

ChaMPIon/Pro 1.1.1 User Manual

Document Version 0.9
May 20, 2004

<http://www.mpi-softtech.com>
<http://www.verari.com>

Copyright © 1997-2004, MPI Software Technology, Inc. All rights reserved.
Copyright © 2004, Verari Systems, Inc. All rights reserved.
The Verari Systems Distinctive Logo is a trademark of Verari Systems, Inc.

Contents

INTRODUCTION

Purpose	1
Intended Audience.....	1
Conventions Used in the Manual	1
Organization of the manual	2
Documentation	2
Contact Information.....	3

CHAPTER 1 **PRODUCT OVERVIEW**

Overview of ChaMPIon/Pro	5
Features of ChaMPIon/Pro	6
<i>High-performance</i>	6
<i>Scalability</i>	7
<i>MPI-2 Compliance</i>	7
<i>Thread safety</i>	7
<i>Compliance with the MPI Standard “Strict Progress Rule”</i> ...	7
<i>Multi-device support</i>	8
<i>MPI-IO</i>	8
<i>Tools and Debuggers</i>	8
<i>Software Engineering process</i>	9
System Requirements	9
<i>Hardware</i>	9
<i>Operating System</i>	10
<i>Networking</i>	11
<i>File System</i>	11
<i>Software development environment</i>	12

CHAPTER 2 INSTALLATION

Installation on Linux 14
To Install ChaMPIon/Pro on Linux:..... 15
 Installation on MC/OS 16
 Installation on Mac OS X 17
To install ChaMPIon/Pro on Mac OS X:..... 17

CHAPTER 3 USING CHAMPION/PRO SDK

Building MPI Programs on Linux 20
Include Files 20
Library Files 21
Executable Utilities 21
Man Pages 22
Supported Compilers 22
Using cmpicc..... 23
Using cmpic++ 25
Using cmpifc 26
Using cmpif90c 28
Using the Profiling Interface 30
 Building MPI Program on MC/OS 30
Using the Profiling Interface 30
 Building MPI Programs on Mac OS X 31
Using cmpicc..... 31
Using cmpic++ 32
Using cmpifc 33
Using cmpif90c 35
Using the Profiling Interface 36

CHAPTER 4 RUNNING MPI PROGRAMS

Running Applications on Linux 38
Using cmpirun..... 38
Remote Shell Utility 51

<i>Redirection of stdin, stderr, and stdout</i>	51
<i>Exporting User Environment</i>	52
<i>Runtime Parameters</i>	52
<i>Selection of Communication Devices</i>	52
<i>Selection of File System</i>	53
<i>Scalable Startup</i>	54
<i>Connections on Demand (TCP Device only)</i>	54
<i>Cancellation of MPI Jobs</i>	55
<i>“Watching” MPI Ranks</i>	55
<i>Debugging in the GNU Debugger (GDB)</i>	56
<i>Debugging in TotalView™</i>	56
<i>Performance Tuning</i>	57
Running Applications on MC/OS	59
<i>Using mpirun</i>	59
<i>Machines File Format</i>	60
<i>Process Group File Format</i>	61
<i>Using mpiload</i>	62
<i>Using runmc or mpimc_load directly</i>	64
<i>Working with Licensing</i>	68
<i>Custom Startup</i>	70
<i>Runtime Parameters</i>	71
<i>Redirection of stdin, stderr, and stdout</i>	71
Running Applications on Mac OS X	72
<i>Using cpmirun</i>	72
<i>Remote Shell Utility</i>	85
<i>Redirection of stdin, stderr, and stdout</i>	85
<i>Exporting User Environment</i>	85
<i>Runtime Parameters</i>	85
<i>Selection of Communication Devices</i>	86
<i>Selection of File System</i>	87
<i>Scalable Startup</i>	87
<i>Connections on Demand (TCP Device only)</i>	88
<i>Cancellation of MPI Jobs</i>	89
<i>“Watching” MPI Ranks</i>	89
<i>Debugging in the GNU Debugger (GDB)</i>	89
<i>Performance Tuning</i>	90

Introduction

Purpose

This manual contains detailed information about ChaMPIon/Pro services and utilities that are offered to assist with building and executing MPI applications. It can be consulted for help on installation and uninstallation of ChaMPI/Pro. Some background information about configuration and setup of networking is provided.

Intended Audience

This manual has been designed for application developers who have a working knowledge of MPI and methods of writing parallel applications. It is expected that ChaMPIon/Pro will be installed and configured by users who are experienced with system administration and networking on the target platform.

Training and consulting for cluster architecture, configuration, administration and management are available from MPI Software Technology, Inc. Additionally, comprehensive training and consulting on scalable application design and “refactoring” of sequential applications for scalable computing are available. For more information, reach us as at support@mpi-softech.com.

Conventions Used in the Manual

The following typographical conventions are followed throughout the manual:

- This typeface is used to identify commands.

Example: `cmpicc -o app app.c`

- Notes are in *Italics* providing additional information on the current topic.
Example: *Note: Do not run normal installation step more than once.*
- You will also find warnings indicating a potential problem or a pitfall. There are not too many of them but when you come across one, pay attention to it.

Organization of the manual

This manual consists of Chapters 1 through 4, offering information on installation and uninstallation of ChaMPIon/Pro and how to use the product.

- Chapter 1 serves as an introduction to ChaMPIon/Pro and provides a brief [overview](#) of the features and optimizations of the product.
- Chapter 2 helps you [get started](#) with the product and discusses the installation and uninstallation process.
- Chapter 3 provides information on how to [use the ChaMPIon/Pro SDK](#).
- Chapter 4 describes the method to [run MPI applications](#) on Linux and MC/OS and Mac OS X platforms.

Documentation

The ChaMPIon/Pro SDK comes with the following documents:

- *ChaMPIon/Pro User Manual* in PDF format, which contains detailed information about ChaMPIon/Pro.
- *MPI-2 Reference Manual* is in PDF format, which contains a comprehensive listing of the MPI functions and provides many examples.
- *README* in plain text format, which contains important release information for this distribution.

Contact Information

As a commercial product, ChaMPIon/Pro is fully supported via MPI Software Technology's professional grade maintenance and support package. This service includes telephone and e-mail support and periodic software updates. For information on obtaining the product for both evaluation copy and purchasing you may contact us at:

Postal address: MPI Software Technology, Inc.
110, 12th Street North, Suite D103
Birmingham, AL 35203, USA

Phone: +1 (205) 314 3471

Fax: +1 (205) 314 3475

E-mail: championpro@mpi-softtech.com

Product Overview

1

This chapter serves as an introduction to ChaMPIon/Pro. The chapter begins with an overview of the product and proceeds to give a detailed insight into the features of ChaMPIon/Pro and system requirements.

It includes the following sections:

- Overview of ChaMPIon/Pro
- Features of ChaMPIon/Pro
- System Requirements

Overview of ChaMPIon/Pro

ChaMPIon/Pro is the next-generation, commercial, scalable middleware product from MPI Software Technology, Inc. ChaMPIon/Pro is a thread-safe, high-performance, robust, message passing middleware for clusters and dedicated multi-computers. ChaMPIon/Pro is compliant with the MPI-2.1 specification. This Software is subject to an End-User License Agreement (EULA). By using this software you agree to be bound by the terms of the EULA.

ChaMPIon/Pro supports C, C++, Fortran 77, and Fortran 90 bindings of the MPI-2 Standard. “Using the ChaMPIon/Pro SDK” section of this document describes the supported compilers and how to use the MPI profiling interface.

ChaMPIon/Pro is available for clusters as well as for multi-computer and supercomputer platforms. The following communication interfaces (devices) are supported:

- Transmission Control Protocol (TCP)
- Symmetric Multiprocessor (SMP)
- Myrinet™ (GM-1 and GM-2)
- Infiniband (Mellanox VAPI)
- IBM LAPI (IBM AIX/SP systems)
- Portals
- Quadrics QsNet™ (ELAN)
- RACE/RACE++ (Mercury MC/OS).

MercutIO, the MPI I/O component of ChaMPIon/Pro, supports the following file systems:

- UNIX File System (UFS),
- Network File System (NFS), v3,

- Parallel Virtual File System (PVFS),
- Cluster File Systems Lustre,
- Panasas File System (PanFS), and
- IBM GPFS

The base configuration of ChaMPIon/Pro for clusters is with support for TCP and SMP communication and for NFS and UFS file systems. High-speed network devices and specialized parallel file systems are optional, added-value features that may require special hardware and drivers. All ChaMPIon/Pro configurations include the base TCP+SMP and NFS+UFS capabilities.

Please verify the supported platforms and software versions with your network and file system provider before ordering ChaMPIon/Pro. The “System Requirements” and “Running MPI Programs” sections provide additional information regarding communication devices and file systems.

Features of ChaMPIon/Pro

High-performance

ChaMPIon/Pro delivers maximum performance to user applications by using efficient communication protocols and file I/O operations that allow for sustained high data throughput and low processing overhead. In addition, ChaMPIon/Pro has been designed to lower CPU utilization for communication purposes, which facilitates overlapping of computation, communication, and file I/O. In addition, ChaMPIon/Pro has a number of architectural optimizations that facilitate improved performance of collective operations, persistent mode of communication, and derived data types. ChaMPIon/Pro supports high-speed networks with OS bypass, such as Myrinet and InfiniBand.

Scalability

ChaMPIon/Pro has been designed to use a small amount of system resources for internal purposes even when the MPI jobs grow to thousands of processes. The startup facilities enable rapid launching of a large number of processes. Multiple jobs can be executed at the same time from one startup console as well as on overlapping compute hosts/nodes.

MPI-2 Compliance

ChaMPIon/Pro provides a full MPI-2.1 implementation, including C, C++, Fortran 77, and Fortran 90 bindings, as well as the standard Profiling interfaces.

Thread safety

ChaMPIon/Pro, except on MC/OS, is thread safe and a thread aware implementation of MPI, which enables hybrid parallel models using message passing between cluster nodes and multithreading for intra-node concurrency. This allows for exploitation of multi-grained parallelism. Using this feature, users can write multi-threaded programs in which all threads make concurrent MPI calls. This feature enables communication and file I/O to be handled more asynchronously and facilitates overlapping of communication and computation. Thread safety makes ChaMPIon/Pro friendly to compiler-level concurrency systems such as OpenMP. Using the terminology of the MPI-2 standard, the thread-compliance of ChaMPIon/Pro is “MPI_THREAD_MULTIPLE”.

Compliance with the MPI Standard “Strict Progress Rule”

The Strict Progress Rule supports strong communication and computation overlap. It requires message passing to continue regardless of whether the user’s MPI application is involved in a computationally intensive section of

code where message-passing functions may not ordinarily be called. ChaMPIon/Pro ensures independent message progress, which guarantees timely delivery of long messages without polling by the user thread.

ChaMPIon/Pro uses independent message progress based on a progress thread. Once a user request is posted, this request will be completed regardless of the behavior of the user process. It does not require user processes to call the library functions frequently in order to guarantee timely completion of asynchronous requests. Independent progress leads to increased effective bandwidth and also meets the requirement of the strict interpretation of the MPI Progress rule.

Multi-device support

ChaMPIon/Pro supports multiple concurrent devices on most target systems. For example, the user can start an MPI job that uses simultaneously the TCP and SMP devices or the GM and SMP devices. ChaMPIon/Pro's multi-device design allows for minimum interference of devices, which allows faster devices to operate at full speed.

MPI-IO

MPI-IO, the I/O part of the MPI-2 standard, is a parallel I/O interface designed specifically for portable, high performance parallel I/O. MPI-IO supports features that promise increased I/O parallelism and guarantee I/O portability, such as non-contiguous file access, collective I/O, asynchronous I/O, file pre-allocation, shared file pointers, and portable data representation. This feature is not yet available for MC/OS.

Tools and Debuggers

ChaMPIon/Pro interoperates with parallel programming tools, such as parallel debuggers (e.g., TotalView), performance analysis software, and

job schedulers for clusters. Support for LSF, OpenPBS, PBS/Pro, BProc, and our Cycles@Work scheduler is provided.

Software Engineering process

ChaMPIon/Pro has no legacy code limitations. The code base of this product is fully developed by MPI Software Technology, Inc. A strong software engineering approach has been consistently applied to the design, development, and release. This delivers especially high value for large systems, long-running applications, and applications that exploit a variety of MPI features and functions.

System Requirements

This section lists the system requirements of ChaMPIon/Pro.

Hardware

Linux

- Processor:
IA32: Intel Pentium II, Pentium III, Pentium IV, Xeon and 100% compatible processors, including AMD processors

IA64: Itanium, Itanium2, and AMD Opteron
- Disk space: 8 MB.
- RAM: 128MB.

MC/OS

- Processor: Motorola Power PC (PPC) G3, G4, or G5

- Disk space: 8 MB.
- RAM: 32 MB.

Mac OS X

- Processor: G4 or G5 System
- Disk Space: 8 MB
- RAM: 512MB [1GB or more Recommended]

Operating System

Linux IA32

Officially supported RedHat Linux versions with standard kernels.

Linux IA64

Officially supported RedHat Linux and United Linux versions with standard kernels.

