication. In this case, any DCE authentication errors are written to the node's partition manager daemon log (if you have set the **MP_PMDLOG** environment variable or its associated command line flag **-pmdlog** to *yes*).

In order to use DCE with POE, the following is required:

1. A valid set of DCE credentials created by a **dce_login** to a valid principal.
2. A valid set of Kerberos Version 5 principals, created by **klogin** or an entry in a *.k5login* file in the user's home directory.
3. The system administrator must have properly set up the PMD service principal as part of the SP Security administration and configuration steps. For more information on configuring the PMD service principal, refer to *IBM Parallel Environment for AIX: Installation*

Using AIX user authorization

When compatibility authentication is enabled as the SP Security method of choice, POE will require users to have remote execution authority in the system. Users will need to be authorized to system nodes via either */etc/hosts.equiv* or *.rhosts* entries, as described in *IBM Parallel Environment for AIX: Installation*.

Ex.1008 / Page 65 of 186
Intel Corporation et al

AIX user authorization is the default, and is the only security mechanism available when POE is used in a standalone RISC/6000 workstation environment without the **ssp.clients** fileset installed.

Running POE with MALLOCDEBUG

Submitting a POE job that uses **MALLOCDEBUG** with an *align:n* option of other than 8 may result in undefined behavior. To allow a POE parallel program to run with an *align:n* option other than 8, you will need to create a script file. For example, say the POE program is named *myprog*. You could create the following script file:

```
MALLOCTYPE=debug
MALLOCDEBUG=align:0
myprog myprog_options
```

Once you had created the script file, you could then run the script file using the **poe** command. For example, if the script file were named *myprog.sh* you would enter:

```
poe myprog.sh <poe_options> <myprog_options>
```

Instead of:

```
poe myprog <poe_options> <myprog_options>
```

Chapter 3. Managing POE jobs

This chapter describes the tasks involved with managing POE jobs. It includes the following:

- Scenarios for allocating nodes with LoadLeveler
- Scenarios for submitting a batch job using LoadLeveler
- Appropriate environment variable information to use when running your applications.

Multi-task core file

With the **MP_COREDIR** environment variable, you can create a separate directory to save a core file for each task. The corresponding command line option is **-coredir**. Creating this type of directory is useful when you are running a parallel job on one node, and the job dumps a core file. By checking the directory, you can see which task dumped the file. When setting **MP_COREDIR**, you specify the first attribute of the directory name. The second attribute is the task id. If you do not specify a directory, the default is *coredir*. The subdirectory containing each task's core file is named *coredir.taskid*.

You can also disable the creation of a new subdirectory to save a core file, by specifying **-coredir** or **MP_COREDIR** with a value of "none". When disabled, core files will be written to /tmp instead of a new directory and instead of your current directory.

The following examples show what happens when you set the environment variable:

```
Example 1:


MP_COREDIR=my_parallel_cores

MP_PROCS=2


run generates core files


Core files will be located at:


/current directory/my_parallel_cores.0/core

/current directory/my_parallel_cores.1/core


Example 2:


MP_COREDIR not specified

MP_PROCS=2
```

© Copyright IBM Corp. 2000, 2001

```
run generates core files


Core files will be located at:


/current directory/coredir.0/core
/current directory/coredir.1/core


Example 3:


MP_COREDIR=none
MP_PROCS=2


run generates core files


Core files will be located at:


/tmp/core
```

Note: If multiple tasks run on a node, and each generates a core file, only the last core file written will be retrieved. The others are overwritten.

Specifying the format of corefiles or suppressing corefile generation

Using the **MP_COREFILE_FORMAT** environment variable (or its associated command-line flag **-corefile_format**), you can determine the format of corefiles generated when processes terminate abnormally — you can specify either traditional AIX corefiles or lightweight corefiles that conform to the Parallel Tool Consortium's Standardized Lightweight Corefile Format (LCF). You can also use the **MP_COREFILE_FORMAT** environment variable and **-corefile_format** flag to suppress corefile generation entirely.

If the MP_COREFILE_FORMAT environment variable or -corefile_format flag:	**Then:**	**For more information, see:**
is not set/used	standard AIX corefiles will be generated when processes terminate abnormally.	"Generating standard AIX corefiles" on page 51
specifies the string ″STDERR″	the corefile information will be output to standard error when processes terminate abnormally.	"Writing corefile information to standard error" on page 51

Ex.1008 / Page 68 of 186
Intel Corporation et al

If the MP_COREFILE_FORMAT environment variable or -corefile_format flag:	Then:	For more information, see:
specifies any other string	lightweight corefiles will be generated when processes terminate abnormally.	“Generating lightweight corefiles”

Note: Although the AIX operating system provides its own lightweight corefile subroutine and environment variable (**LIGHTWEIGHT_CORE**), be aware that it is intended for serial programs only. When using the AIX **LIGHTWEIGHT_CORE** environment variable with parallel programs compiled with the POE compiler scripts, the resulting output is unpredictable. For this reason, you should use the POE lightweight core file flags and environment variables for parallel programs.

Generating standard AIX corefiles

By default, POE processes that terminate abnormally generate standard AIX corefiles. Since this is the default behavior, you will not typically need to explicitly specify that standard AIX corefiles should be generated. If, however, the **MP_COREFILE_FORMAT** environment variable has previously been set, you will need to unset it in order to once again get the default behavior. To unset the **MP_COREFILE_FORMAT** environment variable, you would

ENTER

```
unset MP_COREFILE_FORMAT
```

Writing corefile information to standard error

As described in “Generating standard AIX corefiles”, POE processes that terminate abnormally will, by default, generate standard AIX corefiles. If you prefer, you can instruct POE to write the stack trace or lightweight corefile information to standard error instead. To do this, set the **MP_COREFILE_FORMAT** environment variable to the string ″STDERR″. As with most POE environment variables, you can temporarily override the value of **MP_COREFILE_FORMAT** using its associated command-line flag — **corefile_format**. For example, to specify that lightweight corefile information should be written to standard error, you could:

Set the MP_COREFILE_FORMAT environment variable:	Use the -corefile_format flag when invoking the program:
ENTER export MP_COREFILE_FORMAT=*STDERR*	**ENTER poe** *program* **-corefile_format** *STDERR*

Generating lightweight corefiles

By default, POE processes that terminate abnormally generate standard AIX corefiles. Often, however, traditional AIX corefiles are insufficient for debugging your program. This is because traditional AIX corefiles provide information that is too low-level for you to get a general picture of the overall status of your program. In addition, traditional AIX corefiles tend to be large and so can consume too much, if not all, available disk space. In being written out, theses corefiles can take up an unacceptable amount of CPU time and network bandwidth. These problems are especially acute in a large-scale parallel-processing environment (such as the SP running PE), when the problems can be multiplied by hundreds or thousands of processes.

Ex.1008 / Page 69 of 186
Intel Corporation et al

To address these problems with traditional core files, the Parallel Tools Consortium (a collaborative body of parallel-programming researchers, developers, and users from governmental, industrial, and academic sectors) has developed a corefile format called the Standardized Lightweight Corefile Format (LCF). As its name implies, a lightweight corefile does not have the often unnecessary low-level detail found in a traditional corefile; instead a lightweight corefile contains thread stack traces (listings of function calls that led to the error). Because of its smaller size, a lightweight corefile can be generated without consuming as much disk space, CPU time, and network bandwidth as a traditional AIX corefile. In addition, the LCF format can be a more useful aid in debugging threaded programs.

Using the **MP_COREFILE_FORMAT** environment variable (or its associated command-line flag **-corefile_format**), you can specify that POE should generate lightweight corefiles instead of standard AIX corefiles. To do this, simply specify the lightweight corefile name. For example, to specify the lightweight corefile name *light_core*, you could:

Set the MP_COREFILE_FORMAT environment variable:	**Use the -corefile_format flag when invoking the program:**
ENTER export MP_COREFILE_FORMAT=*light_core*	**ENTER poe** *program* **-corefile_format** *light_core*

One lightweight corefile (in this example, named *light_core*) for each process will be saved in a separate subdirectory.

By default, these subdirectories will be prefixed by the string ″*coredir*″ and suffixed by the task id (as in *coredir.0*, *coredir.1*, and so on. You can specify a prefix other than the default *coredir* by setting the **MP_COREDIR** environment variable or **-coredir** flag as described in “Multi-task core file” on page 49.

Note: By setting **-coredir** or **MP_COREDIR** to ″none″, you can bypass saving lightweight core files in a new subdirectory, and have them saved in /tmp instead.

In addition to developing the LCF standard, the Parallel Tools Consortium has also created command-line and graphical user interface tools (not distributed by IBM) that you can use to analyze lightweight corefiles. To use these tools, you will first want to merge the separate lightweight corefiles into a single file — with each separate lightweight corefile’s information appended, one after another, into the single lightweight corefile. To merge the separate lightweight corefiles into a single file, you could, for example, use the **mcpgath** command (as described in “mcpgath” on page 76) or you could create and use your own script.

Note: The lightweight corefile stack traces, and, by extension, the lightweight corefile browsers, will be able to show source code line numbers only if your program is compiled with the **-g** option. Otherwise, locations will be shown by relative address within the module. The **-g** flag is a standard compiler flag that produces an object file with symbol table references. For more information on the **-g** option, refer to its use on the **cc** command as described in *IBM AIX 5L Version 5.1: Commands Reference*

For more information on the Standard Lightweight Corefile Format or the Lightweight Corefile Browser (LCB) project, refer to *http://www.ptools.org/projects/lcb* on the World Wide Web. For information about the Parallel Tools Consortium, refer to *http://www.ptools.org* on the World Wide Web.

Ex.1008 / Page 70 of 186
Intel Corporation et al

Stopping a POE job

You can stop (suspend) a POE job by pressing <**Ctrl-z**> or by sending POE a SIGTSTP signal. POE stops, and sends a SIGSTOP signal to all the remote tasks, which stops them. To resume the parallel job, issue the **fg** or **bg** command to POE. A SIGCONT signal will be sent to all the remote tasks to resume them.

Cancelling and killing a POE job

You can cancel a POE job by pressing <**Ctrl-c**> or <**Ctrl-**>. This sends POE a SIGINT or SIGQUIT signal respectively. POE terminates all the remote tasks and exits.

If POE on the home node is killed or terminated before the remote nodes are shut down, direct communication with the parallel job will be lost. In this situation, use the **poekill** script as a POE command, or individually via **rsh**, to terminate the partition. **poekill** kills all instantiations of the program name on a remote node by sending it a SIGTERM signal. See the **poekill** script in */usr/lpp/ppe.poe/bin*, and the description of the **poekill** command in “Appendix A. Parallel environment commands” on page 73.

Note: *Do not* kill the **pmd**s using the **poekill** command. This will ensure that your remote processes will continue running.

Detecting remote node failures

POE and the Partition Manager use a *pulse* detection mechanism to periodically check each remote node to ensure that it is actively communicating with the home node. You specify the time interval (or *pulse* interval), of these checks with the **-pulse** flag or the **MP_PULSE** environment variable. During an execution of a POE job, POE and the Partition Manager daemons check at the interval you specify that each node is running. When a node failure is detected, POE terminates the job on all remaining nodes and issues an error message.

The default pulse interval is 600 seconds (10 minutes). You can increase or decrease this value with the **-pulse** flag or the **MP_PULSE** environment variable. To completely disable the pulse function, specify an interval value of 0 (zero). For the PE debugging facility **MP_PULSE** is disabled.

Considerations for using the SP switch and SP switch 2

The SP switch and SP switch 2 support dedicated User Space (US) and IP sessions, running concurrently on a single node. Users of IP communication programs that are not using LoadLeveler may treat these adapters like any other IP-supporting adapter. In this case, the adapter name is *css0*.

While US message passing programs must use LoadLeveler to allocate nodes, IP message passing programs may use LoadLeveler, but are not required to. When using LoadLeveler, nodes may be requested by name or number from one system pool only. When specifying node pools, the following rules apply:

- All the nodes in a pool should support the same combination of IP and US protocols. In other words, all the nodes should be able to run:
  - the IP protocol

    or
  - the US protocol

or
  - the IP and US protocols concurrently.
- In order to run the IP protocol, the IP switch addresses must be configured and started. In order to run the US protocol, the switch node numbers must be configured. For more information regarding these protocols and LoadLeveler, see *Using and Administering LoadLeveler* .
- By default, pool requests for the US message passing protocol also request exclusive use of the node(s). As long as a node was allocated through a pool request (and not through a specific node request), LoadLeveler will not allocate concurrent IP message passing programs on the node. You can override this default so that the node can be used for both IP and US programs by specifying “multiple” CPU usage.
- By default, requests for the IP message passing protocol also request multiple use of the node; LoadLeveler can allocate both IP and US message passing programs on this node. You can override this default so that the node is designated for exclusive use by specifying “unique” CPU usage.
- When running a batch parallel program under LoadLeveler, the adapter and CPU are allocated as specified by the *network keyword* in the LoadLeveler Job Command File. See *Using and Administering LoadLeveler* for more information.

Scenarios for allocating nodes with LoadLeveler

This section provides some examples of how someone would allocate nodes using LoadLeveler.

Scenario 1: Explicit allocation

A POE user, Paul, wishes to run a US job 1 in nodes A, B, C, and D. He doesn’t mind sharing the node with other jobs, as long as they are not also running in US. To do this, he specifies MP_EUIDEVICE=css0, MP_EUILIB=us, MP_PROCS=4, MP_CPU_USE=multiple, and MP_ADAPTER_USE=dedicated. In his host file, he also specifies:

```
node_A
node_B
node_C
node_D
```

The POE Partition Manager (PM) sees that this is a US job, and asks LoadLeveler for dedicated use of the css0 adapter on nodes A, B, C, and D and shared use of the CPU on those nodes. LoadLeveler then allocates the nodes to the job, recording that the css0/US session on A, B, C, and D has been reserved for dedicated use by this job, but that the node may also be shared by other users.

While job 1 is running, another POE user, Dan, wants to run another US job, job 2, on nodes B and C, and is willing to share the nodes with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=us, and MP_PROCS=2, MP_CPU_USE=multiple, and MP_ADAPTER_USE=dedicated. In his host file, he also specifies:

```
node_B
node_C
```

The PM, as before, asks LoadLeveler for dedicated use of the css0/US adapter on nodes B and C. LoadLeveler determines that this adapter has already been reserved for dedicated use on nodes B and C, and does not allocate the nodes again to job 2. The allocation fails, and POE job 2 cannot run.

While job 1 is running, a second POE user, John, wishes to run IP/switch job 3 on nodes A, B, C, and D, but doesn't mind sharing the node and the SP switch with other users. He specifies MP_EUIDEVICE=css0, MP_EUILIB=ip, MP_PROCS=4, MP_CPU_USE=multiple, and MP_ADAPTER_USE=shared. In his host file, he also specifies;

```
node_A
node_B
node_C
node_D
```

The POE PM asks LoadLeveler, as requested by John, for shared use of the css0/ip adapter and CPU on nodes A, B, C, and D. LoadLeveler determines that job 1 permitted other jobs to run on those nodes as long as they did not use the css0/US session on them. The allocation succeeds, and POE IP/switch job 3 runs concurrently with POE US job 1 on A, B, C, and D.

The scenario above, illustrates a situation in which users do not mind sharing nodes with other users' jobs. If a user wants his POE job to have dedicated access to nodes or the css0 adapter on nodes, he would indicate that in the environment by setting MP_CPU_USE=unique instead of *multiple*. If job 1 had done that, then job 3 would not have been allocated to those nodes and, therefore, would not have been able to run.

Scenario 2: Implicit allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H.

Job 1: Job1 is interactive, and requests 4 nodes for US using MP_RMPOOL.

```
MP_PROCS=4

MP_RMPOOL=2

MP_EUILIB=us
```

LoadLeveler allocates nodes A, B, C, and D for dedicated adapter (forced for US) and dedicated CPU (default for MP_RMPOOL).

Job 2: Job 2 is interactive, and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=shared
host.list

     @2
```

POE forces the adapter request to be dedicated, even though the user specified shared. Multiple (shared CPU) is supported, but in this case LoadLeveler doesn't have six nodes, either for CPU or for adapter, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU, but LoadLeveler only has four nodes available for CPU use, so the job fails.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates nodes E, F, and G.

Job 5: Job 5 is interactive and requests two nodes for IP using MP_RMPOOL.

```
MP_PROCS=2

MP_RMPOOL=2

MP_EUILIB=ip
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates two nodes from the list E, F, G, H (the others are assigned as dedicated to job 1).

Scenario 3: Implicit allocation

In this scenario, all nodes have both css0/US and css0/IP sessions configured, and are assigned to pool 2.

In this example, we have eight nodes; A, B, C, D, E, F, G, H

Job 1: Job 1 is interactive and requests four nodes for US using host.list.

```
MP_PROCS=4

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=dedicated
host.list

     @2
```

LoadLeveler allocates nodes A, B, C, and D for dedicated adapter (forced for US), and shared CPU.

Job 2: Job 2 is interactive and requests six nodes for US using host.list.

```
MP_PROCS=6

MP_HOSTFILE=./host.list

MP_EUILIB=us

MP_CPU_USE=multiple
MP_ADAPTER_USE=shared
host.list

     @2
```

POE forces the adapter request to be dedicated, even though the user has specified shared. Multiple (shared CPU) is supported, but in this case, LoadLeveler doesn't have six nodes for the adapter request, so the job fails.

Job 3: Job 3 is interactive and requests six nodes for IP using MP_RMPOOL.

```
MP_PROCS=6

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates six nodes for IP from the pool.

Job 4: Job 4 is interactive and requests three nodes for IP using MP_RMPOOL.

```
MP_PROCS=3

MP_HOSTFILE=NULL

MP_EUILIB=ip

MP_RMPOOL=2
```

The defaults are shared adapter and shared CPU. LoadLeveler allocates three nodes from the pool.

Submitting a batch POE job using IBM LoadLeveler

Note: POE version 3.2.0 is only compatible with LoadLeveler version 3.1. Submitting a POE version 3.2.0 batch job with an earlier version of LoadLeveler is not supported.

This section is intended for users who wish to submit batch POE jobs using IBM LoadLeveler, version 3.1. Refer to *Using and Administering LoadLeveler* for more information.

To submit a POE job using LoadLeveler, you need to build a LoadLeveler job file, which specifies:

- The number of nodes to be allocated

- Any POE options, passed via environment variables using Loadleveler's *environment* keyword, or passed as command line options using LoadLeveler's *argument* keyword.
- The path to your POE executable (usually /usr/bin/poe).
- Adapter specifications using the network keyword.

The following POE environment variables, or associated command line options, are validated, but not used, for batch jobs submitted using LoadLeveler.

- **MP_PROCS**
- **MP_RMPOOL**
- **MP_EUIDEVICE**
- **MP_EUILIB**
- **MP_MSG_API** (except for programs that use LAPI and also use the LoadLeveler **requirements** keyword to specify **Adapter=**"*hps_user*")
- **MP_HOSTFILE**
- **MP_SAVEHOSTFILE**
- **MP_RESD**
- **MP_RETRY**
- **MP_RETRYCOUNT**
- **MP_ADAPTER_USE**
- **MP_CPU_USE**
- **MP_NODES**
- **MP_TASKS_PER_NODE**

To run **myprog** on five nodes, using a Token ring adapter for IP message passing, with the message level set to the **info** threshold, you could use the following LoadLeveler job file. The arguments **myarg1** and **myarg2** are to be passed to **myprog**.

