Bull
HPC BAS3

User's Guide

86 A2 30EM Rev02

Bull
HPC BAS3

User's Guide

Subject: HPC user’s applications.

Special Instructions: Refer to SRB.

Software Supported: HPC BAS3 V2

Software/Hardware required: Refer to SRB.

Date: November 2005

Bull S.A.
CEDOC
Atelier de reprographie
357, Avenue Patton BP 20845
49008 ANGERS Cedex 01
FRANCE

86 A2 30EM Rev02

Copyright © Bull S.A., 2004, 2005

Bull acknowledges the rights of proprietors of trademarks mentioned herein.

Your suggestions and criticisms concerning the form, contents and presentation of this manual are invited.
A form is provided at the end of this manual for this purpose.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical or otherwise without the prior written permission of the publisher.

Bull disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer. In no event is Bull liable to anyone for any indirect, special, or consequential damages.
The information and specifications in this document are subject to change without notice.
Consult your Bull Marketing Representative for product or service availability.

86 A2 30EM Rev02

Preface

Scope and Objectives

The purpose of this guide is to describe the tools allowing development, tests and optimization of applications offered on Bull High Performance Computing (HPC) and to explain how to use them.

Intended Readers

This guide is for users of HPC applications.

Prerequisites

The installation of the all hardware and software components of the HPC must be completed. The HPC administrator must have performed basic administration tasks (creation of users, definition of the file systems, network configuration, etc).

Structure

This guide is organized as follows:

Chapter 1. Provides an Introduction to the HPC environment.

Chapter 2. Describes the user's environment available on HPC (cluster access and use of the files systems).

Chapter 3. Defines how to use Compilers

Chapter 4. Describes Debugging Tools.

Two types of programming libraries are used when developing programs for HPC environment: **Parallel libraries** and **Mathematical libraries**. They are described in the following chapters:

Chapter 5. Lists parallel programming libraries that use MPI communication interfaces.

Chapter 6. Describes scientific libraries and the sets of scientific functions delivered with Bull HPC BAS3 and how these may be invoked.

Chapter 7. Explains the benefits brought by modules: Using modules it is possible to test an application in different compiler and library environments, in order to decide which is the best according to the results criteria.

Chapter 8. Describes job management using TORQUE.

Chapter 9. Deals with Performance Tools and how to optimize your programs using these tools.

Appendix A provides a Troubleshooting section enabling you to resolve some common problems.

Appendix B is a list of Acronyms used in this guide.

Bibliography

- NovaScale 40xx and NovaScale 5xxx documentation.
- *Bull HPC BAS3 Installation and Configuration Guide* (86 A2 31EG)
- *Bull HPC BAS3 Administrator's Guide* (86 A2 31EM)
- *A Software Release Bulletin (SRB)* provides release-specific information and installation instructions (86 A2 31EJ).
- Bull Remote Hardware Management CLI Reference Manual (86 A2 88EM) describes the NS commands.
- Intel® Itanium® 2 Processor *Reference Manual for Software Development and Optimization.*

Web links

http://www.bull.com/novascale/hpc.html

http://www.intel.com/design/itanium2/documentation.htm

http://www.linuxhpc.org/

Syntax Notation

Commands entered by the user and messages displayed by the system to illustrate explanations are in a frame in "Courier" font. Example:

```
BIOS Intel
```

Commands, files, directories and other items whose names are predefined by the system are in "Bold". Example:

The **/etc/sysconfig/dump** file.

Table of Contents

4. Debugging Tools

5. Parallel Libraries

6. Scientific Libraries

7. Modules Package

1. Introduction to the HPC Environment

The term HPC (High Performance Computing) describes large scientific applications that require a powerful computation facility with extremely accurate results, and that are able to process a large amount of data.

The Bull HPC delivery provides a full environment for development, which includes optimized scientific libraries, Fortran and C/C++ compilers, MPI libraries, as well as debugging, performance and optimization tools.

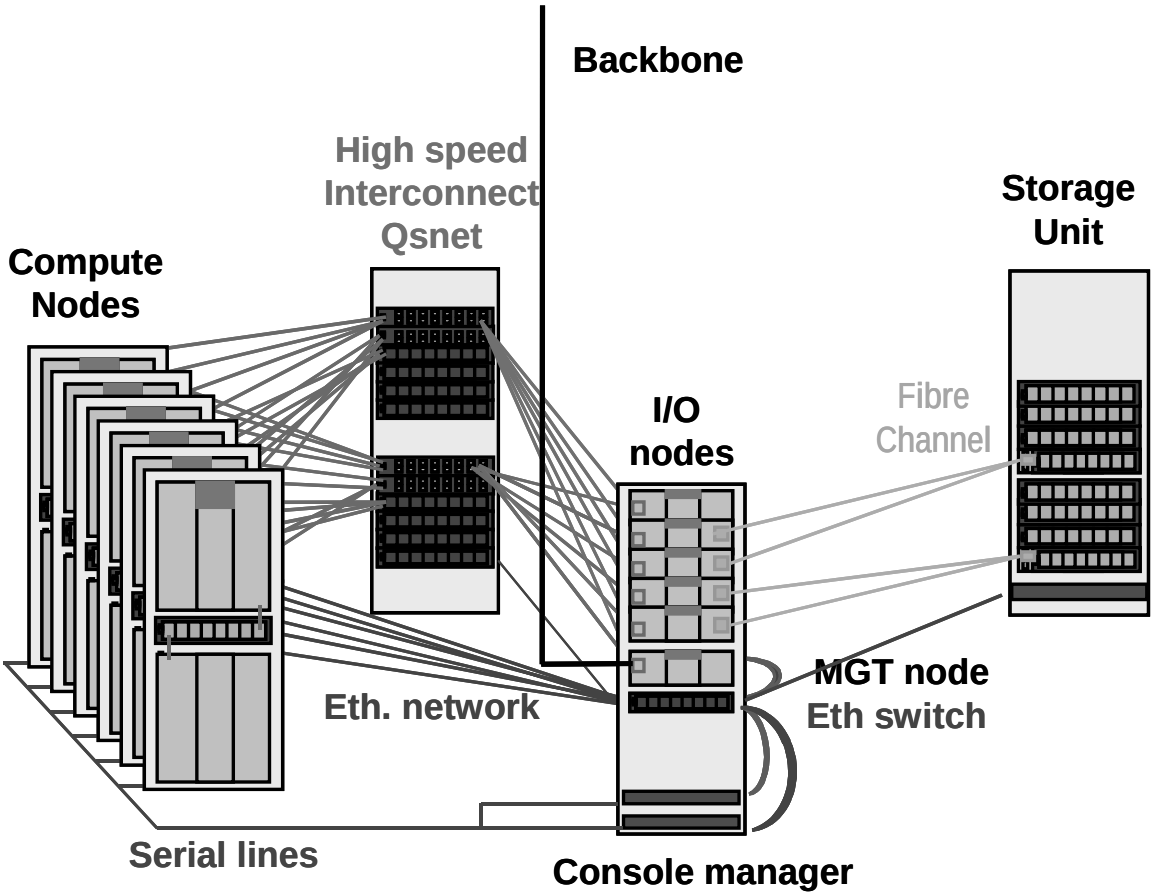
This User's Guide describes all these software components and explains how to work within this environment.

1.1 Hardware Configuration

A typical cluster infrastructure is composed of

- **Compute nodes** for intensive calculation.
- **Input/Output nodes** to store data in storage units.
- **Management node** to administer, manage and exploit the cluster.
- **High speed interconnect** switch and boards to transfer data between compute nodes and I/O nodes.
- **Administration Network** including Ethernet and serial networks which are used for cluster management and maintenance.
- **Backbone** is the link between the HPC system and the external world.
- **Storage:** Each node has to have an internal disk drive. Each node may be attached to a Bull SJ family (external SCSI cabinet JBOD) or to a Bull SR family (external cabinet with RAID) or to a Bull FDA family external fiber RAID cabinet or to Data Direct Network S2A appliances. The supported fibre adapter is the Emulex LP 9802.

This kind of infrastructure is shown in the diagram below.



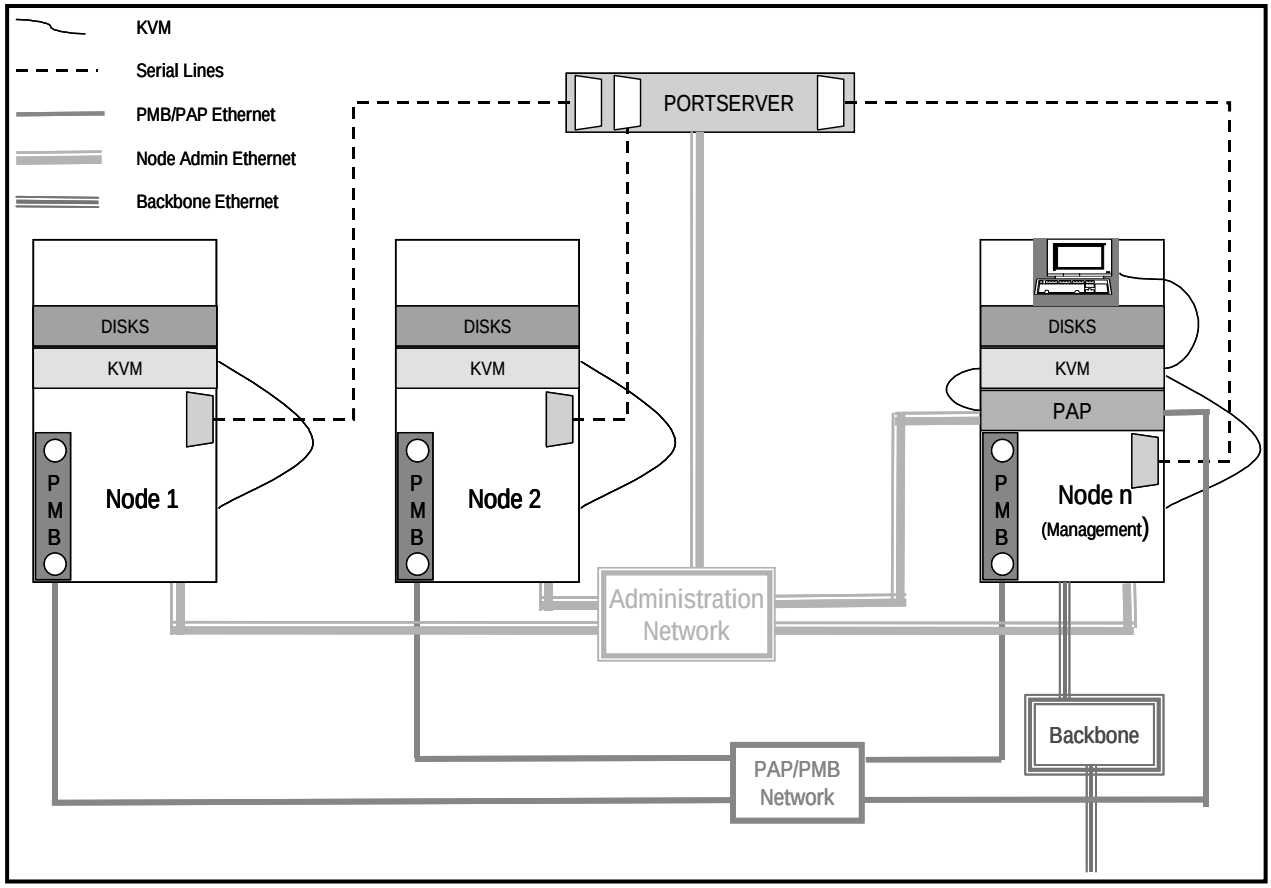
1.2 Administration Network and Backbone

The systems administration needs different kind of network support (Ethernet or serial) according to:

- the status of the system (down, running, frozen)
- the nature of the request (hardware, software) to be sent.

The **Administration network** includes **two** separate **Ethernet networks** (One for general software management of the cluster and the other for hardware management) and **one Serial network**. These networks provide the management node or on the PAM with all the information needed to control and manage the cluster.

The figure below shows a general outline of an administration network and backbone.



1.2.1 Serial Network

In a large computer rack system it is impossible to have a graphical console with keyboard and mouse for each node. Serial port concentrators on an Ethernet line are used. The **Serial network** is used for hardware and software services.

- It provides console management monitoring for all devices (nodes, disk units, switches, etc.) from the management node, and is used when Linux is no longer running.
- It provides facilities in order to obtain dump information from the Ethernet administration network when the system is frozen. It is also used to access firmware and to debug the system.
- All the devices of the cluster which has a terminal server are connected to the serial line network. Each serial line (asynchronous RS232c) is connected from COM1 to the serial multiport concentrator, for example PortServer from Digiboard.

- The PortServer itself is connected to the administration network via Ethernet.

1.2.2 PAP/PMB Network

PAP/PMB network connects all PMB and PAP.