MC/OS

- CE: MC/OS versions 5.8.0 and MC/OE 6.0.0
- Host: Solaris, vxWorks, or Windows

Mac OS X

- Mac OS X
- Apple Mac OS X 10.3.x [Panther]

Networking

Except for MC/OS, all other target platforms must satisfy these requirements in order ChaMPIon/Pro to function properly:

- Operational TCP/IP protocol stack
- Correctly configured mechanism for machine name resolution, such as DNS or `/etc/hosts`.
- Correctly configured Remote Shell services RSH or SSH.

Optional high-speed network support

Mellanox InfiniBand/VAPI, Myrinet GM (with network hardware and software cards installed according to the network manufacturer's instructions). It is very important to verify that the network driver support available on user's platform matches the drivers supported by the requested version of ChaMPIon/Pro.

If a high-speed network option is used, there must still be support for TCP/IP either through a second network interface (e.g., Ethernet) or by providing TCP/IP over the high-speed network interface.

File System

For MPI-IO support on NFS, NFS version 3 or higher is recommended. NFS version 3 is required since `fcntl` locking works correctly only in this version and higher. Also, the file system needs to be mounted with the `'noac'` (no attribute caching) option. Enabling this attribute would require administrative privileges.

Software development environment

Linux

- C/C++: GNU Compiler Collection (gcc/g++) version 3.x, Intel C/C++ version 8.0
- Fortran 77/90: GNU Fortran77 (g77), Intel Fortran 77/90 version 8.0.

MC/OS

- C/C++: ccmc and ccmc++
- Fortran 77/90: Not supported

Mac OS X

- C/C++: GNU Compiler Collection (gcc/g++) [Included with Xcode development software from Apple computer]
- Fortran 77/90: g95-3.5 can be obtained from <http://hpc.sourceforge.net> but Apple does not provide a Fortran compiler with the operating system. Support is limited.

Installation

2

This chapter explains the set up procedure of ChaMPion/Pro on Linux , MC/OS, and Mac OS X platforms. It is assumed that you have acquired a valid license key before you start installation.

This chapter covers the following sections:

- Installation on Linux
- Installation on MC/OS
- Installation on Mac OS X

Installation on Linux

ChaMPIon/Pro is distributed as an RPM package. This package must be installed on all cluster nodes as well on all nodes used for development of MPI programs. The installation is done through the RPM utility. Root access is required to perform installation.

The ChaMPIon/Pro installation CD comes with the most up to date installation instructions in a README file, alongside the RPM package. Please refer to this README for specific installation details.

Note: Any old ChaMPIon/Pro installations must be removed before the installation of this version. In order for this product to function correctly, a license key file must be set up.

The name of the ChaMPIon/Pro distribution is formed as follows:

```
cmpipro-<product_version>-<revision>.<dev>.<fs>.<os>.<arch>.rpm
```

where:

- `<product_version>` is the specific version of ChaMPIon/Pro
- `<revision>` indicates the revision of the distribution file,
- `<dev>` indicates the network support provided in the distribution (can be one of `t` for TCP, `gm` for GM-1, `gm2` for GM-2, `vapi` for InfiniBand-Mellanox/VAPI, or a combination of these),
- `<fs>` is the file system supported by the MPI I/O module of ChaMPIon/Pro (can be one of `nfs`, `pvfs`, `lustre`, or `panfs`),
- `<os>` is the operating system version (e.g., `rh80`, `rh9`, `rhel`, or `ul`),
- `<arch>` is the processor architecture of the target platform (can be one of `i386`, `ia64`, or `x86-64`).

Please verify that the ChaMPIon/Pro distribution that you have obtained from MPI Software Technology, Inc. matches your cluster environment.

To Install ChaMPIon/Pro on Linux:

1. The default location of ChaMPIon/Pro is `/usr`. It is recommended that the installation is done in the default location.

If you have ChaMPIon/Pro or other MPI implementations installed on the target platform, there is a possibility that ChaMPIon/Pro header files will overwrite the existing files. This will make compiling and running MPI programs linked against the previously installed MPI impossible. To alleviate this difficulty, ChaMPIon/Pro is provided as a relocatable RPM. See step 3 for installation with relocation.

2. [Optional] If you have a license key before installing the RPM, export the following environment variables as root:

CMPIPRO_KEY: the key string
CMPIPRO_LICENSE: the name of the license file (defaults to
`/usr/doc/compipro-1.1.1/licensekey`)

An example of how to do this in bash might be:

```
prompt$ export CMPIPRO_KEY="YOUR_KEY_HERE"  
prompt$ export CMPIPRO_LICENSE="/path/to/file/licensekey"
```

It is suggested to use the default location and name for ChaMPIon/Pro's license key file. This can be achieved by not setting CMPIPRO_LICENSE. If you use a location or file name other than the default ones, you will need to pass the `-lic_file` option to `cmpirun` for every MPI job. See documentation and man pages for `cmpirun` details.

3. If default installation location is used, execute as root the command:

```
rpm -i <ChaMPIon/Pro RPM>
```

Alternatively, if you would like to install ChaMPIon/Pro in a location other than the default one, please use the following command:

```
rpm -i --prefix /safe/path <ChaMPIon/Pro RPM>
```

You can use “`rpm -ql cmpipro`” to find the location of the executable utilities (*cmpirun*, *cmpicc*, *cmpic++*, *cmpifc*, and *cmpif90c*) and adjust your PATH appropriately.

Note: The rpm may not be relocatable on RedHat 8.0 because of a problem with the RPM manager on this RedHat version.

4. [Optional] If you wish to install the license key after installing the rpm (in case you did not select the optional step 2 above), or if you wish to install a new license key, overwriting the previous license key, run the following command as root and follow directions:

```
/path/to/cmpipro/bin/cmpipro-conf  
(usually /usr/bin/cmpipro-conf)
```

Installation on MC/OS

Champion/Pro for MC/OS is distributed as a gzip'ed tar archive (.tar.gz) for Solaris and vxWorks hosts or as a WinZip archive (.zip) for Windows hosts. In this archive are subdirectories with the libraries, header files, scripts, binaries, and example programs as well as documentation. To install, unpack the archive in a suitable location.

In order to execute Champion/Pro's utilities, add Champion/Pro's bin directory to your path or copy the contents of this directory into your Mercury software bin directory `$MC_ROOT_DIR/bin/$MC_HOST_DIR`.

In order for Champion/Pro to function correctly, a license key file must be set up. In Solaris (or under Cygwin in Windows, which is not officially supported), this can be done by running the `cmpipro-conf` shell script. Other host systems do not utilize this license file scheme. See the section on Working with Licensing for MC/OS.

Installation on Mac OS X

ChaMPIon/Pro is distributed as a Mac OS X Installer package. This package only needs to be installed on the nodes where programs are going to be built. Installation can be done one of two ways:

- Using the Mac OS X graphical installer by double-clicking the package icon
- Using the command line "installer" application.

The ChaMPIon/Pro installation CD comes with the most up-to-date installation instructions in the README file, alongside the installer package. Please refer to the README for specific installation details.

The name of the ChaMPIon/Pro distribution is formed as follows:

ChaMPIonPro-<product_version>-<revision>.<dev>.<fs>.pkg

where:

- <product_version> is the specific version of ChaMPIon/Pro.
- <revision> indicates the revision of the installer package.
- <dev> indicates the network support provided in the distribution (can be one of t for TCP, gm for GM-1, gm2 for GM-2, vapi for InfiniBand -Mellanox/VAPI, or a combination of these)
- <fs> is the file system supported by the MPI I/O module of ChaMPIon/Pro (currently only nfs is supported on Mac OS X)

Please verify that the ChaMPIon/Pro distribution that you have obtained matches your cluster environment.

To install ChaMPIon/Pro on Mac OS X:

1. The default location for ChaMPIon/Pro is */opt/cmpipro*. This is the only supported installation location. This directory was chosen to avoid conflicts overwriting other MPI implementation's *mpi.h* files.

2. Use the Graphical Installer tool by double clicking the installation package. Follow the on screen instructions. Or use the shell installer utility:

```
"sudo installer -pkg <ChampionPro Package> -target/".
```

3. Configure your path to include */opt/cmpipro/bin*.
4. Install your license key by running `sudo cmpipro-conf` and follow the prompts.

Using ChaMPIon/Pro SDK **3**

This chapter describes the contents of the ChaMPIon/Pro Software Development Kit (SDK) and the method to build MPI programs. The SDK provides all necessary libraries, include files, and build utilities for developing C and Fortran 77/90 MPI programs. Example MPI programs and makefiles provided with the distribution may also be used as guides.

This chapter contains the following sections:

- Building MPI Programs on Linux
- Building MPI Programs on MC/OS
- Building MPI Programs on Mac OS X

Building MPI Programs on Linux

Champion/Pro provides support for C, C++, Fortran 77, and Fortran 90 bindings as specified in the MPI-2 standard. Special purpose build utilities are provided to assist in building MPI programs. This section contains the list of all include files, library files, script utilities, and man pages along with a brief description of each file.

Note: A listing of files installed with Champion/Pro can be found by running the command `rpm -ql cmpipro`.

Include Files

Files	Use
<code>mpi.h</code>	To be included by C programs.
<code>mpi.hpp</code>	To be included by C++ programs.
<code>mpif.h</code>	To be included by Fortran 77 programs.
<code>mpi.f90</code>	To be used for building the Fortran 90 MPI module.
<code>mpi.mod</code>	To be used in Fortran90 programs with “USE MPI”. If this file does not exist, please see the instructions provided in the section “Using <code>mpif90c</code> ” on how to build it.
<code>pmpi.h</code>	Contains the MPI profiling interface, to be included by C profilers.
<code>pmpi.hpp</code>	Contains the MPI profiling interface, to be included by C++ profilers.

Library Files

Files	Use
<code>libcmapi.a</code>	Core ChaMPIon/Pro library that contains the MPI-2 implementation without the MPI I/O API. This library provides C bindings of the MPI-2 API.
<code>libcmapi_io.a</code>	Core ChaMPIon/Pro library that contains the full MPI-2 implementation including the MPI I/O API. This library provides C bindings of the MPI-2 API.
<code>libcmapi_fort.a</code>	MPI-2 Fortran bindings library, without MPI I/O API.
<code>libcmapi_fort_io.a</code>	Full MPI-2 Fortran bindings library, including MPI I/O API.
<code>libcmapi_cpp.a</code>	Full MPI-2 C++ bindings.
<code>libcmapi_tv_dll.so</code>	Shared object that provides ToalView's public interface.
<code>libbafsa.a</code>	Low level portable I/O library enabling multifile system support.

Executable Utilities

Utilities	Use
<code>cmpicc</code>	Executable for quick command line compiling of MPI programs written in C with CMPI.
<code>cmpic++</code>	Executable for quick command line compiling of MPI programs written in C++ with CMPI.

<code>cmpifc</code>	Executable for quick command line compiling of MPI programs written in Fortran77 with CMPI.
<code>cmpif90c</code>	Executable for quick command line compiling of MPI programs written in Fortran90 with CMPI.
<code>cmpirun</code>	Utility to run MPI programs.
<code>cmpipro-conf</code>	Utility to set up ChaMPIon/Pro license key.

Man Pages

`cmpicc.1`
`cmpifc.1`
`cmpic++.1`
`cmpirun.1`
`cmpif90c.1`

For GM, the `GM_HOME` environment variable must be set to the GM software installation directory before running these scripts. For VAPI, the `MTHOME` environment variable must be set to the InfiniBand driver installation directory (the Mellanox driver will automatically set this variable for users). This is necessary in order to allow MPI scripts to find and link with proper driver libraries.

Supported Compilers

ChaMPIon/Pro supports the following compilers:

C/C++ applications:

- GNU gcc and g++ 3.2 or later
- Intel C/C++ version 8.0 or later

Fortran 77/90 applications:

- GNU Fortran g77,
- Intel Fortran 8.0 or later,

Using **mpicc**

mpicc is a wrapper around the C compiler and is used for building MPI programs with ChaMPIon/Pro. **mpicc** can be used on the build command line in place of the compiler. Any option that is not recognized by **mpicc** will be passed on to the compiler as a compiler option.

The default compiler used by **mpicc** is **gcc**. The compiler invoked by **mpicc** can be controlled in the following two ways:

1. Using the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The supported compiler families are GNU and INTEL. When `MSTI_COMPILER` is set to GNU, **mpicc** invokes `gcc`; when `MSTI_COMPILER` is set to INTEL, **mpicc** invokes `icc`.
2. Using **mpicc**'s command line options `-gnu`, `-icc`, and `-ccl`. These options instruct **mpicc** what compiler is to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

Note: User programs must include `mpi.h`.

The following **mpicc** options are supported (as displayed by the **mpicc** help screen):

Options	Use
<code>-h, --help</code>	Show the help screen with all options.
<code>-echo</code>	Show what the command line that mpicc constructs is.
<code>-c</code>	Compile only, do not link.
<code>-gnu</code>	Use GNU C compiler (<code>gcc</code>).
<code>-icc</code>	Use Intel C/C++ compiler.

Options	Use
<code>-ccl <compiler></code>	User specified compiler.
<code>-io</code>	Build an MPI program that uses MPI I/O calls.
<code>-bproc</code>	Build a program with BProc support.
<code>-tv</code>	Build for use with TotalView debugger.