```
#!/bin/ksh

# @ input = myjob.in

# @ output = myjob.out

# @ error = myjob.error

# @ environment = COPY_ALL; \

    MP_EUILIB=ip; \

    MP_INFO_LEVEL=2

# @ executable = /usr/bin/poe

# @ arguments = myprog myarg1 myarg2

# @ min_processors = 5

# @ requirements = (Adapter == "tokenring")

# @ job_type = parallel

# @ checkpoint = no
```

To run **myprog** on 12 nodes from pool 2, using the User Space message passing interface with the message threshold set to **attention**, you could use the following LoadLeveler job file. See the documentation provided with the LoadLeveler program product for more information.

Ex.1008 / Page 76 of 186
Intel Corporation et al

```
#!/bin/ksh

# @ input = myusjob.in

# @ output = myusjob.out

# @ error = myusjob.error

# @ environment = COPY_ALL; MP_EUILIB=us

# @ executable = /usr/bin/poe

# @ arguments = myprog -infolevel 1

# @ min_processors = 12

# @ requirements = (Pool == 2) && (Adapter == "hps_user")

# @ job_type = parallel

# @ checkpoint = no
```

Notes:

1. The first token of the *arguments* string in the LoadLeveler job file must be the name of the program to be run under POE, unless:
   - You use the **MP_CMDFILE** environment variable or the **-cmdfile** command line option
   - The file you specify with the keyword *input* contains the name(s) of the programs to be run under POE.
2. When setting the environment string, make sure that no white space characters follow the backslash, and that there is a space between the semicolon and backslash.
3. When LoadLeveler allocates nodes for parallel execution, POE and task 0 will be executed on the same node.
4. When LoadLeveler detects a condition that should terminate the parallel job, a SIGTERM will be sent to POE. POE will then send the SIGTERM to each parallel task in the partition. If this signal is caught or ignored by a parallel task, LoadLeveler will ultimately terminate the task.
5. Programs that call the **usrinfo** function with the **getinfo** parameter, or programs that use the **getinfo** function, are not guaranteed to receive correct information about the owner of the current process.
6. Programs that use LAPI and also the LoadLeveler *requirements* keyword to specify **Adapter=**″**hps_user**″, must set the **MP_MSG_API** environment variable or associated command line option accordingly.
7. If the value of the **MP_EUILIB**, **MP_EUIDEVICE**, or **MP_MSG_API** environment variable that is passed as an argument to POE differs from the specification in the network statement of the job command file, the network specification will be used, and an attention message will be printed.

Submitting an interactive POE job using an IBM LoadLeveler command file

POE users may specify a LoadLeveler job command file to be used for an interactive job. Using a LoadLeveler job command file provides the capability to:

- Exploit new or existing LoadLeveler functionality not available using POE options, such as specification of:

- task geometry
- blocking factor
- machine order
- consumable resources
- memory requirements
- disk space requirements
- machine architecture

For more information on the LoadLeveler functionality you can exploit, refer to *Using and Administering LoadLeveler*.

- Run parallel jobs without specifying a hostfile or pool, thereby causing LoadLeveler to select nodes for the parallel job from any in its cluster.
- Specify that a job should run from more than 1 pool.

You can use a LoadLeveler job command file with or without a host list file. If you have created a LoadLeveler job command file for node allocation (either independently or in conjunction with a host list file), you need to set the **MP_LLFILE** environment variable (or use the **-llfile** flag when invoking the program) to specify the file. You can specify the LoadLeveler job command file using its relative or full path name. For example, say the LoadLeveler job command file is named *file.cmd* and is located in the directory */u/dlecker*. You could:

Set the MP_LLFILE environment variable:	Use the -llfile flag when invoking the program:
ENTER export MP_LLFILE=/u/dlecker/file.cmd	**ENTER poe** *program* **-llfile /u/dlecker/file.cmd**

When the **MP_LLFILE** environment variable, or the **-llfile** command-line option is used, the following POE node/adapter specifications are ignored.

- **MP_RMPOOL**
- **MP_EUIDEVICE**
- **MP_EUILIB**
- **MP_RESD**
- **MP_MSG_API**
- **MP_ADAPTER_USE**
- **MP_CPU_USE**
- **MP_NODES**
- **MP_TASKS_PER_NODE**
- **MP_PROCS** (when a host list file is not used.)

When using this option, the following restrictions apply.

- Cannot be used for batch POE jobs.
- The host list file cannot contain pool requests.
- The **MP_PROCS** environment variable or the **-procs** command-line flag must be used if a host list file is used, otherwise only 1 parallel task will be run on the first host listed in the host list file.
- Certain LoadLeveler keywords are not allowed in the LoadLeveler job command file when it is being used for an interactive POE job. Refer to the manual *Using and Administering LoadLeveler* for a listing of these keywords.

Generating an output LoadLeveler job command file

When using LoadLeveler for submitting an interactive job, you can, provided you are not already using a LoadLeveler job command file, generate an output LoadLeveler job command file. This output LoadLeveler job command file contains the LoadLeveler settings that result from the environment variables and/or command line options for the current invocation of POE. If you are unfamiliar with LoadLeveler and its job command files, this provides an easy starting point for creating LoadLeveler job command files. Once you create an output LoadLeveler job command file, you can then, for subsequent submissions, modify it to contain additional LoadLeveler specifications (such as new LoadLeveler functionality available only through using a LoadLeveler job command file).

Be aware that you cannot generate a LoadLeveler job command file if you are already using one; in other words, if the **MP_LLFILE** environment variable or the **-llfile** command line flag is used. You also cannot generate an output LoadLeveler job command file if you are submitting a batch job.

To generate a LoadLeveler job command file, you can use the **MP_SAVE_LLFILE** environment variable to specify the name that the output LoadLeveler job command file should be saved as. You can specify the output LoadLeveler job command file name using a relative or full path name. As with most POE environment variables, you can temporarily override the value of **MP_SAVE_LLFILE** using its associated command-line flag **-save_llfile**. For example, to save the output LoadLeveler job command file as *file.cmd* in the directory */u/wlobb*, you could:

Set the MP_SAVE_LLFILE environment variable:	Use the -save_llfile flag when invoking the program:
ENTER export MP_SAVE_LLFILE=/u/wlobb/file.cmd	**ENTER poe** *program* **-save_llfile /u/wlobb/file.cmd**

Running programs under the C shell

During normal configuration of an SP system, the Automount Daemon (amd) is used to mount user directories. amd's maps use the symbolic file system links, rather than the physical file system links. While the Korn shell keeps track of file system changes, so that a directory is always available, this mapping does not take place in the C shell. This is because the C shell only maintains the physical file system links. As a result, users that run POE from a C shell may find that their current directory (for example /a/moms/fileserver/sis), is not known to amd, and POE fails with message 0031-214 (unable to change directory).

By default, POE uses the Korn shell **pwd** command to obtain the name of the current directory. This works for C shell users if the current directory is either:

- The home directory
- Not mounted by amd.

If neither of the above are true (for example, if the user's current directory is a subdirectory of the home directory), then POE provides another mechanism to determine the correct amd name; the **MP_REMOTEDIR** environment variable.

POE recognizes the **MP_REMOTEDIR** environment variable as the name of a command or Korn shell script that echoes a fully-qualified file name. **MP_REMOTEDIR** is run from the current directory from which POE is started.

Ex.1008 / Page 79 of 186
Intel Corporation et al

If you do not set **MP_REMOTEDIR**, the command defaults to **pwd**, and is run as **ksh -c pwd**. POE sends the output of this command to the remote nodes and uses it as the current directory name.

You can set **MP_REMOTEDIR** to some other value and then export it. For example, if you set **MP_REMOTEDIR=″echo /tmp**″, the current directory on the remote nodes becomes /tmp on that node, regardless of what it is on the home node.

The script **mpamddir** is also provided in */usr/lpp/ppe.poe/bin*, and the setting **MP_REMOTEDIR=mpamddir** will run it. This script determines whether or not the current directory is a mounted file system. If it is, the script searches the amd maps for this directory, and constructs a name for the directory that is known to amd. You can modify this script or create additional ones that apply to your installation.

Note: Programs that depend upon the name of the current directory for correct operation may not function properly with an alternate directory name. In this case, you should carefully evaluate how to provide an appropriate name for the current directory on the home nodes.

If you are executing from a subdirectory of your home directory, and your home directory is a mounted file system, it may be sufficient to replace the C shell name of the mounted file system with the contents of $HOME. One approach would be:

```
export MP_REMOTEDIR=pwd.csh
```

or for C shell users:

```
setenv MP_REMOTEDIR pwd.csh
```

where the file **pwd.csh** is:

```
#!/bin/csh -fe

# save the current working directory name

set oldpwd = `pwd`

# get the name of the home directory

cd $HOME

set hmpwd = `pwd`

# replace the home directory prefix with the contents of $HOME

set sed_home = `echo $HOME | sed 's/\//\\\//g'`

set sed_hmpwd = `echo $hmpwd | sed 's/\//\\\//g'`

set newpwd = `echo $oldpwd | sed "s/$sed_hmpwd/$sed_home/"`

# echo the result to be used by amd

echo $newpwd
```

Using MP_CSS_INTERRUPT

The **MP_CSS_INTERRUPT** environment variable may take the value of either **yes** or **no**. By default it is set to **no**. In certain applications, setting this value to **yes** will provide improved performance.

The following briefly summarizes some general application characteristics that could potentially benefit from setting **MP_CSS_INTERRUPT=yes**.

Applications which have the following characteristics may see performance improvements from setting the POE environment variable **MP_CSS_INTERRUPT** to **yes**:

- Applications that use nonblocking send or receive operations for communication.
- Applications that have non-synchronized sets of send or receive pairs. In other words, the send from node0 is issued at a different point in time with respect to the matching receive in node1.
- Applications that do not issue waits for nonblocking send or receive operations immediately after the send or receive, but rather do some computation prior to issuing the waits.

In all of the previous cases, the application is taking advantage of the asynchronous nature of the nonblocking communication subroutines. This essentially means that the calls to the nonblocking send or receive routines do not actually ensure the transmission of data from one node to the next, but only post the send or receive and then return immediately back to the user application for continued processing. However, since the SP communication subsystem is a user space protocol and executes within the user's process, it must regain control from the application to complete asynchronous requests for communication.

The SP communication subsystem can regain control from the application in any one of three different methods:

1. Any subsequent calls to the SP communication subsystem to post send or receive, or to wait on messages.
2. A timer signal is received periodically to allow the communication subsystem to do recovery from transmission errors.
3. If the value of **MP_CSS_INTERRUPT** is set to **yes**, the communication subsystem device driver will send a signal to the user application when data is received or buffer space is available to transmit data.

Method 1 and Method 2 are always enabled. Method 3 is controlled by the POE environment variable **MP_CSS_INTERRUPT**, and is enabled when this variable is set to **yes**.

For those applications that have the characteristics mentioned previously, this implies that when using asynchronous communication the completion of the communication must occur through one of the these three methods. In the case that **MP_CSS_INTERRUPT** is not enabled, only the first two methods are available to process communication. Depending upon the amount of time between the non-synchronized send or receive pairs, or between the nonblocking send or receive and the corresponding waits, the actual transmission of data may only complete at the matching wait call. If this is the case, it is possible that an application may see a performance degradation due to unnecessary processor stalling waiting for communication.

Ex.1008 / Page 81 of 186
Intel Corporation et al

As an example, consider the following application template, where both processors execute the same code, and processor 0 sends and receives data from processor 1.

```
DO LOOP

   MP_SEND (A ...., msgid1)

   MP_RECV (B ...., msgid2)


   MP_WAIT (msgid2, nbytes)


   COMPUTE LOOP1 (uses B)


   MP_WAIT (msgid1, nbytes)


   COMPUTE LOOP2 (modifies A)

ENDDO
```

In this example, application B is guaranteed to be received after the wait for `msgid2`, and more than likely the data is actually received during the wait call. B can then be safely used in the `compute loop1`. A is not guaranteed to be sent until the wait for `msgid1`. Therefore, A cannot be modified until after this wait.

With **MP_CSS_INTERRUPT=no**, it is likely that processor0 receives B during the wait for `msgid2`, and enters the `compute loop1` before the send of A has completed. In this case, processor1 will stall waiting for the completion of the wait for `msgid2`, which will not complete until processor0 completes the `compute loop1` and reaches the wait for `msgid1`. The stalling of processor1 is directly related to the non-continuous flow of communication. If **MP_CSS_INTERRUPT=yes**, when the communication is ready to complete, the communication subsystem device driver sends a signal to the application and causes the application to immediately complete the communication. Therefore data flow is continuous and smooth. The send of A can be completed, even during the `compute loop1`, preventing the stalling of processor1 and improving overall performance of this application.

Finally, note that there is a cost associated with handling the signals when **MP_CSS_INTERRUPT** is set to **yes**. In some cases, this cost can degrade application performance. Therefore, only use **MP_CSS_INTERRUPT** for those applications that require it. For the IP version of the library, **MP_CSS_INTERRUPT=yes** enables UDP to send a SIGIO signal when a message packet is received.

Note: MPI-IO and MPI-1-sided communication enables interrupts when in use, and disables them afterwards. If either the **MP_CSS_INTERRUPT** environment variable or one to the control functions is explicitly set, MPI will not alter the value for the remainder of the run.

Support for performance improvements

POE provides interfaces to improve interrupt mode latency.

Interrupt mode control

When a node receives a packet and an interrupt is generated, the interrupt handler checks its tables for the process identifier (PID) of the user process and notifies the process. The process signal handler or service thread polls for at least two times the interrupt delay, checking to see if more packets will arrive. Waiting for more packets avoids the cost of incurring an interrupt each time a new packet arrives (interrupt processing is very expensive). However, the more packets that arrive, the more delay time is increased. Therefore, with these functions you can tune the delay parameter based on your application, and/or dynamically turn interrupts on or off at selected nodes.

For an application with few nodes exchanging small messages, it will help latency if you keep the interrupt delay small. For an application with a large number of nodes, or one which exchanges large messages, keeping the delay parameter large will help the bandwidth. A large delay allows multiple read transmissions to occur in a single read cycle. You should experiment with different values and use the functions described below to achieve desired performance, depending on the communication pattern.

MP_INTRDELAY is the environment variable which allows you to set the delay parameter for how long the signal handler or service thread waits for more data. The delay specified in the environment variable is set during initialization, before running the program. In this way, user programs can tune the delay parameter without having to recompile existing applications. If none is specified, the default value of 1 microsecond is used. The application can tune this parameter based on the communication pattern it has in different parts of the application.

PE provides five application programming interfaces to help you enable or disable interrupts on specific tasks, based on the communication patterns of the tasks. If a task is frequently in the communication library, then the application can turn interrupts off for that particular task for the duration of the program. The application can enable interrupts when the task is not going to be in the communication subsystem often. The enable or disable interfaces override the setting of the **MP_CSS_INTERRUPT** environment variable.

The first two functions allow you to query what the current delay parameter is and to set the delay parameter to a new value.

int mpc_queryintrdelay() - for C programs
void mp_queryintrdelay(int rc) - for Fortran programs

This function returns the current interrupt delay (in microseconds). If none was set by the user, the default is returned.

int mpc_setintrdelay(int val) - for C programs
void mp_setintrdelay(int val, int rc) - for Fortran programs

This function sets the delay parameter to the value, in microseconds, specified by “val”. The function can be called at multiple places within the program to set the delay parameter to different values during execution.

The following three functions allow you to control dynamically masking interrupts on individual nodes, and query the state of interrupts. In the current system only “all” nodes or “none” can be selected to statically enable or disable running in interrupt mode.

int mpc_queryintr() - for C programs
void mp_queryintr(int rc) - for Fortran programs

This function returns 0 if the node on which it is executed has interrupts turned off, and it returns 1 otherwise.

int mpc_disableintr() - for C programs
void mp_disableintr(int rc) - for Fortran programs

This function disables interrupts on the node on which it is executed. Return code = 0, if successful, -1 otherwise.

int mpc_enableintr() - for C programs
void mp_enableintr(int rc) - for Fortran programs

This function enables interrupts on the node on which it is executed. Return code = 0, if successful, -1 otherwise.

Note: The last two of the previous functions override the setting of the environment variable **MP_CSS_INTERRUPT**. These functions may be useful in reducing latency if the application is doing blocking recv/wait and interrupts are otherwise enabled. Interrupts should be turned off before executing blocking communication calls and turned on immediately after those calls.

You can use all of the above functions for programs running IP.

Rcvncall improvements

The **mpc_wait** function can be called just before re-posting the Rcvncall instead of in the beginning of the Rcvncall handler, if information provided by the wait function call (like length of message) is already available. This removes the wait time from the critical path for latency. The wait function provides the message id, the length of the message, and also cleans up the resources used by the previously posted Rcvncall. This applies to the signal-handling MPL library only.

Parallel file copy utilities

During the course of developing and running parallel applications on numerous nodes, the potential need exists to efficiently copy data and files to and from a number of places. POE provides three utilities for this reason:

1. **mcp** - to copy a single file from the home node to a number of remote nodes. This was discussed briefly in “Step 2: Copy files to individual nodes” on page 10.
2. **mcpscat** - to copy a number of files from task 0 and scatter them in sequence to all tasks, in a round robin order.
3. **mcpgath** - to copy (or gather) a number of files from all tasks back to task 0.

mcp is for copying the same file to all tasks. The input file must reside on task 0. You can copy it to a new name on the other tasks, or to a directory. It accepts the source file name and a destination file name or directory, in addition to any POE command line argument, as input parameters.

mcpscat is intended for distributing a number of files in sequence to a series of tasks, one at a time. It will use a round robin ordering to send the files in a one to

one correspondence to the tasks. If the number of files exceeds the number of tasks, the remaining files are sent in another round through the tasks.

mcpgath is for when you need to copy a number of files from each of the tasks back to a single location, task 0. The files must exist on each task. You can optionally specify to have the task number appended to the file name when it is copied.

Both **mcpscat** and **mcpgath** accept the source file names and a destination directory, in addition to any POE command line argument, as input parameters. You can specify multiple file names, a directory name (where all files in that directory, not including subdirectories, are copied), or use wildcards to expand into a list of files as the source. Wildcards should be enclosed in double quotes, otherwise they will be expanded locally, which may not produce the intended file name resolution.

These utilities are actually message passing applications provided with POE. Their syntax is described in “Appendix A. Parallel environment commands” on page 73.

Chapter 4. Monitoring program execution using the Program Marker Array

The Program Marker Array (shown in Figure 1) is an X-Windows run-time monitoring tool. This window consists of a number of small squares called *lights* that change color under program control. Each task in a parallel program has its own row of lights, and Parallel Utility Function calls from those tasks can change light colors. The calls can also send strings to the PM Array.

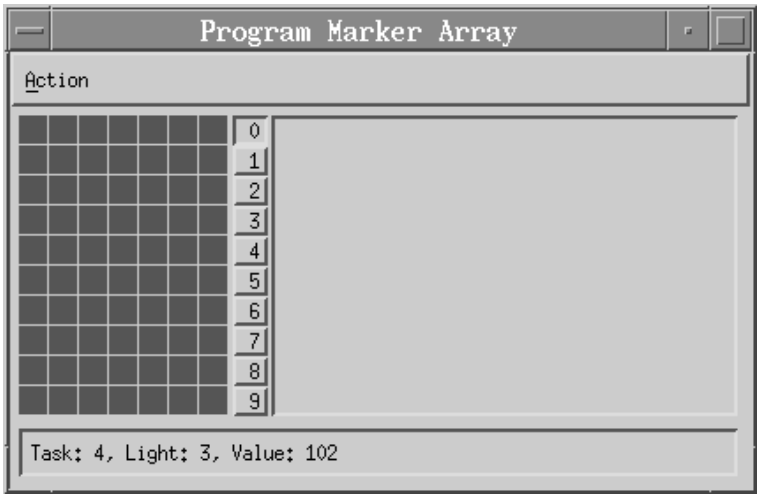

Figure 1. The Program Marker Array

The ability to color lights on, and send strings to, the PM Array window enables a parallel program to provide you with immediate visual feedback as it executes. A program could begin by coloring lights red and then slowly move through the spectrum towards blue as it executes. If a program takes a long time to run, this would give you an indication that it was indeed progressing. Should the program not be progressing, the PM Array would indicate that as well. For example, lights "stuck" on a particular color could indicate that the program is stuck as well. The strings displayed could provide additional information on the program's progress. In addition, the Program Marker Array is distributed as source code, so you can customize the program as you see fit. The source code is located in the directory */usr/lpp/ppe.poe/samples/marker*.

In order to use the PM Array to monitor program execution, you need to:

1. In your C, C++, and/or Fortran program, insert subroutine calls to color lights on, and send strings to, the PM Array. This is described in "Step 1: Call PM Array parallel utility functions" on page 70.
2. Compile and link the program using one of the POE compiler commands as described in "Step 1: Compile the program" on page 8. These commands call the C, C++, or Fortran compilers while linking in the Partition Manager interface and Parallel Utility subroutines.
3. Set the environment variable **MP_PMLIGHTS** equal to the number of lights you would like displayed per program task. Alternatively, you can set the number of lights using a command-line flag when invoking your parallel program. See "Step 3: Set the number of lights" on page 70.

© Copyright IBM Corp. 2000, 2001

4. Issue the **pmarray** command to start the PM Array program as described in "Step 4: Open the PM array window" on page 71.
5. Invoke your parallel program and monitor its execution using the PM Array. This is described in "Step 5: Invoke the program and monitor its execution" on page 71.

Note: The use of the Program Marker Array with threaded programs is not supported. Threaded programs should not include calls to Program Marker Array utility functions.

Step 1: Call PM Array parallel utility functions

In order for the PM Array to display meaningful information at run time, you need to place calls to Parallel Utility Functions within your program. At run time, your program can then:

- color lights on, and/or send output strings to, the PM Array Window. This is done by calling (in C programs) the mpc_marker or (in Fortran programs) the MP_MARKER Parallel Utility Function.
- determine the number of lights displayed per task row. This is done by calling (in C programs) the mpc_nlights or (in Fortran programs) the MP_NLIGHTS Parallel Utility Function. Since the number of lights displayed for each task on the PM Array can vary from run to run, this capability is important. It enables your program to learn exactly how many lights are available to be set. It returns an integer value that can then be used by the program to resolve some conditional expression.

The syntax of these Parallel Utility Functions is shown in *IBM Parallel Environment for AIX: MPI Subroutine Reference*.

Step 2: Compile the program

Once you have inserted calls to the **mpc_marker** and **mpc_nlights** functions or the **MP_MARKER** and **MP_NLIGHTS** subroutines into your program, you can compile it. Since this is the same procedure you follow when regularly compiling a parallel program with POE, see page 8 for more information. see "Step 1: Compile the program" on page 8 for more information.

Step 3: Set the number of lights

When you open the PM Array window in the next step, the number of rows in the PM Array are set to the number of program tasks – the current setting of **MP_PROCS**. You can also specify the number of lights you want displayed per task row. To do this, set the environment variable **MP_PMLIGHTS** or specify the **-pmlights** command-line flag in "Step 5: Invoke the program and monitor its execution" on page 71.

For example, say you want five lights displayed per task in the PM Array. You could:

Set the MP_PMLIGHTS environment variable:	**Use the -pmlights flag when invoking your executable:**
ENTER **export MP_PMLIGHTS=***5*	**ENTER** **poe** *program* **-pmlights** *5*

As with most POE command-line flags, the **-pmlights** flag temporarily overrides its associated environment variable.

Notes:

1. Setting the **MP_PMLIGHTS** environment variable, or the **-pmlights** flag, to *0* indicates that you do not want your program to communicate with the PM Array tool.
2. If you reset the **MP_PMLIGHTS** environment variable, or the **-pmlights** flag, after the Program Marker Array tool is started, it will usually reset to the new number of lights. The only time it will not, however, is when the new value of **MP_PMLIGHTS** is *0*.

Step 4: Open the PM array window

The **pmarray** command starts the PM Array program. You will probably want to use the **&** operator so the program runs in the background and does not tie up the aixterm window.

ENTER

pmarray &

The PM Array window opens. The number of task rows displayed in the PM Array is equal to the current setting of **MP_PROCS**. The number of lights per task row is determined by the current setting of **MP_PMLIGHTS**. If, when you invoke your program in Step 5: Invoke the program and monitor its execution, you override either of the environment variables using its associated command-line flag, the PM Array redisplays with the new number of rows and/or lights.

Notes:

1. The PM Array connects to the Partition Manager using a socket assigned, by default, to port 9999. If you get an error message indicating that the port is in use, specify a different port by setting the **MP_USRPORT** environment variable before entering the **pmarray** command. For example, to specify port 9998:

   ENTER

   export MP_USRPORT=*9998*
2. The **pmarray** program must be restarted by the user if the parallel application that is being restarted uses these utility functions. The restarted program will only reflect changes that result after the parallel applications were restarted.

Step 5: Invoke the program and monitor its execution

Finally, you invoke your program. As the program runs, the Parallel Utility Function calls placed within it change the color of lights on the PM Array. With appropriate mouse clicks on this window, you can:

- display details of a light
- display output strings from a task
- close the window and discontinue monitoring

Displaying details of a light

Each light on the PM Array is associated with a particular task, has a particular light number, and has a particular color value. You can display these details for each of the lights on the PM Array.

For example, say you have coded the PM Array subroutines into your program so that the lights slowly move during execution through the spectrum over color values

0 to 99. As the program runs, the lights start off black, and then turn brown, green, blue, and so on. By watching the lights as they change color, you get a general idea of the program's progress. For a more precise indication of the program's progress, you could display the actual color value number for a light. In this example, the closer this light's value is to 99, the closer execution is to being complete.

To display details of a light:

PLACE
the cursor over any light on the PM Array.

PRESS
the left mouse button.

The following information displays in the text area at the bottom of the PM Array window:
- the task identifier number
- the light number
- the color value number

This information is not updated until you select another light.

Displaying task output

You can display output strings sent by the tasks of your program in the output display area of the PM Array window. This is the area to the right of the PM Array, and the strings displayed there are the ones you specified on the mpc_marker or MP_MARKER subroutine calls. Only one task's strings are displayed in this area at a time. By default, output from task 0 is displayed. You can select the task and display its output instead by pressing its task push button. Each task has a push button. It is just to the right of the task's row on the PM Array, and is labeled with the task identifier. To select, for example, task 3:

PRESS
the task pushbutton labeled 3.

Output strings from task 3 are displayed in the output display area. Only one string is displayed at a time.

Note: If a task not currently selected has sent new output to the PM Array window, its task push button will appear yellow.

Step 6: Close the PM Array window

The PM Array window remains open after your parallel program completes executing. You could then repeat Step 5: Invoke the program and monitor its execution to monitor the same, or a different program's execution. To close the PM Array window when you are done monitoring:

SELECT
Action → Quit

Appendix A. Parallel environment commands

This appendix contains the manual pages for the PE commands discussed throughout this book. Each manual page is organized into the sections listed below. The sections always appear in the same order, but some appear in all manual pages while others are optional.

NAME
Provides the name of the command described in the manual page, and a brief description of its purpose.

SYNOPSIS
Includes a diagram that summarizes the command syntax, and provides a brief synopsis of its use and function. If you are unfamiliar with the typographic conventions used in the syntax diagrams, see “Conventions and terminology used in this book” on page x.

FLAGS
Lists and describes any required and optional flags for the command.

DESCRIPTION
Describes the command more fully than the **NAME** and **SYNOPSIS** sections.

ENVIRONMENT VARIABLES
Lists and describes any applicable environment variables.

EXAMPLES
Provides examples of ways in which the command is typically used.

FILES
Lists and describes any files related to the command.

RELATED INFORMATION
Lists commands, functions, file formats, and special files that are employed by the command, that have a purpose related to the command, or that are otherwise of interest within the context of the command.

© Copyright IBM Corp. 2000, 2001

mcp

NAME

mcp – Allows you to propagate a copy of a file to multiple nodes on an IBM POWERparallel® system.

SYNOPSIS

mcp *infile* [*outfile*] [*POE options*]

In the previous command synopsis, the *infile* is the name of the file to be copied. You can copy to a new name by specifying an *outfile*. If you do not provide the outfile name, the file will be placed in its current directory on each node. The outfile can be either an explicit output file name or a directory name. When a directory is specified, the file is copied with the same name to that directory.

DESCRIPTION

The **mcp** command allows you to propagate a copy of a file to multiple nodes on an IBM RS/6000 SP. The file must initially reside (or be NFS-mounted) on at least one node.

mcp is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest. The input file must be readable from the node assigned to task 0.

Note: A POE job loads faster if a copy of the job resides on each node. For this reason, it is suggested that you use **mcp** to copy your executable to a file system such as /tmp, which resides on each node.

Return codes are:

129
incorrect usage

130
error opening input file

131
error opening *to* file on originating node

132
error writing data to *to* file on originating node

133
no room on remote node's file system

134
error opening file on remote node

135
error writing data on remote node

136
error renaming temp file to file name

137
input file is empty

138
invalid block size

139
error allocating storage

ENVIRONMENT VARIABLES

MP_BLKSIZE
sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. To copy a file from your current directory to the current directory on all nodes of a 16-processor system, using the User Space protocol, enter:

```
mcp filename -procs 16 -euilib us
```

2. To copy a filename from your current directory to the /tmp directory on all nodes of a 16-processor system, using IP, enter:

```
mcp filename /tmp -procs 16 -euilib ip
```

3. To copy a file from your current directory to a different filename on all nodes of a 16-processor system, enter:

```
mcp filename /tmp/newfilename -procs 16
```

RELATED INFORMATION

Commands: **rcp**(1)

mcpgath

NAME

mcpgath – Takes files from each task of tasks 0 through task N and copies them back in sequence to task 0.

SYNOPSIS

mcpgath [**-ai**] *source ... destination* [*POE options*]

Source is one of the following:

- one or more existing file names - files will be copied with the same names to the destination directory on task 0. Each file name specified must exist on all tasks involved in the copy.
- a directory name - all files in that directory on each task are copied with the same names to the destination directory on task 0.
- an expansion of file names, using wildcards - files are copied with the same names to the destination directory. All wildcarded input strings must be enclosed in double quotes.

Destination is an existing destination directory name to where the data will be copied. The destination directory must be the last item specified before any POE flags.

FLAGS

-a An optional flag that appends the task number to the end of the file name when it is copied to task 0. This is for task identification purposes, to know where the data came from. The **-a** and **-i** flags can be combined to check for existing files appended with the task number.

-i An optional flag that checks for duplicate or existing files of the same name, and does not replace any existing file found. Instead, issues an error message and continues with the remaining files to be copied. The **-a** and **-i** flags can be combined to check for existing files appended with the task number.

See Chapter 2. Executing parallel programs for information on POE options.

DESCRIPTION

The **mcpgath** function determines the list of files to be gathered on each task. This function also resolves the source file, destination directory, and path names with any meta characters, wildcard expansions, and so on, to come up with valid file names. Enclose wildcards in double quotes, otherwise they will be expanded locally on the task from where the command is issued, which may not produce the intended file name resolution.

mcpgath is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest.

Return codes are:

129
invalid number of arguments specified

130
invalid option flag specified

131
unable to resolve input file name(s)

132
could not open input file for read

133
no room on destination node's file system

134
error opening file output file

135
error creating output file

136
error writing to output file

137
MPI_Send of data failed

138
final **MPI_Send** failed

139
MPI_Recv failed

140
invalid block size

141
error allocating storage

142
total number of tasks must be greater than one

ENVIRONMENT VARIABLES

MP_BLKSIZE
sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. You can copy a single file from all tasks into the destination directory. For example, enter:

```
mcpgath -a hello_world /tmp -procs 4
```

This will copy the file *hello_world* (assuming it is a file and not a directory) from tasks 0 through 3 as to task 0:

```
From task 0:  /tmp/hello_world.0

From task 1:  /tmp/hello_world.1
```

```
From task 2:  /tmp/hello_world.2

From task 3:  /tmp/hello_world.3
```

2. You can specify any number of files as source files. The destination directory must be the last item specified before any POE flags. For example:

```
mcpgath -a file1.a file2.a file3.a file4.a file5.a /tmp -procs 4
```

will take *file1.a* through *file5.a* from the local directory on each task and copy them back to task 0. All files specified must exist on all tasks involved. The file distribution will be as follows:

```
From Task 0: /tmp/file1.a.0

From Task 1: /tmp/file1.a.1

From Task 2: /tmp/file1.a.2

From Task 3: /tmp/file1.a.3

From Task 0: /tmp/file2.a.0

From Task 1: /tmp/file2.a.1

From Task 2: /tmp/file2.a.2

From Task 3: /tmp/file2.a.3

From Task 0: /tmp/file3.a.0

From Task 1: /tmp/file3.a.1

From Task 2: /tmp/file3.a.2

From Task 3: /tmp/file3.a.3

From Task 0: /tmp/file4.a.0

From Task 1: /tmp/file4.a.1

From Task 2: /tmp/file4.a.2

From Task 3: /tmp/file4.a.3

From Task 0: /tmp/file5.a.0

From Task 1: /tmp/file5.a.1

From Task 2: /tmp/file5.a.2

From Task 3: /tmp/file5.a.3
```

3. You can specify wildcard values to expand into a list of files to be gathered. For this example, assume the following distribution of files before calling **mcpgath**:

```
Task 0 contains file1.a and file2.a

Task 1 contains file1.a only
```

```
Task 2 contains file1.a, file2.a, and file3.a

Task 3 contains file4.a, file5.a, and file6.a
```

Enter:

```
mcpgath -a "file*.a" /tmp -procs 4
```

This will pass the wildcard expansion to each task, which will resolve into the list of locally existing files to be copied. This results in the following distribution of files on task 0:

```
From Task 0: /tmp/file1.a.0

From Task 0: /tmp/file2.a.0

From Task 1: /tmp/file1.a.1

From Task 2: /tmp/file1.a.2

From Task 2: /tmp/file2.a.2

From Task 2: /tmp/file3.a.2

From Task 3: /tmp/file4.a.3

From Task 3: /tmp/file5.a.3

From Task 3: /tmp/file6.a.3
```

4. You can specify a directory name as the source, from which the files to be gathered are found. For this example, assume the following distribution of files before calling **mcpgath**:

```
Task 0 /test contains file1.a and file2.a

Task 1 /test contains file1.a only

Task 2 /test contains file1.a and file3.a

Task 3 /test contains file2.a, file4.a, and file5.a
```

Enter:

```
mcpgath -a /test /tmp -procs 4
```

This results in the following file distribution:

```
From Task 0: /tmp/file1.a.0

From Task 0: /tmp/file2.a.0

From Task 1: /tmp/file1.a.1

From Task 2: /tmp/file1.a.2

From Task 2: /tmp/file3.a.2
```

```
From Task 3: /tmp/file2.a.3

From Task 3: /tmp/file4.a.3

From Task 3: /tmp/file5.a.3
```

mcpscat

NAME

mcpscat – Takes a number of files from task 0 and scatters them in sequence to all tasks, in a round robin order.

SYNOPSIS

mcpscat [**-f**] [**-i**] *source ...*
destination
[*POE options*]

Source can be one of the following:

- a single file name - file is copied to all tasks
- a single file name that contains a list of file names (**-f** option)
- two or more file names - files will be distributed in a round robin order to the tasks
- an expansion of file names, using wildcards - files will be distributed in a round robin order to the tasks
- a directory name - all files in that directory are copied in a round robin order to the tasks.

Destination is an existing destination directory name to where the data will be copied.

FLAGS

-f Is an optional flag that indicates that the first file contains the names of the source files that are to be scattered. Each file name, in the file, must be specified on a separate line. No wildcards are supported when this option is used. Directory names are not supported in the file either. When this option is used, the **mcpscat** parameters should consist of a single source file name (for the list of files) and a destination directory. The files will then be scattered just as if they had all been specified on the command line in the same order as they are listed in the file.

-i Checks for duplicate or existing files of the same name, and does not replace any existing file found. Instead, issues an error message and continues with the remaining files to be copied. Without this flag, the default action is to replace any existing files with the source file.

See Chapter 2. Executing parallel programs for information on POE options.

DESCRIPTION

The **mcpscat** function determines the order in which to distribute the files, using a round robin method, according to the list of nodes and number of tasks. Files are sent in a one-to-one correspondence to the nodes in the list of tasks. If the number of files specified is greater than the number of nodes, the remaining files are sent in another round through the list of nodes. Enclose wildcards in double quotes, otherwise they will be expanded locally on the task from where the command is issued, which may not produce the intended file name resolution.

mcpscat is a POE program and, therefore, all POE options are available. You can set POE options with either command line flags or environment variables. The number of nodes to copy the file to (**-procs**), and the message passing protocol used to copy the file (**-euilib**) are the POE options of most interest.

Return codes are:

129
invalid number of arguments specified

130
invalid option flag specified

131
unable to resolve input file name(s)

132
could not open input file for read

133
no room on destination node's file system

134
error opening file output file

135
error creating output file

136
MPI_Send of data failed

137
final **MPI_Send** failed

138
MPI_Recv failed

139
failed opening temporary file

140
failed writing temporary file

141
error renaming temp file to filename

142
input file is empty (zero byte file size)

143
invalid block size

144
error allocating storage

145
number of tasks and files do not match

146
not enough memory for list of file names

ENVIRONMENT VARIABLES

MP_BLKSIZE
sets the block size used for copying the data. This can be a value between 1 and 8,000,000 (8 megabytes). The default is 100,000 (100K).

EXAMPLES

1. You can copy a single file to all tasks into the destination directory. For example, enter:

   ```
   mcpscat filename /tmp -procs 4
   ```

   This will take the file and distribute it to tasks 0 through 3 as */tmp/filename*.
2. You can specify any number of files as source files. The destination directory must be the last item specified before any POE flags. For example:

   ```
   mcpscat file1.a file2.a file3.a file4.a file5.a /tmp -procs 4
   ```

   will take *file1.a* through *file5.a* from the local directory and copy them in a round robin order to tasks 0 through 3 into */tmp*. The file distribution will be as follows:

   ```
   Task 0: /tmp/file1.a

   Task 1: /tmp/file2.a

   Task 2: /tmp/file3.a

   Task 3: /tmp/file4.a

   Task 0: /tmp/file5.a
   ```

3. You can specify the source files to copy in a file. For example:

   ```
   mcpscat -f file.list /tmp -procs 4
   ```

   will produce the same results as the previous example if as *file.list* contains five lines with the file names *file1.a* through *file5.a* in it.
4. You can specify wildcard values to expand into a list of files to be scattered. Enter:

   ```
   mcpscat "file*.a" /tmp -procs 4
   ```

   Assuming Task 0 contains *file1.a*, *file2.a*, *file3.a*, *file4.a*, and *file5.a* in its home directory, this will result in a similar distribution as in the previous example.
5. You can specify a directory name as the source, from which the files to be scattered are found. Assuming */test* contains *myfile.a*, *myfile.b*, *myfile.c*, *myfile.d*, *myfile.f*, and *myfile.g* on Task 0, enter:

   ```
   mcpscat /test /tmp -procs 4
   ```

   This results in the following file distribution:

   ```
   Task 0: /tmp/myfile.a
   ```

```
Task 1: /tmp/myfile.b

Task 2: /tmp/myfile.c

Task 3: /tmp/myfile.d

Task 0: /tmp/myfile.f

Task 1: /tmp/myfile.g
```

mpamddir

NAME

mpamddir – echoes an amd-mountable directory name.

SYNOPSIS

mpamddir

or, if you're using the Parallel Environment for AIX:

export *MP_REMOTEDIR=mpamddir*

This script determines whether or not the current directory is a mounted file system. If it is, it looks to see if it appears in the amd maps, and constructs a name for the directory that is known to amd. You can modify this script, or create additional ones that apply to your installation.

By default, POE uses the Korn shell **pwd** command to obtain the name of the current directory to pass to the remote nodes for execution. This works for C shell users if the current directory is:

- The home directory
- Not mounted by amd, the AutoMount Daemon.

If this is not the case, (for example, if the user's current directory is a subdirectory of the home directory), then you can supply your own script for providing the name of the current directory on the remote nodes.

To use **mpamddir** as the script for providing the name, export the environment variable **MP_REMOTEDIR**, and set it to **mpamddir**.

RELATED INFORMATION

Commands: **ksh**(1), **poe**(1), **csh**(1)

mpcc

NAME

mpcc – Invokes a shell script to compile C programs.

SYNOPSIS

mpcc [*cc_flags*]... *program.c*

The **mpcc** shell script compiles C programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **cc** command can also be used on **mpcc**. For a complete listing of these flag options, refer to the manual page for the AIX **cc** command. Typical options to **mpcc** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** debugger.

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

DESCRIPTION

The **mpcc** shell script invokes the **xlc** command. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpcc** to the **xlc** command, so any of the **xlc** options can be used on the **mpcc** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically loaded.

ENVIRONMENT VARIABLES

MP_PREFIX
: sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C program, enter:

```
mpcc program.c -o program
```

FILES

When you compile a program using **mpcc**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)

RELATED INFORMATION

Commands: **mpcc_r**(1), **mpCC**(1), **mpCC_r**(1), **mpxlf**(1), **cc**(1), **pdbx**(1), **xprofiler**(1)

mpcc_r

NAME

mpcc_r – Invokes a shell script to compile C programs which use threaded MPI.

SYNOPSIS

mpcc_r [*cc_flags*]... *program.c*

The **mpcc_r** shell script compiles C programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and (optionally) Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlc_r** or **cc_r** command can also be used on **mpcc_r**. For a complete listing of these flag options, refer to the manual page for the AIX **cc_r** command. Typical options to **mpcc_r** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** debugger.

-o
names the executable.

-cpp
enables the use of full C++ bindings in MPI..

-d7
compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, the POSIX standard libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information,

Ex.1008 / Page 106 of 186
Intel Corporation et al

see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-q64
enables compiling of 64-bit applications.

DESCRIPTION

The **mpcc_r** shell script invokes the **xlc_r** command. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpcc_r** to the **xlc_r** command, so any of the **xlc_r** options can be used on the **mpcc_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_UTE
Setting this variable to *yes* causes the UTE library to be added to the link step, allowing the user to collect data from the application using PE Benchmarker. For more information, see IBM Parallel Environment for AIX: Operation and Use, Volume 2.

EXAMPLES

To compile a C program, enter:

```
mpcc_r program.c -o program
```

FILES

When you compile a program using **mpcc_r**, the following libraries are automatically selected:
/usr/lpp/ppe.poe/lib/libmpi_r.a (Message Passing Interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpCC**(1), **mpCC_r**(1), **mpcc**(1), **cc**(1), **pdbx**(1), **xprofiler**(1)

mpCC

NAME

mpCC – Invokes a shell script to compile C++ programs.

SYNOPSIS

mpCC [*xlC_flags*]... *program.C*

The **mpCC** shell script compiles C++ programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlC** command can also be used on **mpCC**. For a complete listing of these flag options, refer to the manual page for the **xlC** command. Typical options to **mpCC** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

DESCRIPTION

The **mpCC** shell script invokes the **xlC** command. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpCC** to the **xlC** command, so any of the **xlC** options can be used on the **mpCC** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX
: sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a C++ program, enter:

```
mpCC program.C -o program
```

FILES

When you compile a program using **mpCC**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)

RELATED INFORMATION

Commands: **mpCC_r**(1), **mpcc**(1), **mpcc_r**(1), **mpxlf**(1), **xlC**(1), **pdbx**(1), **xprofiler**(1)

mpCC_r

NAME

mpCC_r – Invokes a shell script to compile C++ programs which use threaded MPI.

SYNOPSIS

mpCC_r [*xlC_flags*]... *program.C*

The **mpCC_r** shell script compiles C++ programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and (optionally) Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlC_r** command can also be used on **mpCC_r**. For a complete listing of these flag options, refer to the manual page for the **xlC_r** command. Typical options to **mpCC_r** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-d7
compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, the POSIX standard libraries in AIX are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-q64
enables compiling of 64-bit applications.

DESCRIPTION

The **mpCC_r** shell script invokes the **xlC_r** command. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in "Creating a static executable" on page 9 in place of this script.

Flags are passed by **mpCC_r** to the **xlC_r** command, so any of the **xlC_r** options can be used on the **mpCC_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

ENVIRONMENT VARIABLES

MP_PREFIX
: sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_UTE
: Setting this variable to *yes* causes the UTE library to be added to the link step, allowing the user to collect data from the application using PE Benchmarker. For more information, see IBM Parallel Environment for AIX: Operation and Use, Volume 2.

EXAMPLES

To compile a C++ program, enter:

```
mpCC_r program.C -o program
```

FILES

When you compile a program using **mpCC_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpcc_r**(1), **mpcc**(1), **mpCC**(1), **xlC**(1), **pdbx**(1), **xprofiler**(1)

mpiexec

NAME

mpiexec – Invokes the Parallel Operating Environment (POE) for loading and executing programs on remote processor nodes. This command invokes the **poe** command.

SYNOPSIS

mpiexec -n *partition_size program*

The **mpiexec** command is described in the MPI-2 standard as a portable way of starting MPI jobs; it is provided here to conform with that standard. The **mpiexec** command invokes **poe** the run the specified *program*. The **mpiexec** command translates the **-n** flag to the **-procs** flag for the **poe** command. The **mpiexec** command passes all other arguments unchanged to the **poe** command. Refer to the **poe** command man page for additional details on its flags.

FLAGS

In addition to the **-n** flag described below, all **poe** command flags are accepted, and passed unchanged to the **poe** command.

If you are familiar with the description of the **mpiexec** command in the MPI-2 standard, please note that we have chosen to implement only the command syntax required for compliance with that standard. The optional flags have not been implemented, as our **poe** command, which is invoked by the **mpiexec** command, offers sufficient functionality.

-n
: Translated to the **-procs** flag and passed to the **poe** command. This determines the number of program tasks. Valid values are any number from 1 to 4096. If not set, the default is 1.

EXAMPLES

To invoke an MPI program *sample* to run as five tasks:

```
mpiexec -n 5 sample
```

RELATED INFORMATION

Commands: **poe**(1)

mpxlf

NAME

mpxlf – Invokes a shell script to compile Fortran programs.

SYNOPSIS

mpxlf [*xlf_flags*]... *program.f*

The **mpxlf** shell script compiles Fortran programs while linking in the Partition Manager, Message Passing Interface (MPI), and/or Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** debugger.

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

DESCRIPTION

The **mpxlf** shell script invokes the **xlf** command. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpxlf** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script's path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
: sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a Fortran program, enter:

```
mpxlf program.f -o program
```

FILES

When you compile a program using **mpxlf**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)

RELATED INFORMATION

Commands: **mpcc**(1), **xlf**(1), **pdbx**(1), **xprofiler**(1)

mpxlf_r

NAME

mpxlf_r – Invokes a shell script to compile Fortran programs and link them into a threaded environment.

SYNOPSIS

mpxlf_r [*xlf_flags*]... *program.f*

The **mpxlf_r** shell script compiles Fortran programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and (optionally) Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf_r**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf_r** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references. This object file is needed for debugging with the **pdbx** debugger.

-o
names the executable.

-d7
compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, the POSIX standard libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-q64
enables compiling of 64-bit applications.

DESCRIPTION

The **mpxlf_r** shell script invokes the **xlf** command. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in "Creating a static executable" on page 9 in place of this script.

Flags are passed by **mpxlf_r** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script's path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_UTE
Setting this variable to *yes* causes the UTE library to be added to the link step, allowing the user to collect data from the application using PE Benchmarker. For more information, see IBM Parallel Environment for AIX: Operation and Use, Volume 2.

EXAMPLES

To compile a Fortran program, enter:

```
mpxlf_r program.f -o program
```

FILES

When you compile a program using **mpxlf_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpcc_r**(1), **xlf**(1), **pdbx**(1), **xprofiler**(1)

mpxlf90

NAME

mpxlf90 – Invokes a shell script to compile Fortran 90 programs.

SYNOPSIS

mpxlf90 [*xlf_flags*]... *program.f*

The **mpxlf90** shell script compiles Fortran 90 programs while linking in the Partition Manager, Message Passing Interface (MPI), and Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf90**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf90** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

DESCRIPTION

The **mpxlf90** shell script invokes the **xlf** command. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpxlf90** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf90** shell script. The communication subsystem library

implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script's path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
: sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a Fortran 90 program, enter:

```
mpxlf90 program.f -o program
```

FILES

When you compile a program using **mpxlf90**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)

RELATED INFORMATION

Commands: **mpcc**(1), **mpCC**(1), **xlf**(1), **mpxlf**(1), **xprofiler**(1)

mpxlf90_r

NAME

mpxlf90_r – Invokes a shell script to compile Fortran 90 programs and link them into a threaded environment.

SYNOPSIS

mpxlf90_r [*xlf_flags*]... *program.f*

The **mpxlf90_r** shell script compiles Fortran 90 programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and (optionally) Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlf** command can also be used on **mpxlf90_r**. For a complete listing of these flag options, refer to the manual page for the **xlf** command. Typical options to **mpxlf90_r** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-d7
compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, POSIX standard libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-q64
enables compiling of 64-bit applications.

DESCRIPTION

The **mpxlf90_r** shell script invokes the **xlf** command. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in "Creating a static executable" on page 9 in place of this script.

Flags are passed by **mpxlf90_r** to the **xlf** command, so any of the **xlf** options can be used on the **mpxlf90_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script's path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_UTE
Setting this variable to *yes* causes the UTE library to be added to the link step, allowing the user to collect data from the application using PE Benchmarker. For more information, see IBM Parallel Environment for AIX: Operation and Use, Volume 2.

EXAMPLES

To compile a Fortran 90 program, enter:

```
mpxlf90_r program.f -o program
```

FILES

When you compile a program using **mpxlf90_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpcc_r**(1), **xlf**(1), **mpxlf_r**(1), **pdbx**(1), **xprofiler**(1)

mpxlf95

NAME

mpxlf95 – Invokes a shell script to compile Fortran 95 programs.

SYNOPSIS

mpxlf95 [*xlf95_flags*]... *program.f*

The **mpxlf95** shell script compiles Fortran 95 programs while linking in the Partition Manager, Message Passing Interface (MPI), and Message Passing Library (MPL).

FLAGS

Any of the compiler flags normally accepted by the **xlf95** command can also be used on **mpxlf95**. For a complete listing of these flag options, refer to the manual page for the **xlf95** command. Typical options to **mpxlf95** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

DESCRIPTION

The **mxlf95** shell script invokes the **xlf95** command. In addition, the Partition Manager and message passing interface are automatically linked in. The script creates an executable that dynamically binds with the message passing libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpxlf95** to the **xlf95** command, so any of the **xlf95** options can be used on the **mpxlf95** shell script. The communication subsystem library

implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script's path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

EXAMPLES

To compile a Fortran 95 program, enter:

```
mpxlf95 program.f -o program
```

FILES

When you compile a program using **mpxlf95**, the following libraries are automatically selected:
/usr/lpp/ppe.poe/lib/libmpi.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe.a (PE common routines)

RELATED INFORMATION

Commands: **mpcc**(1), **mpCC**(1), **xlf**(1), **mpxlf**(1), **xprofiler**(1), **mpxlf90**(1)

mpxlf95_r

NAME

mpxlf95_r – Invokes a shell script to compile Fortran 95 programs and link them into a threaded environment.

SYNOPSIS

mpxlf95_r [*xlf_flags*]... *program.f*

The **mpxlf95_r** shell script compiles Fortran 95 programs while linking in the Partition Manager, the threaded implementation of Message Passing Interface (MPI), and (optionally) Low-level Applications Programming Interface (LAPI).

FLAGS

Any of the compiler flags normally accepted by the **xlf95** command can also be used on **mpxlf95_r**. For a complete listing of these flag options, refer to the manual page for the **xlf95** command. Typical options to **mpxlf95_r** include:

-v causes a “verbose” output listing of the shell script.

-g
Produces an object file with symbol table references.

-o
names the executable.

-d7
compiles the program with POSIX Threads Draft 7 base MPI and compatibility libraries. Otherwise, POSIX standard libraries are used.

-l (lower-case L)
names additional libraries to be searched. Several libraries are automatically included, and are listed below in the CONTEXT section.

Note: Not all AIX libraries are thread safe. Verify that your intended use is supported.

-I (upper-case i)
names directories for additional includes. The directory */usr/lpp/ppe.poe/include* or the appropriate subdirectory is included automatically. Command line or makefile hard coding of include paths for PE header files should normally be avoided. Such specifications will take precedence over the directory selected by the script and may result in generating incorrect code.

-p
enables profiling with the **prof** command. For more information, see the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-pg
enables profiling with the **xprofiler** and **gprof** commands. For more information, see the “Xprofiler” chapter and the appendix on “Profiling Programs” in *IBM Parallel Environment for AIX: Operation and Use, Volume 2*

-q64
enables compiling of 64-bit applications.

DESCRIPTION

The **mpxlf95_r** shell script invokes the **xlf95** command. In addition, the Partition Manager and data communication interfaces are automatically linked in. The script creates an executable that dynamically binds with the communication subsystem libraries. If you wish to create a statically bound application, use the instructions in “Creating a static executable” on page 9 in place of this script.

Flags are passed by **mpxlf95_r** to the **xlf95** command, so any of the **xlf95** options can be used on the **mpxlf95_r** shell script. The communication subsystem library implementation is dynamically linked when you invoke the executable using the **poe** command. The value specified by the **MP_EUILIB** environment variable or the **-euilib** flag will then determine which communication subsystem library implementation is dynamically linked.

There are three versions of *mpif.h* supplied and the compilation scripts will provide the include path to select the correct version. A user specified include path provided through makefile or compilation command line flag will be searched before the script’s path. If any user specified include path provides an inappropriate copy of *mpif.h*, the script will not be able to override and select the appropriate copy.

ENVIRONMENT VARIABLES

MP_PREFIX
sets an alternate path to the scripts library. If not set or NULL, the standard path */usr/lpp/ppe.poe* is used. If this environment variable is set, then all libraries are prefixed by *$MP_PREFIX/ppe.poe*.

MP_UTE
Setting this variable to *yes* causes the UTE library to be added to the link step, allowing the user to collect data from the application using PE Benchmarker. For more information, see IBM Parallel Environment for AIX: Operation and Use, Volume 2.

EXAMPLES

To compile a Fortran 95 program, enter:

```
mpxlf95_r program.f -o program
```

FILES

When you compile a program using **mpxlf95_r**, the following libraries are automatically selected:

/usr/lpp/ppe.poe/lib/libmpi_r.a (Message passing interface, collective communication routines)
/usr/lpp/ppe.poe/lib/libppe_r.a (PE common routines)
The following library is selected if it exists as a symbolic link to /usr/lpp/ssp/css/lib/liblapi_r.a:

```
/usr/lib/liblapi_r.a
```

RELATED INFORMATION

Commands: **mpcc_r**(1), **xlf95**(1), **mpxlf_r**(1), **mpxlf95**(1), **pdbx**(1), **xprofiler**(1)

pmarray

NAME

pmarray – Starts the Program Marker Array, which is an X-Windows tool for monitoring a parallel executable's run.

SYNOPSIS

pmarray

The **pmarray** command starts the Program Marker Array X-Windows tool prior to invoking **poe**. This tool is used for run-time monitoring.

FLAGS

None.

DESCRIPTION

The **pmarray** command starts the Program Marker Array. This X-Windows run-time monitoring tool consists of a number of small squares, or lights. Each task in a parallel program has its own row of lights. Using calls to the Parallel Utility Functions enables a parallel program to control the appearance of the Program Marker Array Window. Calls to MP_MARKER (for Fortran programs) or mpc_marker (for C programs) enables a program to color lights on, and/or send output strings to the Window. Calls to MP_NLIGHTS (for Fortran programs) and mpc_nlights (for C programs) enables a program to determine the number of lights displayed per task row.

The Program Marker Array is not supported with threaded programs.

ENVIRONMENT VARIABLES

This command responds to the following environment variables:

MP_PMLIGHTS
Indicates the number of lights displayed per row on the Array. (If, when you invoke **poe**, you override **MP_PMLIGHTS** using the **-pmlights** flag, the Array will redisplay with the new number of lights per row.)

MP_PROCS
Indicates the number of program tasks. The **pmarray** command sets the number of task rows displayed in the Array equal to the value of **MP_PROCS**. (If, when you invoke **poe**, you override **MP_PROCS** using the **-procs** flag, the Array will redisplay with the new number of rows.)

MP_USRPORT
Indicates the port id used by the Partition Manager to connect to the Array. By default, the Partition Manager connects to the Array using a socket assigned to port 9999. If you get an error message indicating that the port is in use, specify a different port. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10000.

EXAMPLES

To start the Program Marker Array program as a background process, and open its window, enter:

```
pmarray &
```

FILES

/usr/lpp/X11/lib/X11/app-defaults/PMarray (Xdefaults file)

/usr/lpp/ppe.poe/samples/marker/PMarray.ad (X-Windows resource information)

RELATED INFORMATION

Commands: **poe**(1)

Subroutines: **mpc_marker**(3), **MP_MARKER**(3), **mpc_nlights**(3), **MP_NLIGHTS**(3)

"Monitoring Program Execution Using the Program Marker Array" in *IBM Parallel Environment for AIX: Operation and Use, Volume 1*

poe

NAME

poe – Invokes the Parallel Operating Environment (POE) for loading and executing programs on remote processor nodes.

SYNOPSIS

poe [**-h**] [*program*] [*program_options*]...
[**-adapter_use** *adapter_specifier*]
[**-cpu_use** *cpu_specifier*]
[**-euidevice** *device_specifier*]
[**-euilib** {*ip* | *us*}]
[**-euilibpath** *path_specifier*]
[{**-hostfile** | **-hfile**} *host_file_name*]
[**-pmd_version** {*2* | *3*}]
[**-procs** *partition_size*]
[**-pulse** *interval*]
[**-resd** {**yes** | **no**}]
[**-retry** *retry_interval*]
[**-retrycount** *retry_count*]
[**-msg_api** {**MPI** | **LAPI** | **MPI,LAPI**}]
[**-rmpool** *pool_ID*]
[**-nodes** *number_of_nodes*]
[**-tasks_per_node** *number_of_tasks*]
[**-savehostfile** *output_file_name*]
[**-cmdfile** *commands_file*]
[**-llfile** *loadleveler_job_command_file_name*]
[**-newjob** {**yes** | **no**}]
[**-pgmmodel** {**spmd** | **mpmd**}]
[**-save_llfile** *output_file_name*]
[**-labelio** {**yes** | **no**}]
[**-stdinmode** {**all** | **none** | *task_ID*}]
[**-stdoutmode** {**unordered** | **ordered** | *task_ID*}]
[{**-infolevel** | **-ilevel**} *message_level*]
[**-pmdlog** {**yes** | **no**}]
[**-buffer_mem** *memory_size*]
[**-clock_source** {**AIX** | **SWITCH**}]
[**-css_interrupt** {**yes** | **no**}]
[**-eager_limit** *size_limit*]
[**-hints_filtered** {**yes** | **no**}]
[**-intrdelay** *delay_parameter*]
[**-max_typedepth** *maximum_depth*]
[**-ionodefile** *io_node_file_name*]
[**-shared_memory** {**yes** | **no**}]
[**-use_flow_control** {**yes** | **no**}]
[**-thread_stacksize** *stacksize*]
[**-single_thread** {**no** | **yes**}]
[**-wait_mode** {**poll** | **yield** | **sleep**}]
[**-polling_interval** *interval*]
[**-retransmit_interval** *interval*]
[**-io_buffer_size** *buffer_size*]
[**-io_errlog** {**yes** | **no**}]

[**-coredir** *directory_prefix_string* | **none**]
[**-corefile_format** { *lightweight_corefile_name* | **STDERR** }]
[**-euidevelop** {**yes** | **no** | **deb** | **min** | **nor**}]
[**-pmlights** *number_of_lights*]
[**-usrport** *port_ID*] [*fence_string additional_options*]

The **poe** command invokes the Parallel Operating Environment for loading and executing programs on remote processor nodes. The operation of POE is influenced by a number of POE environment variables. The flag options on this command are each used to temporarily override one of these environment variables. User *program_options* can be freely interspersed with the flag options. If no *program* is specified, POE will either prompt you for programs to load, or, if the **MP_CMDFILE** environment variable is set, will load the partition using the specified commands file.

FLAGS

The **-h** flag, when used, must appear immediately after **poe**, and causes the **poe** man page, if it exists, to be printed to the screen.

The remaining flags you can specify on this command are used to temporarily override POE environment variables. For more information on valid values, and on what a particular flag sets, refer to the description of its associated environment variable in the ENVIRONMENT VARIABLES section. The following flags are grouped by function.

The following Partition Manager control flags override the associated environment variables.

-adapter_use
    MP_ADAPTER_USE
-cpu_use
    MP_CPU_USE
-euidevice
    MP_EUIDEVICE
-euilib
    MP_EUILIB
-euilibpath
    MP_EUILIBPATH
-hostfile or **-hfile**
    MP_HOSTFILE
-pmd_version
    MP_PMD_VERSION
-procs
    MP_PROCS
-pulse
    MP_PULSE
-resd
    MP_RESD
-retry
    MP_RETRY
-retrycount
    MP_RETRYCOUNT
-msg_api
    MP_MSG_API

-rmpool
MP_RMPOOL
-nodes
MP_NODES
-tasks_per_node
MP_TASKS_PER_NODE
-savehostfile
MP_SAVEHOSTFILE

The following Job Specification flags override the associated environment variables.
-cmdfile
MP_CMDFILE
-llfile
MP_LLFILE
-newjob
MP_NEWJOB
-pgmmodel
MP_PGMMODEL
-save_llfile
MP_SAVE_LLFILE

The following I/O Control flags override the associated environment variables.
-labelio
MP_LABELIO
-stdinmode
MP_STDINMODE
-stdoutmode
MP_STDOUTMODE

The following generation of diagnostic information flags override the associated environment variables.
-infolevel or **-ilevel**
MP_INFOLEVEL
-pmdlog
MP_PMDLOG

The following Message Passing flags override the associated environment variables.
-buffer_mem
MP_BUFFER_MEM
-clock_source
MP_CLOCK_SOURCE
-css_interrupt
MP_CSS_INTERRUPT
-eager_limit
MP_EAGER_LIMIT
-hints_filtered
MP_HINTS_FILTERED
-intrdelay
MP_INTRDELAY
-max_typedepth
MP_MAX_TYPEDEPTH
-ionodefile
MP_IONODEFILE
-shared_memory
MP_SHARED_MEMORY

-use_flow_control
MP_USE_FLOW_CONTROL
-thread_stacksize
MP_THREAD_STACKSIZE
-single_thread
MP_SINGLE_THREAD
-wait_mode
MP_WAIT_MODE
-polling_interval
MP_POLLING_INTERVAL
-retransmit_interval
MP_RETRANSMIT_INTERVAL
-io_buffer_size
MP_IO_BUFFER_SIZE
-io_errlog
MP_IO_ERRLOG

The following corefile generation flags override the associated environment variables.

-coredir
MP_COREDIR
-corefile_format
MP_COREFILE_FORMAT

The following are miscellaneous flags:

-euidevelop
Overrides the **MP_EUIDEVELOP** environment variable.

-pmlights
Determines the number of lights displayed (per row) on the Program Marker Array. This overrides the **MP_PMLIGHTS** environment variable. For more information on the Program Marker Array, refer to the manual page for the **pmarray** command.

-usrport
Overrides the **MP_USRPORT** environment variable.

DESCRIPTION

The **poe** command invokes the Parallel Operating Environment for loading and executing programs on remote nodes. You can enter it at your home node to:

- load and execute an SPMD program on all nodes of your partition.
- individually load the nodes of your partition with an MPMD job.
- load and execute a series of SPMD and MPMD programs, in individual job steps, on the same partition.
- run non-parallel programs on remote nodes.

The operation of POE is influenced by a number of POE environment variables. The flag options on this command are each used to temporarily override one of these environment variables. User *program_options* can be freely interspersed with the flag options, and *additional_options* not to be parsed by POE can be placed after a *fence_string* defined by the **MP_FENCE** environment variable. If no *program* is specified, POE will either prompt you for programs to load, or, if the **MP_CMDFILE** environment variable is set, will load the partition using the specified commands file.

The environment variables and flags that influence the operation of this command fall into distinct categories of function. They are:

- **Partition Manager control**. The environment variables and flags in this category determine the method of node allocation, message passing mechanism, and the PULSE monitor function.
- **Job specification**. The environment variables and flags in this category determine whether or not the Partition Manager should maintain the partition for multiple job steps, whether commands should be read from a file or STDIN, and how the partition should be loaded.
- **I/O control**. The environment variables and flags in this category determine how I/O from the parallel tasks should be handled. These environment variables and flags set the input and output modes, and determine whether or not output is labeled by task id.
- **Generation of diagnostic information**. The environment variables and flags in this category enable you to generate diagnostic information that may be required by the IBM Support Center in resolving PE-related problems.
- **Message Passing Interface**. The environment variables and flags in this category enable you to specify values for tuning message passing applications.
- **Corefile generation.** The environment variables and flags in this category govern aspects of corefile generation including the directory name into which corefiles will be saved, or the corefile format (standard AIX or lightweight).
- **Miscellaneous**. The additional environment variables and flags in this category enable additional error checking, and set a dispatch priority class for execution.

ENVIRONMENT VARIABLES

The environment variable descriptions in this section are grouped by function.

The following environment variables are associated with Partition Manager control.

MP_ADAPTER_USE
Determines how the node's adapter should be used. The US communication subsystem library does not require dedicated use of the SP switch on the node. Adapter use will be defaulted, as in Table 4 on page 20, but shared usage may be specified. Valid values are *dedicated* and *shared*. If not set, the default is dedicated for US jobs, or shared for IP jobs. The value of this environment variable can be overridden using the **-adapter_use** flag.

MP_CPU_USE
Determines how the node's CPUs should be used. The US communication subsystem library does not require unique CPU use on the node. CPU use will be defaulted, as in Table 4 on page 20, but multiple use may be specified. Valid values are *multiple* and *unique*. If not set, the default is *unique* for US jobs, or *multiple* for IP jobs. The value of this environment variable can be overridden using the **-cpu_use** flag.

MP_EUIDEVICE
Determines the adapter set to use for message passing. Valid values are *en0* (for Ethernet), *fi0* (for FDDI), *tr0* (for token-ring), *css0* (for the SP system's high performance switch feature), and *csss* (for the SP switch 2 high performance adapter).

MP_EUILIB
Determines the communication subsystem library implementation to use for communication – either the IP communication subsystem or the US communication subsystem. In order to use the US communication

subsystem, you must have an SP system configured with its high performance switch feature. Valid, case-sensitive, values are **ip** (for the IP communication subsystem) or **us** (for the US communication subsystem). The value of this environment variable can be overridden using the **-euilib** flag.

MP_EUILIBPATH
Determines the path to the message passing and communication subsystem libraries. This only needs to be set if an alternate library path is desired. Valid values are any path specifier. The value of this environment variable can be overridden using the **-euilibpath** flag.

MP_HOSTFILE
Determines the name of a host list file for node allocation. Valid values are any file specifier. If not set, the default is *host.list* in your current directory. The value of this environment variable can be overridden using the **-hostfile** or **-hfile** flags.

MP_PMD_VERSION
Determines which version of the partition manager daemon should be used. It is intended for situations where you need to submit a parallel job from a POE version 3 node to a set of nodes that have not yet been upgraded to version 3. Valid values are **2** (use the version 2 partition manager daemon — pmv2) and **3** (use the version 3 partition manager daemon — pmv3). If not set, the default is to use the version 3 partition manager daemon. You can override this environment variable with the **-pmd_version** flag.

MP_PROCS
Determines the number of program tasks. Valid values are any number from 1 to 4096. If not set, the default is 1. The value of this environment variable can be overridden using the **-procs** flag.

MP_PULSE
The interval (in seconds) at which POE checks the remote nodes to ensure that they are communicating with the home node. The default interval is 600 seconds (10 minutes). To disable the pulse function, specify an interval of 0 (zero) seconds. The pulse function is automatically disabled when running the **pdbx** debugger. You can override the value of this environment variable with the **-pulse** flag.

MP_REMOTEDIR
Specifies the name of a script which echoes the name of the current directory to be used on the remote nodes. By default, the current directory is the current directory at the time that POE is run. You may need to specify this if the AutoMount Daemon is used to mount user file systems, and the user is not using the Korn shell.

The script **mpamddir** is provided for mapping the C shell directory name to an AutoMount Daemon name.

MP_RESD
Determines whether or not the Partition Manager should connect to LoadLeveler to allocate nodes. Valid values are either **yes** or **no**, and there is no default. The value of this environment variable can be overridden using the **-resd** flag.

MP_RETRY
Determines the period of time (in seconds) between processor node allocation retries if there are not enough processor nodes immediately available to run a program. This is only valid if you are using LoadLeveler.

Valid values are any integer greater than or equal to 0. The default is 0 (no retry). The value of this environment variable can be overridden using the **-retry** flag.

MP_RETRYCOUNT
The number of times (at the interval set by **MP_RETRY**) that the Partition Manager should attempt to allocate processor nodes. Valid values are any integer greater than or equal to 0. If not set, the default is 0. The value of this environment variable can be overridden using the **-retrycount** flag.

MP_MSG_API
Indicates to POE which message passing API is being used by the parallel tasks. You need to set this environment variable if a parallel task is using LAPI alone or in conjunction with MPI. You do not need to set it if a parallel task is using MPI only. The value of this environment variable can be overridden using the **-msg_api** flag.

MP_RMPOOL
Determines the name or number of the pool that should be used for non-specific node allocation. This environment variable/command-line flag only applies to LoadLeveler. Valid values are any identifying pool name or number. There is no default. The value of this environment variable can be overridden using the **-rmpool** flag.

MP_NODES
Specifies the number of physical nodes on which to run the parallel tasks. It may be used alone or in conjunction with **MP_TASKS_PER_NODE** and/or **MP_PROCS**, as described in Table 6 on page 27. The value of this environment variable can be overridden using the **-nodes** flag.

MP_TASKS_PER_NODE
Specifies the number of tasks to be run on each of the physical nodes. It may be used in conjunction with **MP_NODES** and/or **MP_PROCS**, as described in Table 6 on page 27, but may not be used alone. The value of this environment variable can be overridden using the **-tasks_per_node** flag.

MP_SAVEHOSTFILE
The name of an output host list file to be generated by the Partition Manager. Valid values are any relative or full path name. The value of this environment variable can be overridden using the **-savehostfile** flag.

MP_TIMEOUT
Controls the length of time POE waits before abandoning an attempt to connect to the remote nodes. The default is 150 seconds. **MP_TIMEOUT** also changes the length of time the communication subsystem will wait for a connection to be established during message passing initialization.

If the SP security method is ″dce and compatibility″, you may need to increase the **MP_TIMEOUT** value to allow POE to wait for the DCE servers to respond (or timeout if the servers are down).

MP_CKPTFILE
Defines the base name of the checkpoint file when checkpointing a program. See “Checkpointing and restarting programs” on page 45 for more information.

MP_CKPTDIR
Defines the directory where the checkpoint file will reside when checkpointing a program. See “Checkpointing and restarting programs” on page 45 for more information.

The following environment variables are associated with Job Specification.

MP_CMDFILE
Determines the name of a POE commands file used to load the nodes of your partition. If set, POE will read the commands file rather than STDIN. Valid values are any file specifier. The value of this environment variable can be overridden using the **-cmdfile** flag.

MP_LLFILE
Determines the name of a LoadLeveler job command file for node allocation. If you are performing specific node allocation, you can use a LoadLeveler job command file in conjunction with a host list file. If you do, the specific nodes listed in the host list file will be requested from LoadLeveler. Valid values are any relative or full path name. The value of this environment variable can be overridden using the **-llfile** environment variable.

MP_NEWJOB
Determines whether or not the Partition Manager maintains your partition for multiple job steps. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-newjob** flag.

MP_PGMMODEL
Determines the programming model you are using. Valid values are **spmd** or **mpmd**. If not set, the default is **spmd**. The value of this environment variable can be overridden using the **-pgmmodel** flag.

MP_SAVE_LLFILE
When using LoadLeveler for node allocation, the name of the output LoadLeveler job command file to be generated by the Partition Manager. The output LoadLeveler job command file will show the LoadLeveler settings that result from the POE environment variables and/or command-line options for the current invocation of POE. If you use the **MP_SAVE_LLFILE** environment variable for a batch job, or when the **MP_LLFILE** environment variable is set (indicating that a LoadLeveler job command file should participate in node allocation), POE will show a warning and will not save the output job command file. Valid values are any relative or full path name. The value of this environment variable can be overridden using the **-save_llfile** flag.

The following environment variables are associated with I/O Control.

MP_LABELIO
Determines whether or not output from the parallel tasks are labeled by task id. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-labelio** flag.

MP_STDINMODE
Determines the input mode – how STDIN is managed for the parallel tasks. Valid values are:

all all tasks receive the same input data from STDIN.

none no tasks receive input data from STDIN; STDIN will be used by the home node only.

n STDIN is only sent to the task identified (*n*).

If not set, the default is **all**. The value of this environment variable can be overridden using the **-stdinmode** flag.

MP_HOLD_STDIN
Determines whether or not sending of STDIN from the home node to the remote nodes is deferred until the message passing partition has been established. Valid values are **yes** or **no**. If not set, the default is **no**.

MP_STDOUTMODE
Determines the output mode – how STDOUT is handled by the parallel tasks. Valid values are:

unordered
all tasks write output data to STDOUT asynchronously.

ordered
output data from each parallel task is written to its own buffer. Later, all buffers are flushed, in task order, to STDOUT.

a task id
only the task indicated writes output data to STDOUT.

If not set, the default is **unordered**. The value of this environment variable can be overridden using the **-stdoutmode** flag.

The following environment variables are associated with the generation of diagnostic information.

MP_INFOLEVEL
Determines the level of message reporting. Valid values are:

0 error

1 warning and error

2 informational, warning, and error

3 informational, warning, and error. Also reports diagnostic messages for use by the IBM Support Center.

4, 5, 6
Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center.

If not set, the default is **1** (warning and error). The value of this environment variable can be overridden using the **-infolevel** or **-ilevel** flags.

MP_PMDLOG
Determines whether or not diagnostic messages should be logged to a file in */tmp* on each of the remote nodes. Typically, this environment variable/command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem. Valid values are **yes** or **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-pmdlog** flag.

MP_DEBUG_INITIAL_STOP
Determines the initial breakpoint in the application where **pdbx** will get control. **MP_DEBUG_INITIAL_STOP** should be specified as *file_name:line_number*. The *line_number* is the number of the line within the source file *file_name*; where *file_name* has been compiled with **-g**. The line number has to be one that defines executable code. In general, this is a line of code for which the compiler generates machine level code. Another way to view this is that the line number is one for which debuggers will accept a breakpoint. Another valid string for **MP_DEBUG_INITIAL_STOP** would be the *function_name* of the desired initial stopping point in the

debugger. If this variable is not specified, the default is to stop at the first executable source line in the main routine. This environment variable has no associated command-line flag.

MP_PMDSUFFIX
Determines a string to be appended to the Partition Manager daemon service, or executable (when using LoadLeveler).

The PMD service in */etc/services* is named *pmv3*. By setting **MP_PMDSUFFIX**, you can append a string to *pmv3*. If **MP_PMDSUFFIX** is set to *"abc"*, for example, the service requested in */etc/services* is *pmv3abc*.

When using LoadLeveler, the string is appended to the partition manager daemon executable name, */etc/pmdv3*. By setting **MP_PMDSUFFIX**, you can append a string to *pmdv3*. If **MP_PMDSUFFIX** is set to *"abc"*, for example, the partition manager daemon that gets run on each node is */etc/pmdv3abc*.

If the **MP_PMD_VERSION** environment variable (or **-pmd_version** flag) is set to a value of *2*, the appropriate Partition Manager daemon service name and executable name is appended to *pmv2* and *pmdv2* respectively.

This permits testing of alternate versions of the Partition Manager daemon. Typically, this environment variable is used only under the direction of the IBM Support Center in resolving a PE-related problem. Valid values are any string. This environment variable has no associated command-line flag.

The following environment variables are associated with the Message Passing Interface.

MP_BUFFER_MEM
Changes the maximum size of memory used by the communication subsystem to buffer early arrivals. The default is 2.8 megabytes for IP and 64 megabytes for US. However, if checkpointing a program, for US the default will be 2.8 megabytes. If you are using this environment variable to change the maximum size of memory used by the communication subsystem while checkpointing a program, please be aware that the amount of space needed for the checkpointing files will be increased by the value of **MP_BUFFER_MEM**.

MP_CLOCK_SOURCE
Determines whether or not to use the SP switch clock as a time source. Valid values are **AIX** and **switch**. There is no default value. The value of this environment variable can be overridden using the **-clock_source** flag.

MP_CSS_INTERRUPT
Determines whether or not arriving message packets cause interrupts. This may provide better performance for certain applications. Valid values are **yes** and **no**. If not set, the default is **no**.

MP_EAGER_LIMIT
Changes the threshold value for message size, above which rendezvous protocol is used.

MP_HINTS_FILTERED
Determines whether MPI info objects reject hints (key/value pairs) which are not meaningful to the MPI implementation. In filtered mode, an **MPI_INFO_SET** call which provides a key/value pair that the implementation does not understand will behave as a no-op. A subsequent **MPI_INFO_GET** call will find that the hint does not exist in the info object.

In unfiltered mode, any key/value pair is stored and may be retrieved. Applications which wish to use MPI info objects to cache and retrieve key/value pairs other than those actually understood by the MPI implementation must use unfiltered mode. The option has no effect on the way MPI uses the hints it does understand. In unfiltered mode, there is no way for a program to discover which hints are valid to MPI and which are simply being carried as uninterpreted key/value pairs.

Providing an unrecognized hint is not an error in either mode.

Valid values for this environment variable are *yes* and *no*. If set to *yes*, unrecognized hints are be filtered. If set to *no*, they will not. If this environment variable is not set, the default is *yes*. The value of this environment variable can be overridden using the **-hints_filtered** command-line flag.

MP_INTRDELAY
Allows user programs to tune the interruptdelay parameter without having to recompile existing applications.

MP_MAX_TYPEDEPTH
Changes the maximum depth of user-defined message data types.

MP_IONODEFILE
The name of a parallel I/O node file — a text file that lists the nodes that should be handling parallel I/O. This enables you to limit the number of nodes that participate in parallel I/O, guarantee that all I/O operations are performed on the same node, and so on. Valid values are any relative or full path name. If not specified, all nodes will participate in parallel I/O operations. The value of this environment variable can be overridden using the **-ionodefile** command-line flag.

MP_SHARED_MEMORY
Determines whether or not tasks running on the same node should use shared memory (instead of the SP switch) for message passing. Valid values are **yes** and **no**. If not set, the default is **no**. The value of this environment variable can be overridden using the **-shared_memory** flag.

MP_SYNC_ON_CONNECT
Determines whether or not the internal synchronization of MPI initialization is disabled, thereby reducing the amount of network traffic. Valid values are **yes** and **no**. If not set, the default is **yes**.

MP_USE_FLOW_CONTROL
Applies flow control to MPI programs so that a sender can't outpace a receiver.

MP_THREAD_STACKSIZE
Determines the additional stacksize allocated for user programs executing on an MPI service thread. If you allocate insufficient space, the program may encounter a SIGSEGV exception.

MP_SINGLE_THREAD
Avoids mutex lock overheads in a single threaded program. This is an optimization flag, with values of **no** and **yes**. The default value is **no**, which means multiple user message passing threads are assumed.

Note: MPI-IO cannot be used if this is set to **YES**. Results are undefined if this is **YES**, with multiple message passing threads in use.

MP_WAIT_MODE
Determines how a thread or task behaves when it discovers it is blocked, waiting for a message to arrive. Values are **poll**, **yield**, **sleep**, and **nopoll**. The default mode for the signal handling library is **poll** for US, and **sleep** for IP (except on SMP nodes). The default for IP on an SMP node is **poll** if the number of tasks (for this job) on the node does not exceed the number of processors on the node. If the number of tasks does exceed the number of processors, the default is **sleep**.

MP_POLLING_INTERVAL
Defines the polling interval in microseconds. The maximum interval is approximately 2 billion microseconds (2000 seconds). The default is 180,000 microseconds for IP, and 400,000 microseconds for US.

MP_RETRANSMIT_INTERVAL
MP_RETRANSMIT_INTERVAL=*nnnnn* and its command line equivalent, **-retransmit_interval**=*nnnnn*, control how often the communication subsystem library checks to see if it should retransmit packets that have not been acknowledged. The value *nnnnn* is the number of polling loops between checks. The acceptable range is 1000 to 400000. The default is 10,000 for UDP and 400,000 for User Space.

MP_IO_BUFFER_SIZE
Indicates the default size of the data buffer used by MPI-IO agents. For example:

```
export MP_IO_BUFFER_SIZE=16M
```

sets the default size of the MPI-IO data buffer to 16MB. The default value of the environment variable is the number of bytes corresponding to 16 file blocks. This value depends on the block size associated with the file system storing the file. Valid values are any positive size up to 128MB. The size can be expressed as a number of bytes, as a number of KB (1024 bytes), using the letter **k** as a suffix, or as a number of MB (1024 * 1024 bytes), using the letter **m** as a suffix.

MP_IO_ERRLOG
Indicates whether to turn on error logging for I/O operations. For example:

```
export MP_IO_ERRLOG=yes
```

turns on error logging. When an error occurs, a line of information will be logged into file */tmp/mpi_io_errdump.app_name.userid.taskid*, recording the time the error occurs, the POSIX file system call involved, the file descriptor, and the returned error number.

The following are corefile generation environment variables:

MP_COREDIR
Creates a separate directory for each task's core file. The value of this environment variable can be overridden using the **-coredir** flag. A value of ″none″ signifies to bypass creating a new directory resulting in core files written to /tmp.

MP_COREFILE_FORMAT
Determines the format of corefiles generated when processes terminate abnormally. If not set, POE will generate standard AIX corefiles. If set to the string ″*STDERR*″, output will go to standard error. If set to any other string, POE will generate a lightweight corefile (conforming to the Parallel Tool consortium's Standardized Lightweight Corefile Format) for each process in your partition. The string you specify is the name you want to assign to

each lightweight corefile. By default, these lightweight corefiles will be saved to subdirectories prefixed by the string *coredir* and suffixed by the task id (as in *coredir.0*, *coredir.1*, and so on). You can specify a prefix other than the default *coredir* by setting the **MP_COREDIR** environment variable. The value of this environment variable can be overridden using the **-corefile_format** flag.

The following are miscellaneous environment variables:

MP_EUIDEVELOP
Determines whether or not the message passing interface performs more detailed checking during execution. This additional checking is intended for developing applications, and can significantly slow performance. Valid values are **yes** or **no**, **deb** (for “debug”), **nor** (for “normal”), and **min** (for “minimum”). The **debug** and **min** values are used to enable and disable parameter checking. If not set, the default is **no**. The value of this environment variable can be overridden using the **-euidevelop** flag.

MP_FENCE
Determines a *fence_string* to be used for separating options you want parsed by POE from those you do not. Valid values are any string, and there is no default. Once set, you can then use the *fence_string* followed by *additional_options* on the **poe** command line. The *additional_options* will not be parsed by POE. This environment variable has no associated command-line flag.

MP_NOARGLIST
Determines whether or not POE ignores the argument list. Valid values are **yes** and **no**. If set to **yes**, POE will not attempt to remove POE command-line flags before passing the argument list to the user’s program. This environment variable has no associated command-line flag.

MP_PMLIGHTS
Indicates the number of lights displayed per row on the Program Marker Array.

MP_PRIORITY
Determines a dispatch priority adjustment class for execution. See *IBM Parallel Environment for AIX: Installation* for more information on dispatch priority classes. Valid values are any of the dispatch priority classes set up by the system administrator in the file */etc/poe.priority*. This environment variable has no associated command-line flag.

MP_USRPORT
Indicates the port id used by the Partition Manager to connect to the Program Marker Array. By default, the Partition Manager connects to the Array using a socket assigned to port 9999. If you get an error message indicating that the port is in use, specify a different port. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10000.

EXAMPLES

1. Assume the **MP_PGMMODEL** environment variable is set to **spmd**, and **MP_PROCS** is set to *6*. To load and execute the SPMD program *sample* on the six remote nodes of your partition, enter:

```
poe sample
```

2. Assume you have an MPMD application consisting of two programs – *master* and *workers*. These programs are designed to run together and communicate via calls to message passing subroutines. The program *master* is designed to run on one processor node. The *workers* program is designed to run as separate tasks on any number of other nodes. The **MP_PGMMODEL** environment variable is set to **mpmd**, and **MP_PROCS** is set to *6*. To individually load the six remote nodes with your MPMD application, enter:

   ```
   poe
   ```

   Once the partition is established, the **poe** command responds with the prompt:

   ```
   0:host1_name>
   ```

   To load the *master* program as task 0 on host1_name, enter:

   ```
   master
   ```

   The **poe** command responds with a prompt for the next node to load. When you have loaded the last node of your partition, the **poe** command displays the message `Partition loaded...` and begins execution.
3. Assume you want to run three SPMD programs – *setup*, *computation*, and *cleanup* – as job steps on the same partition of nodes. The **MP_PGMMODEL** environment variable is set to *spmd*, and **MP_NEWJOB** is set to **yes**. You enter:

   ```
   poe
   ```

   Once the partition is established, the **poe** command responds with the prompt:

   ```
   Enter program name (or quit):
   ```

   To load the program *setup*, enter:

   ```
   setup
   ```

   The program setup executes on all nodes of your partition. When execution completes, the **poe** command again prompts you for a program name. Enter the program names in turn. To release the partition, enter:

   ```
   quit
   ```

4. To check the process status (using the non-parallel command **ps**) for all remote nodes in your partition, enter:

   ```
   poe ps
   ```

FILES

host.list (Default host list file)

RELATED INFORMATION

Commands: **mpcc**(1), **mpcc_r**(1), **mpCC**(1), **mpCC_r**(1), **mpxlf**(1), **mpxlf_r**(1), **pdbx**(1), **pmarray**(1), **xprofiler**(1)

poeckpt

NAME

poeckpt –takes a checkpoint of an interactive, non-LoadLeveler POE job.

SYNOPSIS

poeckpt [**-?**] [**-H**] [**-k**] [**-u** *username*] *<pid>*

FLAGS

-? Provides a short usage message.

-H
Provides help information.

-k Specifies that the job is to be terminated after a successful checkpoint.

-u
Specifies the owner of the resulting checkpoint file (used only when root invokes the **poeckpt** command).

<pid>
The process id of the POE process for the job to be checkpointed.

DESCRIPTION

poeckpt will checkpoint an interactive POE job, ensuring that job is a non-LoadLeveler POE job, running stand-alone. The process id specified corresponds to the POE process id for the job to be checkpointed. If the process specified is not a POE process or if a POE job is running under LoadLeveler, the command will fail. If the terminate option is specified and the POE job cannot be checkpointed, the terminate option is ignored and the POE job continues to run.

The **poeckpt** command will block until the checkpoint operation completes. Interrupting this command by pressing **Ctrl-c** will cause the checkpoint to be aborted.

This command must be run as the user who owns the specified process or as root. When the **-u** flag is specified and the process is being run by root, **poeckpt** will change the ownership of the checkpoint files to the user name specified. The **-u** flag is ignored when **poeckpt** is run by a non-root user.

Return codes are:

0 if successful

-1 if unsuccessful; with error message(s) containing reasons for failure.

Note: For checkpoint failures, the primary errors reported are actual error numbers as documented in */usr/include/sys/errno.h*. The secondary errors provide additional error information and are documented in */usr/include/sys/chkerror.h*. There may also be further error information reported in string format as “error data”.

poekill

NAME

poekill – terminates all remote tasks for a given program.

SYNOPSIS

poe poekill pgm_name [**poe_options**]

or

rsh remote_node poekill pgm_name

poekill is a Korn shell script that searches for the existence of running programs (named **pgm_name**) owned by the user, and terminates them via SIGTERM signals. If run under POE, **poekill** uses the standard POE mechanism for identifying the set of remote nodes (host.list, LoadLeveler, and so on). If run under rsh, **poekill** applies only to the node specified as remote_node.

FLAGS

When run as a POE program, standard POE flags apply.

DESCRIPTION

poekill determines the user id of the user that submitted the command. It then uses the id to obtain a list of active processes, which is filtered by the **pgm_name** argument into a scratch file in /tmp. The file is processed by an awk script that sends a SIGTERM signal (15) to each process in the list, and echoes the action back to the user. The scratch file is then erased, and the script exits with code of 0.

If you do not provide a pgm_name, an error message is printed and the script exits with a code of 1.

The pgm_name can be a substring of the program name.

RELATED INFORMATION

Commands: **rsh**(1), **poe**(1), **kill**(1)

poerestart

NAME

poerestart – is a command that can be used to restart an interactive POE job.

SYNOPSIS

poerestart [**-?**] [**-H**] [**-s**] <*file*>

FLAGS

-? Provides a short usage message.

-H
Provides extended help information.

-s Specifies that the same hosts should be used for the restarted job as were used for the job that was checkpointed.

<file>
The checkpoint file for the POE process.

DESCRIPTION

poerestart will restart a previously checkpointed interactive POE job, from the checkpoint file specified. Only an interactive job, stand-alone or running under LoadLeveler, can be restarted. A batch POE job cannot be restarted with this command.

Interrupting the **poerestart** command by pressing **Ctrl-c** will cause the restart operation to be aborted.

This command must be run as the user who owned the original checkpointed process.

ENVIRONMENT VARIABLES

This command responds to the following environment variables:

MP_HOSTFILE
specifies the name of the hostfile to be used. This setting is ignored if the **-s** flag is specified.

MP_RMPOOL
specifies the name of the LoadLeveler pool from which nodes will be selected to restart the job. It is an error to use this specification if the originally checkpointed POE job was not being run under LoadLeveler. This setting is ignored if:

- The **-s** flag is specified.
- **MP_HOSTFILE** is set.
- A *host.list* file exists in the directory from which the command is run.
- **MP_LLFILE** is set.

MP_LLFILE
specifies the name of the LoadLeveler job command file to be used for specification of the restarted job. This **must** be specified if the originally

checkpointed POE job used the **-llfile** command-line option or the **MP_LLFILE** environment variable for job specification. This **cannot** be used if the originally checkpointed POE job did not use the **-llfile** command-line option or the **MP_LLFILE** environment variable for job specification.

NOTES

1. When restarting a non-LoadLeveler job, or a LoadLeveler job that does not use **MP_RMPOOL** or **MP_LLFILE**, the hosts will be determined using the following:
   - The **-s** flag.
   - The **MP_HOSTFILE** environment variable.
   - A *host.list* file.
2. When **MP_LLFILE** is not being used, one of the following **must** be true:
   - The **-s** flag is specified.
   - The **MP_HOSTFILE** environment variable is set.
   - A *host.list* file exists in the directory from which the command is being run.
   - The **MP_RMPOOL** environment variable is set.
3. The following may be used in conjunction with the **MP_LLFILE** environment variable:
   - The **-s** flag.
   - The **MP_HOSTFILE** environment variable.
   - A *host.list* file in the directory from which the command is being run.
4. Any POE environment variables other than those indicated above are not used by the restarted POE.
5. The task geometry (tasks that are common within a node) for the restarted task must be the same as the originally started task.
6. This command may not be used to restart from a checkpoint file of a POE batch job. If the file provided to the **poerestart** command was generated from the checkpoint of a batch POE job, the **poerestart** command will return with no error message printed. The #@error file specified in the original batch job (if present) will contain a message indicating that this error occurred.

Retun codes are:

0 if successful

-1 if unsuccessful; with error messages containing reasons for failure.

Appendix B. POE environment variables and command-line flags

This appendix contains tables that summarize the POE environment variables and command-line flags that are discussed throughout this book. You can set these variables and flags to influence the execution of parallel programs and the operation of certain tools. A command-line flag temporarily overrides its associated environment variable. The tables divide the variables and flags by function:

- Table 7 on page 132 summarizes the variables and flags for controlling the partition manager. These variables and flags enable you to specify such things as an input or output host list file and the method of node allocation. For a complete description of the variables and flags summarized in this table, see "Chapter 2. Executing parallel programs" on page 7.
- Table 8 on page 134 summarizes the variables and flags for Job Specifications. These variables and flags determine whether or not the partition manager should maintain the partition for multiple job steps, whether commands should be read from a file or STDIN, and how the partition should be loaded. For a complete description of the variables and flags summarized in this table, see "Chapter 2. Executing parallel programs" on page 7.
- Table 9 on page 136 summarizes the variables and flags for determining how I/O from the parallel tasks should be handled. These variables and flags set the input and output modes, and determine whether or not output is labeled by task ID. For a complete description of the variables and flags summarized in this table, see "Managing standard input, output, and error" on page 37.
- Table 10 on page 136 summarizes the variables and flags for collecting diagnostic information. These variables and flags enable you to generate diagnostic information that may be required by the IBM Support Center in resolving PE-related problems.
- Table 11 on page 138 summarizes the variables and flags for the Message Passing Interface (MPI). You can use these variables and flags to change message sizes and memory sizes, as well as other message passing information.
- Table 12 on page 142 summarizes the variables and flags for core file generation.
- Table 13 on page 143 summarizes some miscellaneous variables and flags. These variables and flags provide control for the Program Marker Array, enable additional error checking, and let you set a dispatch priority class for execution.

You can use the POE command-line flags on the **pdbx** and **poe** commands. You can also use some of the following flags on program names when individually loading nodes from STDIN or a POE commands file. The flags you can use are mainly those having to do with parallel trace collection. They are:

- **-euidevelop**
- **-infolevel** or **-ilevel**

In the tables that follow, a check mark (✓) denotes those flags you can use when individually loading nodes. For more information on individually loading nodes, see "Invoking an MPMD program" on page 30.

© Copyright IBM Corp. 2000, 2001

Table 7. Variables and flags for partition manager control

Environment variable Command-line flag	Set:	Possible values	Default
MP_ADAPTER_USE -adapter_use	How the node's adapter should be used. The US communication subsystem library does not require dedicated use of the SP Switch on the node. Adapter use will be defaulted, as in Table 4 on page 20, but shared use may be specified.	One of the following strings: **dedicated** Only a single program task can use the adapter. **shared** A number of tasks on the node can use the adapter.	For IP jobs: **shared** For US jobs: **dedicated**
MP_CPU_USE -cpu_use	How the node's CPUs should be used. The US communication subsystem library does not require unique CPU use on the node. CPU use will be defaulted, as in Table 4 on page 20, but multiple use may be specified.	One of the following strings: **multiple** Your program can share the node with other users. **unique** Only your program's tasks can use the node.	For IP jobs: **multiple** For US jobs: **unique**
MP_EUIDEVICE -euidevice	The adapter set to use for message passing: Ethernet, Fiber Distributed Data Interface (FDDI), SP Switch, SP Switch2, or token ring.	One of the following strings: **css0** SP Switch **csss** SP Switch2 **en0** Ethernet **fi0** FDDI **tr0** token ring	The adapter set that is used as the external network address.
MP_EUILIB -euilib	The communication subsystem library implementation to use for communication: either the IP communication subsystem or the User Space (US) communication subsystem. Programs that use LAPI must set **MP_EUILIB** (or **-euilib**) to **us**. In order to use the US communication subsystem, you must have an SP system configured with its SP Switch feature.	One of the following strings: **ip** The IP communication subsystem. **us** The US communication subsystem. **Note:** This specification is case-sensitive.	**ip**
MP_EUILIBPATH -euilibpath	The path to the message passing and communication subsystem libraries. This only needs to be set if the libraries are moved, or an alternate set is being used.	Any path specifier.	**/usr/lpp/ppe.poe/lib**
MP_HOSTFILE -hostfile or -hfile	The name of a host list file for node allocation.	Any file specifier or the word NULL.	**host.list** in the current directory.

Table 7. Variables and flags for partition manager control (continued)

Environment variable **Command-line flag**	**Set:**	**Possible values**	**Default**
MP_NODES -nodes	To specify the number of processor nodes on which to run the parallel tasks. It can be used alone or in conjunction with **MP_PROCS** or **MP_TASKS_PER_NODE** or both, as described in Table 6 on page 27.	Any number from 1 to the maximum supported configuration.	None
MP_PMD_VERSION -pmd_version	The version of the partition manager daemon to be used for parallel jobs.	**2** The Parallel Environment version 2 partition manager daemon (**pmv2**). **3** The Parallel Environment version 3 partition manager daemon (**pmv3**).	**3** The Parallel Environment version 3 partition manager daemon (**pmv3**).
MP_PROCS -procs	The number of program tasks.	Any number from **1** to the maximum supported configuration.	**1**
MP_PULSE -pulse	The interval (in seconds) at which POE checks the remote nodes to ensure that they are actively communicating with the home node. **Note: Pulse** is ignored for **pdbx**.	An integer greater than or equal to **0**. **0** turns the pulse off.	**600**
MP_REMOTEDIR (no associated command-line flag)	The name of a script which echoes the name of the current directory to be used on the remote nodes.	Any file specifier.	None
MP_RESD -resd	Whether or not the partition manager should connect to LoadLeveler to allocate nodes. **Note:** When running POE from a workstation that is external to the LoadLeveler cluster, the **LoadL.so** fileset must be installed on the external node (see *Using and Administering LoadLeveler* and *IBM Parallel Environment for AIX: Installation* for more information).	**yes** **no**	None

Ex.1008 / Page 151 of 186
Intel Corporation et al

Table 7. Variables and flags for partition manager control (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_RETRY -retry	The period of time (in seconds) between processor node allocation retries if there are not enough processor nodes immediately available to run a program. This is only valid if you are using LoadLeveler.	An integer greater than or equal to 0.	0 (no retry)
MP_RETRYCOUNT -retrycount	The number of times (at the interval set by **MP_RETRY**) that the partition manager should attempt to allocate processor nodes.	An integer greater than or equal to 0.	0
MP_RMPOOL -rmpool	The name or number of the pool that should be used for non-specific node allocation. This environment variable/command-line flag only applies to LoadLeveler.	An identifying pool name or number.	None
MP_SAVEHOSTFILE -savehostfile	The name of an output host list file to be generated by the partition manager.	Any relative or full path name.	None
MP_TASKS_PER_NODE -tasks_per_node	To specify the number of tasks to be run on each of the physical nodes. It may be used in conjunction with **MP_NODES** or **MP_PROCS** or both, , as described in Table 6 on page 27, but may not be used alone.	Any number from 1 to the maximum supported configuration.	None

Table 8. Variables and flags for job specification

Environment variable Command-line flag	Set:	Possible values	Default
MP_CKPTDIR -ckptdir	The directory where the checkpoint file will reside. See “Checkpointing and restarting programs” on page 45 for more information.	Any path specifier	The directory from which POE is run
MP_CKPTFILE -ckptfile	The base name of the checkpoint file. See “Checkpointing and restarting programs” on page 45 for more information.	Any file specifier	None

Ex.1008 / Page 152 of 186
Intel Corporation et al

Table 8. Variables and flags for job specification (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_CMDFILE -cmdfile	The name of a POE commands file used to load the nodes of your partition. If set, POE will read the commands file rather than STDIN.	Any file specifier	None
MP_LLFILE -llfile	The name of a LoadLeveler job command file for node allocation. If you are performing specific node allocation, you can use a LoadLeveler job command file in conjunction with a host list file. If you do, the specific nodes listed in the host list file will be requested from LoadLeveler.	Any path specifier	None
MP_NEWJOB -newjob	Whether or not the partition manager maintains your partition for multiple job steps.	**yes** **no**	**no**
MP_PGMMODEL -pgmmodel	The programming model you are using.	**mpmd** **spmd**	**spmd**
MP_SAVE_LLFILE -save_llfile	When using LoadLeveler for node allocation, the name of the output LoadLeveler job command file to be generated by the partition manager. The output LoadLeveler job command file will show the LoadLeveler settings that result from the POE environment variables and/or command-line options for the current invocation of POE. If you use the **MP_SAVE_LLFILE** environment variable for a batch job, or when the **MP_LLFILE** environment variable is set (indicating that a LoadLeveler job command file should participate in node allocation), POE will show a warning and will not save the output job command file.	Any relative or full path name.	None

Ex.1008 / Page 153 of 186
Intel Corporation et al

Table 9. Variables and flags for I/O control

Environment variable **Command-line flag**	**Set:**	**Possible values**	**Default**
MP_HOLD_STDIN (no associated command-line flag)	Whether or not sending of STDIN from the home node to the remote nodes is deferred until the message passing partition has been established.	**yes** **no**	**no**
MP_LABELIO -labelio	Whether or not output from the parallel tasks is labeled by task id.	**yes** **no**	**no** **yes** (for **pdbx**)
MP_STDINMODE -stdinmode	The input mode. This determines how input is managed for the parallel tasks.	**all** All tasks receive the same input data from STDIN. **none** No tasks receive input data from STDIN; STDIN will be used by the home node only. *task-id* STDIN is only sent to the task specified by *task-id*.	**all**
MP_STDOUTMODE -stdoutmode	The output mode. This determines how STDOUT is handled by the parallel tasks.	One of the following: **ordered** Output data from each parallel task is written to its own buffer. Later, all buffers are flushed, in task order, to STDOUT. **unordered** All tasks write output data to STDOUT asynchronously. *task-id* Only the task specified by *task-id* writes output data to STDOUT.	**unordered**

Table 10. Variables and flags for diagnostic information

Environment variable **Command-line flag**	**Set:**	**Possible values**	**Default**
Note: The checkmark symbol (✓) denotes flags you can use when individually loading nodes.			
MP_DEBUG_INITIAL_STOP (no associated command-line flag)	The initial breakpoint in the application where **pdbx** will get control.	One of the following: *"filename":line_number* *function_name*	The first executable source line in the main routine.

Table 10. Variables and flags for diagnostic information (continued)

Environment variable **Command-line flag**	**Set:**	**Possible values**	**Default**
MP_INFOLEVEL -infolevel ✓ -ilevel ✓	The level of message reporting.	One of the following integers: **0** Error. **1** Warning and error. **2** Informational, warning, and error. **3** Informational, warning, and error. Also reports high-level diagnostic messages for use by the IBM Support Center. **4** Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center. **5** Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center. **6** Informational, warning, and error. Also reports high- and low-level diagnostic messages for use by the IBM Support Center.	**1**
MP_PMDLOG -pmdlog	Whether or not diagnostic messages should be logged to a file in **/tmp** on each of the remote nodes. Typically, this environment variable or command-line flag is only used under the direction of the IBM Support Center in resolving a PE-related problem.	**yes** **no**	**no**

Table 10. Variables and flags for diagnostic information (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_PMDSUFFIX (no associated command-line flag)	Determines a string to be appended to the partition manager daemon service, or executable (when using LoadLeveler). The PMD service in **/etc/services** is named **pmv3**. By setting **MP_PMDSUFFIX**, you can append a string to **pmv3**. If **MP_PMDSUFFIX** is set to **abc**, for example, the service requested in **/etc/services** is **pmv3abc**. When using LoadLeveler, the string is appended to the partition manager daemon executable name, **/etc/pmdv3**. By setting **MP_PMDSUFFIX**, you can append a string to **pmdv3**. If **MP_PMDSUFFIX** is set to **abc**, for example, the partition manager daemon that gets run on each node is **/etc/pmdv3abc**. If the **MP_PMD_VERSION** environment variable (or **-pmd_version** flag) is set to a value of **2**, the appropriate partition manager daemon service name and executable name is appended to **pmv2** and **pmdv2** respectively. This permits testing of alternate versions of the partition manager daemon. Typically, this environment variable is used only under the direction of the IBM Support Center in resolving a PE-related problem.	Any string	None

Table 11. Variables and flags for message passing

Environment variable Command-line flag	Set:	Possible values	Default
MP_BUFFER_MEM -buffer_mem	To change the maximum size of memory used by the communication subsystem to buffer early arrivals.	*nnnnn* *nnn*K (where: K = 1024 bytes) *nn*M (where: M = 1024*1024 bytes)	64MB (US) 2 800 000 bytes (IP)

Table 11. Variables and flags for message passing (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_CLOCK_SOURCE -clock_source	To use the SP Switch clock as a time source. See *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.	**AIX** **SWITCH**	None. See *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.
MP_CSS_INTERRUPT -css_interrupt	To specify whether or not arriving packets generate interrupts. Using this environment variable may provide better performance for certain applications. Setting this variable explicitly will suppress the MPI-directed switching of interrupt mode, leaving the user in control for the rest of the run. For more information, see MPI_FILE_OPEN in *IBM Parallel Environment for AIX: MPI Subroutine Reference*.	**yes** **no**	**no**
MP_EAGER_LIMIT -eager_limit	To change the threshold value for message size, above which rendezvous protocol is used. To ensure that at least 32 messages can be outstanding between any two tasks, MP_EAGER_LIMIT will be adjusted based on the number of tasks according to the table in the Default column, when the user has specified neither MP_BUFFER_MEM, nor MP_EAGER_LIMIT, nor MP_USE_FLOW_CONTROL.	An integer less than or equal to 64K, in one of these formats: *nnnnn* *nnn*K (where: K = 1024 bytes)	Number of Tasks — Default Value 1 to 16 — 4096 17 to 32 — 2048 33 to 64 — 1024 65 to 128 — 512 129 to 256 — 256 257 to the maximum number of tasks supported by the implementation — 128
MP_HINTS_FILTERED -hints_filtered	To specify whether or not MPI **info** objects reject hints (*key* and *value* pairs) that are not meaningful to the MPI implementation.	**yes** **no**	**yes**
MP_IO_BUFFER_SIZE -io_buffer_size	To specify the default size of the data buffer used by MPI-IO agents.	An integer less than or equal to 128M, in one of these formats: *nnnnn* *nnn*K (where: K = 1024 bytes) *nnn*M (where: M = 1024*1024 bytes)	The number of bytes that corresponds to 16 file blocks.

Table 11. Variables and flags for message passing (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_IO_ERRLOG -io_errlog	To specify whether or not to turn on I/O error logging.	**yes** **no**	**no**
MP_IONODEFILE -ionodefile	To specify the name of a parallel I/O node file — a text file that lists the nodes that should be handling parallel I/O. Setting this variable enables you to limit the number of nodes that participate in parallel I/O or to guarantee that all I/O operations are performed on the same node. See “Determining which nodes will participate in parallel I/O” on page 44 for more information.	Any relative path name or full path name.	None. All nodes will participate in parallel I/O.
MP_INTRDELAY -intrdelay	To tune the delay parameter without having to recompile existing applications (in microseconds).	An integer greater than or equal to **0**	**1**
MP_MAX_TYPEDEPTH -max_typedepth	To change the maximum depth of message derived datatypes.	An integer greater than or equal to **1**	**5**
MP_MSG_API -msg_api	To indicate to POE which message-passing API is being used by the parallel tasks. You need to set this environment variable if a parallel task is using LAPI alone or in conjunction with MPI. You do not need to set it if a parallel task is using MPI only.	**MPI LAPI** or **MPI,LAPI**	**MPI**
MP_PIPE_SIZE>/p> -pipe_size	To use the selected pipe size (16KB, 32KB, or 64KB) as the transmission buffer to communicate between tasks on a job. This is only effective if you are using the SP Switch2 and user space jobs.	16, 32, or 64	64
MP_POLLING_INTERVAL -polling_interval	To change the polling interval (in microseconds).	An integer between 1 and 2 billion	180 000 (IP) 400 000 (US)

Ex.1008 / Page 158 of 186
Intel Corporation et al

Table 11. Variables and flags for message passing (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_RETRANSMIT_INTERVAL -retransmit_interval	To control how often the communication subsystem library checks to see if it should retransmit packets that have not been acknowledged. The value *nnnn* is the number of polling loops between checks.	An integer between 1000 and 400 000	10 000 (UDP) 400 000 (US)
MP_SHARED_MEMORY -shared_memory	To specify the use of shared memory (instead of IP or the SP Switch) for message passing between tasks running on the same node.	**yes** **no**	**no**
MP_SINGLE_THREAD -single_thread	To avoid lock overheads in a program that is known to be single-threaded. Neither MPI-IO nor MPI 1-sided communication can be used if this variable is set to **yes**. Results are undefined if this variable is set to **yes** with multiple message threads in use. See *IBM Parallel Environment for AIX: MPI Programming Guide* for more information.	**yes** **no**	**no**
MP_SYNC_ON_CONNECT (no associated command-line flag)	To disable the internal synchronization of MPI initialization, thereby reducing the amount of network traffic. Applications that involve a large number of tasks may benefit from setting this environment variable to **no**. MP_SYNC_ON_CONNECT should only be set to **no** when it is certain that the network hardware and software are sound, and the application involves a large number of tasks that might otherwise flood the network.	**yes** **no**	**yes** or, if MP_PROCS > 128: **no**

Table 11. Variables and flags for message passing (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_THREAD_STACKSIZE -thread_stacksize	To specify the additional stack size allocated for user subroutines running on an MPI service thread. If you do not allocate enough space, the program may encounter a SIGSEGV exception or more subtle failures.	*nnnnn* *nnn*K (where: K = 1024 bytes) *nn*M (where: M = 1024*1024 bytes)	**0**
MP_TIMEOUT (no associated command-line flag)	To change the length of time (in seconds) the communication subsystem will wait for a connection to be established during message-passing initialization.	Any number greater than **0**. If set to **0** or a negative number, the value is ignored.	**150**
MP_USE_FLOW_CONTROL -use_flow_control	To throttle the sender before the number of outstanding eager send messages can overflow the early arrival buffer at a destination. Flow control insures that programs with weak synchronization and aggressive use of small messages will never overflow early arrival buffers. When flow control is on, the setting of this variable affects the default value of **MP_EAGER_LIMIT** for 16 or more tasks, if the user has specified neither **MP_BUFFER_MEM** nor **MP_EAGER_LIMIT**.	**yes** **no**	**yes**
MP_WAIT_MODE -wait_mode	To specify how a thread or task behaves when it discovers it is blocked, waiting for a message to arrive.	**poll** **sleep** **yield**	**poll** (for US) **poll** (for IP on SMP nodes, if the number of tasks for this job on the node does not exceed the number of processors on the node) otherwise: **sleep**

Table 12. Variables and flags for core file generation

Environment variable Command-line flag	Set:	Possible values	Default
MP_COREDIR -coredir	To create a separate directory for each task's core file.	Any valid directory name, or **none** to bypass creating a new directory.	*coredir.taskid*

Table 12. Variables and flags for core file generation (continued)

Environment variable Command-line flag	Set:	Possible values	Default
MP_COREFILE_FORMAT -corefile_format	The format of core files generated when processes terminate abnormally.	The string **STDERR** (to specify that the lightweight core file information should be written to standard error) or any other string (to specify the lightweight core file name).	Standard AIX core files are generated if this variable (or flag) is not set or specified.

Table 13. Other variables and flags

Environment variable Command-line flag	Set:	Possible values	Default
MP_DBXPROMPTMOD (no associated command-line flag)	A modified **dbx** prompt. The **dbx** prompt *\n(dbx)* is used by the **pdbx** command as an indicator denoting that a **dbx** subcommand has completed. This environment variable modifies that prompt. Any value assigned to it will have a "." prepended and will then be inserted in the *\n(dbx)* prompt between the "x" and the ")". This environment variable is useful when the string *\n(dbx)* is present in the output of the program being debugged.	Any string.	None
MP_EUIDEVELOP -euidevelop ✓	Whether or not the message passing interface performs more detailed checking during execution. This additional checking is intended for developing applications, and can significantly slow performance. You can also enable and disable parameter checking with **deb** (for "debug") and **min** (for "minimum").	• **deb** (for "debug") • **min** (for "minimum") • **no** or **nor** (for "normal") • **yes** (for "develop")	**no**
MP_FENCE (no associated command-line flag)	A "fence" character string for separating arguments you want parsed by POE from those you do not.	Any string.	None
MP_NOARGLIST (no associated command-line flag)	Whether or not POE ignores the argument list. If set to **yes**, POE will not attempt to remove POE command-line flags before passing the argument list to the user's program.	**yes** **no**	**no**

Ex.1008 / Page 161 of 186
Intel Corporation et al

Table 13. Other variables and flags (continued)

Environment variable **Command-line flag**	**Set:**	**Possible values**	**Default**
MP_PMLIGHTS -pmlights	The number of lights displayed (per row) on the program marker array.	An integer greater than or equal to 0.	**0**
MP_PRIORITY (no associated command-line flag)	A dispatch priority class for execution. See *IBM Parallel Environment for AIX: Installation* for more information on dispatch priority classes.	Any of the dispatch priority classes set up by the system administrator in the file **/etc/poe.priority**.	None
MP_USRPORT -usrport	The port ID used by the partition manager to connect to the program marker array.	Any positive integer less than 32 767. Standard TCP/IP practice suggests using ports greater than 5000 and less than 10 000.	**9999**

Ex.1008 / Page 162 of 186
Intel Corporation et al

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 give 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 (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, 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 sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

© Copyright IBM Corp. 2000, 2001

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department LJEB/P905
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AFS
AIX
AIX/L
AIX/L (logo)

AIX 5L
e (logo)
ESCON
IBM
IBM (logo)
IBMLink
LoadLeveler
Micro Channel
pSeries
RS/6000
SP

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, MS-DOS, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

PC Direct is a trademark of Ziff Communications Company in the United States, other countries, or both and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Acknowledgments

The PE Benchmarker product includes software developed by the Apache Software Foundation, *http://www.apache.org*.

Glossary

A