The Platform Administration Processor (PAP) is used to power on or to power off a node and to get hardware information (state of processors or temperature for example) of all the hardware components of the cluster. using a separate Ethernet network. One PAP can control from between 1 to 16 nodes.

This network has no links with the other networks. It includes

- PAP (Platform Administration Processor)
- The PMB (Platform Management Board) of each node.
- 100 Mb/s Ethernet switch

1.2.3 Actual Administration network

The **Actual Administration network**.is an Ethernet network which facilitates the supervision of the operating systems, middleware and applications from management node. This network joins all the eth0 native ports of each node through a 100 Mb/s network. It is also connected to the PAP.

This network has no links to the other networks and includes 100 Mb/s Ethernet switch(es)

1.2.4 Backbone

The **Backbone** is the link of the cluster with the external world. This network links all the eth1 ports of each node and external networks through a 100 Mb/s network including Ethernet switches.

1.2.5 Ethernet Network and Switches

Depending on the model the switches are managed either:

- directly by Ethernet
- or through a serial line which allows the configuration of the network management over Ethernet. For this you have to plug in a serial line to the switch and all the Ethernet cables specified to be plugged in to this switch.

Useful parameters can be set up at this step including multicast management, ARP management and Fast Spanning Tree. Check the manufacturer's documentation to define which options to be set on the device.

1.2.6 Main Console and Hardware Management Commands

Hardware management

Hardware management administration and maintenance tools give you immediate insight into system status and configuration. They can be used to operate, monitor, and configure the server. The following are used:

- PAM commands available on the PAP platform. For details about PAM commands see the *Bull - NovaScale 5xx0 & 6xx0 - User's Guide*.
- NS-commands installed on the management node. These commands invoke the PAM which use the administration network. These are described in the *Bull Remote Hardware Management CLI Reference Manual*.

Console management

ConMan is a console management program designed to support a large number of console devices and simultaneous users. It currently supports local serial devices and remote terminal servers (via the telnet protocol).

ConMan and the advantages of **ConMan** for a simple telnet connection are described in the *Bull HPC BAS3 Administrator's Guide*.

Sometimes, accessing the console is the only way to diagnose and correct software failures and to debug the kernel: **Kdb** is active before the Ethernet driver is loaded, thus, usable only with an asynchronous line. The console provides:

- Access to the firmware shell (BIOS/EFI) in order to get and modify NvRAM information, to choose boot parameters from the kernel or the disk on which the node has to boot or boot from a CD-ROM to install an OS.
- Boot monitoring.
- Boot interventions like interactive file system check (**fsck**) at boot.
- Telnet sessions, using the following command:

```
telnet <serial IP address>
```

where <serial IP address> is the IP address dedicated to the node on the **PortServer**.

1.3 High Speed Interconnect.

The network includes Quadrics Interconnect and Elan4 to ensure data transfer between the nodes of the cluster.

High speed interconnects uses **QsNet^{II}** technology from **Quadrics** for interconnection. Effective bandwidth of 900 MB/s (from user space to user space), latency lower than 5 µs and the possibility to interconnect up to 4096 nodes are its main characteristics.

1.4 Software Configuration

1.4.1 Operating System and installation

The HPC **Bull Advanced Server** (or BAS) is based on a standard Linux distribution combined with a number of open source utilities to take advantage of the best from the open systems community. This open systems base combined with technology from Bull and its partners results in a powerful and complete solution for development, execution and management of parallel and serial applications simultaneously.

All cluster's nodes function using the same Linux distribution.

In order to ensure the necessary compatibility level, all the nodes involved in a computation must have the same level of software products. The image of one node type is stored as a reference image to be deployed on other nodes.

Software installation is done by firstly creating an image on a node then loading this image onto the management node and then distributing the image to the other nodes using the **System Installer Suite** (SIS) utility. This distribution is performed over the Administration Network.

1.4.2 Using the Cluster's Nodes

All nodes may be used to run user applications on but one or more of the nodes may also be used for other purposes such as login services, job launching, and cluster administration. Each node runs the same core Linux environment.

System administrators perform the majority of their administrative tasks from a node designated as the **management node**, which is used to boot cluster’s. This management node is also capable of providing system services using daemons.

The BAS software is pre-configured to monitor all critical systems and to report the information through one centralized interface. The monitoring technology for data-collection includes the **PAP** and the **Nagios plugins**. The user-interface technology is **Nagios**. Nagios events can be monitored with a browser. Nagios also manages some storage systems like DDN or FDA.

Following the initial configuration and communication over the administrative network, the administrator uses **Nagios** and **Ganglia** to collect statistics and to monitor events concerning usage and health information (hardware parameters) from the individual nodes collected by scalable software.

The BAS Software also provides a scalable event-logging utility, **syslog-ng**. This utility aggregates cluster events of interest to a single location.

In addition to monitoring and logging global information, it is sometimes necessary for the administrator to perform the same command on all nodes, or on some subset of nodes, of the system. This possibility is provided by the global parallel shell utility, **pdsh**.

mkCDrec or **mkDVDrec** provides the ability to save and restore management node on CD or DVD media.

1.4.3 Application Development

When a user logs onto the Bull Advanced Server system, the login session is directed to one of several nodes. Upon logging onto the system, the users may then develop, execute and manage their applications. For development, the environment consists of:

1. Standard Linux tools such as **gcc** (collection of free compilers that can compile C/C++ and Fortran), **gdb GnuDebugger**, and other third-party tools including the Intel® V9 Fortran and C Compilers and the Intel® Debugger **idb**.

2. Optimized parallel libraries that are a part of the BAS Software. These libraries include **MPIBull message-passing library**. **MPI_Bull** provides binary compatibility across the **Quadrics QSNet** interconnects.

3. The "**Modules**" software provides a way to predefine and switch from different environments each of which includes, for instance, a compiler, a debugger and libraries which are compatible with each other. So it's easy to invoke one given environment and make tests to compare the results with other environments.

1.4.4 Job Operation

For job execution and workload management BAS Software provides an integrated resource management, scheduling and job launch mechanism based on **RMS from Quadrics** and **Torque batch manager.**

The resource manager (**RMS** from **Quadrics**) federates all compute nodes. **RMS** is responsible for the allocation of resources to jobs. The resources are provided by nodes that are designated as compute resources. Processes of the job are assigned to and executed on these allocated resources. **RMS** also provides the means to arrange the cluster into distinct partitions.

Serial or parallel jobs may be scheduled for execution within a given partition, provided that the partition has sufficient resources e.g. memory or CPUs, to execute the job. The entire system can be designated as a single partition, allowing parallel jobs to run across all of the CPUs of the cluster. Alternatively, the system administrator can divide the system into smaller partitions.

1.4.5 Data and Files

Application file **I/O** may be performed to locally mounted storage devices, or alternatively, to remote storage using **NFS** or **Lustre** (CFS) file systems for high performance and high availability. By using a separate interconnect for administration and I/O, the Bull cluster System administrator is able to isolate user application traffic from administrative operations and monitoring. With this separation, application I/O performance and process communication can be made more predictable while still enabling administrative operations to proceed..

1.4.6 Performance Tools and Optimization

There are two major kinds of tools

1. Performance tools like **pfmon** and **top_pfb**. **Pfmon** provides an access to performance registers of Itanium 2 located in the Performance Monitoring Unit. **Top_pfb** is a Bull complement to standard **pfmon**.
2. Profiling tools using the **PAPI** library. For instance **cprof** makes it possible to search in a program for the location of sequences which are consuming certain types of events.

2. User's Environment

2.1 Introduction

This chapter describes how the HPC environment can be accessed and how to use file systems.

Typical connections and use of a HPC system for a user are described below.

- The user logs onto the HPC system through the management node only or through login node where the configuration includes specials login node(s). From these nodes the users can launch their jobs.
- Compilation may be done on all nodes which have compilers installed on them. The best method is that compilers are placed on management nodes or interactive nodes only so as to not interfere with performance on compute nodes.

2.2 Cluster Access

The **SSH** command has to be used to access a cluster node.

2.2.1 ssh syntax and description

```
ssh [-l login_name] hostname | user@hostname [command]
```

```
ssh [-afgknqstvxACNTX1246] [-b bind_address] [-c cipher_spec]
    [-e escape_char] [-i identity_file] [-l login_name] [-m mac_spec]
    [-o option] [-p port] [-F configfile] [-L port:host:hostport]
    [-R port:host:hostport] [-D port] hostname | user@hostname [command]
```

ssh (SSH client) is a program for logging onto a remote machine and for executing commands on this machine. It is intended to replace **rlogin** and **rsh**, and to provide secure encrypted communications between two untrusted hosts over an insecure

network. **X11** connections and arbitrary TCP/IP ports can also be forwarded over the secure channel. **ssh** connects and logs onto the specified hostname. The user must prove his/her identity to the remote machine using one of several methods depending on the protocol version used.

2.3 Global File System

Two major kinds of file systems are generally used in a HPC environment. These are **NFS -** a distributed file system and **LUSTRE -** a parallel file system.

Lustre is specially designed for the needs of high performance systems with a large data bandwidth. This design allows **Lustre** to take advantage of the QSNET^{II} network. Metadata and data pass through this high flow, weak latency interconnect network.

Lustre is an Open Source product under a GPL License.

Data and metadata are stored under **ldiskfs** local files. **ldiskfs** is an **ext3** file system with special patches for Lustre.

QSNET^{II} networks allows a flow of 900 MB/s for one rail (link). This flow can be limited by the flow of the disk bay and depends upon the Input/output typology.

For details about Lustre administration refer to *Bull HPC BAS3 Administrator's Guide*.

2.3.1 Using Lustre

The usual way of using Lustre is the following:

- Each user has a private directory under **/home_nfs**.
- Each user has a private directory under the **Lustre** file system. Generally data under **Lustre** file system is not saved, and can be destroyed by the cluster's administrator whenever he needs. If you want to keep it, you have to copy it under **NFS**.
- The Lustre File System is mounted, on a user's request on defined compute nodes.

- There are two possibilities when running an application on a HPC system:
  - The code is within the Lustre environment (it must have been copied from NFS before launch) and the results are generated under Lustre.
  - The code is within the NFS environment and results are generated under Lustre (output files must have been directed for the application to /mnt/lustre/user's directory)

For example:

To copy NFS files to Lustre file system using **prun** enter:

```
prun -p my_partition -N1 -n1 cp –r
~/home_nfs/‘whoami‘/pathname /mnt/lustre/‘whoami‘/pathname
```

3. Compilers

3.1 Overview

Compilers play an essential role in exploiting the full potential of Itanium 2 processors. These processors use EPIC (Explicit Parallel Instruction set Computing), which enables several instructions to be executed in parallel. This parallelism must therefore be detected and exploited at compiler level. For these reasons Bull recommends Intel compilers (C/C++ and Fortran).

GNU compiler versions are also available for users familiar with this type software. However, these compilers are unable to exploit the EPIC instruction set.

3.2 Intel Fortran Compiler

The current version of the Intel Fortran 95 compiler is version 8. The main features of this compiler are the following ones:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Preloading of data
- Conditional instruction prediction
- Speculative loading
- Optimization of the software pipeline

This compiler complies with the Fortran 95 ISO standard. It is also compatible with GNU products. **emacs** and **gbd** tools can be used with this compiler. It also supports big endian encoded files. Finally, this compiler allows the development of applications combining programs written in C and Fortran.

The compiler supports multithreading functionality:

- **OpenMP 2.0** for Fortran is supported. The compiler accepts **OpenMP** pragmas and generates a multithreaded application.
- Automatic parallelization: a compiler option detects parallelism (in particular in the computation loops) and generates a multithreaded application.

The compilers are located in the directory **/opt/intel/compilo_<version_nb>**.

To use the compilers you have to update your environment as described below.

Several versions of compilers can be installed, so you have to source the environment you have chosen by running the command:

```
source /opt/intel/compilo_<version_nb>/l_fc_pc_<package_nb>/bin/ifortvars.sh
```

For example:

- To use the compiler Fortran 8.0, enter:

```
source /opt/intel/compilo_8.0/l_fc_pc_8.0.039_pl044.1/bin/ifortvars.sh
```

- To use the compiler Fortran 8.1, enter:

```
source /opt/intel/compilo_8.1/l_fc_8.1.019//bin/ifortvars.sh
```

- To display the version of the active compiler, enter:

```
 ifort -V
```

- To obtain the documentation of the compiler, enter:

```
/opt/intel/compiler_8/l_fc_pc_8.0.049/doc
```

3.2.1 Defaults Options

Intel (**ifort**) Fortran compilers have the default options below:

I -72: Assigns column numbers with fixed format

I -02: Optimization level (software pipelining, unrolling, inlining)

I -align: alignment (memory reorganization)

I -save: static allocation for local variables **bss**

I -nomodule: compiles with F90 in the current directory

I -tpp2: Code optimization for Itanium II

I -IPF_fp_speculationfast: maximizes uses of floating point registers

I -IPF_fltacc-: enables optimizations that affect floating point accuracy.