Note: Options other than the ones that are listed above will be passed as compiler or linker options.

Example uses of **mpicc** are as follows:

- `mpicc -c app.c`
- `mpicc -c app.c -o app.o`
- `mpicc -icc gcc3 app.c -o app`
- `mpicc -io -o ioapp ioapp.c`
- `mpicc -tv -o dbgapp ioapp.c`

Using `cmpic++`

The `cmpic++` utility is a wrapper around the C++ compiler and is used for building MPI programs with ChaMPIon/Pro. Its use is identical to the use of `mpicc`. `cmpic++` can be used on the build command line in place of the compiler. Any option that is not recognized by `cmpic++` will be passed on to the compiler as a compiler option.

The default compiler used by `cmpic++` is `g++`. The compiler invoked by `cmpic++` can be controlled in the following two ways:

1. Using the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The supported compiler families are GNU and INTEL. When `MSTI_COMPILER` is set to GNU, `cmpic++` invokes `g++`; when `MSTI_COMPILER` is set to INTEL, `cmpic++` invokes `icc`.
2. Using `cmpic++`'s command line options `-gnu`, `-icc`, and `-ccl`. These options instruct `cmpic++` what compiler to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

Note: User programs must include `mpi.hpp`.

The following table lists the supported options:

Options	Use
<code>-h, --help</code>	Show the help screen with all options.
<code>-echo</code>	Show what the command line <code>cmpic++</code> constructs is.
<code>-c</code>	Compile only, do not link.
<code>-gnu</code>	Use GNU C++ compiler (<code>g++</code>).
<code>-icc</code>	Use Intel C/C++ compiler.
<code>-ccl <compiler></code>	User specified compiler.
<code>-tv</code>	Build for use with TotalView debugger.

Note: Options other than the ones that are listed above will be passed as compiler or linker options.

Using **cmpifc**

cmpifc is a wrapper around the FORTRAN 77 compiler and is used for building MPI programs with ChaMPIon/Pro. **cmpifc** can be used on the build command line in place of the compiler. Any option that is not recognized by **cmpic++** will be passed on to the compiler as a compiler option.

The default compiler used by **cmpifc** is **g77**. The compiler invoked by **cmpifc** can be controlled in the following two ways:

1. Using the value of the **MSTI_COMPILER** environment variable, which specifies a compiler family. The supported compiler families are GNU and INTEL. When **MSTI_COMPILER** is set to GNU, **cmpifc** invokes **g77**; when **MSTI_COMPILER** is set to INTEL, **cmpifc** invokes **ifort**.
2. The second mechanism for selecting compiler is by using **cmpifc**'s command line options **-gnu**, **-ifort**, **-pgf**, and **-ccl**. These options instruct **cmpifc** what compiler to be used for building MPI programs. The command line options take precedence to **MSTI_COMPILER**.

Note: User FORTRAN 77 programs must have the 'include mpif.h' statement in the beginning.

The following table lists the supported options for **cmpifc**:

Options	Use
-h, --help	Show the help screen with all the options.
-echo	Show what the command line that cmpifc constructs is.
-c	Compile only, do not link.

Options	Use
<code>-gnu</code>	Use GNU Fortran77 compiler (g77).
<code>-ifort</code>	Use Intel Fortran 'ifort' compiler.
<code>-pgf</code>	Use Portland group's pgf77 compiler.
<code>-ccl <compiler></code>	User specified Fortran 77 compiler.
<code>-io</code>	Build with MPI IO support.
<code>-bproc</code>	Link with BProc support.
<code>-tv</code>	Compile for use with TotalView debugger.

Note: Options other than the ones that are specified here will be passed as compiler or linker options.

Example uses of **cmpifc**:

- `cmpifc -c app.f`
- `cmpifc -o fapp.o fapp.f`
- `cmpifc -ifort fapp.f -o fapp`

Using `cmpif90c`

cmpif90c is a wrapper around the FORTRAN90 compiler and is used for building MPI programs with ChaMPIon/Pro. `cmpif90c` can be used on the build command line in place of the compiler. Any option that is not recognized by `cmpif90c` will be passed on to the compiler as a compiler option.

The default compiler used by **cmpif90c** is Intel **ifort**. The compiler invoked by `cmpif90c` can be controlled in the following two ways:

1. The first uses the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The only supported compiler family at present is `INTEL`. When `MSTI_COMPILER` is set to `INTEL`, `cmpif90c` invokes `ifort`.
2. The second mechanism for selecting compiler is by using `cmpif90c`'s command line options `-ifort`, `-pgf`, and `-ccl`. These options instruct `cmpif90c` what compiler to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

According to the MPI-2 standard, FORTRAN90 programs must use an MPI module. ChaMPIon/Pro provides the source code for its MPI module in `mpi.f90`. For some ChaMPIon/Pro distributions, the MPI module will be provided in a pre-built form in the `mpi.mod` file. If the `mpi.mod` file does not exist, please follow instructions given below for building it.

To build `mpi.mod` file:

1. Select the FORTRAN 90 compiler of your choice, e.g., `f90`,
2. Compile `mpi.f90` using a command line of the form: `f90 -c mpi.f90`
3. The above compilation will produce `mpi.mod`, place `mpi.mod` in the include directory of ChaMPIon/Pro, where `mpi.h` is stored.

If during building with `cmpif90c` a link error related to an unresolved symbol 'cmpipro' occurs, this is most likely caused by the way the FORTRAN 90 compiler produces the symbol name for the common block defined in `mpi.f90`. Choose the appropriate options for your FORTRAN 90 compiler in order to generate the common block name that is expected

by the ChaMPIon/Pro FORTRAN bindings library, namely `cmpipro_` for Absoft or `cmpipro_` for other compilers.

The following table lists the supported options for **cmpif90c**:

Options	Use
<code>-h, --help</code>	Show the help screen with all the options.
<code>-echo</code>	Show what the command line that <code>cmpif90c</code> constructs is.
<code>-c</code>	Compile only, do not link.
<code>-ifort</code>	Use Intel Fortran 'ifort' compiler.
<code>-pgf</code>	Use Portland group's <code>pgf77</code> compiler.
<code>-ccl <compiler></code>	User specified Fortran90 compiler.
<code>-io</code>	Build with MPI IO support.
<code>-bproc</code>	Link with BProc support.
<code>-tv</code>	Compile for use with TotalView debugger.

Note: Options other than the ones that are specified here will be passed as compiler or linker options.

Example uses of **cmpif90c**:

- `cmpif90c -c app.f`
- `cmpif90c -o fapp.o fapp.f`
- `cmpif90c -ifort fapp.f -o fapp`

Using the Profiling Interface

In order to use the Profiling Interface, the user should include the `pmpi.h` header file. ChaMPIon/Pro's Profiling interface symbols are provided in the main `libcmpi.a` library and no additional libraries need to be linked in, except the user profiler. Only C profiling interface is provided.

Building MPI Program on MC/OS

ChaMPIon/Pro supports the GNU and Green Hills `ccmc` and `ccmc++` compilers on MC/OS. Fortran compilers are not supported.

ChaMPIon/Pro does not provide build utilities for MC/OS™ hosts. Users must instead use `ccmc` or `ccmc++` directly. Below is a typical command line for building ChaMPIon/Pro programs:

```
ccmc -t ppc -I/path/to/cmpipro/include/files/ -o app app.c -  
      L/path/to/cmpipro/libraries -lcmpi
```

If `MPI_HOME` is set to the location where the ChaMPIon/Pro distribution archive was unpacked, the following examples show how MPI programs can be built with ChaMPIon/Pro:

```
ccmc -t ppc7400 -I $MPI_HOME/include -o app app.c -L  
      $MPI_HOME/lib -lcmpi  
ccmc -t ppc_le -I $MPI_HOME/include -o app.le app.c -L  
      $MPI_HOME/lib -lcmpi
```

Using the Profiling Interface

In order to use the Profiling Interface, the user should include the `pmpi.h` header file. The user profiler must be linked in. Only C profiling interface is provided.

Building MPI Programs on Mac OS X

Using **mpicc**

mpicc is a wrapper around the C compiler and is used to build MPI programs with ChaMPIon/Pro. **mpicc** can be used on the build command line in place of the compiler. Any option that is not recognized by **mpicc** will be passed on to the compiler as a compiler option.

The default compiler used by **mpicc** is **gcc**. The compiler invoked by **mpicc** can be controlled in the following two ways:

1. Using the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The supported compiler is GNU. When `MSTI_COMPILER` is set to GNU, **mpicc** invokes `gcc`.
2. Using **mpicc**'s command line options `-gnu` and `-ccl`. These options instruct **mpicc** what compiler is to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

Note: User programs must include `mpi.h`.

The following **mpicc** options are supported (as displayed by the **mpicc** help screen):

Options	Use
<code>-h, --help</code>	Show the help screen with all options.
<code>-echo</code>	Show what the command line that mpicc constructs is.
<code>-c</code>	Compile only, do not link.
<code>-gnu</code>	Use GNU C compiler (<code>gcc</code>).
<code>-ccl <compiler></code>	User specified compiler.

Options	Use
<code>-io</code>	Build an MPI program that uses MPI I/O calls.
<code>-pmpi</code>	Use the PMPI profiling interface.

Note: Options other than the ones that are listed above will be passed as compiler or linker options.

Example uses of **cmpicc** are as follows:

- `cmpicc -c app.c`
- `cmpicc -c app.c -o app.o`

Using **cmpic++**

The **cmpic++** utility is a wrapper around the C++ compiler and is used for building MPI programs with ChaMPIon/Pro. Its use is identical to the use of **cmpicc**. **cmpic++** can be used on the build command line in place of the compiler. Any option that is not recognized by **cmpic++** will be passed on to the compiler as a compiler option.

The default compiler used by **cmpic++** is **g++**. The compiler invoked by **cmpic++** can be controlled in the following two ways:

1. Using the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The supported compiler is GNU . When `MSTI_COMPILER` is set to `GNU`, **cmpic++** invokes `g++`.
2. Using **cmpic++**'s command line options `-gnu` and `-ccl`. These options instruct **cmpic++** what compiler to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

Note: User programs must include `mpi.hpp`.

The following table lists the supported options:

Options	Use
-h, --help	Show the help screen with all options.
-echo	Show what the command line <code>cmpic++</code> constructs is.
-c	Compile only, do not link.
-gnu	Use GNU C++ compiler (<code>g++</code>).
-ccl <compiler>	User specified compiler.

Note: Options other than the ones that are listed above will be passed as compiler or linker options.

Using `cmpifc`

cmpifc is a wrapper around the FORTRAN 77 compiler and is used for building MPI programs with ChaMPIon/Pro. `cmpifc` can be used on the build command line in place of the compiler. Any option that is not recognized by `cmpic++` will be passed on to the compiler as a compiler option.

The default compiler used by **cmpifc** is `g77`. The compiler invoked by `cmpifc` can be controlled in the following two ways:

1. Using the value of the `MSTI_COMPILER` environment variable, which specifies a compiler family. The supported compiler is GNU. When `MSTI_COMPILER` is set to GNU, `cmpifc` invokes `g77`.
2. The second mechanism for selecting compiler is by using `cmpifc`'s command line options `-gnu` and `-ccl`. These options instruct `cmpifc` what compiler to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

Note: User FORTRAN 77 programs must have the 'include mpif.h' statement in the beginning.

The following table lists the supported options for **cmpifc**:

Options	Use
-h, --help	Show the help screen with all the options.
-echo	Show what the command line that cmpifc constructs is.
-c	Compile only, do not link.
-gnu	Use GNU Fortran77 compiler (g77).
-ccl <compiler>	User specified Fortran 77 compiler.
-io	Build with MPI IO support.

Note: Options other than the ones that are specified here will be passed as compiler or linker options.

Example uses of **cmpifc**:

- `cmpifc -c app.f`
- `cmpifc -o fapp.o fapp.f`

Using `cmpif90c`

`cmpif90c` is a wrapper around the FORTRAN90 compiler and is used to build MPI programs with ChaMPIon/Pro. `cmpif90c` can be used on the build command line in place of the compiler. Any option that is not recognized by `cmpif90c` will be passed on to the compiler as a compiler option.

The compiler invoked by `cmpif90c` can be controlled by using `cmpif90c`'s command line option `-cc1`. This option instructs `cmpif90c` what compiler to be used for building MPI programs. The command line options take precedence to `MSTI_COMPILER`.

According to the MPI-2 standard, FORTRAN90 programs must use an MPI module. ChaMPIon/Pro provides the source code for its MPI module in `mpi.f90`. For some ChaMPIon/Pro distributions, the MPI module will be provided in a pre-built form in the `mpi.mod` file. If the `mpi.mod` file does not exist, please follow instructions given below for building it.

To build `mpi.mod` file:

1. Select the FORTRAN 90 compiler of your choice, e.g., `f90`,
2. Compile `mpi.f90` using a command line of the form: `f90 -c mpi.f90`
3. The above compilation will produce `mpi.mod`, place `mpi.mod` in the include directory of ChaMPIon/Pro, where `mpi.h` is stored.

If during building with `cmpif90c` a link error related to an unresolved symbol 'cmpipro' occurs, this is most likely caused by the way the FORTRAN 90 compiler produces the symbol name for the common block defined in `mpi.f90`. Choose the appropriate options for your FORTRAN 90 compiler in order to generate the common block name that is expected by the ChaMPIon/Pro FORTRAN bindings library, namely `cmpipro_` for Absoft or `cmpipro_` for other compilers.