address. A value, possibly a character or group of characters that identifies a register, a device, a particular part of storage, or some other data source or destination.

AIX. Abbreviation for Advanced Interactive Executive, IBM's licensed version of the UNIX operating system. AIX is particularly suited to support technical computing applications, including high-function graphics and floating-point computations.

AIXwindows Environment/6000. A graphical user interface (GUI) for the RS/6000. It has the following components:

- A graphical user interface and toolkit based on OSF/Motif
- Enhanced X-Windows, an enhanced version of the MIT X Window System
- Graphics Library (GL), a graphical interface library for the application programmer that is compatible with Silicon Graphics' GL interface.

API. See *application programming interface.*

application. The use to which a data processing system is put; for example, topayroll application, an airline reservation application.

application programming interface. A set of programming functions and routines that provide access between the Application layer of the OSI seven-layer model and applications that want to use the network. It is a software interface.

argument. A parameter passed between a calling program and a called program or subprogram.

attribute. A named property of an entity.

B

bandwidth. The difference, expressed in hertz, between the highest and the lowest frequencies of a range of frequencies. For example, analog transmission by recognizable voice telephone requires a bandwidth of about 3000 hertz (3 kHz). The bandwidth of an optical link designates the information-carrying capacity of the link and is related to the maximum bit rate that a fiber link can support.

blocking operation. An operation that does not complete 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 operation. A communication operation in which one processor sends (or broadcasts) a message to all other processors.

buffer. A portion of storage used to hold input or output data temporarily.

C

C. A general-purpose programming language. It was formalized by Uniforum in 1983 and the ANSI standards committee for the C language in 1984.

C++. A general-purpose programming language that is based on the C language. C++ includes extensions that support an object-oriented programming paradigm. Extensions include:

- strong typing
- data abstraction and encapsulation
- polymorphism through function overloading and templates
- class inheritance.

call arc. The representation of a call between two functions within the Xprofiler function call tree. It appears as a solid line between the two functions. The arrowhead indicates the direction of the call; the function it points to is the one that receives the call. The function making the call is known as the *caller,* while the function receiving the call is known as the *callee.*

chaotic relaxation. An iterative relaxation method which uses a combination of the Gauss-Seidel and Jacobi-Seidel methods. The array of discrete values is divided into sub-regions that can be operated on in parallel. The sub-region boundaries are calculated using Jacobi-Seidel, whereas the sub-region interiors are calculated using Gauss-Seidel. See also *Gauss-Seidel.*

client. A function that requests services from a server and makes them available to the user.

cluster. A group of processors interconnected through a high-speed network that can be used for high-performance computing. A cluster typically provides excellent price/performance.

collective communication. A communication operation that involves more than two processes or tasks. Broadcasts, reductions, and the **MPI_Allreduce** subroutine are all examples of collective communication operations. All tasks in a communicator must participate.

© Copyright IBM Corp. 2000, 2001

command alias. When using the PE command line debugger **pdbx**, you can create abbreviations for existing commands using the **pdbx alias** command. These abbreviations are known as *command aliases.*

Communication Subsystem (CSS). A component of the Parallel System Support Programs that provides software support for the SP Switch. CSS provides two protocols: Internet Protocol (IP) for LAN-based communication and user space (US) as a message-passing interface that is optimized for performance over the switch. See also *Internet Protocol* and *user space.*

communicator. An MPI object that describes the communication context and an associated group of processes.

compile. To translate a source program into an executable program.

condition. One of a set of specified values that a data item can assume.

control workstation. A workstation attached to the RS/6000 SP SP that serves as a single point of control allowing the administrator or operator to monitor and manage the system using Parallel System Support Programs.

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. The current program state is needed for the programmer to diagnose and correct the problem.

core file. A file that preserves the state of a program, usually just before a program is terminated for an unexpected error. See also *core dump.*

current context. When using the **pdbx** debugger, control of the parallel program and the display of its data can be limited to a subset of the tasks belonging to that program. This subset of tasks is called the *current context.* You can set the current context to be a single task, multiple tasks, or all the tasks in the program.

D

data decomposition. A method of breaking up (or decomposing) a program into smaller parts to exploit parallelism. One divides the program by dividing the data (usually arrays) into smaller parts and operating on each part independently.

data parallelism. Refers to situations where parallel tasks perform the same computation on different sets of data.

dbx. A symbolic command line debugger that is often provided with UNIX systems. The PE command line debugger **pdbx** is based on the **dbx** debugger.

debugger. A debugger provides an environment in which you can manually control the execution of a program. It also provides the ability to display the program' data and operation.

distributed shell (dsh). A PSSP command that lets you issue commands to a group of hosts in parallel. See *IBM Parallel System Support Programs for AIX: Command and Technical Reference* for details.

domain name. The hierarchical identification of a host system (in a network), consisting of human-readable labels, separated by decimals.

E

environment variable. 1) A variable that describes the operating environment of the process. Common environment variables describe the home directory, command search path, and 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 baseband local area network (LAN) that allows multiple stations to access the transmission medium at will without prior coordination, avoids contention by using carrier sense and deference, and resolves contention by using collision detection and delayed retransmission. Ethernet uses carrier sense multiple access with collision detection (CSMA/CD).

event. An occurrence of significance to a task — the completion of an asynchronous operation such as an input/output operation, for example.

executable. A program that has been link-edited and therefore can be run in a processor.

execution. To perform the actions specified by a program or a portion of a program.

expression. In programming languages, a language construct for computing a value from one or more operands.

F

fairness. A policy in which tasks, threads, or processes must be allowed eventual 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.

FDDI. See *Fiber Distributed Data Interface.*

Ex.1008 / Page 168 of 186
Intel Corporation et al

Fiber Distributed Data Interface (FDDI). An American National Standards Institute (ANSI) standard for a local area network (LAN) using optical fiber cables. An FDDI LAN can be up to 100 kilometers (62 miles) and can include up to 500 system units. There can be up to 2 kilometers (1.24 miles) between system units and concentrators.

file system. In the AIX operating 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 functions. 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.*

foreign host. See *remote host.*

FORTRAN. One of the oldest of the modern programming languages, and the most popular language for scientific and engineering computations. Its name is a contraction of *FORmula TRANslation.* The two most common FORTRAN versions are FORTRAN 77, originally standardized in 1978, and FORTRAN 90. FORTRAN 77 is a proper subset of FORTRAN 90.

function call tree. A graphical representation of all the functions and calls within an application, which appears in the Xprofiler main window. The functions are represented by green, solid-filled rectangles called function boxes. The size and shape of each function box indicates its CPU usage. Calls between functions are represented by blue arrows, called call arcs, drawn between the function boxes. See also *call arcs.*

function cycle. A chain of calls in which the first caller is also the last to be called. A function that calls itself recursively is not considered a function cycle.

functional decomposition. A method of dividing the work in a program to exploit parallelism. One divides the program into independent pieces of functionality, which are distributed to independent processors. This method is in contrast to data decomposition, which distributes the same work over different data to independent processors.

functional parallelism. Refers to situations where parallel tasks specialize in particular work.

G

Gauss-Seidel. An iterative relaxation method for solving Laplace's equation. It calculates the general solution by finding particular solutions to a set of discrete points distributed throughout the area in question. The values of the individual points are obtained by averaging the values of nearby points. Gauss-Seidel differs from Jacobi-Seidel in that, for the i+1st iteration, Jacobi-Seidel uses only values calculated in the ith iteration. Gauss-Seidel uses a mixture of values calculated in the ith and i+1st iterations.

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 variable defined in one portion of a computer program and used in at least one other portion of the computer program.

gprof. A UNIX command that produces an execution profile of C, COBOL, FORTRAN, or Pascal programs. The execution profile is in a textual and tabular format. It is useful for identifying which routines use the most CPU time. See the man page on **gprof**.

graphical user interface (GUI). A type of computer interface consisting of a visual metaphor of a real-world scene, often of a desktop. Within that scene are icons, which represent actual objects, that the user can access and manipulate with a pointing device.

GUI. See *Graphical user interface.*

H

HIPPI. High performance parallel interface.

hook. A **pdbx** command that lets you re-establish control over all tasks in the current context that were previously unhooked with this command.

home node. The node from which an application developer compiles and runs his program. The home node can be any workstation on the LAN.

host. A computer connected to a network that provides an access method to that network. A host provides end-user services.

host list file. A file that contains a list of host names, and possibly other information, that was defined by the application which reads it.

host name. The name used to uniquely identify any computer on a network.

hot spot. A memory location or synchronization resource for which multiple processors compete excessively. This competition can cause a disproportionately large performance degradation when one processor that seeks the resource blocks, preventing many other processors from having it, thereby forcing them to become idle.

I

IBM Parallel Environment for AIX (PE). A licensed program that provides an execution and development

environment for parallel C, C++, and FORTRAN programs. PE also includes tools for debugging, profiling, and tuning parallel programs.

installation image. A file or collection of files that are required in order to install a software product on a RS/6000 workstation or on SP system nodes. These files are in a form that allows them to be installed or removed with the AIX **installp** command. See also *fileset*, *licensed program*, and *package.*

IBM Parallel System Support Programs for AIX (PSSP). A comprehensive suite of applications that is used to manage an RS/6000 SP system as a full-function parallel-processing system. PSSP provides a single point of control for administrative tasks and helps increase productivity by letting administrators view, monitor, and control how the system operates.

Internet. The collection of worldwide networks and gateways that function as a single, cooperative virtual network.

Internet Protocol (IP). 1) The TCP/IP protocol that provides packet delivery between the hardware and user processes. 2) The SP Switch library, provided with the Parallel System Support Programs, that follows the IP protocol of TCP/IP.

IP. Internet Protocol.

J

Jacobi-Seidel. See *Gauss-Seidel.*

Jumpshot. A public domain tool, developed at Argonne National Laboratory, for visualizing program performance. Jumpshot reads files in a scalable log file format, called SLOG. The PE Benchmarker toolset provides the **slogmerge** utility to enable you to convert UTE files into the SLOG format.

K

Kerberos. A publicly available security and authentication product that works with the Parallel System Support Programs software to authenticate the execution of remote commands.

kernel. The core portion of the UNIX operating system that controls the resources of the CPU and allocates them to the users. The kernel is memory-resident, is said to run in *kernel mode* (in other words, at higher execution priority level than *user mode*), and is protected from user tampering by the hardware.

L

LAPI. See *low-level communication API.*

Laplace's equation. A homogeneous partial differential equation used to describe heat transfer, electric fields, and many other applications.

The dimension-free version of Laplace's equation is:

latency. The time interval between the instant at which an instruction control unit initiates a call for data transmission, and the instant at which the actual transfer of data (or receipt of data at the remote end) begins. Latency is related to the hardware characteristics of the system and to the different layers of software that are involved in initiating the task of packing and transmitting the data.

licensed program. A collection of software packages sold as a product that customers pay for to license. A licensed program can consist of packages and filesets a customer would install. These packages and filesets bear a copyright and are offered under the terms and conditions of a licensing agreement. See also *fileset* and *package.*

lightweight corefiles. An alternative to standard AIX corefiles. Corefiles 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 corefiles.

LoadLeveler. A job management system that works with POE to let users run jobs and match processing needs with system resources, in order to make better use of the system.

local variable. A variable that is defined and used only in one specified portion of a computer program.

loop unrolling. A program transformation that makes multiple copies of the body of a loop, also placing the copies within the body of the loop. The loop trip count and index are adjusted appropriately so the new loop computes the same values as the original. This transformation makes it possible for a compiler to take additional advantage of instruction pipelining, data cache effects, and software pipelining.

See also *optimization.*

low-level communication API (LAPI). A low-level (low overhead) message-passing protocol that uses a one-sided, active-message-style interface to transfer messages between processes. LAPI is an IBM proprietary interface that exploits the SP Switch adapters.

M

menu. A list of options displayed to the user by a data processing system, from which the user can select an action to be initiated.

Ex.1008 / Page 170 of 186
Intel Corporation et al

message catalog. A file created using the AIX Message Facility from a message source file that contains application error and other messages, which can later be translated into other languages without having to recompile the application source code.

message passing. Refers to the process by which parallel tasks explicitly exchange program data.

message passing client interface (MPCI). The primary interface to the point¡to¡point message-passing protocols that support the SP Switch and the SP Switch2.

message passing interface (MPI). An industry-standard message-passing protocol that typically uses a two-sided send-receive model to transfer messages between processes.

MIMD. See *multiple instruction stream, multiple data stream.*

MPCI. See *message passing client interface.*

MPMD. See *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.

MPI. See *message passing interface.*

multiple instruction stream, multiple data stream (MIMD). A parallel programming model in which different processors perform different instructions on different sets of data.

N

netCDF file. See *network Common Data Form (netCDF) file*.

network. An interconnected group of nodes, lines, and terminals. A network provides the ability to transmit data to and receive data from other systems and users.

network Common Data Form (netCDF) file. A file using the netCDF format developed at the Unidata Program Center — a program funded by the National Science Foundation (NSF). In the PE Benchmarker toolset, the PVT reads netCDF files containing hardware and operating system profiles output by the PCT.

Network Information Services (NIS). A set of UNIX 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. (1) In a network, the point where one or more functional units interconnect transmission lines. A computer location defined in a network. (2) In terms of the RS/6000 SP, a single location or workstation in a network. An SP node is a physical entity (a processor).

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 was completed. For example, a nonblocking receive will not wait until a message is sent, but a blocking receive will wait. A nonblocking receive will return a status value that indicates whether or not a message was received.

O

object code. The result of translating a computer program to a relocatable, low-level form. Object code contains machine instructions, but symbol names (such as array, scalar, and procedure names), are not yet given a location in memory. Contrast with *source code.*

optimization. A widely-used (though not strictly accurate) term for program performance improvement, especially for performance improvement done by a compiler or other program translation software. An optimizing compiler is one that performs extensive code transformations in order to obtain an executable that runs faster but gives the same answer as the original. Such code transformations, however, can make code debugging and performance analysis very difficult because complex code transformations obscure the correspondence between compiled and original 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. A number of filesets that have been collected into a single installable image of licensed programs. Multiple filesets can be bundled together for installing groups of software together. See also *fileset* and *licensed program.*

Parallel Environment. See *IBM Parallel Environment for AIX.*

parallelism. The degree to which parts of a program may be concurrently executed.

parallelize. To convert a serial program for parallel execution.

Ex.1008 / Page 171 of 186
Intel Corporation et al

Parallel System Support Programs. See *IBM Parallel System Support Programs for AIX.*

Parallel Operating Environment (POE). An execution environment that smooths the differences between serial and parallel execution. POE lets you submit and manage parallel jobs.

parameter. (1) In FORTRAN, a symbol that is given a constant value for a specified application. (2) An item in a menu for which the operator specifies a value or for which the system provides a value when the menu is interpreted. (3) A name in a procedure that is used to refer to an argument that is passed to the procedure. (4) A particular piece of information that a system or application program needs to process a request.

partition. (1) A fixed-size division of storage. (2) In terms of the RS/6000 SP, a logical definition of nodes to be viewed as one system or domain. System partitioning is a method of organizing the SP into groups of nodes for testing or running different levels of software of product environments.

Partition Manager. The component of the Parallel Operating Environment (POE) that allocates nodes, sets up the execution environment for remote tasks, and manages distribution or collection of standard input (STDIN), standard output (STDOUT), and standard error (STDERR).

PCT. See *Performance Collection Tool.*

pdbx. The parallel, symbolic command line debugging facility of PE. **pdbx** is based on the **dbx** debugger and has a similar interface.

PE. See *IBM Parallel Environment for AIX.*

PE Benchmarker. A suite of applications and utilities that you can use to analyze the performance of programs run within IBM Parallel Environment for AIX. The PE Benchmarker toolset consists of the Performance Collection Tool (PCT), the Profile Visualization Tool (PVT), and a set of Unified Trace Environment (UTE) utilities.

Performance Collection Tool (PCT). Part of the PE Benchmarker toolset, this tool enables you to collect either MPI and user event data or hardware and operating system profiles for one or more application processes. Because it is built on dynamic instrumentation technology, the PCT enables you to insert instrumentation probes into a target application while the target application is running.

performance monitor. A utility that displays how effectively a system is being used by programs.

PID. See *process identifier.*

point-to-point communication. A communication operation which 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.

POE. See *Parallel Operating Environment.*

pool. Groups of nodes on an SP that are known to LoadLeveler, and are identified by a pool name or number.

procedure. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked 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. It 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 via 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.

process identifier (PID). An integer used by the Unix kernel to uniquely identify a process. PIDs are returned by the **fork system** call and can be passed to **wait()** or **kill()** to perform actions on the given process.

prof. A utility that produces an execution profile of an application or program. **prof** is useful for identifying which routines use the most CPU time.

Profile Visualization Tool (PVT). Part of the PE Benchmarker toolset, this tool reads the hardware or operating system profiles output by the PCT in netCDF format. The PVT enables you to summarize the collected information in reports.

profiling. The act of determining how much CPU time is used by each function or subroutine in a program. The histogram or table produced is called the execution profile.

Program Marker Array. An X-Windows run time monitor tool provided with Parallel Operating Environment, used to provide immediate visual feedback on a program's execution.

PSSP. See *IBM Parallel System Support Programs for AIX.*

pthread. A thread that conforms to the POSIX Threads Programming Model.

PVT. See *Profile Visualization Tool.*

R

reduced instruction set computer (RISC). A computer that uses a small, simplified set of frequently-used instructions for rapid execution.

reduction operation. An operation, usually mathematical, that reduces a collection of data by one or more dimensions. For example, the arithmetic SUM operation is a reduction operation which reduces an array to a scalar value. Other reduction operations include MAXVAL and MINVAL.

remote host. Any host on a network except the one at which a particular operator is working.

remote shell (rsh). A command supplied with both AIX and the Parallel System Support Programs that lets you issue commands on a remote host.

Report. In Xprofiler, a tabular listing of performance data that is derived from the **gmon.out** files of an application. Xprofiler generates five types of reports. Each type of report presents different statistical information for an application.

RISC. See *reduced instruction set computer.*

RS/6000 SP. A scalable parallel system arranged in various physical configurations that provides a high-powered computing environment.

S

segmentation fault. A system-detected error, usually caused by referencing an non-valid memory address.

server. A functional unit that provides shared services to workstations over a network — a file server, a print server, or a mail server, for example.

shell script. A sequence of commands that are to be executed by a shell interpreter such as the Bourne shell (**sh**), the C shell (**csh**), or the Korn shell (**ksh**). Script commands are stored in a file in the same form as if they were typed at a terminal.

signal handling. A type of communication that is used by message passing libraries. Signal handling involves using AIX signals as an asynchronous way to move data in and out of message buffers.

single program, multiple data (SPMD). A parallel programming model in which different processors execute the same program on different sets of data.

SLOG file. A file using the scalable log file format. The Jumpshot tool from Argonne National Laboratory reads files of this format. The **slogmerge** utility of the PE Benchmarker toolset converts UTE files into the SLOG file format.

source code. The input to a compiler or assembler, written in a source language. Contrast with *object code.*

source line. A line of source code.

SP. See *RS/6000 SP.*

SPMD. See *single program, multiple data.*

SP Switch. The high-performance message-passing network of the RS/6000 SP system that connects all processor nodes.

standard error (STDERR). (1) An output file intended to be used for error messages for C programs. (2) In many workstation-based operating systems, the output stream to which error messages or diagnostic messages are sent.

standard input (STDIN). In the AIX operating system, the primary source of data entered into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command.

standard output (STDOUT). In the AIX operating system, the primary destination of data produced by a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can go to a file or to another command.

STDERR. See *standard error.*

STDIN. See *standard input.*

STDOUT. See *standard output.*

stencil. A pattern of memory references used for averaging. A 4-point stencil in two dimensions for a given array cell, x(i,j), uses the four adjacent cells, x(i-1,j), x(i+1,j), x(i,j-1), and x(i,j+1).

subroutine. (1) A sequence of instructions whose execution is invoked by a call. (2) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. (3) A group of instructions that can be part of another routine or can be called by another program or routine.

synchronization. The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

system administrator. (1) The person at a computer installation who designs, controls, and manages the use

of the computer system. (2) The person who is responsible for setting up, modifying, and maintaining the Parallel Environment.

System Data Repository. A component of the Parallel System Support Programs software that provides configuration management for the SP system. It manages the storage and retrieval of system data across the control workstation, file servers, and nodes.

T

target application. See *DPCL target application.*

task. A unit of computation analogous to an AIX process.

thread. A single, separately dispatchable, unit of execution. There may be one or more threads in a process, and each thread is executed by the operating system concurrently.

tracing. In PE, the collection of information about the execution of the program. This information is accumulated into a trace file that can later be examined.

tracepoint. Tracepoints are places in the program that, when reached during execution, cause the debugger to print information about the state of the program.

trace record. In PE, a collection of information about a specific event that occurred during the execution of your program. For example, a trace record is created for each send and receive operation that occurs in your program (this is optional and may not be appropriate). These records are then accumulated into a trace file that can later be examined.

U

Unified Trace Environment (UTE). An environment that is used to generate, analyze, and visualize trace events for applications running on IBM RS/6000 SP systems.

unrolling loops. See *loop unrolling.*

US. See *user space.*

user. (1) A person who requires the services of a computing system. (2) Any person or any thing that may issue or receive commands and message to or from the information processing system.

user space (US). A version of the message passing library that is optimized for direct access to the SP Switch, that maximizes the performance capabilities of the SP hardware.

UTE. See *Unified Trace Environment.*

UTE interval files. A Unified Trace Environment file that is distinct from an AIX trace file by the inclusion of interval information. While an AIX trace file has a time stamp indicating the point in time when an event occurred, UTE interval files also determine how long an event lasts before encountering the next event. Because they include this duration information, UTE interval files are easier to visualize than traditional AIX event trace files.

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. (1) In programming languages, a named object that may take different values, one at a time. The values of a variable are usually restricted to one data type. (2) A quantity that can assume any of a given set of values. (3) A name used to represent a data item whose value can be changed while the program is running. (4) A name used to represent data whose value can be changed, while the program is running, by referring to the name of the variable.

view. (1) To display and look at data on screen.

(2) A special display of data, created as needed. A view temporarily ties two or more files together so that the combined files can be displayed, printed, or queried. The user specifies the fields to be included. The original files are not permanently linked or altered; however, if the system allows editing, the data in the original files will be changed.

X

X Window System. The UNIX industry's graphics windowing standard that provides simultaneous views of several executing programs or processes on high resolution graphics displays.

Xprofiler. An AIX tool that is used to analyze the performance of both serial and parallel applications, via a graphical user interface. Xprofiler provides quick access to the profiled data, so that the functions that are the most CPU-intensive can be easily identified.

Bibliography

This bibliography helps you find product documentation related to the RS/6000 SP hardware and software products.

You can find most of the IBM product information for RS/6000 SP products on the World Wide Web. Formats for both viewing and downloading are available.

PE documentation is shipped with the PE licensed program in a variety of formats and can be installed on your system. See “Accessing PE documentation online” and “Parallel Environment (PE) publications” on page 158 for more information.

This bibliography also contains a list of non-IBM publications that discuss parallel computing and other topics related to the RS/6000 SP.

Information formats

Documentation supporting RS/6000 SP software licensed programs is no longer available from IBM in hardcopy format. However, you can view, search, and print documentation in the following ways:

- On the World Wide Web
- Online (on the product media and via the SP Resource Center)

Finding documentation on the World Wide Web

Most of the RS/6000 SP hardware and software books are available from the IBM Web site at:

http://www.ibm.com/servers/eserver/pseries

The serial and parallel programs that you find in the *IBM Parallel Environment for AIX: Hitchhiker’s Guide* are also available from this Web site, in the same location as the PE online library.

You can view a book, download a Portable Document Format (PDF) version of it, or download the sample programs from the *IBM Parallel Environment for AIX: Hitchhiker’s Guide*.

At the time this manual was published, the Web address of the *RS/6000 SP Product Documentation Library* page was:

http://www.rs6000.ibm.com/resource/aix_resource/sp_books

However, the structure of the IBM Web site may change over time.

Accessing PE documentation online

On the same medium as the PE product code, IBM ships PE man pages, HTML files, and PDF files. To use the PE online documentation, you must first install these filesets:

- **ppe.html**
- **ppe.man**
- **ppe.pdf**

© Copyright IBM Corp. 2000, 2001

To view the PE HTML publications, you need access to an HTML document browser such as Netscape. The HTML files and an index that links to them are installed in the **/usr/lpp/ppe.html** directory. Once the HTML files are installed, you can also view them from the RS/6000 SP Resource Center.

To view the PE PDF publications, you need access to the Adobe Acrobat Reader. The Acrobat Reader is shipped with the AIX Bonus Pack and is also freely available for downloading from the Adobe Web site at:

http://www.adobe.com

To successfully print a large PDF file (approximately 300 or more pages) from the Adobe Acrobat reader, you may need to select the “Download Fonts Once” button on the Print window.

If you have installed the SP Resource Center on your SP system, you can access it by entering this command:

```
/usr/lpp/ssp/bin/resource_center
```

If you have the SP Resource Center on CD-ROM, see the **readme.txt** file for information about how to run it.

RS/6000 SP publications

SP planning publications

The following publications are related to this book only if you run parallel programs on the RS/6000 SP. These books are not related if you use a network cluster that is made up of IBM @server pSeries processors, IBM RS/6000 processors, or both.

- *IBM RS/6000 SP: Planning, Volume 1, Hardware and Physical Environment*, GA22-7280
- *IBM RS/6000 SP: Planning, Volume 2, Control Workstation and Software Environment*, GA22-7281

SP software publications

GPFS publications

- *IBM General Parallel File System for AIX: Administration and Programming Reference*, SA22-7452
- *IBM General Parallel File System for AIX: Concepts, Planning, and Installation Guide*, GA22-7453
- *IBM General Parallel File System for AIX: Data Management API Guide*, GA22-7435
- *IBM General Parallel File System for AIX: Problem Determination Guide*, GA22-7434

LoadLeveler publications

- *IBM LoadLeveler for AIX 5L: Diagnosis and Messages Guide*, GA22-7277
- *IBM LoadLeveler for AIX 5L: Using and Administering*, SA22-7311

Parallel Environment (PE) publications

- *IBM Parallel Environment for AIX: Hitchhiker’s Guide*, SA22-7424
- *IBM Parallel Environment for AIX: Installation*, GA22-7418

Ex.1008 / Page 176 of 186
Intel Corporation et al

- *IBM Parallel Environment for AIX: Messages*, GA22-7419
- *IBM Parallel Environment for AIX: MPI Programming Guide*, SA22-7422
- *IBM Parallel Environment for AIX: MPI Subroutine Reference*, SA22-7423
- *IBM Parallel Environment for AIX: Operation and Use, Volume 1*, SA22-7425
- *IBM Parallel Environment for AIX: Operation and Use, Volume 2*, SA22-7426

PSSP publications

The following publications are related to this book only if you run parallel programs on the RS/6000 SP. These books are not related if you use a network cluster that is made up of IBM @server pSeries processors, IBM RS/6000 processors, or both.

- *IBM Parallel System Support Programs for AIX: Administration Guide*, SA22-7348
- *IBM Parallel System Support Programs for AIX: Command and Technical Reference*, SA22-7351
- *IBM Parallel System Support Programs for AIX: Diagnosis Guide*, GA22-7350
- *IBM Parallel System Support Programs for AIX: Installation and Migration Guide*, GA22-7347
- *IBM Parallel System Support Programs for AIX: Messages Reference*, GA22-7352
- *IBM Parallel System Support Programs for AIX: Planning, Volume 2*, GA22-7281
- *IBM Parallel System Support Programs for AIX: Managing Shared Disks*, SA22-7349

RS/6000 Cluster Technology (RSCT) publications

- *IBM RS/6000 Cluster Technology: Event Management Programming Guide and Reference*, SA22-7354
- *IBM RS/6000 Cluster Technology: Group Services Programming Guide and Reference*, SA22-7355
- *IBM RS/6000 Cluster Technology: First Failure Data Capture Programming Guide and Reference*, SA22-7454

AIX publications

You can find links to the latest AIX publications on the IBM Web site at:

http://www.ibm.com/servers/aix/library/techpubs.html

DCE publications

You can view a DCE book or download a PDF version of it from the IBM Web site at:

http://www.ibm.com/software/network/dce/library

Red books

IBM's International Technical Support Organization (ITSO) has published a number of redbooks related to the RS/6000 SP. For a current list, see the IBM Web site at:

http://www.ibm.com/redbooks

Non-IBM publications

Here are some non-IBM publications that you might find helpful.

- Almasi, G. and A. Gottlieb. *Highly Parallel Computing*, Benjamin-Cummings Publishing Company, Inc., 1989.
- Bergmark, D., and M. Pottle. *Optimization and Parallelization of a Commodity Trade Model for the SP1*. Cornell Theory Center, Cornell University, June 1994.
- Foster, I. *Designing and Building Parallel Programs*, Addison-Wesley, 1995.
- Gropp, William, Ewing Lusk, and Anthony Skjellum. *Using MPI*, The MIT Press, 1994.

  As an alternative, you can use SR28-5757 to order this book through your IBM representative or IBM branch office serving your locality.
- Koelbel, Charles H., David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance FORTRAN Handbook*, The MIT Press, 1993.
- Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard, Version 1.1*, University of Tennessee, Knoxville, Tennessee, June 6, 1995.
- Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface, Version 2.0*, University of Tennessee, Knoxville, Tennessee, July 18, 1997.
- Pfister, Gregory, F. *In Search of Clusters*, Prentice Hall, 1998.
- Snir, M., Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI: The Complete Reference* The MIT Press, 1996.
- Spiegel, Murray R. *Vector Analysis* McGraw-Hill, 1959.

Permission to copy without fee all or part of Message Passing Interface Forum material is granted, provided the University of Tennessee copyright notice and the title of the document appear, and notice is given that copying is by permission of the University of Tennessee. ©1993, 1997 University of Tennessee, Knoxville, Tennessee.

For more information about the Message Passing Interface Forum and the MPI standards documents, see:

http://www.mpi-forum.org

Index

Special Characters

-buffer_mem command-line flag 138
-clock_source command-line flag 139
-css_interrupt command-line flag 139
-hints_filtered command-line flag 139
-intrdelay command-line flag 140
-io_buffer_size command-line flag 139
-io_errlog command-line flag 140
-ionodefile command-line flag 140
-max_typedepth command-line flag 140
-pipe_size command-line flag 140
-polling_interval command-line flag 140
-retransmit_interval command-line flag 141
-shared_memory command-line flag 141
-single_thread command-line flag 141
-thread_stacksize command-line flag 142
-use_flow_control command-line flag 142
-wait_mode command-line flag 142

A

abbreviated names x
acknowledgments 147
acronyms for product names x
adapter 12
AIX 1
allocating nodes 54
application 2
application programming interface (API) 65
argument 36
attribute 49
authorized access 7

B

bandwidth 1
breakpoint 120
buffer 40

C

C 8
C++ 1
C shell 61
cancelling a POE job 53
checkpointing programs 45
cluster 1
collective communication 87, 89
command-line flags
  -buffer_mem 138
  -clock_source 139
  -css_interrupt 139
  -eager_limit 139
  -hints_filtered 139
  -intrdelay 140
  -io_buffer_size 139
  -io_errlog 140
command-line flags *(continued)*
  -ionodefile 140
  -max_typedepth 140
  -pipe_size 140
  -polling_interval 140
  -retransmit_interval 141
  -shared_memory 141
  -single_thread 141
  -thread_stacksize 142
  -use_flow_control 142
  -wait_mode 142
command-line flags, POE 16, 131
commands, PE 73
Communication Subsystem (CSS) 2
communication subsystem library 2
compiling parallel programs 8
condition 59
control workstation 10
conventions x

D

dbx 4
debugger 4
dynamic libraries 9

E

environment variables
  MP_BUFFER_MEM 138
  MP_CLOCK SOURCE 139
  MP_CSS_INTERRUPT 139
  MP_EAGER_LIMIT 139
  MP_HINTS_FILTERED 139
  MP_INTRDELAY 140
  MP_IO_BUFFER_SIZE 139
  MP_IO_ERRLOG 140
  MP_IONODEFILE 140
  MP_MAX_TYPEDEPTH 140
  MP_POLLING_INTERVAL 140
  MP_RETRANSMIT_INTERVAL 141
  MP_SHARED_MEMORY 141
  MP_SINGLE_THREAD 141
  MP_SYNC_ON_CONNECT 141
  MP_THREAD_STACKSIZE 142
  MP_TIMEOUT 142
  MP_USE_FLOW_CONTROL 142
  MP_WAIT_MODE 142
  PIPE_SIZE 140
environment variables, POE 16, 131
executable 7
executing parallel programs 7
execution 1
execution environment 11
expression 70

© Copyright IBM Corp. 2000, 2001

F

file system 10
flag 8
flags, command-line
  -buffer_mem 138
  -clock_source 139
  -css_interrupt 139
  -eager_limit 139
  -hints_filtered 139
  -intrdelay 140
  -io_buffer_size 139
  -io_errlog 140
  -ionodefile 140
  -max_typedepth 140
  -pipe_size 140
  -polling_interval 140
  -retransmit_interval 141
  -shared_memory 141
  -single_thread 141
  -thread_stacksize 142
  -use_flow_control 142
  -wait_mode 142
Fortran 1
function 41, 53

G

gprof 4

H

home node 3
host list file 17
host name 17

I

IBM Parallel Environment for AIX 1
Internet Protocol (IP) 2

K

kernel 13
killing a POE job 53

L

latency 1
LoadLeveler 54
LoadLeveler, submitting a batch POE job to 57
Low-level Application Programming Interface (LAPI) 2

M

message catalog 30
message passing 2
message passing call 2
Message Passing Interface (MPI) 2
Message Passing Library (MPL) 2
message passing program 2
message passing routine 2
mixed system 1
MP_BUFFER_MEM environment variable 138
MP_CLOCK_SOURCE environment variable 139
MP_CSS_INTERRUPT environment variable 139
MP_HINTS_FILTERED environment variable 139
MP_INTRDELAY environment variable 140
MP_IO_BUFFER_SIZE environment variable 139
MP_IO_ERRLOG environment variable 140
MP_IONODEFILE environment variable 140
MP_MAX_TYPEDEPTH environment variable 140
MP_PIPE_SIZE environment variable 140
MP_POLLING_INTERVAL environment variable 140
MP_RETRANSMIT_INTERVAL environment variable 141
MP_SHARED_MEMORY environment variable 141
MP_SINGLE_THREAD environment variable 141
MP_SYNC_ON_CONNECT environment variable 141
MP_THREAD_STACKSIZE environment variable 142
MP_TIMEOUT environment variable 142
MP_USE_FLOW_CONTROL environment variable 142
MP_WAIT_MODE environment variable 142
MPMD (Multiple Program Multiple Data) 1

N

node 1
nonblocking operation 63

O

option 20

P

Parallel Environment (PE), overview 1, 4
parallel file copy utilities 66
Parallel Operating Environment (POE) 7
  executing parallel programs 7
parallel profiling capability 4
parallel programs 7
  compiling 8
  controlling program execution 35, 50
  executing 7
  monitoring execution using the Program Marker Array 69
Parallel Utility Function 41
parallelizing 2
parameter 59
partition 1
Partition Manager 3
pdbx 4
PE commands 73
  mcp 73
  mcpgath 75
  mcpscat 80
  mpamddir 84
  mpcc 85
  mpCC 89
  mpcc_r 87

PE commands *(continued)*
mpCC_r 91
mpiexec 93
mpxlf 94
mpxlf_r 96
mpxlf90 99
mpxlf90_r 101
mpxlf95 104
mpxlf95_r 106
pmarray 109
poe 111
poeckpt 126
poekill 127
poerestart 128
POE
commands file, loading nodes individually using 31
commands file, reading job steps from 33
compiling parallel programs 8
controlling program execution using 35, 50
executing non-parallel programs using 34
invoking executables in 28, 35
setting up execution environment 11
POE command-line flags 16, 131
-adapter_use 20, 132
-buffer_mem 114, 138
-ckptdir 134
-ckptfile 134
-clock_source 114, 139
-cmdfile 31, 33, 135
-coredir 142
-corefile_format 143
-cpu_use 20, 132
-css_interrupt 114, 139
-eager_limit 114, 139
-euidevelop 124, 143
-euidevice 25, 132
-euilib 24, 132
-euilibpath 24, 132
-hfile 22, 132
-hints_filtered 114, 139
-hostfile 22, 132
-ilevel 120, 137
-infolevel 120, 137
-intrdelay 114, 140
-io_buffer_size 115, 139
-io_errlog 115, 140
-ionodefile 140
-labelio 42, 136
-llfile 135
-max_typedepth 114, 140
-msg_api 140
-newjob 32, 135
-nodes 133
-pgmmodel 29, 135
-pipe_size 140
-pmd_version 133
-pmdlog 120, 137
-pmlights 110, 144
-polling_interval 115, 140
-procs 16, 133
-pulse 53, 133
POE command-line flags *(continued)*
-resd 23, 133
-retransmit_interval 115, 141
-retry 36, 134
-retrycount 36, 134
-rmpool 26, 134
-save_llfile 135
-savehostfile 21, 134
-shared_memory 141
-single_thread 115, 141
-stdinmode 38, 136
-stdoutmode 41, 136
-tasks_per_node 134
-thread_stacksize 115, 142
-use_flow_control 115, 142
-usrport 115, 144
-wait_mode 115, 142
generating diagnostic logs using 43
labeling task output using 42
maintaining partition for multiple job steps using 32
making POE wait for available nodes using 36
managing standard input using 38
managing standard output using 40
setting number of task processes 16
setting the message reporting level using 42
specifying a commands file using 31, 33
specifying a host list file 22
specifying adapter set for message passing using 25
specifying additional error checking using 35
specifying communication subsystem library implementation using 23
specifying programming model using 29
POE environment variables 16, 131
generating diagnostic logs using 43
labeling task output using 42
maintaining partition for multiple job steps using 32
making POE ignore arguments using 36
making POE wait for available nodes using 36
managing standard input using 38
managing standard output using 40
MP_ADAPTER_USE 20, 132
MP_BUFFER_MEM 121, 138
MP_CKPTDIR 35, 45, 134
MP_CKPTFILE 35, 45, 134
MP_CLOCK_SOURCE 121, 139
MP_CMDFILE 31, 33, 135
MP_COREDIR 142
MP_COREFILE_FORMAT 143
MP_CPU_USE 20, 132
MP_CSS_INTERRUPT 121, 139
MP_DBXPROMPTMOD 143
MP_DEBUG_INITIAL_STOP 120, 136
MP_EAGER_LIMIT 121, 139
MP_EUIDEVELOP 124, 143
MP_EUIDEVICE 13, 132
MP_EUILIB 13, 132
MP_EUILIBPATH 24, 132
MP_FENCE 124, 143
MP_HINTS_FILTERED 121, 139
MP_HOLD_STDIN 35, 136

POE environment variables *(continued)*
MP_HOSTFILE 12, 129, 132
MP_INFOLEVEL 120, 137
MP_INTRDELAY 122, 140
MP_IO_BUFFER_SIZE 123, 139
MP_IO_ERRLOG 123, 140
MP_IONODEFILE 140
MP_LABELIO 35, 136
MP_LLFILE 129, 135
MP_MAX_TYPEDEPTH 122, 140
MP_MSG_API 140
MP_NEWJOB 32, 135
MP_NOARGLIST 124, 143
MP_NODES 133
MP_PGMMODEL 29, 135
MP_PMD_VERSION 133
MP_PMDLOG 120, 137
MP_PMDSUFFIX 121, 138
MP_PMLIGHTS 110, 144
MP_POLLING_INTERVAL 123, 140
MP_PRIORITY 124, 144
MP_PROCS 11, 133
MP_PULSE 133
MP_REMOTEDIR 61, 133
MP_RESD 12, 133
MP_RETRANSMIT_INTERVAL 123, 141
MP_RETRY 35, 134
MP_RETRYCOUNT 35, 134
MP_RMPOOL 13, 129, 134
MP_SAVE_LLFILE 135
MP_SAVEHOSTFILE 21, 134
MP_SHARED_MEMORY 122, 141
MP_SINGLE_THREAD 122, 141
MP_STDINMODE 35, 136
MP_STDOUTMODE 35, 136
MP_SYNC_ON_CONNECT 122, 141
MP_TASKS_PER_NODE 134
MP_THREAD_STACKSIZE 122, 142
MP_TIMEOUT 118, 142
MP_USE_FLOW_CONTROL 122, 142
MP_USRPORT 110, 144
MP_WAIT_MODE 123, 142
PIPE_SIZE 140
setting number of task processes 16
setting the message reporting level using 42
specifying a commands file using 31, 33
specifying a host list file 22
specifying adapter set for message passing using 25
specifying additional error checking using 35
specifying communication subsystem library implementation using 23
specifying programming model using 29
process 34
prof 4
Program Marker Array 69
displaying details of light on 71
displaying task output on 72
Parallel Utility Functions for 70
setting the number of lights on 70
starting 71

R

remote node 3
restarting programs 45

S

serial program 1
shell script 3
source code 1
SPMD (Single Program Multiple Data) 1
standard error (STDERR) 37
standard input (STDIN) 37
standard output (STDOUT) 37
static executable 9
stopping a POE job 53
subroutine 7
system administrator 1

T

task 1
threads 2
trademarks 146

U

user 7
User Space (US) 2

V

variable 3
variables, environment
MP_BUFFER_MEM 138
MP_CLOCK_SOURCE 139
MP_CSS_INTERRUPT 139
MP_EAGER_LIMIT 139
MP_HINTS_FILTERED 139
MP_INTRDELAY 140
MP_IO_BUFFER_SIZE 139
MP_IO_ERRLOG 140
MP_IONODEFILE 140
MP_MAX_TYPEDEPTH 140
MP_POLLING_INTERVAL 140
MP_RETRANSMIT_INTERVAL 141
MP_SHARED_MEMORY 141
MP_SINGLE_THREAD 141
MP_SYNC_ON_CONNECT 141
MP_THREAD_STACKSIZE 142
MP_TIMEOUT 142
MP_USE_FLOW_CONTROL 142
MP_WAIT_MODE 142
PIPE_SIZE 140

Ex.1008 / Page 182 of 186
Intel Corporation et al

Reader's Comments– We'd like to hear from you

IBM Parallel Environment for AIX
Operation and Use, Volume 1
Using the Parallel Operating Environment
Version 3 Release 2

Publication No. SA22-7425-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SA22-7425-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie NY 12601-5400

Fold and Tape **Please do not staple** Fold and Tape

SA22-7425-01

Cut or Fold
Along Line

Program Number: 5765-D93

SA22-7425-01