3.2.2 Compiling for the first time

- **-p**
  allows the use of gprof.

NB. The options below may result in a loss of performance

- **-assume dummy_aliases**
  Forces the compiler to assume that dummy (formal) arguments to procedures share memory locations with other dummy arguments or with variables shared through use association, host association, or common block use. These program semantics slow performance, so you should only specify **-assume dummy_aliases** for the subprograms called that depend on such aliases. The use of dummy aliases violates the FORTRAN-77 and Fortran 95/90 standards but occurs in some older programs.
- **-c**
  If you use **-c** when compiling multiple source files, also specify **-ooutputfile** to compile many source files together into one object file. Separate compilations prevent certain interprocedural optimizations, such as when using multiple f90 commands or using **-c** without the **-ooutputfile** option.
- **-check bounds**
  Generates extra code for array bounds checking at run time.
- **-check overflow**
  Generates extra code to check integer calculations for arithmetic overflow at run time. Once the program is debugged, omit this option to reduce executable program size and slightly improve run-time performance.
- **-fpe 3**
  Using this option slows program execution. It enables certain types of floating-point exception handling, which can be expensive.
- **-g**
  Generate extra symbol table information in the object file. Specifying this option also reduces the default level of optimization to **-O0** or **-O0** (no optimization). Note: The **-g** option only slows your program down when no optimization level is specified, in which case **-g** turns on **-O0**, which slows the compilation down. If -g, -O2 are specified, the code runs a tmuch the same speed as if **-g** were not specified.
- **-inline none -inline manual** Prevents the inlining of all procedures except statement functions.

- **-save**
  Forces the local variables to retain their values from the last terminated invocation. This may change the output of your program for floating-point values as it forces operations to be carried out in memory rather than in registers, which in turn causes more frequent rounding of your results.
- **-O0**
  Turns off optimizations. Can be used during the early stages of program development or when you use the debugger.
- **-vms** Controls certain VMS-related run-time defaults, including alignment. If you specify the **-vms** option, you may need to also specify the **-align records** option to obtain optimal run- time performance.

3.2.3 Options for migration

- **-O0**
  inhibit optimization options
- **-warn all**
  all warnings are issued
- **-check all**
  all error messages such as “address out of array limits” are reported to user.
- **-fpe 3**
  enables certain types of floating point exception handling, which may be expensive in resources.
- **- traceback**
  gives more information about crashes at execution time

3.3 Intel C/C++ Compiler

The current version of the Intel C/C++ compiler is version 8. The main features of this compiler are the following ones:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Preloading of data
- Conditional instruction prediction
- Speculative loading
- Optimization of the software pipeline

This compiler complies with ISO standard Ansi C/C++ and ISO standard C/C++. It is also compatible with GNU products. A GNU C object or source code can therefore be compiled with an Intel compiler. **emacs** and **gbd** tools can also be used with this compiler.

The compiler supports multithreading functionality:

- **OpenMP** 2.0 for C/C++ is supported. The compiler accepts **OpenMP** pragmas and generates a multithreaded application.
- Automatic parallelization: a compiler option detects parallelism (in particular in the computation loops) and generates a multithreaded application.

For more details, visit Intel web site www.intel.com.

The compilers are located in the directory **/opt/intel/compilo_<version_nb>**.

To use the compilers you have to update your environment as described below.

Several versions of compilers can be installed, so you have to source the environment you have chosen by running the command:

```
source /opt/intel/compilo_<version_nb>/l_cc_pc_<package_nb>/bin/iccvars.sh
```

For example:

- To use the C/C++ 8.0 compiler, enter:

```
source /opt/intel/compilo_8/l_cc_pc_8.0.058_pl063.1/bin/iccvars.sh
```

- To use the C/C++ 8.1 compiler, enter:

```
source /opt/intel/compilo_8.1/l_cc_pc_8.1.022/bin/iccvars.sh
```

- To display the version of the active compiler, enter:

```
 icc -V
```

3.3.1 Defaults Options

The C Intel® compiler has the following default options.

I -02: optimization level (software pipelinig, unrolling, inlining)

I -ipo1

I -Ob1: allows inlining for functions declared __inline.

I -alias-args: you can define alias for arguments

I -falias: Aliasing in the program

I -ffnalias: Aliasing in the functions

I -IPF_fltacc-: enables optimizations that affect floating point accuracy

3.3.2 Options for migration

-O0: inhibit optimization options

-w2: all warnings are issued

-Wp64 displays a diagnostic for the migration

3.4 GNU Compilers

Gcc, a collection of free compilers that can compile both C/C++ and Fortran, is part of the installed Linux distribution.

4. Debugging Tools

4.1 Overview

There are two types of debuggers: symbolic debuggers and non-symbolic debuggers.

A symbolic debugger gives access to a program's symbols. This means that:

- you can access the lines of the source code file
- you can access the program variables by name.

Whereas a non-symbolic debugger gives access only to the lines of the machine code program and top physical addresses.

4.2 GDB

GDB stands for Gnu DeBugger. It is a powerful debugger, which can be used through a command line interface, or a graphical interface such as **XXGDB** or **DDD** (Data Display Debugger). It is also possible to use an emacs/xemacs interface.

GDB supports parallel applications and threads.

GDB is published as part of the GNU license.

4.3 IDB

IDB is a debugger delivered with the Intel compilers. It can be used with C/C++ and F90 programs.

4.4 TOTALVIEW

TotalView™, provided by **Etnus**, is a symbolic debugger for C, C++ and Fortran (77, 90 and HPF). It can debug **PVM** or **MPI** applications. **TotalView™** fully supports multi process programs and works just as well on mono-processor, SMP, clustered, distributed and MPP systems.

TotalView™ accepts new processes and threads exactly as they were generated by the application, whatever the processor they execute on. You can also connect to a process started up outside T**otalView™**. Data tables can be filtered, displayed and viewed in order to monitor program behavior. Finally, you can descend into the objects and structures of the program.

TotalView™ is an Xwindows application. Context-sensitive help provides you with basic information. The full documentation is available on the **Etnus** web site.

TotalView is located in the directory **/opt/totalview**.

Before running **TotalView**, update your environment using the following command:

```
source /opt/totalview/totalview-vars.sh
```

Then enter:

```
totalview&
```

For detailed information, please refer to http://www.etnus.com/.

4.5 Dmalloc

The debug memory allocation or dmalloc library is a memory management routine which provides powerful debugging facilities configurable at runtime. These facilities include such things as memory-leak tracking, fence-post write detection, file/line number reporting, and general logging of statistics.

For detailed information about dmalloc, please refer to http://dmalloc.com/ .

4.6 Electric Fence

Electric Fence is a **malloc** debugger for Linux and Unix. It stops your program on the exact instruction that overruns or under-runs a **malloc()** buffer.

Electric Fence helps you detect two common programming bugs:

- Software that overruns the boundaries of a **malloc()** memory allocation.
- Software that effects a memory allocation that has been released by **free()**

You can use the following example, replacing **`icc --version`** by the command line of your program.

```
[test@host]$LD_PRELOAD=/usr/local/tools/ElectricFence-
2.2.2/lib/libefence.so.0.0  icc --version

Electric Fence 2.2.0 Copyright (C) 1987-1999 Bruce Perens <bruce@perens.com>

……..
```

For detailed information about Electric Fence please refer to http://perens.com/FreeSoftware/ .

5. Parallel Libraries

5.1 Overview of Parallel libraries

A common approach to parallel programming is to use a message passing library, where a process uses library calls to exchange messages (information) with another process. This message passing allows processes running on multiple processors to cooperate.

Simply stated, **MPI** (Message Passing Interface) provides a standard for writing message-passing programs. An **MPI** application is a set of autonomous processes, each one running its own code, and communicating through calls to subroutines of the **MPI** library.

Bull provides different MPI for use in the HPC environment.

- The recommended one is **MPI_Bull**. This is the Bull MPI library optimized for the Novascale architecture. This component is able to run applications on **Quadrics** interconnected cluster environment and on a single node. **MPI_Bull** is divided into two parts: a global static or dynamic library with which the application is compiled and a dynamic elan (Quadrics environment) or mono single node library called when the program is running.
- MPICH_Ethernet is provided to run applications in a Ethernet environment instead of the Quadrics interconnect environment.
- Bull also enables the use of **LAM MPI** and **PVM**.

Programming with MPI

It is not in the scope of the present guide to describe how to program with MPI. Please, refer to the Web, where you will find complete information. For example, you can refer to the following site: http://www.idris.fr

5.2 MPI_Bull

5.2.1 MPI_Bull Features

Bull has developed a **MPI** (Message Passing Interface) library, called **MPI_Bull**, enabling the exchange of messages between processes in a distributed environment. This model of communication is used by parallel applications.

The **MPI_Bull** library conforms with the MPI-1 standard (refer to *MPI: A Message-Passing Interface Standard*, dated May 5, 1994). http://www-unix.mcs.anl.gov/mpi

MPI_Bull library is optimized for NovaScale hardware architectures. **Quadrics Elan3** or **Elan4** ensure hardware interconnection.

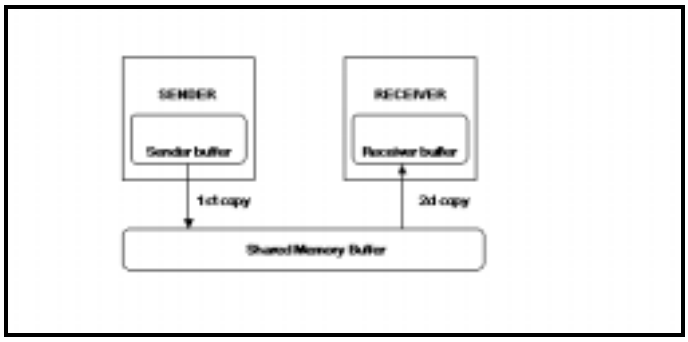
MPI_Bull has been developed from MPICH 1.2.4 Open Source library to which several improvements were made:

- **Mapping of processes on processors**, in order to inhibit the scheduler actions (the ideal operation is one process by processor).
- **Introduction of the Futex mechanism** (Fast User mode muTEX), for locks management in the interface.
- **Improvement of MPI_Barrier algorithm** (processes synchronization).
- **Optimization of collective operations** (**broadcast** for example).
- **Allocation of a memory area** to benefit from the advantages of the NUMA architecture of NovaScale 5160.
- **Optimization of data copies** between processes located on the same machine, using the MDM module (MPI Data Mover module), for Linux kernel 2.6 and higher.
- **Cpuset** use to locate processes
- **Thread Safety**: **MPI_Bull** is thread safe (MPI-2 functionality).
- **Introduction of One-Sided communications** which is also a MPI-2 functionality.
- **Introduction of a MPI_Bull profiler called Profilecomm** and which allows applications to be profiled using MPI_Bull.
- **Integration of Quadrics** Elan library.
- **Integration of mono library** allowing applications to run on a single node.

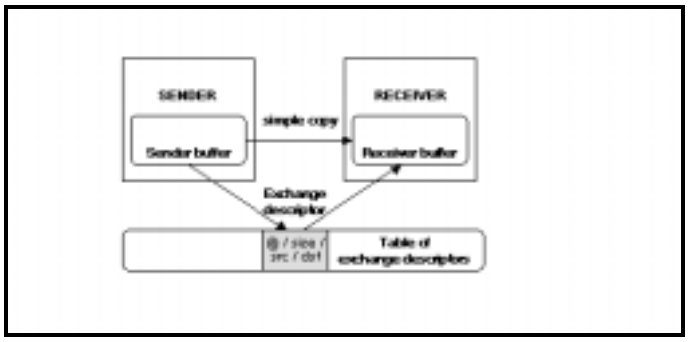
Note: For more information about MPICH 1.2.4 Open Source library, please refer to: http://www-unix.mcs.anl.gov/mpi/mpich/.

5.2.2 The MPI Data Mover module mdm

With MPICH in an intra-machine communications scheme, the sender copies data into a shared memory buffer and then the receiver copies it in turn in their own memory space. Therefore, two copies are required, as illustrated below:



With **MPI_Bull**, the **mdm** module uses one copy only, by directly copying the source buffer into the destination one, as illustrated here:



The **mdm** module is also used in one-sided communications and manages longer lasting communications (Memory window allocation).

mdm integrates a trace tool, a profiling tool and a **KDB** module.

The **trace tool** logs the last events for each processor. This tool has the advantage of not influencing the application's behavior, enabling problems of concurrent access to data and of processes synchronization to be solved.

The **KDB module** allows access to traces when a crash occurs, enabling errors to be detected more efficiently.

The **profiling tool** helps to outline the critical parts of the application code. Its implementation is similar to that for trace tools, and leads to minimal loss of performance.