The following table lists the supported options for **cmpif90c**:

Options	Use
<code>-h, --help</code>	Show the help screen with all the options.
<code>-echo</code>	Show what the command line that <code>cmpif90c</code> constructs is.
<code>-c</code>	Compile only, do not link.
<code>-ccl <compiler></code>	User specified Fortran90 compiler.
<code>-io</code>	Build with MPI IO support.

Note: Options other than the ones that are specified here will be passed as compiler or linker options.

Example uses of **cmpif90c**:

- `cmpif90c -c app.f`
- `cmpif90c -o fapp.o fapp.f`

Using the Profiling Interface

In order to use the Profiling Interface, the `pmpi.h` header file needs be included. ChaMPIon/Pro's profiling interface symbols are provided in the `libcmpi.a` library and you will need to link in `libcmpi.a` or `libcmpi_io.a` to resolve all symbols.

Running MPI Programs

4

This chapter describes the method to run MPI programs. The SDK provides all necessary libraries, include files, and build utilities for developing C and Fortran 77/90 MPI programs. Example MPI programs and makefiles provided with the distribution may also be used as guides.

This chapter includes the following topics:

- Running MPI Applications on Linux
- Running MPI Applications on MC/OS
- Running MPI Applications on Mac OS X

Running Applications on Linux

For starting MPI jobs, ChaMPIon/Pro uses a special-purpose utility called **cmpirun**. This utility depends on a working *rsh* or *ssh* services in order to spawn jobs on remote nodes. The **cmpirun** utility expects to be able to establish remote shell sessions to each remote node and login without a password. This can be accomplished by use of the standard Berkeley rhosts authentication mechanism or setup passwordless ssh.

Using cmpirun

The command-line specification for **cmpirun** varies depending on whether the user chooses a proggroup or machines configuration file.

cmpirun Options

The following is the list of options available for **cmpirun**.

Note: Not all options will be present in all configurations of ChaMPIon/Pro.

`-h, --help`

Displays the full help screen.

`-version`

Displays product version information.

`-np <num>`

Number of processes requested for the MPI job. Requires a machines or a process group specification. Used most often with machines files. If this option is not specified then one process per machine entry in the machines file will be started. For example, if the machines file contains 16 machines, then the size of the MPI job will be 16 processes, equivalent to `-np 16`.

`-mf, -mach_file, -machinefile <file>`

Specifies a machines file. This file is a list of machine names, IP addresses, or a mixture of both identifying the machines to be used for executing the individual processes of the MPI job. If no file is specified through this option, then a machines file in the search order `./machines; /etc/machines` is expected. If such a file does not exist, an error is reported. The MPI ranks are assigned to machines in ascending order, e.g., rank 0 to the first machine in the file, rank 1 to the second, etc. If the argument `<num>` of the `-np` option is smaller than the number of machines in the machines file, the requested MPI processes are started on the first `<num>` machines, one per machine. If `<num>` is greater than the number of machines, the MPI processes are started in a round-robin fashion.

`-pg, -pg_file, -p4pg <file>`

Specifies a process group (procgrou) file. Procgrou files allow for more flexible specification of MPI jobs than machines files. Users can explicitly define the mapping of processes to machines. The procgrou format allows for different executables or different locations of the same executable, including different parameter lists. The size of the MPI job is computed as a sum of all processes started on the machines listed in the procgrou file. The `-np` option can be used to specify a smaller number of processes to be started using the same procgrou file. For example, if a procgrou file defines 16 ranks on 5 machines and `-np 8` is used, only the first 8 ranks will be started.

`-wd <dir>`

Specifies the working directory for the processes of the MPI job. The default working directory is the current working directory.

`-local_root`

Starts rank 0 on the local machine. This may be beneficial for applications with a graphical front-end.

`-root_name <name>`

Specifies a machine (possibly other than the local one) to be used for executing rank 0 of the MPI job. This option may also be used if the machine has multiple names associated with it and ‘hostname’ returns a name that is not suitable for the MPI job.

`-net_file <file>`

This feature is not fully supported in this release and will be fully enabled in future releases.

`-lic_file <file>`

Path to file containing the license key. The default location is in the installation directory.

`-no_smp`

Turns off support for SMP communication for processes on the same node. By default, SMP communication is turned on for processes that are executed on the same node.

`-poll`

Turns on polling mode of the library for Myrinet or InfiniBand communication. SMP support and thread safety will be disabled. By default, when this option is not used, interrupt-based synchronization is used and the highest level of thread-safety enabled.

`-watch <list>`

Terminate the entire job if any rank in <list> exits. The format of <list> is a comma-separated list of ranks or any valid perl list.

Example lists are: 0; “0,4”; “0..5,10..13”.

By default, MPI jobs will not terminate if one or more ranks exit abnormally. This option allows users to specify a set of “watched” ranks, whose exit will cause the entire job to be terminated after a period of time.

`-watch_timeout <seconds>`

Specifies the period (in seconds) of the time-out for termination of the entire MPI job after a “watched” process exits. The default period is 60 seconds.

`-universe_size <num>`

Sets the value of the `MPI_UNIVERSE_SIZE` attribute of `MPI_COMM_WORLD` to `<num>`. This value can also be controlled by the `MSTI_UNIVERSE_SIZE` environment variable. The command line option takes precedence.

`-group_size <num>`

Size of rank groups for initial library setup. Larger group sizes may cause faster library initialization for large-scale jobs. The default value is 64.

`-scale_level <level>`

Scalable startup level: 1, 2 or 3. The startup of MPI process relies on the standard `rsh` or `ssh` remote shell services. The client utilities of these services have certain scalability limitations. For example, the `rsh` client distributed with most common Linux versions is limited to 252 simultaneous sessions. The OpenSSH client has a similar limitation, although the limit is much higher - more than 1,500. In order to accommodate large runs with thousands of processes, far exceeding the limitations of `rsh` and `ssh`, a scalable startup mechanism is provided. This mechanism is turned on with the use of the `-scale_level` option. Supported levels are 1, 2, and 3.

Depending on whether `rsh` or `ssh` is used for remote process startup, the number of processes started by a given level is as follows:

- `rsh`: level 1 up to 500 processes; level 2 up to 1000 processes; level 3 up to 2000 processes
- `ssh`: level 1 up to 2000 processes; level 2 up to 4000 processes; level 3 up to 8000 processes

It is recommended that `ssh` is used for remote process startup of large-scale jobs.

`-timeout <seconds>`

Sets the timeout period in seconds during MPI setup. It is intended to abort jobs when there is some misconfiguration of the network, which does not allow the MPI job to initialize properly. For large scale-jobs, this timeout may need to be increased to allow all processes to perform initial setup before the timeout expires. The default timeout is 60 seconds.

Batch Scheduler Options

`-lsf`

Use machines allocated by LSF. The machines list is extracted from LSF's environment variable `LSB_MCPU_HOSTS`.

`-tslsf`

Use machines allocated by LSF. The machines list is extracted from LSF's environment variable `LSB_MCPU_HOSTS`. This startup is executed according to the LSF Generic MPI Integration, which allows LSF to have better control over parallel jobs and the resources allocated for these jobs.

`-pbs`

Use machines allocated by PBS. The list of machines is obtained from the file specified in PBS's environment variable `PBS_NODEFILE`.

`-bproc`

Configure job for BProc. BProc uses integer numbers for identifying cluster nodes instead of machine names.

`-bjssub`

Configure job for BProc via `bjssub`.

`-caw`

Use machines allocated by Cycles@Work. The list of machines is obtained from the file specified in Cycles@Work environment variable `CAW_NODES_LIST_FILE`. See description of `-mf` option for determining the number of processes and rank assignment to machines.

Debug Options

`-verbose`

Enables verbose mode of the startup script. Information about the mapping of ranks to machines is printed along with additional verbose information.

`-mpi_debug`

Enables extra checking and additional error information in the library.

`-mpi_verbose`

Enables verbose mode of the startup script. Information about the mapping of ranks to machines is printed along with additional verbose information.

`-port <num>`

TCP communication is used for initial library initialization. All ports are generated automatically by the startup procedure. For debugging purposes, this option can be used to assign a specific port to rank 0.

`-enable_corefiles`

Enables dump of core files. Core dump is disabled by default as dumps of a large number of process images can fill up the file system with core files and cause problems. Therefore, core dumps must be requested explicitly.

`-tv`

Starts the parallel job under the TotalView debugger.

`-gdb <list>`

Runs the ranks in `<list>` under the gdb debugger. This requires X and a shared file system for the working directory. The format of `<list>` is a comma-separated list of ranks or any valid perl list. Example lists are: 0; "0,4"; "0..5,10..13".

Tuning Options

`-tcp_cod`

TCP connections on demand. TCP communication is implemented over TCP sockets, which are connection oriented. Sockets connect pairs of processes and have internal kernel buffer space associated with them. This buffer space is proportional to the number of sockets each process establishes. For largescale jobs using TCP, establishing all sockets in an “all-to-all” fashion, is time and resource consuming. Many applications use data exchanges among a limited number of processes, e.g., neighboring processes in virtual 2-D or 3-D process grids as in CFD applications. For such applications, the bulk of the socket connections will not be used, thus resulting in a significant waste of resources. This option instructs the library to establish only TCP socket connections that are requested by the application algorithm, thus the name “connections on demand”. This option has effect only on TCP communication and will be ignored by SMP or high-speed network communication.

`-tcp_long <size>`

TCP message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the TCP device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

`-tcp_buffers <num>`

Maximum number of buffers for unexpected messages used by the TCP device. By default, buffers for unexpected messages are allocated as needed until the virtual space of the operating system is exhausted, which causes the memory allocation operation to be rejected. This option can be used to impose an upper limit on the memory used for unexpected messages. Processes that need a large amount of memory for unexpected messages usually indicate that the parallel algorithm suffers from a severe load imbalance. Such algorithms can be improved to achieve a better parallel efficiency. This option can be used for performance debugging purposes. The total amount of memory allocated by the library is the product of the allocated buffers for unexpected messages and the size of each buffer, controlled by the `-tcp_long` option.

`-tcp_sockbuf <size>`

Size of kernel TCP socket buffers. Each socket has a pair of internal send and receive buffers maintained by the kernel. This option controls the size of these buffers. Changing the value of the TCP socket buffers can lead to a better effective point-to-point throughput of the TCP device.

`-smp_long <size>`

SMP message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the SMP device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed and memory speed. The optimal value of this option can be determined experimentally.

`-smp_buffers <num>`

Maximum number of buffers for unexpected messages used by the SMP device. Unlike network devices, the SMP device requires that the memory space used by the device be defined before initialization of the library. The library will not allocate buffers above the upper limit and if more buffers are needed, the library will abort execution due to insufficient resources.

`-gm_long <size>`

GM message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the GM device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

`-gm_buffers <num>`

Maximum number of buffers for unexpected messages used by the GM device. By default, buffers for unexpected messages are allocated as needed until the virtual space of the operating system is exhausted, which causes the memory allocation operation to be rejected. This option can be used to impose an upper limit on the memory used for unexpected messages. Processes that need a large amount of memory for unexpected messages usually indicate that the parallel algorithm suffers from a severe load imbalance. Such algorithms can be improved to achieve a better parallel efficiency. This option can be used for performance debugging purposes. The total amount of memory allocated by the library is the product of the allocated buffers for unexpected messages and the size of each buffer, controlled by the `-gm_long` option.

`-gm_vlong <num>`

The GM device uses a special protocol for exchanging very long messages, independent of how much memory can be registered with the operating system. This option controls the message size beyond which this protocol will be used.

`-vapi_long <num>`

VAPI message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the VAPI device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

`-vapi_vlong <num>`

The VAPI device uses a special protocol for exchanging very long messages, independent of how much memory can be registered with the operating system. This option controls the message size beyond which this protocol will be used.

`-vapi_send_bufs <num>`

The VAPI device uses a number of pre-registered buffers for sending messages. These buffers are shared among all InfiniBand connections. This option allows users to increase the number of send buffers, which may result in improved communication performance.

`-vapi_recv_bufs <num>`

The VAPI device uses a number of pre-registered buffers for receiving messages. This option allows users to increase the number of receive buffers per connection, which may result in improved communication performance

Environment Variables

The following environment variables affect the behavior of **cmpirun**.

`MPI_COMM`

Selects the medium (device) for communication between ChaMPIon/Pro processes. Allowed values are: `<unset>`, `TCP`, `GM`, and `VAPI`. If the variable is not set, the value that provides the fastest communication by the particular installation of ChaMPIon/Pro is assumed. Only one value can be selected at a time. `MPI_COMM` can be used as a run-time selector of communication devices.

For example, once an MPI program is built with ChaMPIon/Pro that supports TCP and Myrinet/GM, this program can be executed using alternatively TCP and GM devices without recompilation.

`MSTI_PRINT`

Controls the level of verbosity of ChaMPIon/Pro. Allowed values are: `DEBUG`, `VERBOSE`, `API`, and `PTL`. Any combination of these values can be used for setting `MSTI_PRINT`. If more than one value is used, they need to be assigned as a quoted comma-separated list. Example values of `MSTI_PRINT` are: `DEBUG`; `API`; `"API, PTL"`; `"API, DEBUG, VERBOSE"`.