See *Using the mdm* Module, on page 5-5 for information about using these tools.

5.2.3 Using MPI_Bull

MPI_Bull can work in two different environments - with the Quadrics interconnect or on a single node. For both environments the compilation mode is the same. No specific configuration is required. You simply have to compile your application, for example appli.exe, using the script **mpicc** (C), **mpiCC** (C++), **mpif77** or **mpif90**. **MPI_Bull** can work in two different modes:

For example: the following command allows the appli.c code to be compiled using the MPI library:

```
$ mpicc -o /appli.exe /appli.c
```

5.2.4 MPI_Bull and Quadrics interconnect environment

The application is launched with the **prun** command. **prun** is part of **RMS** from **Quadrics.**

Example: the following command launches the **appli.exe** application on `3` processes (**-n** flag), on the `2` first nodes (**-N** flag) of the RMS partition `partition` (**-p** flag):

```
$ prun -n 3 -N 2 -p partition ./appli.exe
```

For more details about the **prun** command and the options available, run:

```
 $ prun -h.
```

For more information about using **RMS** and its configuration, please refer to **Quadrics** documentation.

5.2.5 MPI_Bull on a single node

The application is launched with the **mprun** command (Bull specific command).

Example: the following command launches the **appli.exe** application on `4` processes (**-n** flag).

```
$ mprun -n 4 ./appli.exe
```

5.2.6 Using the mdm Module

The **mdm** module (*see The MPI Data Mover module mdm*, on page 5-3)is an integral part of **MPI_Bull**, it has no specific options. However it does include a trace and profiling tools, enabling trace data to be analyzed.

Information related to profiling is in **/proc/mdm/profiler**. To display the profile, run:

```
$ cat /proc/mdm/profiler
mdm profiling data ======================
```

The trace tool is relevant when an application behaves abnormally. To view the events that occurred on the different processors consult **/proc/mdm/trace/<n>**, where **n** is the CPU number. For example:

```
$ cat /proc/mdm/trace/0
ITC            UID     DESCRIPTION
================================================
```

You may also observe in real time the events occurring for the process whose rank is r and the application whose MPI jobkey is p with the following command. **/proc/mdm/<p>/trace/rank_<r>.**

```
$ cat /proc/mdm/145231/trace/rank_0
ITC        UID        DESCRIPTION
====================================
```

The status of an application may be known by reading the file **/proc/mdm/<p>/status**, where **p** is the MPI jobkey.

```
$ cat /proc/mdm/145231/status
```

Also, one-sided communications window descriptors are available under the directory **/proc/mdm/<p>/onesided/**, where **p** is the MPI jobkey. In this directory, you may consult the file **rank_<r>**, where **r** is the process rank.

```
$ cat /proc/mdm/145231/onesided/rank_0
ID       WIN                BADDR              SIZE
================================================================
```

To display the MDM module release number, run:

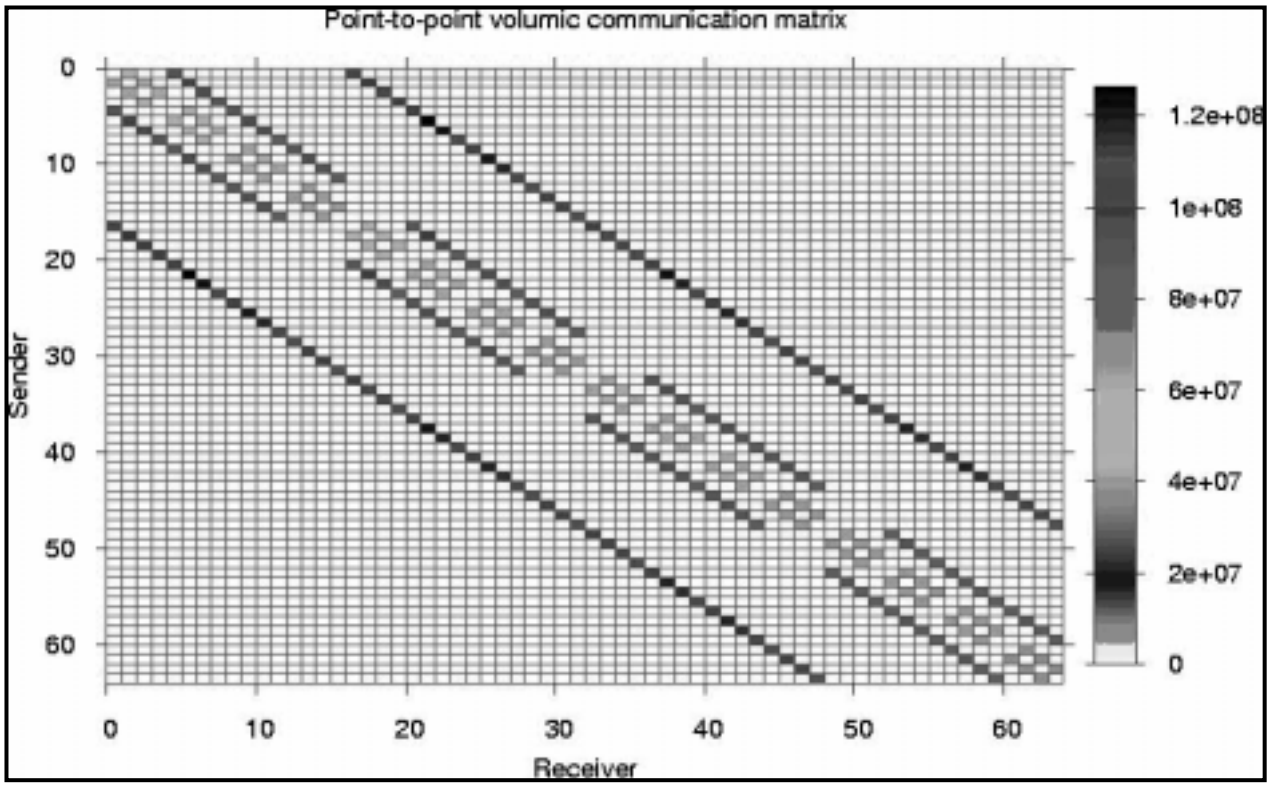
```
$ cat /proc/mdm/version
MDM module release mdm (mdm:aravis) 1.0.0-0 [ version kunlock dynamic
32cpus pt2pt profiler ] {mpi_bull >= 1.0.0}
```

5.2.7 Using Profilecomm

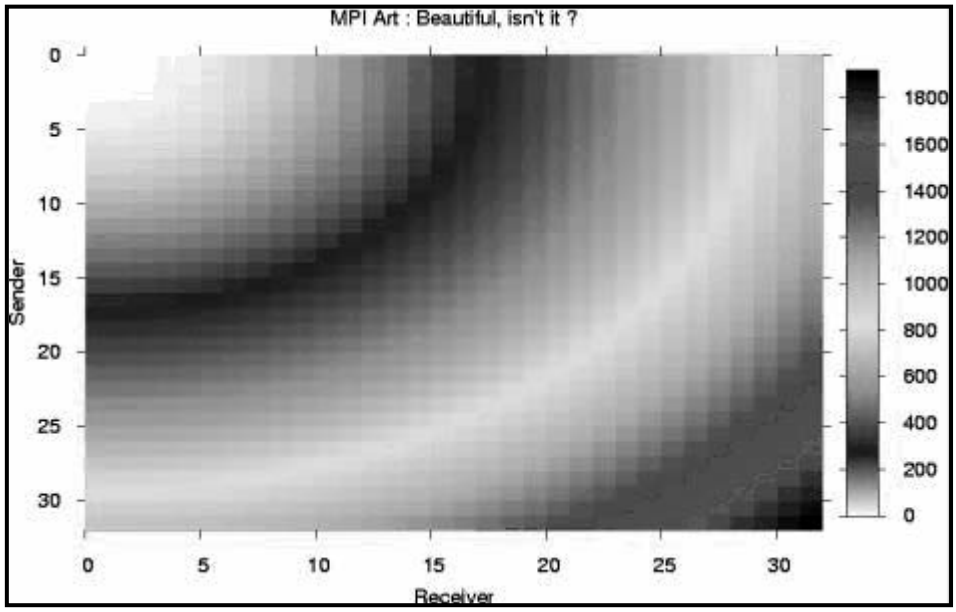
Profilecomm is a profiling tool, developed by Bull for its own **MPI_Bull** implementation. This is a non-intrusive tool allowing counter information accumulated during the application's run to be displayed. **MPI_Bull** has been modified to include **Profilecomm** calls. See the *Performance Tools* chapter for further information.

Example:

The following picture is a graphic of the matrix representing message exchanges between 32 processes.



The following picture presents the results of an application run with **MPI_Bull**, using a linear color scale. Each color represents the quantity of data exchanged during the application run.



5.3 MPICH_Ethernet

Bull supplies **mpich_Etherne**t (version 1.2.6), which is to be used with an Ethernet interconnect.

Modify the file **/opt/mpi/mpich_ethernet-1.2.6/share/machines.LINUX** in order to set, the name and the number of processors for each machine. For example:

```
ns0:4
ns1:4
ns2:4
ns3:4
```

Next, the program using **mpich_ethernet** must be compiled using a command similar to the one below where **np** is the number of processes, and **appli.exe** is the name of the application using MPI:

```
$mpirun –np 4 ./appli.exe
```

For more details, see the *Installation and User's Guide to MPICH, a portable implementation of MPI* which is available from `http://www-unix.mcs.anl.gov/mpi/mpich`

5.4 LAM MPI

5.4.1 Description

The LAM MPI library was developed by the University of Notre Dame (Indiana). This implementation is based on a system of daemons running on each node which ensure messages are sent and received. It is also possible to launch different processes that communicate between themselves. The typical example is a "master" process with several "slave" processes. They are all different, communicate via MPI and are launched on several different nodes.

Release 7.0 introduces the concept of **SSI** (System Service Interface) and **RPI** (Request Progression Interface) modules. These modules are chosen and loaded at runtime. The RPI modules available today are:

- Myrinet (gm)
- Shared Memory (sysv et usysv)
- TCP (tcp)
- Check point restart TCP (crtcp).

After the install step has been performed, the library will be located in the folder that has been specified with the –prefix flag.

For more information, please refer to www.lam-mpi.org .

5.4.2 Installing and Compiling LAM MPI

Download

The Lam MPI is available on the BAS CD. Please download the lam-7.1.1-1.Bull.ia64.rpm file and install it.

Installation

To install Lam MPI a relocation prefix has to be specified. Otherwise the installation may erase many **MPI_Bull** files. Run the following command:

```
$rpm –ivh –relocate /opt/mpi/lam-mpi7.1.1/ lam-7.1.1-1.Bull.ia64.rpm
```

Increase the number of semaphores.

The **sysv RPI** uses semaphores. For a NovaScale 5160 system it may be necessary to increase the number of available semaphores. To do this one must be the root:

```
#  cat /proc/sys/kernel/sem
250     32000   32      128
# echo "50000 50000 100 500" > /proc/sys/kernel/sem
# cat /proc/sys/kernel/sem
50000   50000   100     500
#
```

5.4.3 Using ssh

When first connecting to a node using **ssh**, a warning message will be printed. **LAM** may be confused by these messages. Thus it is recommended to establish the first connection to each node manually. The following command should return the date without having to enter a password.

```
ssh -n `hostname` date
```

Examples

The commands below indicate how to compile and launch the programs of the folder named `examples`.

```
$  export MANPATH=/opt/mpi/lam-mpi7.1.1/man:$MANPATH
$  export LAMHOME=/opt/mpi/lam-mpi7.1.1/
$  export PATH=$LAMHOME/bin:$PATH
$  cat hostfile
clus1admin cpu=4
$  lamboot hostfile

LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University

$  lamnodes
n0      clus1admin:4:origin,this_node
$
```

- If there is a problem with the **lamboot** command, check the **/etc/hosts** file as well as the output of the **hostname –f** command.

- Compile the examples files:

```
make examples
```

- Launch an example:

```
$  mpirun C examples/hello/hello
mpirun(31370): 0x60000fffffffa2cc, ip=0x40000000000383a0
mpirun(31370): unaligned access to 0x60000fffffffa2cc, ip=0x40000000000383a0
Hello, world!  I am 0 of 4
Hello, world!  I am 1 of 4
Hello, world!  I am 2 of 4
Hello, world!  I am 3 of 4
```

- One can get rid of the `unaligned access to` warning messages with the following command:

```
prctl --unaligned=silent
```

Tests

```
export LAMHOME=/opt/mpi/lam-mpi7.1.1/
unset CC FC CXX
export MANPATH=$LAMHOME/man:$MANPATH
export PATH=$LAMHOME/bin:$PATH
export CPPFLAGS="-I$LAMHOME/include"   FLIBS="-Vaxlib"
make
make –k check
```

The second make command executes 142 tests (13+8+21+4+17+3+1+12+9+2+3+1+12+32+4). It should not be executed on a large cluster.