`DEBUG` prints more messages when internal errors are detected by the library. It also turns extra checking of user input parameters to MPI calls. `DEBUG` should be used when error conditions are present. `DEBUG` is equivalent to passing the command line flag `-mpi_debug` to **cmpirun**.

`VERBOSE` prints information related to the initialization of the MPI processes and communication subsystem. It is equivalent to passing the command line flag `-mpi_verbose` to `mpirun`. API causes ChaMPIon/Pro to printout a message for every MPI call. The message contains the name of the call and a complete list of all parameters passed to this call. This is useful to trace the execution of user's code and to verify that the correct arguments are passed to MPI.

`PTL` is the lowest level of verbose information. It prints messages related to the internal operation of the MPI library. It should be used only when a problem is reported to customer support.

`MSTI_UNIVERSE_SIZE`

Determines the value of the attribute `MPI_UNIVERSE_SIZE` to `MPI_COMM_WORLD`. The command line parameter `-universe_size` overrides the value of this variable.

Machines File Format

The machines file is a simple list of machine names. The machines command-line specification for `mpirun` is as follows:

```
mpirun -np # [-mf <machines file>] program [args..]
```

The search order for locating the machines file is as follow:

1. machines file specified with `-mach_file` or `-mf`,
2. A file named machines in the current working directory,
3. A file named machines in the `/etc` directory,
4. Otherwise: an error occurs.

Below is an example of a typical machines file. The symbol `#` is used for commenting out the rest of the line.

```
# Machines file format
#
# Ranks assigned to machines in Round Robin order.
# machine ranks
# ----- ---- -
# name1 0,3,6
# name2 1,4,7
# name3 2,5
#
name1
name2
name3
```

Process Group File Format

A *procgrou* file explicitly specifies the following:

- Names of the machines to run on
- Number of processes to run on each machine
- Name and path to the executable
- Ranks each machine should be assigned

The format of the *procgrou mpirun* command specification is as follows:

```
cmprun [options] -pg_file | -pg <proc group file>
```

There is no default location of the *procgrou* file. A file name of a *procgrou* file must be supplied with the `-pg_file` option to **cmprun**, or an error will be reported. Below is an example of a typical *procgrou* file.

```
# Process group file has the following format:
#
# Ranks assigned to machines in order that they are listed.
#
```

```
# The following will occur:
# machine #procs running ranks
# -----
# name 1 4 /fileserver/apps/test 0,1,2,3
# name 2 3 /fileserver/apps/test2 4,5,6
# name 3 2 /fileserver/apps/test3 7,8
#
name1 4 /fileserver/apps/test
name2 3 /fileserver/apps/test2
name3 2 /fileserver/apps/test3
```

Remote Shell Utility

ChaMPIon/Pro relies on the standard *rsh* or *ssh* remote shell services for remote startup of MPI processes. The default remote startup utility in most distributions of ChaMPIon/Pro is *rsh*. In order to change to *ssh*, please edit the *mpirun* perl script and replace the source line `my $rsh = 'rsh';` with `my $rsh = 'ssh';`. This change needs to be done on all nodes on which ChaMPIon/Pro has been installed.

Redirection of **stdin**, **stderr**, and **stdout**

ChaMPIon/Pro redirects **stderr** and **stdout** of all MPI processes to the console where **cmprun** was executed. The output is serialized per line. The **stdin** stream is redirected only to the MPI process with rank 0.

Exporting User Environment

ChaMPIon/Pro exports the entire user environment by default.

Runtime Parameters

There are a number of run-time tunable parameters for the ChaMPIon/Pro

library to enable users to customize the library for their own specific applications and environment. These parameters are specified as command line options to **cmpirun**. These parameters include:

- Cutoff message size between long and short communication protocols for all devices (TCP, SMP, GM, and VAPI).
- Setting a limit on the maximum number of buffers for unexpected messages.

Note: To see a full list of options use `cmpirun -h` at the command line and the man page for `cmpirun`.

Each of the runtime-controlled parameters has a default value set in the library. If the user does not request a specific value, the library will use the default values. In order to see the values of the runtime parameters, start the MPI job and the pass `-mpi_verbose` flag to **cmpirun**. This will cause ChaMPIon/Pro to print information about its configuration, including the values of the runtime parameters.

Selection of Communication Devices

ChaMPIon/Pro is based on multi-device architecture. TCP and SMP devices are provided in most versions of ChaMPIon/Pro. Special high-speed network support is provided with specialized packages. The basic package of ChaMPIon/Pro contains the TCP and SMP devices. If the basic version has been installed, selection of the devices is not necessary. ChaMPIon/Pro automatically picks the SMP device whenever two processes are started on the same machine. If the processes are on different machines, then ChaMPIon/Pro uses the TCP networking device.

In ChaMPIon/Pro distributions with high-speed network support, such as Myrinet or InfiniBand, a run-time selector is provided for device selection. User applications need not be rebuilt for every device. Once, compiled and linked with ChaMPIon/Pro, the application can use any of the available devices with the installed package. When two or more networking devices

are present, such as when TCP and GM, `mpirun` reads the value of the `MPI_COMM` environment variable in order to perform selection of the network media for communication. The possible values of the environment variable are currently TCP, GM and VAPI. The GM option is used for Myrinet support, and the VAPI option is used for Mellanox VAPI support. If the installed ChaMPIon/Pro package is with high-speed network support, when `MPI_COMM` is not set, the default communication device is the high-speed device.

The default selection of the SMP device for intra-host/node communication can be turned off by the use of the `-no_smp` flag with **`mpirun`**. In this case, the intra-host/node communication will be implemented through the loopback capabilities of the network device (TCP, GM or VAPI).

Note: Before disabling the SMP device, make sure that the selected network device can support loopback communication.

Selection of File System

The basic package of ChaMPIon/Pro contains support for UFS and NFS. ChaMPIon/Pro automatically detects the appropriate file-system back-end to be invoked based on the filename (passed as argument). However, in case of files stored in the Lustre file system, the prefix “lustre:” needs to be passed along with the filename to the MPI I/O application. This prefix is required, as this file system does not export the necessary functionalities required for automatic detection of the file system type. Once this feature is made available in Lustre, it shall be enabled in ChaMPIon/Pro as well.

Scalable Startup

ChaMPIon/Pro relies on the standard `rsh` or `ssh` remote shell services for remote startup of MPI processes. The client utilities of these services have certain scalability limitations. For example, the `rsh` client distributed with most common Linux versions is limited to 252 simultaneous sessions. The

OpenSSH client has a similar limitation, although the limit is much higher – more than 1,500. In order to accommodate large runs with thousands of processes, far exceeding the limitations of rsh and ssh, ChaMPIon/Pro provides a scalable startup mechanism. This mechanism is turned on by the use of the `-scale_level <level>` option to `cmpirun`. Supported levels are 1, 2, and 3. Depending on whether rsh or ssh is used for remote process startup, the number of processes started by a given level is as follows:

- rsh: level 1 up to 500 processes; level 2 up to 1000 processes; level 3 up to 2000 processes
- ssh: level 1 up to 2000 processes; level 2 up to 4000 processes; level 3 up to 8000 processes

Note: For large-scale jobs, it is recommended that ssh is used for remote process startup.

Connections on Demand (TCP Device only)

The TCP protocol is connection oriented. In order to exchange data between pairs of processes, the ChaMPIon/Pro library needs to establish TCP socket connections between these processes. Each socket has a send and a receive buffer maintained in the kernel space of the operating system. Establishing a large number of sockets is a time and resource consuming operation, which limits scalability. ChaMPIon/Pro supports a mechanism, called “TCP connections on demand”, which when engaged will delay the creation of socket connections until the moment they are needed. If two processes never communicate between themselves during the course of the MPI job, a socket connection will not be established. In many common parallel applications, the processes do not participate in all-to-all style of communication. For such applications, the creation of a large number of unused connections can be avoided by passing `-tcp_cod` flag to `cmpirun`. This will instruct ChaMPIon/Pro to turn on its “TCP connections on demand” mechanism. This mechanism is implemented only for the TCP device and is turned off by default.

Cancellation of MPI Jobs

The **cmpirun** utility allows the user to cancel “hanging” MPI applications or to terminate long running MPI programs. Even though processes are started on remote machines, they will be terminated if the user presses the <CTRL>-C key combination.

“Watching” MPI Ranks

In its default mode of operation, ChaMPIon/Pro allows the MPI jobs to continue when one or more ranks exit abnormally. This feature is useful for long-running jobs that can sustain certain level of process loss, especially applications that are created according to the master-slave model. However, in certain cases it is useful to terminate the entire MPI job if specific ranks exit abnormally, such as in a master-slave application when the master process dies. Evidently, this MPI job will not produce the expected results but may occupy the resources allocated to it for a long time, thus wasting precious computation resources. ChaMPIon/Pro provides a “rank watch” mechanism that allows the user to enumerate some or all ranks that will cause the entire job to exit if a “watched” process exits. This mode is turned on with the `-watch <rank_list>` option to **cmpirun**. The argument `<rank_list>` specifies the ranks to be “watched.” This list can be any valid perl list. For instance,

```
cmpirun -np 256 -mf mfile -watch "0,4-6" long_running_app
```

This command will start a 256-way job on the machines listed in `mfile` and will terminate the entire job if ranks 0, 4, 5, or 6 exit before the rest. Before terminating the entire job, ChaMPIon/Pro uses a time-out to allow slower processes of normally completing jobs to exit without being terminated. The default value of this time-out is 60 seconds. It can be controlled by the user with the `-watch_timeout <seconds>` option to **cmpirun**.

Debugging in the GNU Debugger (GDB)

ChaMPIon/Pro applications can be debugged interactively using the GNU Debugger (GDB). If **cmpirun** is used with a machines file the command line to start a debugging session with **gdb** is as follows:

```
cmpirun -np 2 -mf machines_file -gdb "1,3,8..9" application
      app_args
```

The argument given to the `-gdb` flag is a valid perl list specifies the ranks for which **gdb** should be run. An instance of **gdb** will be run for each rank that is specified. A working X-Windows configuration and correct `DISPLAY` and `xhost` settings are required for debugging with **gdb**.

Debugging in TotalView™

ChaMPIon/Pro applications can be debugged interactively using TotalView. If **cmpirun** is used with a machines file the command line to start a debugging session with TotalView is as follows:

```
cmpirun -np 2 -mf machines -tv application app_args
```

The `-tv` flag requires no arguments. A single instance of TotalView will be run for the entire parallel job. A working X-Windows configuration and TotalView installation (possibly including a working flexlm setup) is required for such debugging.

Performance Tuning

Champion/Pro offers a number of performance tuning parameters that may help with adjusting the performance and scalability behavior of the library to the specific user's environment. One of the important run-time options is the synchronization and message progress mode of the library.

Champion/Pro supports two modes:

- blocking (default)
- polling (turned on with the `-poll` option of `mpirun`).

The blocking mode uses interrupts for notification of message completion and it also provides independent message progress. This mode complies with the strict interpretation of the MPI Progress Rule. In this mode Champion/Pro uses very little of the host CPU time for communication and provides for an efficient overlapping of communication, computation, and file I/O. The blocking mode adds certain constant processing overhead related to interrupt handling and context switches, which results in increased latency for very short messages as measured by ping-pong point-to-point tests. The blocking mode is thread safe.

The polling mode is not thread safe and the message progress is implemented in the MPI calls of the user program, i.e., the message progress depends on the calling sequence of the user code. The polling mode achieves lower latency than the blocking mode at the expense of using more of the host CPU cycles for communication purposes. The polling mode has lower latency as measure by pure communication benchmarks. Typically, applications that use a large number of short messages, which cannot exploit asynchrony and overlapping will benefit of using the polling mode.

Another option related to the communication performance of Champion/Pro is the switchover size between the short and long message protocols of the library. Champion/Pro has a default size for each device that can be

inspected by the use of the `-mpi_debug` flag to `mpirun`. This default size may not match best the particular user environment. Using `mpirun`'s `-<dev>_long` options for the specific communication device (GM, VAPI, TCP, SMP, or ELAN), the user can change the default protocol switchover size. Important factors that can affect the value of the protocol switchover size are CPU speed, memory speed and size, cache size, and network speed.

Running Applications on MC/OS

In this section, references to “hosts” or “host types” have a specific meaning in the Mercury computing environment. A host is typically a computer that runs Solaris, Windows, or vxWorks and serves as a front-end to the user and sets up the Compute Environments (CEs).

Depending on the host type, ChaMPIon/Pro jobs on MC/OS can be started in one of the following ways:

- With the *mpirun* script
- With the *mpiload* script
- Through the direct use of the standalone binary loader `mpimc_load` or the standard MC/OS utility `runmc`.

The *mpirun* and *mpiload* scripts are only supported on Solaris hosts; they are known to work under most circumstances in Windows with Cygwin installed but this method is not supported. For Windows or vxWorks hosts, the only supported method is to use `mpimc_load` or `runmc` directly.

Using *mpirun*

The command-line specification for *mpirun* varies depending on whether the user chooses a process group (`progroup`) or machines configuration file.

See *mpirun*'s online help for information on all command line options and their use. The help screen can be displayed by executing “*mpirun -help*.” Of specific interest are the `-lic_key` and `-lic_file` options that allow the user to specify ChaMPIon/Pro's license key that is needed for the proper operation of the ChaMPIon/Pro software.

Machines File Format

The machines file is a list of CE ID's. The machines command-line specification for mpirun is as follows:

```
mpirun -np # [-mach_file <machines file>] [opts] program
        [args..]
```

If no machines file is specified, a default file called machines in the current working directory is used.

Examples:

```
mpirun -np 32 appname
mpirun -np 32 -mach_file myceids -lic_file mylicense appname
        100 5 10
mpirun -h 0x40000 -dc -np 8 -mach_file cefile -race_long
        2048 appname 100 5 10
```

Below is an example of a typical machines file. The symbol '#' is used for commenting out the rest of the line.

```
# Machines file format
#
# One process per CE, no round-robin
CEID 10
CEID 11
CEID 12
CEIDs 13-15
```

Process Group File Format

A procgroup file explicitly specifies the following:

- ID's of the CE's to run on
- Number of processes to run on each machine (only 1 process in this version),
- Name and path to the executable, and
- Arguments to each executable

The format of the procgroup mpirun command specification is as follows:

```
mpirun [options] -pg_file <proc group file>
```

A file name of a process group file must be supplied with the `-pg_file` option to *mpirun*, or an error will be reported.

Examples:

```
mpirun -pg_file pgfile
mpirun -h 0x40000 -dc -pg_file pgfile
```

Below is an example of a typical procgroup file.

```
# Procgroup file
# One process per CE, for a total of 5 processes
CEID      3      /path/to/exename1.ppc arg1
CEID      5      /path/to/exename2.ppc arg1 arg2
CEIDs     8-10  /path/to/exename3.ppc arg3
```

Using mpiload

mpiload performs similar functionality to *mpirun*. It differs from *mpirun* in that it uses a specialized binary utility, `mpimc_load`, for loading the executable image to the selected CE's instead of the standard Mercury `runmc` utility. `mpimc_load` utilizes Mercury's high-speed interconnect for distributing the image. This results in a faster and more scalable startup of MPI jobs. *mpiload* provides three different methods for specifying the target CE's:

- Direct specification on the command line
- Using a *progroup* file
- Using a *machines* file.

The use of the *progroup* or *machines* files is similar to *mpirun* as described above. The generic format of the *mpiload* command line is as follows:

```
mpiload -ce <server_ce_id> [opts] <CE_specification> exec  
[args]
```

The `server_ce_id` is the CE that loads the image of `mpimc_load` through `runmc` and then “ships” this user executable image `exec` to the CE's specified by the `CE_specification`.

IMPORTANT: If `-lic_file` or `-lic_key` options for specifying the license key are passed to *mpiload*, they should follow the `-ce <server_ce_id>` option and should precede the `CE_specification`, as in the examples below:

```
mpiload -ce 10 -lic_file licfile -np 2 -ceids 1 11-12 app  
mpiload -ce 10 -lic_key $LIC_KEY -h 0x200000 -np 2  
-mach_file ces app
```

Specifying Target CEs on the Command Line

The command line format for *mpiload* to specify target CEs directly is as follows:

```
mpiload -ce <id> [opts] -np <num> -ceids <num_blk> <blocks>
      exec [args]
```

Using this format, the *-ce* option specifies which CE to load the *mpimc_load* or *mpimc_loadle* (for little Endian systems) binary, which in turn loads the user application on a number of blocks of CEs specified by *<num_blk>*. An example where the user uses CE 10 for *mpimc_load* and runs the ring program on CEs 11-12 and 14-15 (two blocks of CEs) would be:

```
mpiload -ce 10 -np 4 -ceids 2 11-12 14-15 ./ring
```

The same example, for running the same executable on 32 CE's in one contiguous block of CE ID's 20 through 51 would be:

```
mpiload -ce 10 -np 32 -ceids 1 20-51 ./ring
```

Machines file format

This is similar to the format used by *mpirun*.

```
mpiload -ce <id> [opts] -np <num> -mach_file <file> exec
      [args]
```

Example:

```
mpiload -ce 10 -d -np 4 -mach_file my_mach app
```

Process Group file format

This is similar to the format used by *mpirun*.

```
mpiload -ce <id> [opts] -pg_file <file>
```

Example:

```
mpiload -ce 10 -h 0x400000 -pg_file my_pg
```

Using `runmc` or `mpimc_load` directly

Using `runmc` or `mpimc_load` directly is the only officially supported startup methods on Windows or vxWorks hosts. Of these two methods, using `mpimc_load` is suggested, for anything but small jobs. The *mpirun* and *mpiload* scripts used on Solaris, convert options specific to these utilities to options that are understood by the ChaMPIon/Pro library. In the cases when `runmc` or `mpimc_load` are used, such conversion is not available and the options that are passed on the command line must be the options understood by the library. For the `runmc` and `mpimc_load`, “ChaMPIon/Pro options” is used to denote these options that are passed directly to the library. The library strips these options from the argument list of the application, so that user specific arguments passed to the application will be processed without change.

Using `runmc`

One instance of `runmc` must be run for every rank in a ChaMPIon/Pro job. On Solaris, the *mpirun* script automates this process but on Windows or vxWorks the user must manually perform the necessary steps. Each call of *runmc* is blocking -- returning only when the job finishes -- so each rank may need to be started in a separate command window or run in the background. The basic command line for `runmc` is:

```
runmc -ce <id> [opts] <user executable> [ChaMPIon/Pro opts]  
[application opts]
```

It is a good idea to specify a heap and stack size in [opts] with the `-h` and `-s` flags respectively, otherwise many user applications that need more than the default amounts of memory may fail. For more information, consult the documentation for `runmc` from Mercury Computer Systems.

The ChaMPIon/Pro options must include, at the very least:

- `msti_size` number of total processes in the job
- `msti_rank` rank of the current process being started
- `msti_lic_key` license key (See Using Licensing for more help.)

Other ChaMPIon/Pro options are `-msti_verbose` and `-msti_debug`, which turn on verbose and debug modes of the library. These options correspond to the `-mpi_verbose` and `-mpi_debug` options passed to *mpirun* and *mpiload*.

Examples, assuming the `LIC_KEY` environment variable is set to the license key without dashes, e.g.,
`LIC_KEY=aaaaabbbbbccccddddddeeeefffff:`

```
runmc -ce 2 -h 0x200000 ring -msti_size 1 -msti_rank 0
      -msti_lic_key $LIC_KEY -msti_verbose
```

This will run an MPI job with on process (`np=1`) in verbose mode.

Open two consoles. In the first console type:

```
runmc -ce 2 -h 0x200000 ring -msti_size 2 -msti_rank 0
      -msti_lic_key $LIC_KEY
```

In the second console type:

```
runmc -ce 3 -h 0x200000 ring -msti_size 2 -msti_rank 1
      -msti_lic_key $LIC_KEY
```

This will run an MPI job with 2 processes ($np=2$) on CE 2 and CE 3.

Using `mpimc_load`

The user needs to run only one instance of `mpimc_load` (or `mpimc_loadle` which is the little Endian version) for an MPI job. On Solaris, the `mpiload` script automates this process but on Windows or vxWorks the user must manually perform this step. Using `mpimc_load` is the preferred way for starting MPI jobs on Windows and vxWorks hosts. The `mpimc_load` utility will only return when the entire job is finished. This utility runs on one of the CEs, effectively reducing the number of CEs available for the job. Use `mpimc_loadle` for systems with Windows hosts. To load and execute `mpimc_loadle` on a CE with a Windows host, the following command line can be used:

```
runmc -ce <id> C:\path\to\mpimc_loadle [mpimc_load opts]
      exec [ChamPION/Pro opts] [args]
```

The same command on a vxWorks hosts will look like:

```
runmc "-ce <id> /path/to/mpimc_load [mpimc_load opts] exec
      [ChamPION/Pro opts] [args]"
```

This command loads the `mpimc_load` or `mpimc_loadle` binary on the CE specified by `<id>`. The `[mpimc_load options]` given to `mpimc_load` then specify what CEs will execute the user application. These options are as follows:

```
-h <hsize> -s <ssize> -np <np> -ceids <num_ranges> <ranges>
```

The heap and stack sizes are specified by `-h` and `-s` for the user jobs. The number of processes in the job, not including the process that runs `mpimc_load` itself, must be specified with `-np`. The target CEs to run the user application on are specified by the `-ceids` option, followed by a number that specifies the number of blocks of consecutive CEs to use and then each range, specified as a pair in the form `<start ce>--<end ce>`. An

example for CE specification that would launch a user application on CEs 2, 3, 4, 5, 10, 11, 12, 15, and 16 on a vxWorks host would be:

```
runmc "-ce 2 mpimc_load -h 0x200000 -np 6 -ceids 2 3-5 10-12  
      myapp -msti_lic_key <key>"
```

The same example on a Windows hosts will be as follows:

```
runmc -ce 2 mpimc_loadle -h 0x200000 -np 6 -ceids 2 3-5 10-12  
      myapp -msti_lic_key <key>
```

The ChaMPion/Pro options, at the very least, must include `-msti_lic_key <license_key>`, where the `license_key` should be a single string not containing “-” symbols. This can be achieved by storing the original license key in an a file and manually removing the dashes and then passing the content of the file as an argument to `-msti_lic_key` with `'head -1 <file>'` or by automatically removing the dashes from the original license key stored in a file by passing `'head -1 <file> | sed 's/-//g'`.

Example:

```
echo 00000-11111-22222-33333-44444-55555 > licfile  
runmc -ce 10 mpimc_load -h 0x200000 -np 3 -ceids 1 11-13 ring  
      -msti_lic_key `head -1 licfile | sed 's/-//g'`
```

The same effect can be achieved if the license key with stripped dashes is assigned to an environment variable.

Example:

```
echo 00000-11111-22222-33333-44444-55555 > licfile  
export LIC_KEY=`head -1 licfile | sed 's/-//g'`  
runmc -ce 10 mpimc_load -h 0x200000 -np 3 -ceids 1 11-13 ring  
      -msti_lic_key $LIC_KEY
```

Notice that the `-msti_lic_key` option differs from the `-lic_key` option in that `-msti_lic_key` is placed after the executable whereas `-lic_key` is placed before the executable. In this sense, `-msti_lic_key` is passed as an

argument to the user application, which will be processed by the MPI library, as opposed to `-lic_key`, which is passed as an argument to `mpirun` or `mpiload`.

As opposed to the case when `runmc` is used directly to start the application, the `-msti_size` and `-msti_rank` ChaMPIon/Pro options will be supplied by the `mpimc_load` utility and the user does not need to provide them on the command line.

Working with Licensing

ChaMPIon/Pro requires license keys. This section assumes you have received a valid license key when the software was purchased. If not, you will be unable to use ChaMPIon/Pro. To obtain a license key, please contact sales@mpi-softtech.com.

Depending on the host type, there are several methods of specifying a license key, in addition to custom startup solutions.

1. [Solaris] A default location for the license key in `/etc/cmpipro.conf` (used by `mpirun` and `mpiload`)
2. [Solaris] Specify the location of a license key file with `-lic_file` (used by `mpirun` and `mpiload`).
3. [Solaris] Specify the key in a default environment variable `MSTI_LIC_KEY` (used by `mpirun` and `mpiload`)
4. [Any host type] Specify the key as a command line argument passed after the user executable on the command line using the `-msti_lic_key` option; the license key string should have its “-” symbols stripped when passed to `-msti_lic_key`.

Currently, options one through three are only available and officially supported under Solaris. However, these methods are expected to work under Windows with the Cygwin package as well but are unsupported by MPI Software Technology, Inc. If Process Group configuration files are used for specifying the target CE's and the executables, option four must be

used, regardless of the host type. The `-msti_lic_key` option must be provided in each line of the progroup file after the executable.

Note: At this moment Process Group files are not supported under vxWorks and regular Windows command line prompt.

1. Setting up and using a default key

A default key may be set up on systems with Solaris hosts using the provided `cmpipro-conf` script. Run this script as root and enter your key when prompted. It is recommended to leave the default location. If the default location is used, the user may opt not to specify the location of the key file manually. The key should be found by the `mpirun` or `mpiload` scripts. Under Windows with Cygwin, the key should be created under `<CYGWIN ROOT>\etc`, where `<CYGWIN ROOT>` is where Cygwin is installed (`C:\cygwin` for example).

2. Specifying a key location

To specify the location of the license key file, use the `-lic_file <file>` parameter to either `mpirun` or `mpiload`. If the file is in the current working directory, make sure to specify `./<filename>` rather than just `<filename>`. The `-lic_file` option overrides the default license key, which is installed by `cmpipro-conf` when the default location is accepted.

3. Specifying the key in an environment variable

To set a license key for a login session, the user may export the `MSTI_LIC_KEY` variable to the value of the license key. If this variable is set, `mpirun` and `mpiload` will utilize it. In a Bourne shell, an example might be:

```
MSTI_LIC_KEY=12345-12345-12345-12345-12345-12345
```

```
export MSTI_LIC_KEY
```

4. Specifying the key as a command line argument

To specify a key directly on the command line to *mpirun* or *mpiload*, use the `-lic_key <key>` parameter. This overrides any other method of specifying a license key on any host type in *mpirun* or *mpiload*.