Sample Execution

Following is an example of usage of LAM-MPI.

```
$  laminfo

           LAM/MPI: 7.0.6
            Prefix: /usr/local/lam-7.0.6
      Architecture: ia64-unknown-linux-gnu
     Configured by: gutfreud
     Configured on: Mon Nov  3 10:16:53 CET 2003
    Configure host: clus1admin
        C bindings: yes
      C++ bindings: yes
  Fortran bindings: yes
       C profiling: yes
     C++ profiling: yes
 Fortran profiling: yes
     ROMIO support: yes
      IMPI support: no
     Debug support: no
      Purify clean: no
          SSI boot: globus (Module v0.5)
          SSI boot: rsh (Module v1.0)
          SSI coll: lam_basic (Module v7.0)
```

```
           SSI coll: smp (Module v1.0)
            SSI rpi: crtcp (Module v1.0)
            SSI rpi: lamd (Module v7.0)
            SSI rpi: sysv (Module v7.0)
            SSI rpi: tcp (Module v7.0)
            SSI rpi: usysv (Module v7.0)

$  hostname -f

clus1admin

$  cat hostfile

clus1admin cpu=4

$  lamboot hostfile


LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University

$  lamnodes

n0      clus1admin:4:origin,this_node

$  mpirun C  /usr/local/mpi/lammpi-7.0.6/examples/hello/hello

Hello, world!  I am 0 of 4
Hello, world!  I am 1 of 4
Hello, world!  I am 2 of 4
Hello, world!  I am 3 of 4

$  prctl --unaligned=silent

$  mpirun C  lam-7.0.6/examples/hello/hello

Hello, world!  I am 0 of 4
Hello, world!  I am 1 of 4
Hello, world!  I am 2 of 4
Hello, world!  I am 3 of 4

$  mpirun -v  lam-7.0.6/examples/mandelbrot/myapp

$  xview  lam-7.0.6/examples/mandelbrot/mandel.out

$  lamhalt


LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University

$
```

5.4.4 Using LAM within a Cluster

To use LAM within a cluster, you must of course edit the **hostfile** file to declare on separate lines the name of each node, with the right number of CPUs. The **lamboot** command must be executed only once, on the management node. The program binaries must be visible on each node, each with the same absolute path. The easiest option is to use an **NFS** mounted filesystem.

Each user must add the following line in the .**bashrc** user's specific file:

```
export LAMHOME=/opt/mpi/lam-mpi7.1.1/
export PATH=/$LAMHOME/bin:$PATH
```

5.5 PVM

PVM stands for Parallel Virtual Machine.

PVM has been developed by Oak Ridge National Laboratory. **PVM** makes it possible to use a set of Unix workstations linked by a network as a parallel machine. This set of workstations is a virtual parallel machine. The machines may be distributed geographically. They may be heterogeneous, i.e. composed of different processors working under operating systems that may also differ one from another. Once installed, PVM enables different machines to communicate between themselves using messages exchanged by their processes.

PVM is made up of two elements:

1. A programming interface available for C and Fortran. This library contains the primitives required for generating processes running in parallel on several machines.
2. The daemon of a **PVM** virtual machine is a process running on each node in the cluster. The daemons interexchange messages with instructions supplied by the primitives. These messages are used to load programs on the various nodes of the cluster, to set parameters, to collect results, etc.

PVM has the following features:

- It is public domain software
- It is easy to install
- It is very flexible:
  - it can run on a great variety of different architectures and machines
  - it runs on LANs and other networks (LAN, WAN or a combination of both)
  - it is the user's application which decides where and when the components are to be executed and that determines controls and dependencies
  - it allows programming in various languages (essentially C and Fortran)
  - the virtual machine is easy to define and to modify.

For more information, please refer to www.csm.ornl.gov/pvm/pvm_home.html

5.5.1 Using PVM

Following is an example of a **PVM** session.

To use **PVM**, you have to define the following variables:

```
export PVM_ROOT=/usr/share/pvm3

export PVM_ARCH=LINUXIA64

export PATH=${PVM_ROOT}/lib/${PVM_ARCH}:${PVM_ROOT}/bin/${PVM_ARCH}:${PATH}
```

```
icc -o hello /usr/share/pvm3/examples/hello.c -I /usr/share/pvm3/include -L
/usr/share/pvm3/lib/LINUXIA64/
-lpvm3

pvm

pvm> conf

conf
1 host, 1 data format
            HOST     DTID     ARCH   SPEED       DSIG
            ns0    40000  LINUX64    1000 0x00408c41

pvm> add ns1

add ns1
1 successful
            HOST     DTID
            ns1    80000

pvm> add ns2

add ns2
1 successful
            HOST     DTID
            ns2    c0000

pvm> conf

conf
3 hosts, 1 data format
              HOST     DTID     ARCH   SPEED       DSIG
              ns0    40000  LINUX64    1000 0x00408c41
              ns1    80000  LINUX64    1000 0x00408c41
              ns2    c0000  LINUX64    1000 0x00408c41

pvm> spawn -10 /usr/local/pvm3/bin/LINUX64/hello

spawn -10 /usr/local/pvm3/bin/LINUX64/hello

10 successful
t80001
t80002
t80003
t80004
tc0001
tc0002
tc0003
t40002
t40003
t40004

pvm> halt

halt
Terminated
```

5.5.2 Installing and Compiling PVM

Install the rpm file

PVM is available on the BAS CD.

To install PVM, please use the following command:

```
rpm –ivh –-relocate=/opt/pvm/ pvm-3.4.5-13.Bull.ia64.rpm
```

Creation of environment variables

To do this, create the script **ENV_PVM** (in /opt/envhpc for example):

```
export PVM_ROOT=/opt/pvm
export PVM_ARCH=LINUX64
export PATH=$PVM_ROOT/lib/$PVM_ARCH:$PVM_ROOT/bin/$PVM_ARCH:$PATH
```

An example of this file is given in /home/packages_sources/ ENVIRONMENT_VARIABLES/

Run environment variables:

Set the compilers environment (source **iccvars.sh** and **ifortvars.sh** where the compilers are installed).

Edit configuration files

Go to the /opt/pvm directory

```
cd /opt/pvm
```

Note: The various architectures provided are in the **conf** subdirectory. The one to use is LINUX64.

Make the following changes to **conf/LINUX64.def**:

```
-DRSHCOMMAND=\"/usr/bin/ssh\" (instead of  "/usr/bin/rsh")
```

Make the following changes to **Makefile.aimk**:

```
CC = icc
F77 = ifort
```

Compile and run

```
make all
```

To run PVM and its daemon, simply run the following command:

```
pvm
```

Test the Installation:

- In standalone mode, run:

```
pvm
```

The following prompt should be displayed:

```
pvm>
```

To exit, type:

```
quit
```

The following is displayed:

```
Console : exit handler called
pvmd still running
```

Basic tests:

```
cd bin/LINUX64
./hello
./master1
./fmaster1
```

- In Cluster mode:
  - a **pvm3** directory should be on each node participating in the Parallel Virtual Machine.
  - **pre-requisite**: **ssh** does not ask for a password between nodes participating in the PVM.
  - Modify the **.bashrc** file of user to make it call /**opt/envhpc/ENV_PVM**.
  - Add pvm hosts:

```
Pvm
pvm> add “remote_node_name”
pvm> conf (to see PVM hosts)
```

  - Basic tests

```
ssh “remote_node_name” ‘echo $PVM_ROOT’
cd bin/LINUX64
pvm
pvm> spawn -“remote_node_name” -> hello
```

6. Scientific Libraries

6.1 Overview

Scientific Libraries are sets of tested, validated and optimized functions. They spare the users the development of such subprograms.

The advantages of these scientific libraries are the following ones:

- Transportability
- Support for different types of data (real, complex, double precision, etc)
- Support for different kinds of storage (banded matrix, symmetrical, etc).

Various libraries exist, which are recognized by the scientific community, including the libraries described in this chapter.

Delivery

The scientific libraries **BLACS, SCALAPACK, FFTW, Blocksolve95, SuperLU, PETSC** use **MPI** (Message Passing Interface). They are delivered in several flavours depending on the implementation used to create them. There are currently two implementations:

- **MPI_Bull** , for single nodes or clusters using **Quadrics** interconnect.
- **MPICH_Ethernet**, for clusters using Gigabit Ethernet interconnect.

6.2 Intel Math Kernel Library

This library, which has been optimized by Intel® for its processors, contains among other things, the following libraries: **blas**, **lapack** and **fft**.

An installation leaflet is provided by Bull with the library.

The library is located in the **/opt/intel/mkl<release_nb>/** directory.

To use it, you have to specify the environment by updating the LD_LIBRARY_PATH variable:

```
export LD_LIBRARY_PATH=/opt/intel/mkl<release_nb>/lib/64:$LD_LIBRARY_PATH
```

For example for MKL 7.2:

```
export LD_LIBRARY_PATH=/opt/intel/mkl72/lib/64:$LD_LIBRARY_PATH
```

6.3 Intel Cluster Math Kernel Library

The Intel Cluster Math Kernel Library contains all the highly optimized maths functions of Math Kernel Library plus **ScaLAPACK** for Linux Clusters.

An installation leaflet is provided by Bull with the library.

The Cluster MKL library is located in the **/opt/intel/mkl<release_nb>cluster/** directory.

6.4 BLAS

BLAS stands for Basic Linear Algebra Subprograms.

This library contains linear algebraic operations that include matrixes and vectors. Its functions are divided into three parts:

- Level 1 routines to represent vectors and vector/vector operations.
- Level 2 routines to represent matrixes and matrix/vector operations.
- Level 3 routines mainly for matrix/matrix operations.

This library is included in the Intel MKL package.

6.5 BLACS

BLACS stands for Basic Linear Algebra Communication Subprograms.

BLACS is a specialized communications library using message passing. After defining a process chart, it exchanges vectors, matrices and blocks. It can be compiled for systems using either **MPI** or **PVM**.

BLACS uses MPI and thus it is delivered in two releases, corresponding to the two available MPI interfaces (see *Overview*, on page6-1).

6.5.1 Using BLACS

This paragraph describes the installation and the use of the BLACS library, using either MPICH_Ethernet or the MPIBull implementations of MPI.

The library is located in the directory **/usr/lib/blacs**:

```
$ ls /usr/lib/blacs/blacs*
blacsCinit_MPI-LINUX-0.a
blacsF77init_MPI-LINUX-0.a
blacs_MPI-LINUX-0.a

$
```

Install the rpm file

The **BLACS** rpm file is available on the HPC CD.

It is installed as follows:

– To use it with MPI_Bull:

```
$rpm –ivh –-relocate /opt/blacs/ blacs-mpi_bull-1.6.2-1_p3-1.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet:

```
$rpm –ivh –-relocate /opt/blacs/ blacs-mpich_ethernet-1.2.6-1.1_p3-
1.Bull.ia64.rpm
```

Files to be created

Create the **Bmake.inc** file:

```
$cat Bmake.inc
BTOPdir     = /opt/mpi/blacs
SHELL       = /bin/sh
COMMLIB     = MPI
PLAT        = LINUX
BLACSdir    = $(BTOPdir)/LIB
BLACSDBGLVL = 0
BLACSFINIT  = $(BLACSdir)/blacsF77init_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
BLACSCINIT  = $(BLACSdir)/blacsCinit_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
BLACSLIB    = $(BLACSdir)/blacs_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL).a
MPIdir      = /opt/envhpc/mpich_shmem
MPILIBdir   = $(MPIdir)/lib
MPIINCdir   = $(MPIdir)/include
MPILIB      =
BTLIBS      = $(BLACSFINIT) $(BLACSLIB) $(BLACSFINIT)   $(MPILIB)
INSTdir     = $(BTOPdir)/INSTALL/EXE
TESTdir     = $(BTOPdir)/TESTING/EXE
FTESTexe    = $(TESTdir)/xFbtest_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL)
CTESTexe    = $(TESTdir)/xCbtest_$(COMMLIB)-$(PLAT)-$(BLACSDBGLVL)
SYSINC      = -I$(MPIINCdir)
INTFACE     = -DAdd_
SENDIS      =
BUFF        =
SYSERRORS   =
TRANSCOMM      = -DCSameF77
WHATMPI        =
DEBUGLVL       = -DBlacsDebugLvl=$(BLACSDBGLVL)
DEFS1          = -DSYSINC $(SYSINC) $(INTFACE) $(DEFBSTOP)   $(DEFCOMBTOP)
$(DEBUGLVL)
BLACSDEFS      = $(DEFS1) $(SENDIS) $(BUFF) $(TRANSCOMM)  $(WHATMPI)
$(SYSERRORS)
F77            = mpif90
F77FLAGS       = $(F77NO_OPTFLAGS) -O
F77LOADER      = $(F77)
F77LOADFLAGS   =
CC             = mpicc
CCFLAGS        =
CCLOADER       = $(CC)
CCLOADFLAGS    =
ARCH           = ar
ARCHFLAGS      = r
RANLIB         = ranlib
$
```

Update the environment and compile the library

Set the compilers environment to source **iccvars.sh** and **ifortvars.sh** which are available where the compilers are installed.