When *runmc* and *mpimc_load* are used, the only way of specifying the license key is through the `-msti_lic_key` option passed after the executable. The same option must also be used if *procgroup* files are used; to do so, put `-msti_lic_key` argument on each line of the *procgroup* file as an argument to the user program. The “-” symbols must be stripped from the license key strings when passed as arguments to `-msti_lic_key`. An example is given below:

```
# Procgroup file
#
# One process per CE, no round-robin
CEID 2 /path/to/exe1.ppc -msti_lic_key 000001111122222333334444455555
CEID 5 /path/to/exe2.ppc -msti_lic_key 000001111122222333334444455555
CEIDs 8-10 /path/to/exe3.ppc -msti_lic_key 000001111122222333334444455555
```

Custom Startup

Customers, at their option, may choose to create their own startup mechanism. In order to facilitate this, the following options must be passed as command line parameters to applications compiled with ChaMPIon/Pro:

- `msti_size <N>` The total number of processes in the parallel job.
- `msti_rank <M>` The rank of the given process [0 ... N-1] where N is the parameter passed to `-msti_size`.
- `msti_lic_key <key>` The license key to use.

So, after compiling a user application, a custom startup solution must load the compiled image onto the desired CEs with at least the three above command line parameters included in argv. Each process should receive the same `-msti_size`, the same `-msti_license_key`, and a unique `-msti_rank` with values in the range `[0, job_size-1]`. Failure to do so will result in hanging on startup or other undesired behavior.

Runtime Parameters

There are a number of run-time tunable parameters for the ChaMPIon/Pro library to enable users to customize the library for their own specific applications. These parameters are specified as command line options to `mpirun` and `mpiload`. These parameters include:

- The minimum size using DMA (default 32 bytes)
- The race long protocol size (bytes), default 2048
- The number of library created buffers per peer rank
- Verbose mode
- Debug mode

In addition to the command line options specific to ChaMPIon/Pro, the startup utilities accept options that are specific to `runmc`, such as heap size (`-h`), stack size (`-s`), debug (`-d`), dump core (`-dc`), and others.

Note: To see a full list of arguments that are supported by `mpirun` and `mpiload` use the `-help` option of these utilities as a single argument at the command line.

Redirection of `stdin`, `stderr`, and `stdout`

ChaMPIon/Pro redirects `stderr` and `stdout` of all MPI processes to the OSM I/O server. The output is serialized per line. The `stdin` stream is redirected only to the MPI process with rank 0.

Running Applications on Mac OS X

For starting MPI jobs, ChaMPIon/Pro uses a special-purpose utility called `cmpirun`. This utility depends on a working ssh service in order to spawn jobs on remote nodes. The `cmpirun` utility expects to establish remote shell sessions to each remote node and login without a password. This can be accomplished by configuring ssh for passphraseless keys.

Using `cmpirun`

The command-line specification for **`cmpirun`** varies depending on whether the user chooses a proggroup or machines configuration file.

`cmpirun` Options

The following is the list of options available for **`cmpirun`**.

Note: Not all options will be present in all configurations of ChaMPIon/Pro.

`-h, --help`

Displays the full help screen.

`-version`

Displays product version information.

`-np <num>`

Number of processes requested for the MPI job. Requires a machines or a process group specification. Used most often with machines files. If this option is not specified then one process per machine entry in the machines file will be started. For example, if the machines file contains 16 machines, then the size of the MPI job will be 16 processes, equivalent to `-np 16`.

`-mf, -mach_file, -machinefile <file>`

Specifies a machines file. This file is a list of machine names, IP addresses, or a mixture of both identifying the machines to be used for executing the individual processes of the MPI job. If no file is specified through this option, then a machines file in the search order `./machines; /etc/machines` is expected. If such a file does not exist, an error is reported. The MPI ranks are assigned to machines in ascending order, e.g., rank 0 to the first machine in the file, rank 1 to the second, etc. If the argument `<num>` of the `-np` option is smaller than the number of machines in the machines file, the requested MPI processes are started on the first `<num>` machines, one per machine. If `<num>` is greater than the number of machines, the MPI processes are started in a round-robin fashion.

`-pg, -pg_file, -p4pg <file>`

Specifies a process group (procgrou) file. Procgrou files allow for more flexible specification of MPI jobs than machines files. Users can explicitly define the mapping of processes to machines. The procgrou format allows for different executables or different locations of the same executable, including different parameter lists. The size of the MPI job is computed as a sum of all processes started on the machines listed in the procgrou file. The `-np` option can be used to specify a smaller number of processes to be started using the same procgrou file. For example, if a procgrou file defines 16 ranks on 5 machines and `-np 8` is used, only the first 8 ranks will be started.

`-wd <dir>`

Specifies the working directory for the processes of the MPI job. The default working directory is the current working directory.

`-local_root`

Starts rank 0 on the local machine. This may be beneficial for applications with a graphical front-end.

`-root_name <name>`

Specifies a machine (possibly other than the local one) to be used for executing rank 0 of the MPI job. This option may also be used if the machine has multiple names associated with it and ‘hostname’ returns a name that is not suitable for the MPI job.

`-net_file <file>`

This feature is not fully supported in this release and will be fully enabled in future releases.

`-lic_file <file>`

Path to file containing the license key. The default location is in the installation directory.

`-no_smp`

Turns off support for SMP communication for processes on the same node. By default, SMP communication is turned on for processes that are executed on the same node.

`-poll`

Turns on polling mode of the library for Myrinet or InfiniBand communication. SMP support and thread safety will be disabled. By default, when this option is not used, interrupt-based synchronization is used and the highest level of thread-safety enabled.

`-watch <list>`

Terminate the entire job if any rank in <list> exits. The format of <list> is a comma-separated list of ranks or any valid perl list.

Example lists are: 0; “0,4”; “0..5,10..13”.

By default, MPI jobs will not terminate if one or more ranks exit abnormally. This option allows users to specify a set of “watched” ranks, whose exit will cause the entire job to be terminated after a period of time.

`-watch_timeout <seconds>`

Specifies the period (in seconds) of the time-out for termination of the entire MPI job after a “watched” process exits. The default period is 60 seconds.

`-universe_size <num>`

Sets the value of the `MPI_UNIVERSE_SIZE` attribute of `MPI_COMM_WORLD` to `<num>`. This value can also be controlled by the `MSTI_UNIVERSE_SIZE` environment variable. The command line option takes precedence.

`-group_size <num>`

Size of rank groups for initial library setup. Larger group sizes may cause faster library initialization for large-scale jobs. The default value is 64.

`-scale_level <level>`

Scalable startup level: 1, 2 or 3. The startup of MPI process relies on the standard `rsh` or `ssh` remote shell services. The client utilities of these services have certain scalability limitations. For example, the `rsh` client distributed with most common Linux versions is limited to 252 simultaneous sessions. The OpenSSH client has a similar limitation, although the limit is much higher - more than 1,500. In order to accommodate large runs with thousands of processes, far exceeding the limitations of `rsh` and `ssh`, a scalable startup mechanism is provided. This mechanism is turned on with the use of the `-scale_level` option. Supported levels are 1, 2, and 3.

Depending on whether `rsh` or `ssh` is used for remote process startup, the number of processes started by a given level is as follows:

- `rsh`: level 1 up to 500 processes; level 2 up to 1000 processes; level 3 up to 2000 processes
- `ssh`: level 1 up to 2000 processes; level 2 up to 4000 processes; level 3 up to 8000 processes

It is recommended that ssh is used for remote process startup of large-scale jobs.

Note: The only supported remote startup option on Mac OS X is ssh.

`-timeout <seconds>`

Sets the timeout period in seconds during MPI setup. It is intended to abort jobs when there is some misconfiguration of the network, which does not allow the MPI job to initialize properly. For large scale-jobs, this timeout may need to be increased to allow all processes to perform initial setup before the timeout expires. The default timeout is 60 seconds.

Batch Scheduler Options

`-lsf`

Use machines allocated by LSF. The machines list is extracted from LSF's environment variable `LSB_MCPU_HOSTS`.

`-tslsf`

Use machines allocated by LSF. The machines list is extracted from LSF's environment variable `LSB_MCPU_HOSTS`. This startup is executed according to the LSF Generic MPI Integration, which allows LSF to have better control over parallel jobs and the resources allocated for these jobs.

`-pbs`

Use machines allocated by PBS. The list of machines is obtained from the file specified in PBS's environment variable `PBS_NODEFILE`.

Debug Options

`-verbose`

Enables verbose mode of the startup script. Information about the mapping of ranks to machines is printed along with additional verbose information.

`-mpi_debug`

Enables extra checking and additional error information in the library.

`-mpi_verbose`

Enables verbose mode of the startup script. Information about the mapping of ranks to machines is printed along with additional verbose information.

`-port <num>`

TCP communication is used for initial library initialization. All ports are generated automatically by the startup procedure. For debugging purposes, this option can be used to assign a specific port to rank 0.

`-enable_corefiles`

Enables dump of core files. Core dump is disabled by default as dumps of a large number of process images can fill up the file system with core files and cause problems. Therefore, core dumps must be requested explicitly.

`-gdb <list>`

Runs the ranks in `<list>` under the gdb debugger. This requires X and a shared file system for the working directory. The format of `<list>` is a comma-separated list of ranks or any valid perl list. Example lists are: 0; “0,4”; “0..5,10..13”.

Tuning Options

`-tcp_cod`

TCP connections on demand. TCP communication is implemented over TCP sockets, which are connection oriented. Sockets connect pairs of processes and have internal kernel buffer space associated with them. This buffer space is proportional to the number of sockets each process establishes. For largescale jobs using TCP, establishing all sockets in an “all-to-all” fashion, is time and resource consuming. Many applications use

data exchanges among a limited number of processes, e.g., neighboring processes in virtual 2-D or 3-D process grids as in CFD applications. For such applications, the bulk of the socket connections will not be used, thus resulting in a significant waste of resources. This option instructs the library to establish only TCP socket connections that are requested by the application algorithm, thus the name “connections on demand”. This option has effect only on TCP communication and will be ignored by SMP or high-speed network communication.

```
-tcp_long <size>
```

TCP message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the TCP device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

```
-tcp_buffers <num>
```

Maximum number of buffers for unexpected messages used by the TCP device. By default, buffers for unexpected messages are allocated as needed until the virtual space of the operating system is exhausted, which causes the memory allocation operation to be rejected. This option can be used to impose an upper limit on the memory used for unexpected messages. Processes that need a large amount of memory for unexpected messages usually indicate that the parallel algorithm suffers from a severe load imbalance. Such algorithms can be improved to achieve a better parallel efficiency. This option can be used for performance debugging purposes. The total amount of memory allocated by the library is the

product of the allocated buffers for unexpected messages and the size of each buffer, controlled by the `-tcp_long` option.

`-tcp_sockbuf <size>`

Size of kernel TCP socket buffers. Each socket has a pair of internal send and receive buffers maintained by the kernel. This option controls the size of these buffers. Changing the value of the TCP socket buffers can lead to a better effective point-to-point throughput of the TCP device.

`-smp_long <size>`

SMP message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the SMP device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed and memory speed. The optimal value of this option can be determined experimentally.

`-smp_buffers <num>`

Maximum number of buffers for unexpected messages used by the SMP device. Unlike network devices, the SMP device requires that the memory space used by the device be defined before initialization of the library. The library will not allocate buffers above the upper limit and if more buffers are needed, the library will abort execution due to insufficient resources.

`-gm_long <size>`

GM message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These

protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the GM device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

```
-gm_buffers <num>
```

Maximum number of buffers for unexpected messages used by the GM device. By default, buffers for unexpected messages are allocated as needed until the virtual space of the operating system is exhausted, which causes the memory allocation operation to be rejected. This option can be used to impose an upper limit on the memory used for unexpected messages. Processes that need a large amount of memory for unexpected messages usually indicate that the parallel algorithm suffers from a severe load imbalance. Such algorithms can be improved to achieve a better parallel efficiency. This option can be used for performance debugging purposes. The total amount of memory allocated by the library is the product of the allocated buffers for unexpected messages and the size of each buffer, controlled by the `-gm_long` option.

```
-gm_vlong <num>
```

The GM device uses a special protocol for exchanging very long messages, independent of how much memory can be registered with the operating system. This option controls the message size beyond which this protocol will be used.

`-vapi_long <num>`

VAPI message protocol switchover size. The MPI library uses two different protocols for data exchange depending on the size of user messages. These protocols are often referred to as “short” and “long” protocols. The short protocol is optimized for low latency while the long protocol for high bandwidth. This option controls the switchover message size used for switching between the short and long protocols of the VAPI device. Users can change the switchover size in order to adjust the performance behavior according to their specific platform. The value of this option also controls the size of the internal buffers used by the library for reception of unexpected messages. The factors that usually affect the optimal protocol switchover size are CPU speed, memory speed, and network speed. The optimal value of this option can be determined experimentally.

`-vapi_vlong <num>`

The VAPI device uses a special protocol for exchanging very long messages, independent of how much memory can be registered with the operating system. This option controls the message size beyond which this protocol will be used.

`-vapi_send_bufs <num>`

The VAPI device uses a number of pre-registered buffers for sending messages. These buffers are shared among all InfiniBand connections. This option allows users to increase the number of send buffers, which may result in improved communication performance.

`-vapi_recv_bufs <num>`

The VAPI device uses a number of pre-registered buffers for receiving messages. This option allows users to increase the number of receive buffers per connection, which may result in improved communication performance

Environment Variables

The following environment variables affect the behavior of **cmpirun**.

MPI_COMM

Selects the medium (device) for communication between ChaMPIon/Pro processes. Allowed values are: `<unset>`, `TCP`, `GM`, and `VAPI`. If the variable is not set, the value that provides the fastest communication by the particular installation of ChaMPIon/Pro is assumed. Only one value can be selected at a time. `MPI_COMM` can be used as a run-time selector of communication devices.

For example, once an MPI program is built with ChaMPIon/Pro that supports TCP and Myrinet/GM, this program can be executed using alternatively TCP and GM devices without recompilation.

MSTI_PRINT

Controls the level of verbosity of ChaMPIon/Pro. Allowed values are: `DEBUG`, `VERBOSE`, `API`, and `PTL`. Any combination of these values can be used for setting `MSTI_PRINT`. If more than one value is used, they need to be assigned as a quoted comma-separated list. Example values of `MSTI_PRINT` are: `DEBUG`; `API`; `"API, PTL"`; `"API, DEBUG, VERBOSE"`.

`DEBUG` prints more messages when internal errors are detected by the library. It also turns extra checking of user input parameters to MPI calls. `DEBUG` should be used when error conditions are present. `DEBUG` is equivalent to passing the command line flag `-mpi_debug` to **cmpirun**.

`VERBOSE` prints information related to the initialization of the MPI processes and communication subsystem. It is equivalent to passing the command line flag `-mpi_verbose` to **cmpirun**. `API` causes ChaMPIon/Pro to printout a message for every MPI call. The message contains the name of the call and a complete list of all parameters passed to this call. This is useful to trace the execution of user's code and to verify that the correct arguments are passed to MPI.

PTL is the lowest level of verbose information. It prints messages related to the internal operation of the MPI library. It should be used only when a problem is reported to customer support.

MSTI_UNIVERSE_SIZE

Determines the value of the attribute `MPI_UNIVERSE_SIZE` to `MPI_COMM_WORLD`. The command line parameter `-universe_size` overrides the value of this variable.

Machines File Format

The machines file is a simple list of machine names. The machines command-line specification for *mpirun* is as follows:

```
mpirun -np # [-mf <machines file>] program [args..]
```

The search order for locating the machines file is as follow:

5. machines file specified with `-mach_file` or `-mf`,
6. A file named `machines` in the current working directory,
7. A file named `machines` in the `/etc` directory,
8. Otherwise: an error occurs.

Below is an example of a typical machines file. The symbol `#` is used for commenting out the rest of the line.

```
# Machines file format
#
# Ranks assigned to machines in Round Robin order.
# machine ranks
# ----- ---- -
# name1 0, 3, 6
# name2 1, 4, 7
# name3 2, 5
```

```
#  
name1  
name2  
name3
```

Process Group File Format

A *procgrou*p file explicitly specifies the following:

- Names of the machines to run on
- Number of processes to run on each machine
- Name and path to the executable
- Ranks each machine should be assigned

The format of the *procgrou*p *mpirun* command specification is as follows:

```
cmprun [options] -pg_file | -pg <proc group file>
```

There is no default location of the *procgrou*p file. A file name of a *procgrou*p file must be supplied with the `-pg_file` option to **cmprun**, or an error will be reported. Below is an example of a typical *procgrou*p file.

```
# Process group file has the following format:  
#  
# Ranks assigned to machines in order that they are listed.  
#  
# The following will occur:  
# machine #procs running ranks  
# -----  
# name 1 4 /fileserver/apps/test 0,1,2,3  
# name 2 3 /fileserver/apps/test2 4,5,6  
# name 3 2 /fileserver/apps/test3 7,8  
#  
name1 4 /fileserver/apps/test
```

```
name2 3 /fileserver/apps/test2
name3 2 /fileserver/apps/test3
```

Remote Shell Utility

ChaMPIon/Pro relies on the standard *rsh* or *ssh* remote shell services for remote startup of MPI processes. The default remote startup utility in most distributions of ChaMPIon/Pro is *rsh*. In order it to change to *ssh*, please edit the *mpirun* perl script and replace the source line `my $rsh = 'rsh';` with `my $rsh = 'ssh';`. This change needs to be done on all nodes on which ChaMPIon/Pro has been installed.

Redirection of **stdin**, **stderr**, and **stdout**

ChaMPIon/Pro redirects **stderr** and **stdout** of all MPI processes to the console where **cmpirun** was executed. The output is serialized per line. The **stdin** stream is redirected only to the MPI process with rank 0.

Exporting User Environment

ChaMPIon/Pro exports the entire user environment by default.

Runtime Parameters

There are a number of run-time tunable parameters for the ChaMPIon/Pro library to enable users to customize the library for their own specific applications and environment. These parameters are specified as command line options to **cmpirun**. These parameters include:

- Cutoff message size between long and short communication protocols for all devices (TCP, SMP, GM, and VAPI).
- Setting a limit on the maximum number of buffers for unexpected messages.

Note: To see a full list of options use `mpirun -h` at the command line and the man page for `mpirun`.

Each of the runtime-controlled parameters has a default value set in the library. If the user does not request a specific value, the library will use the default values. In order to see the values of the runtime parameters, start the MPI job and the pass `-mpi_verbose` flag to **mpirun**. This will cause ChaMPIon/Pro to print information about its configuration, including the values of the runtime parameters.

Selection of Communication Devices

ChaMPIon/Pro is based on multi-device architecture. TCP and SMP devices are provided in most versions of ChaMPIon/Pro. Special high-speed network support is provided with specialized packages. The basic package of ChaMPIon/Pro contains the TCP and SMP devices. If the basic version has been installed, selection of the devices is not necessary. ChaMPIon/Pro automatically picks the SMP device whenever two processes are started on the same machine. If the processes are on different machines, then ChaMPIon/Pro uses the TCP networking device.

In ChaMPIon/Pro distributions with high-speed network support, such as Myrinet or InfiniBand, a run-time selector is provided for device selection. User applications need not be rebuilt for every device. Once, compiled and linked with ChaMPIon/Pro, the application can use any of the available devices with the installed package. When two or more networking devices are present, such as when TCP and GM, `mpirun` reads the value of the `MPI_COMM` environment variable in order to perform selection of the network media for communication. The possible values of the environment variable are currently TCP, GM and VAPI. The GM option is used for Myrinet support, and the VAPI option is used for Mellanox VAPI support. If the installed ChaMPIon/Pro package is with high-speed network support, when `MPI_COMM` is not set, the default communication device is the high-speed device.

The default selection of the SMP device for intra-host/node communication can be turned off by the use of the `-no_smp` flag with **cmpirun**. In this case, the intra-host/node communication will be implemented through the loopback capabilities of the network device (TCP, GM or VAPI).

Note: Before disabling the SMP device, make sure that the selected network device can support loopback communication.

Selection of File System

The basic package of ChaMPIon/Pro contains support for UFS and NFS. ChaMPIon/Pro automatically detects the appropriate file-system back-end to be invoked based on the filename (passed as argument). However, in case of files stored in the Lustre file system, the prefix “lustre:” needs to be passed along with the filename to the MPI I/O application. This prefix is required, as this file system does not export the necessary functionalities required for automatic detection of the file system type. Once this feature is made available in Lustre, it shall be enabled in ChaMPIon/Pro as well.

Scalable Startup

ChaMPIon/Pro relies on the standard rsh or ssh remote shell services for remote startup of MPI processes. The client utilities of these services have certain scalability limitations. For example, the rsh client distributed with most common Linux versions is limited to 252 simultaneous sessions. The OpenSSH client has a similar limitation, although the limit is much higher – more than 1,500. In order to accommodate large runs with thousands of processes, far exceeding the limitations of rsh and ssh, ChaMPIon/Pro provides a scalable startup mechanism. This mechanism is turned on by the use of the `-scale_level <level>` option to **cmpirun**. Supported levels are 1, 2, and 3. Depending on whether rsh or ssh is used for remote process startup, the number of processes started by a given level is as follows:

- rsh: level 1 up to 500 processes; level 2 up to 1000 processes; level 3 up to 2000 processes
- ssh: level 1 up to 2000 processes; level 2 up to 4000 processes; level 3 up to 8000 processes

Note: The only supported remote startup option on Mac OS X is ssh.

Connections on Demand (TCP Device only)

The TCP protocol is connection oriented. In order to exchange data between pairs of processes, the ChaMPIon/Pro library needs to establish TCP socket connections between these processes. Each socket has a send and a receive buffer maintained in the kernel space of the operating system. Establishing a large number of sockets is a time and resource consuming operation, which limits scalability. ChaMPIon/Pro supports a mechanism, called “TCP connections on demand”, which when engaged will delay the creation of socket connections until the moment they are needed. If two processes never communicate between themselves during the course of the MPI job, a socket connection will not be established. In many common parallel applications, the processes do not participate in all-to-all style of communication. For such applications, the creation of a large number of unused connections can be avoided by passing `-tcp_cod` flag to **mpirun**. This will instruct ChaMPIon/Pro to turn on its “TCP connections on demand” mechanism. This mechanism is implemented only for the TCP device and is turned off by default.

Cancellation of MPI Jobs

The **mpirun** utility allows the user to cancel “hanging” MPI applications or to terminate long running MPI programs. Even though processes are started on remote machines, they will be terminated if the user presses the <CTRL>-C key combination.

“Watching” MPI Ranks

In its default mode of operation, ChaMPIon/Pro allows the MPI jobs to continue when one or more ranks exit abnormally. This feature is useful for long-running jobs that can sustain certain level of process loss, especially applications that are created according to the master-slave model. However, in certain cases it is useful to terminate the entire MPI job if specific ranks exit abnormally, such as in a master-slave application when the master process dies. Evidently, this MPI job will not produce the expected results but may occupy the resources allocated to it for a long time, thus wasting precious computation resources. ChaMPIon/Pro provides a “rank watch” mechanism that allows the user to enumerate some or all ranks that will cause the entire job to exit if a “watched” process exits. This mode is turned on with the `-watch <rank_list>` option to `cmpirun`. The argument `<rank_list>` specifies the ranks to be “watched.” This list can be any valid perl list. For instance,

```
cmpirun -np 256 -mf mfile -watch "0,4-6" long_running_app
```

This command will start a 256-way job on the machines listed in `mfile` and will terminate the entire job if ranks 0, 4, 5, or 6 exit before the rest. Before terminating the entire job, ChaMPIon/Pro uses a time-out to allow slower processes of normally completing jobs to exit without being terminated. The default value of this time-out is 60 seconds. It can be controlled by the user with the `-watch_timeout <seconds>` option to `cmpirun`.

Debugging in the GNU Debugger (GDB)

ChaMPIon/Pro applications can be debugged interactively using the GNU Debugger (GDB). If `cmpirun` is used with a machines file the command line to start a debugging session with `gdb` is as follows:

```
cmpirun -np 2 -mf machines_file -gdb "1,3,8..9" application
      app_args
```

The argument given to the `-gdb` flag is a valid perl list specifies the ranks for which **gdb** should be run. An instance of **gdb** will be run for each rank that is specified. A working X-Windows configuration and correct `DISPLAY` and `xhost` settings are required for debugging with **gdb**.

Performance Tuning

Champion/Pro offers a number of performance tuning parameters that may help with adjusting the performance and scalability behavior of the library to the specific user's environment. One of the important run-time options is the synchronization and message progress mode of the library.

Champion/Pro supports two modes:

- blocking (default)
- polling (turned on with the `-poll` option of `emprun`).

The blocking mode uses interrupts for notification of message completion and it also provides independent message progress. This mode complies with the strict interpretation of the MPI Progress Rule. In this mode Champion/Pro uses very little of the host CPU time for communication and provides for an efficient overlapping of communication, computation, and file I/O. The blocking mode adds certain constant processing overhead related to interrupt handling and context switches, which results in increased latency for very short messages as measured by ping-pong point-to-point tests. The blocking mode is thread safe.

The polling mode is not thread safe and the message progress is implemented in the MPI calls of the user program, i.e., the message progress depends on the calling sequence of the user code. The polling mode achieves lower latency than the blocking mode at the expense of using more of the host CPU cycles for communication purposes. The polling mode has lower latency as measure by pure communication benchmarks. Typically, applications that use a large number of short

messages, which cannot exploit asynchrony and overlapping will benefit of using the polling mode.

Another option related to the communication performance of ChaMPIon/Pro is the switchover size between the short and long message protocols of the library. ChaMPIon/Pro has a default size for each device that can be inspected by the use of the `-mpi_debug` flag to `cmpirun`. This default size may not match best the particular user environment. Using `cmpirun`'s `-<dev>_long` options for the specific communication device (GM, VAPI, TCP, SMP, or ELAN), the user can change the default protocol switchover size. Important factors that can affect the value of the protocol switchover size are CPU speed, memory speed and size, cache size, and network speed.