```
make mpi   what=clean

make mpi
```

Compile the tester

```
cd TESTING/
make clean
make
```

Tests

The tester is launched using the following commands:

– With MPIBull

```
cd EXE
prun -n 4 –p partition_name xCbtest_MPI-LINUX-0
prun -n 4 –p partition_name xFbtest_MPI-LINUX-0
```

– With MPICH_Ethernet

Set the MPICH_Ethernet environment:

```
export MPI_HOME=/opt/mpi/mpich_ethernet/
export PATH=$MPI_HOME/bin:$PATH
export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
cd EXE
mpirun -np 4 xCbtest_MPI-LINUX-0
mpirun -np 4 xFbtest_MPI-LINUX-0
```

Note that the tester must be launched with at least 4 processes.

6.6 PBLAS

PBLAS stands for Parallel Basic Linear Algebra Subprograms.

PBLAS is the parallelized version of BLAS for distributed memory machines. It requires cyclic distribution by matrix block that the BLACS library offers.

This library is included in the Intel MKL package.

6.7 LAPACK

LAPACK stands for Linear Algebra PACKage.

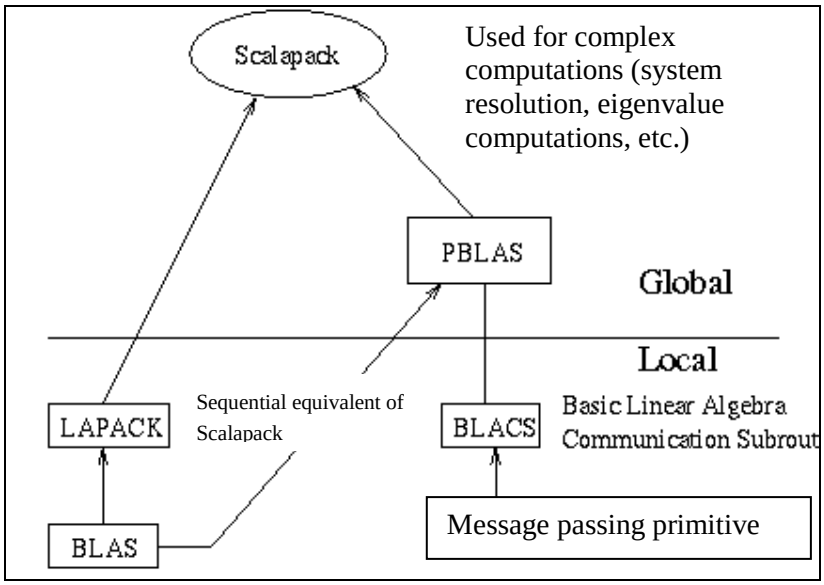
This is a set of Fortran 77 routines used to resolve linear algebra problems such as the resolution of linear systems, eigenvalue computations, matrix computations, etc. However, it has not been developed for a parallel architecture.

This library is included in the Intel MKL package.

6.8 SCALAPACK

SCALAPACK stands for: SCAlable Linear Algebra PACKage.

This library is the scalable version of LAPACK. Both libraries use block partitioning to reduce data exchanges between the different memory levels to a minimum. SCALAPACK is used above all for EIGEN value problems and factorizations (LU, Cholesky and QR). Matrices are distributed using BLACS.



Interdependence of the different mathematical libraries

6.8.1 Using SCALAPACK

Local component routines are called by a single process with arguments residing in local memory.

Global component routines are synchronous and parallel. They are called with arguments that are matrices or vectors distributed over all the processes.

SCALAPACK uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs (see *Overview*, on page6-1).

The **libscalapack.a** file is located in the **/usr/lib/scalapack** directory.

6.8.2 Installing and compiling Scalapack

This paragraph describes the installation of the SCALAPACK library using MPICH_Ethernet 1.2.6, or the MPIBull implementations of MPI.

SCALAPACK uses the BLACS library. Thus BLACS must be installed before installing SCALAPACK.

Install the rpm file

SCALAPACK is available on the HPC CD.

To install SCALAPACK, please use the following command:

– To use it with MPIBull

```
$rpm –ivh –-relocate /opt/scalapack/ scalapack-mpi_bull-1.6.2-1.7-1
.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet

```
$rpm –ivh –-relocate /opt/scalapack/ scalapack-mpich_ethernet-1.2.6-1.7-
1.Bull.ia64.rpm
```

Update the environment and compile

The environment variables have to be updated for the usage of the correct MPI implementation (one of the ENV_ files has to be sourced).

Set the compilers and MPI environment to source **iccvars.sh** and **ifortvars.sh** which are available where the compilers are installed.

– With MPIBull

Nothing else has to be done

– With MPICH_Ethernet

Set the MPICH_Ethernet environment:

```
$ export MPI_HOME=/opt/mpi/mpich_ethernet-1.2.6/
$ export PATH=$MPI_HOME/bin:$PATH
$ export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
```

The library and the tests are compiled using the following command:

```
make clean all exe
```

Tests

There are 102 tests. These tests can be grouped as follows:

- PBLAS test suite: 12 programs
- PBLAS timing suite: 12 programs
- REDIST test suite: 10 programs
- SCALAPACK test suite: 18 programs, which can be compiled with different precision levels.

The variable TOTMEM type PARAMETER can be modified to adapt to the memory of the tested system.

The following shell script executes all the tests:

- If you use MPIBull:

```
#! /bin/csh
setenv OUT runall.out
rm -f $OUT
foreach exe ( xcbrd   xcinv        xcpblas2tst  xctrmr  xdinv xdpblas2tst
xdtrd   xsgemr  xspblas1tim  xsqr    xzevc   xzlu xcdblu  xcllt
xcpblas3tim  xdbrd   xdllt xdpblas3tim  xdtrmr  xspblas1tst  xssep   xzgblu
xznep xzptllt xcdtlu  xcls         xcpblas3tst  xddblu  xdls
xdpblas3tst  xigemr  xshrd   xspblas2tim  xssvd   xzgemr  xzpblas1tim  xzqr
xcevc   xclu         xcpbllt      xddtlu  xdlu         xdpbllt      xitrmr
xsinv   xspblas2tst  xstrd   xzgsep  xzpblas1tst  xcgblu  xcnep xcptllt
xdgblu  xdnep xdptllt xsbrd   xspblas3tim  xstrmr  xzhrd   xzpblas2tim
xztrd xcgemr  xcpblas1tim  xcqr xdgemr  xdpblas1tim  xdqr xsdblu  xsls
xspblas3tst  xzbrd   xzinv xzpblas2tst  xztrmr xcgsep  xcpblas1tst  xcsep
xdpblas1tst  xdsep xsdtlu  xslu    xspbllt      xzdblu  xzllt   xzpblas3tim
xchrd   xcpblas2tim  xctrd        xdhrd   xdpblas2tim  xdsvd xsgblu  xsnep
xsptllt      xzdtlu  xzls    xzpblas3tst xsgsep xsllt xdgsep  xzpbllt xzsep
)
  echo ""            |& tee -a $OUT
  echo "-------------------------------->  testing   $exe "      |&  tee -a
$OUT
  echo ""            |&  tee -a $OUT

prun -n 4   -p parallel    ./$exe |&  tee -a $OUT

  echo ""            |&  tee -a $OUT
end
```

– If you use MPICH_Ethernet:

```
 [SCALAPACK]$ cd TESTING
[TESTING]$ more runall.csh
#! /bin/csh
setenv OUT runall.out
rm -f $OUT
foreach exe ( xcbrd   xcinv        xcpblas2tst  xctrmr  xdinv xdpblas2tst
xdtrd   xsgemr  xspblas1tim  xsqr    xzevc   xzlu xzpbllt xcdblu  xcllt
xcpblas3tim  xdbrd   xdllt xdpblas3tim  xdtrmr  xspblas1tst  xssep   xzgblu
xznep xzptllt xcdtlu  xcls         xcpblas3tst  xddblu  xdls
xdpblas3tst  xigemr  xshrd   xspblas2tim  xssvd   xzgemr  xzpblas1tim  xzqr
xcevc   xclu         xcpbllt      xddtlu  xdlu         xdpbllt      xitrmr
xsinv   xspblas2tst  xstrd   xzgsep  xzpblas1tst  xzsep xcgblu  xcnep
xcptllt xdgblu  xdnep xdptllt xsbrd   xspblas3tim  xstrmr  xzhrd
xzpblas2tim  xztrd xcgemr  xcpblas1tim  xcqr xdgemr  xdpblas1tim  xdqr
xsdblu  xsls    xspblas3tst  xzbrd   xzinv xzpblas2tst  xztrmr xcgsep
xcpblas1tst  xcsep xdpblas1tst  xdsep xsdtlu  xslu    xspbllt      xzdblu
xzllt   xzpblas3tim xchrd   xcpblas2tim  xctrd        xdhrd   xdpblas2tim
xdsvd xsgblu  xsnep   xsptllt      xzdtlu  xzls    xzpblas3tst xsgsep xsllt
xdgsep )
  echo ""            |& tee -a $OUT
  echo "-------------------------------->  testing   $exe "      |&  tee -a
$OUT
  echo ""            |&  tee -a $OUT
  mpirun -np 4    ./$exe  |&  tee -a $OUT
  echo ""            |&  tee -a $OUT
end
[TESTING]$
```

Tests results:

The tests *xcinv xdinv xsinv xzgsep xzinv xcgsep xsgsep xdgsep* will end with an address error if the library is compiled using the -O3, -O2 or -O1 options. When the library is compiled using the -O0 option, all the tests except *xzsep* should occur without any problem.

6.9 Blocksolve 95

BlockSolve95 is a scalable parallel software library intended primarily for the solution of sparse linear systems that arise from physical models, especially problems involving multiple degrees of freedom at each node.

Blocksolve 95 uses MPI and thus it is delivered in two releases, corresponding to the two available MPI (see *Overview*, on page6-1).

The library is located in the directory **/usr/lib/BlockSolve95**.

6.9.1 Installing and Compiling Blocksolve 95

This paragraph describes the installation of the BlockSolve95 library, using the MPICH_Ethernet or MPIBull implementation of MPI.

Blocksolve95 is available on the HPC CD.

BLOCKSOLVE95 is installed as follows:

– To use it with MPIBull:

```
$rpm –ivh –-relocate /opt/blocksolve95/ BlockSolve95-mpi_bull-1.6.2-3.0-1
.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet:

```
$rpm –ivh –-relocate /opt/blocksolve95/ BlockSolve95-mpich_ethernet_bull-
1.2.6-3.0-1.Bull.ia64.rpm
```

6.10 SuperLU

SuperLU is a general purpose library for the direct solution of large, sparse, non symmetric systems of linear equations on high performance machines. The library is written in C and can be called from either C or Fortran.

SuperLU is available in three independent versions:

- **SuperLU** for sequential systems (Version 3.0, October 15, 2003). Note that between version 2.0 and version 3.0 the interfaces of some functions have been modified and are incompatible!
- **SuperLU_SMP** for shared memory systems (Version 1.0, September 1999), which uses POSIX threads.
- **SuperLU_DIST** for distributed memory system using MPI (Version 2.0, March 2003, the latest update dates back to October 15, 2003).

6.10.1 SuperLU Serial

Versions 2.0 and 3.0 of the sequential SuperLU Serial are both provided.

These libraries (both named **superlu_ia64.a**) are located in the directories **/usr/include/SuperLU** and **/usr/lib/SuperLU**.

Installation

SUPERLU is available on the HPC CD.

SUPERLU-SEQ is installed as follows:

```
$rpm –ivh –-relocate /opt/SuperLU-SEQ/ SuperLU-SEQ-2.0-2.Bull.ia64.rpm
```

File modifications

Modify the **make.inc** file as follows:

```
PLAT         = _ia64
BLASDEF      = -DUSE_VENDOR_BLAS
BLASLIB      = -L/opt/envhpc/intel/mkl/lib/64 -lmkl_ipf -lmkl_lapack -lguide
TMGLIB       = tmglib$(PLAT).a
SUPERLULIB   = superlu$(PLAT).a
ARCH         = ar
ARCHFLAGS    = cr
RANLIB       = ranlib
CC           = ecc
CFLAGS       = -O3  -mp -w -ftz
FORTRAN      = efc
FFLAGS       = -O3  -mp -w -cm -ftz
LOADER       = ecc
CDEFS        = -DAdd_
```

Update the environment

Make sure that **$PATH** contains the current folder, as well as the INTEL compilers:

```
echo $PATH
export PATH=$PATH:.
```

Set the compilers environment to source **iccvars.sh** and **ifortvars.sh** which are available where the Intel compilers are installed.

Examples

The folder ***EXAMPLE*** contains 21 programs.

Modify the **dlinsolx2.c** file (not necessary with version 2.0)

```
188     dgssvx(&options, &A1, perm_c, perm_r, etree, equed, R, C,
189            &L, &U, work, lwork, &B1, &X, &rpg, &rcond, ferr, berr,
190            &mem_usage, &stat, &info);
```

The codification consists in replacing “&A”, by “&A1,” at line 188

The same modification has to be done in the **clinsolx2, slinsolx2** and **zlinsolx2**files.

Compile the examples:

```
make clean ; make
```

The test is launched using the command line:

```
./EXECUTABLE_NAME <g10
```

This is not the case for **superlu** which does not require a file to be redirected for its standard input.

A shell script that executes all the tests is shown below:

```
$  cat run.sh
   #! /bin/bash
   export exe=superlu
   ./$exe

for suffix in "" "1" "x" "x1" "x2" ;
do
   for exe in `ls *linsol${suffix}` ;
   do
   ./$exe  < g10
   done
done
$
```

6.10.2 SuperLU_SMP

The library (named **superLU_MT_PTHREAD.a**) is found in the directories **/usr/include/SuperLU** and **/usr/lib/SuperLU**.

Installation

SUPERLU-SMP is available on the HPC CD.

SUPERLU-SMP is installed as follows:

```
$rpm –ivh –-relocate /opt/SuperLU-SMP/ SuperLU-SMP-1.0-2.Bull.ia64.rpm
```

Examples

The folder named **EXAMPLE** contains 5 programs.

A shell script that starts all these programs is shown below:

```
#! /bin/bash
        export procs=4

        for mat in g10 g20 big ;
        do
           for exe in ./f77exm ./pdlinsol ./pdlinsolx ./pdrepeat ./pdspmd ;
           do
             $exe -p $procs  < $mat
           done
        done
```

6.10.3 SuperLU_DIST

SuperLU_DIST uses MPI and is delivered in two releases, corresponding to the two available MPI (see *Overview*, on page6-1).

The library (named **superlu_lnx_ia64.a**) is located in the directories **/usr/include/SuperLU** and **/usr/lib/SuperLU**.

Installation

This paragraph describes the installation of the SuperLU-DIST library, using the MPICH_Ethernet or MPIBull implementation of MPI.

SuperLU-DIST is available on the HPC CD.

SuperLU_DIST is installed as follows:

– To use it with MPIBull:

```
$rpm –ivh –-relocate /opt/SuperLU_DIST/ SuperLU_DIST-mpi_bull-1.6.2-2.0-1
.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet:

```
$rpm –ivh –-relocate /opt/SuperLU_DIST/ SuperLU_DIST-mpich_ethernet-1.2.6-
2.0-1.Bull.ia64.rpm
```

Update the environment

Set the compilers environment to source **iccvars.sh** and **ifortvars.sh** which are available where the compilers are installed.

– With MPIBull

Nothing else has to be done

– With MPICH_Ethernet

Set the MPICH_Ethernet environment:

```
$ export MPI_HOME=/opt/mpi/mpich_ethernet-1.2.6/
$ export PATH=$MPI_HOME/bin:$PATH
$ export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
```

Examples

We provide here the listing of a shell script that executes all the examples of the folder **EXAMPLE**:

```
#! /bin/bash

for exe in  pddrive pddrive1 pddrive1_ABglobal pddrive2 pddrive2_ABglobal
       pddrive3 pddrive3_ABglobal pddrive_ABglobal ;
       do mpirun -np 4 ./$exe  -r 2 -c 2 g20.rua   ; done

for exe in  pddrive4_ABglobal pddrive4  ;
       do mpirun -np 10 ./$exe   g20.rua ; done

for exe in  pddrive pddrive1 pddrive1_ABglobal pddrive2 pddrive2_ABglobal
       pddrive3 pddrive3_ABglobal pddrive_ABglobal ;
       do mpirun -np 4 ./$exe  -r 2 -c 2 big.rua ;  done

for exe in  pddrive4_ABglobal pddrive4  ;
       do mpirun -np 10 ./$exe   big.rua   ; done

for exe in  pzdrive pzdrive1 pzdrive1_ABglobal pzdrive2 pzdrive2_ABglobal
       pzdrive3 pzdrive3_ABglobal pzdrive_ABglobal ;
       do mpirun -np 4 ./$exe  -r 2 -c 2 g20.rua ;    done

for exe in  pzdrive4_ABglobal pzdrive4  ;
       do mpirun -np 10 ./$exe g20.rua   ; done
```

A script that launches the tests with MPI_Bull is shown below:

```
cat run.sh
#! /bin/bash

for exe in pddrive pddrive1 pddrive1_ABglobal pddrive2 pddrive2_ABglobal
        pddrive3 pddrive3_ABglobal pddrive_ABglobal ;
        do prun -p parallel -N 2 -n 4 ./$exe  -r 2 -c 2 g20.rua   ; done

for exe in  pddrive4_ABglobal pddrive4  ;
        do prun -p parallel -N 2 -n 10 ./$exe   g20.rua ; done

for exe in  pddrive pddrive1 pddrive1_ABglobal pddrive2 pddrive2_ABglobal
        pddrive3 pddrive3_ABglobal pddrive_ABglobal ;
        do prun -p parallel -N 2 -n 4 ./$exe  -r 2 -c 2 big.rua ;  done

for exe in  pddrive4_ABglobal pddrive4  ;
        do prun -p parallel -N 2 -n 10 ./$exe   big.rua   ; done

for exe in  pzdrive pzdrive1 pzdrive1_ABglobal pzdrive2 pzdrive2_ABglobal
        pzdrive3 pzdrive3_ABglobal pzdrive_ABglobal ;
        do prun -p parallel -N 2 -n 4 ./$exe  -r 2 -c 2 g20.rua ;    done

for exe in  pzdrive4_ABglobal pzdrive4  ;

do prun -p parallel -N 2 -n 10 ./$exe g20.rua   ; done
```

6.11 FFTW

FFTW stands for Fastest Fourier Transform in the West.

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data.

FFTW uses MPI and thus it is delivered in two releases, corresponding to the two available MPI (see *Overview*, on page6-1).

The library is located in the directories **/usr/lib/fftw**, **/usr/include** and **/usr/doc/**.

6.11.1 Installing and compiling FFTW

Download

The archive file can be downloaded at http://www.fftw.org/download.html

FFTW is available on the HPC CD.

Version

Here is a quotation from a note that comes with version 3.0.1 (July, 6th, 2003):

« We haven't yet added MPI parallel transforms to 3.0.1, so you need to use 2.1.5 for these. Version 3.0.1 does include shared-memory/threads parallel transforms, however.»

This paragraph describes the installation of the FFTW library, using MPICH_Ethernet, or the MPIBull implementations of MPI.

FFTW is installed as follows:

– To use it with MPIBull

```
$rpm –ivh –-relocate /opt/fftw/ FFTW-mpi_bull-1.6.2-2.1.5-1.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet

```
$rpm –ivh –-relocate /opt/fftw/ FFTW-mpich_ethernet-1.2.6-2.1.5-
1.Bull.ia64.rpm
```

Update the environment

Set the compilers environment to source **iccvars.sh** and **ifortvars.sh** which are available where the compilers are installed.

– With MPIBull

Nothing else has to be done

– With MPICH_Ethernet

Set the MPICH_Ethernet environment:

```
$ export MPI_HOME=/opt/mpi/mpich_ethernet-1.2.6/
$ export PATH=$MPI_HOME/bin:$PATH
$ export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
```

6.12 PETSc

PETSc stands for Portable, Extensible Toolkit for Scientific Computation.

PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication (see http://www.mcs.anl.gov/mpi for more details).

PETSc uses MPI and is delivered in two releases, corresponding to the two available MPI (see *Overview*, on page6-1).

The library is located in the directory **/usr/lib**.

6.12.1 Installing and Compiling PETSc

This paragraph describes the installation of the PETSc library, using MPICH_Ethernet, or the MPIBull implementations of MPI.

PETSC is available on the HPC CD.

PETSc is installed as follows:

– To use it with MPIBull

```
$rpm –ivh –-relocate opt/PETSc/ PETSc-mpi_bull-1.6.2-2.2.0-2.Bull.ia64.rpm
```

– To use it with MPICH_Ethernet

```
$rpm –ivh –-relocate /opt/PETSc/ PETSc-mpich_ethernet-1.2.6-2.2.0-1
.Bull.ia64.rpm
```

Update the environment

Set the compilers environment to source **iccvars.sh** and **ifortvars.sh** which are available where the compilers are installed.

– With MPIBull

Nothing else has to be done

– With MPICH_Ethernet

Set the MPICH_Ethernet environment:

```
$ export MPI_HOME=/opt/mpi/mpich_ethernet-1.2.6/
$ export PATH=$MPI_HOME/bin:$PATH
$ export LD_LIRARY_PATH=$MPI_HOME/lib:$LD_LIRARY_PATH
```

Tests

The tests are executed by the following commands:

```
make BOPT=O test
make BOPT=O testexamples
```

The X11 display is also tested.

To use MPIBull, the usage of MPIRUN must be modified:

```
for   file in `grep -lr 'MPIRUN} \{1,\}-np'  * `  ; do cp $file $file.orig
;  sed 's/MPIRUN} \{1,\}-np/MPIRUN} -n /' $fil.orige  > $file ; done
```

Results of the tests:

Some tests fail:

```
make[5]: Entering directory
`/home/gutfreud/CANDIDATS_PROCHAINE_LIVRAISON/PETSc/petsc-
2.1.6/src/sys/examples/tests'
2d1
< rank =   0
Error in PetscSetCommWorld.
```

The expected result contains only one line with « rank = 0 » whereas the test is executed on two processes.

As explained in the tests commentary, some tests encounter round up imprecision:

```
testexamples_1 in: petsc-2.1.6/src/snes/examples/tests
2,4c2,4
<   1 SNES Function norm 0.00955879
<   2 SNES Function norm 7.57667e-05
<   3 SNES Function norm 5.48316e-09
---
>   1 SNES Function norm 0.00955878
>   2 SNES Function norm 7.57662e-05
>   3 SNES Function norm 5.48023e-09
Possible problem with ex1_3, diffs above
```

Quoting the test commentary:

```
=========================================
BEGINNING TO COMPILE AND RUN TEST EXAMPLES
Due to different numerical round-off on certain
machines some of the numbers may not match exactly.
=========================================
```

Where to find the library:

The library is located in the **lib/libO/linux64_intel** folder.

6.13 NETCDF

NetCDF (network Common Data Form) allows the management of data input/output.

NetCDF is an interface for array-oriented data access and a library that provides an implementation of the interface. The NetCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data.

The library is located in the **/usr/bin**, **/usr/include**, **/usr/lib** and **/usr/man** directories.

7. Modules Package

7.1 Overview

Environment modules provide a great way to easily customize your shell environment, and can be done on the fly.

For instance an environment may consist of one set of compatible products including defined releases of Fortran compilers, C compilers, debuggers, mathematical libraries. In this way you can easily reproduce the conditions of a trial, or use only proven environments.

The Modules environment is a program that can read and list module files returning commands, suitable for the shell to interpret, and most importantly to "eval". Module files are a kind of flat database using files.

In Unix a child process cannot modify its parent environment. How does Modules do this then? Modules parses the given modules file and produces the appropriate shell commands to set/unset/append/un-append onto an environment variable. These commands are eval'd by the shell. Each shell providesa mechanism where commands can be executed and the resulting output can, in turn, be executed as shell commands. In the C-shell & Bourne shell and derivatives this is "eval". This is the only way for a child process to modify the parent's (login shell) environment. Hence the module command itself is a shell alias or function that performs these operations. To the user, it looks just like any other command.

This chapter consists of two parts:

- Paragraph 7.2 shows how modules are used with an example.
- Paragraphs 7.3 to 7.8 detail the Modules commands.

For more details, refer to http://modules.sourceforge.net/ .

7.2 Module use example

To display the list of available modules for a cluster enter the following command:

```
module avail
```

The output is similar to the following:

```
------------------------ /opt/modules/version -----------------------
3.1.6

------------------- /opt/modules/3.1.6/modulefiles -------------------
dot         module-info null
module-cvs  modules     use.own

---------------------- /opt/modules/modulefiles ----------------------
configuration/1              intel_fc/8.0.046
configuration/2(default)     intel_fc/8.0.049(default)
configuration/3              intel_fc/8.1.019
intel_cc/8.0.066             intel_idb/8.1.3(default)
intel_cc/8.0.069             intel_mkl/7.0.017(default)
intel_cc/8.0.071(default)    oscar-modules/1.0.3(default)
intel_cc/8.1.022             vampir/4.0.2(default)
```

The available user modules are listed after the **/opt/modules/modulefiles** line.

To load a module use the following command:

```
module load module_name
```

To check the list of loaded modules use the following command

```
module list
```

The **module avail** command shows that some modules are marked as *(default)*. These modules are loaded if the user does not specify the module version number. For example the following commands are the same:

```
module load configuration
```

is equivalent to:

```
module load configuration/2
```

Three configurations are created. These configurations are modules that load other modules automatically.

For example configuration number 2 includes:
- Intel Fortran compiler version 8.0.049
- Intel C compiler version 8.0.071
- Intel debugger version 8.1.3
- MKL version 7.0.017

Note that the **module load** command adds the configuration name to the user prompt (in the module configuration context).

The **module unload** command unloads the module.

```
module unload
```

The **module purge** command clears all the modules from the environment.

```
module purge
```

It is not possible to load simultaneously two "configuration" modules. Loading a "configuration" module unloads the previous one.

It is not possible to load two **intel_cc** or **intel_fc** modules at the same time because it causes conflicts.

7.3 Setting Up the Shell RC Files

This section is a quick tutorial on Shell rc (run-command) files. When a user logs in and if they have **/bin/csh** (**/bin/sh**) as their shell, the first rc file to be parsed by the shell is **/etc/csh.login** & **/etc/csh.cshrc** (**/etc/profile**) (the order is implementation dependent), and then the user's **$HOME/.cshrc** (**$HOME/.kshenv**) and finally **$HOME/.login** (**$HOME/.profile**).
All the other login shells are based on **/bin/csh** and **/bin/sh** with additional features and rc files.
Certain environment variables and aliases (functions) need to be set for Modules to work correctly.
This is handled by the Module init files in **/usr/local/Modules/default/init**, which contains separate init files for each of the various supported shells, where the default is a symbolic link to a module command version.

Global Shell RC Files

Since Modules sets & appends to several environment variables it's a good idea to set them up with the default system values, otherwise the compilers, loaders, man pages, etc may not find the default system paths unless they are listed in the appropriate environment variables.
Look in **./etc/global** for example rc files (**csh.login**, **profile**) which define most of the variables likely to be required. These files are not tailored for all platforms, but are targeted at GNU/Linux systems.

The Modules specific commands are located in **./etc/global** and are named **csh.modules** and **profile.modules**.
These files should be copied to **/etc** or wherever else has been specified with the **--with-etc-path=<path>** option with the configuration script. These files will be sourced from the users' .login and .profile.
Edit these files in order that certain modules will be loaded automatically by all users.
Otherwise it will load only "null" which does nothing.

Skeleton Shell RC ("Dot") Files

If there is not time to set them up individually then the skeleton files will provide a "default" environment when new users are added to the system.
The files are usually placed in **/etc/skel** or wherever else has been specified with the **--with-skel-path=<path>** option with the configuration script and contain a minimal set of "dot" files and directories that every new user should start with.
The skeleton files are copied to the new user's **$HOME** directory using the **-m** option with the **useradd** command.
A set of example "dot" files are located in **./etc/skel** .
Copy everything but the .*.in and CVS files and directories to the skeleton directory.
Edit and tailor according to the system.
If there is a pre-existing set of skeleton files is available, then make sure the following minimum set exists: .cshrc, .login, .kshenv, .profile . They can be automatically updated with: env **HOME=/etc/skel /usr/local/Modules/default/bin/add.modules**.
Inspect the new "dot" files and if OK, then remove all the .*.old (original) files.
An alternative way of setting-up the users' dot files can be found in **./ext**.
This model can be used with the **--with-dot-ext** configure option.

User Shell RC ("Dot") Files

The final step for a functioning Modules environment is to modify the user "dot" files to source the right files. One way to do this is to put a message in the **/etc/motd** telling each user to run the command:

```
/usr/local/Modules/default/bin/add.modules
```

This is a script that parses their existing "dot" files pre-pending the appropriate commands to initialize the Modules environment.
The user can re-run this script which will find and remember the modules they have initially loaded, strip out the previous module initialization and restore it with an upgraded one.
If the user lacks a necessary "dot" file, then the script will copy one over from the skeleton directory.
The user will have to logout and login for it to come into effect.
Another way is for the system administrator to **su - username** to each user and run it interactively.
The process can be semi-automated with a single line command that obviates the need for direct interaction:

```
su - username -c "yes | /usr/local/Modules/default/bin/add.modules"
```

Power users can create a script to directly parse the **/etc/passwd** file to perform this command.

Otherwise just copy the **passwd** file and edit it appropriately to execute this command for each valid user.

7.4 Module Files

Once the above steps have been performed, it is important to have module files in each of the module file directories.
For example, the installation will install the following module files:

```
--------- /usr/local/Modules/3.0.9-rko/modulefiles ----------
dot         module-info modules     null        use.own
```

If the user does not have module files in **/usr/local/Modules/modulefiles** then copy "null" for that directory.
On some systems an empty modulefile directory will cause a core dump, on other systems no problem at all.
Use **`/usr/local/Modules/default/modulefiles/modules`** as a template for creating your own module files.
For more information enter:

```
module load modules
```

This will provide access to the module(1) modulefile(4) man pages, as well as the versions directory.
Study the man pages carefully.
The version directory may look something like this:

```
---------------- /usr/local/Modules/versions ----------------
3.0.5-rko 3.0.6-rko 3.0.7-rko 3.0.8-rko 3.0.9-rko
```

The model you should use for modulefiles is "name/version".
For example the **/usr/local/Modules/modulefiles** directory may have a directory named "netscape" which contains the following module files: 301, 405c, 451c, etc.
When it's displayed with **module avail**, it looks something like this:

```
netscape/301
netscape/405c
netscape/451c(default)
netscape/45c
netscape/46
```

The default is established with .version file in the netscape directory and it looks something like this:

```
#%Module1.0#####################################################
#########
##
## version file for Netscape
##
set ModulesVersion      "451c"
```

If the user enters **module load netscape**, then the default netscape/451c will be used. The default can be changed instantly by editing the .version file to point to a different module file in that directory.
If no .version file exists then Modules will just assume the last module in the alphabetical ordered directory listing as the default.

7.5 Package Location Suggestions

To make Modules a useful tool in your environment, it's a good idea to use some discipline and this may require some work in placing binaries and man pages in suitable locations. Using NFS is a convenient way to scatter modules database and more tools on all cluster nodes.It's a way to stop using **/usr/local/bin** as a catch-all dump for every miscellaneous binary, especially the ones that are not used very often.

There are some scripts to help this along. For this discussion the mythical "foobar" package will be used. The sources are down-loaded in a form of a gzip'd tar file - foobar-1.2.3.tar.gz . Most sources can be placed anywhere, and in most cases will configure and build without any problems. For this example everything will be done from /tmp.

The sources will be unloaded with "tar -xzf foobar-1.2.3.tar.gz" which creates a directory in the current working directory (/tmp) named ./foobar-1.2.3 .
Cd to this directory and run the configuration script

```
./configure --prefix=/usr/local/pkg/foobar/1.2.3
```

This should configure the source to place all necessary files to that location.
Continue the build, typically with:

```
make
make check
make install
```

The binaries, libraries, man pages, and info pages are now placed in /usr/local/pkg/foobar/1.2.3. If you need to create the "root" yourself, then load the "modules" module with "module load modules". Use the **mkroot -m**' script to

create a collection of ./bin, ./etc, ./lib, ./man/ directories. Install the items as needed, then use "mkroot -c" to clean out the empty directories.

Finally, after installing the binaries, etc. create a module file using another module file as a template and place it somewhere in the modulefile hierarchy. Also be sure to keep your original source files somewhere.

7.6 Upgrading Module Commands

The theory is that Modules should use a similar package/version locality as the package environments it helps to define. Switching between versions of the module command should be as easy as switching between different packages via the module command. Suppose there is a change from 3.0.5-rko to version 3.0.6-rko. The goal is to semi-automate the changes to the user "dot" files so that the user is oblivious to the change.

The first step is to install the new module command & files to **/usr/local/Modules/3.0.6-rko/.** Test it out by loading with "module load modules 3.0.6-rko". You may get an error like: 3.0.6-rko(25):ERROR:152: Module 'modules' is currently not loaded This is OK and is likely to go away with future versions.

Make sure the new version is loaded using "module --version". If it seems stable enough then advertise it to the more adventurous users. Once you are satisfied that it works adequately then go into /usr/local/Modules and remove the old "default" symbolic link to the new versions.
For example:

```
cd /usr/local/Modules
rm default; ln -s 3.0.6-rko default
```

This new version is now the default, and will be referenced by all the users that log in and by those that haven't loaded a specific module command version.

7.7 The Module Command

Synopsis

```
module [ switches ] [ sub-command ] [ sub-command-args ]
```

7.7.1 Description

The Module command is a user interface to the Modules package. The Modules package provides dynamic modification’s of the user's environment via *modulefiles*.

Each *modulefile* contains the information needed to configure the shell for an application. Once the Modules package is initialized, the environment can be modified on a per-module basis using the module command which interprets *modulefiles*. Typically *modulefiles* instruct the **module** command to alter or set shell environment variables such as PATH, MANPATH, etc. *modulefiles* may be shared by many users on a system and users may have their own collection to supplement or replace the shared *modulefiles*.

The *modulefiles* are added to and removed from the current environment by the user. The environment changes contained in a *modulefile* can be summarized through the module command as well. If no arguments are given, a summary of the module usage and sub-commands are shown.

The action for the module command to take is described by the sub-command and its associated arguments.

7.7.2 Package Initialization

The Modules package and the module command are initialized when a shell-specific initialization script is sourced into the shell. The script creates the module command as either an alias or a function, creates Modules environment variables, and a snapshot of the environment is saved in **${HOME }/.modulesbeginenv** . The module alias or function executes the **modulecmd** program located in **${MODULESHOME }/bin** and has the shell evaluate the command's output. The first argument to **modulecmd** specifies the type of shell.

The initialization scripts are kept in **${MODULESHOME }/init/shellname** where shellname is the name of the sourcing shell. For example, a C Shell user sources the **${MODULESHOME }/init/csh** script. The sh, csh, tcsh, bash, ksh, and zsh shells are supported by **modulecmd**. In addition, python and perl "shells" are supported and these write the environment changes to stdout as python or perl code.

7.7.3 Examples of initialization

In the following examples, replace ${MODULESHOME } with the actual directory name.

```
C Shell initialization (and derivatives):
source ${MODULESHOME }/init/csh
module load modulefile modulefile ...
```

```
Bourne Shell (sh) (and derivatives):
. ${MODULESHOME }/init/sh
module load modulefile modulefile ...
```

```
Perl:
require "${MODULESHOME }/init/perl";
&module("load modulefile modulefile ...");
```

7.7.4 Modulecmd startup

Upon invocation **modulecmd** sources rc files which contain global, user and *modulefile* specific setups. These files are interpreted as *modulefiles*. See *modulefile* (4) for detailed information.

Upon invocation **of modulecmd** module RC files are sourced in the following order:

- Global RC file as specified by ${MODULERCFILE } or ${MODULESHOME }/etc/rc
- User specific module RC file ${HOME }/.modulerc
- All .modulerc and .version files found during *modulefile* seeking.

7.7.5 Command line switches

The module command accepts command line switches as its first parameter. These may be used to control output format of all the information displayed and the module behavior in case of locating and interpreting module files.

All switches may be entered either in short or long notation. The following switches are accepted:

--force, -f
Force active dependency resolution. This will result in modules found with a **prereq** command inside a module file being loaded automatically. Unloading module files using this switch will result in all required modules which have been loaded automatically using the **-f** switch being unloaded. This switch is experimental at the moment.

--terse, -t
Display **avail** and **list** output in short format.

--long, -l
Display **avail** and **list** output in long format.

--human, -h
Display short output of the **avail** and **list** commands in human readable format.

--verbose, -v
Enable verbose messages during module command execution.

--silent, -s
Disable verbose messages. Redirect **stderr** to **/dev/null** if **stderr** is found not to be a **tty**. This is a useful option for module commands being written into .**cshrc** , .login or .profile files, because some remote shells (as **rsh** (1)) and remote execution commands (like **rdist**) get confused if there is output to **stderr**.

--create, -c
Create caches for **module avail** and **module apropos** . You must be granted write access to the ${MODULEHOME }/*modulefiles*/ directory if you try to invoke module with the **-c** option.

--icase, -i
Case insensitive module parameter evaluation. Currently only implemented for the module apropos command.

--userlvl <lvl>, -u <lvl>
Set the user level to the specified value. The argument of this option may be one of:
novice, nov Novice
expert, exp Experienced module user
advanced, adv Advanced module user

7.7.6 Module Sub-Commands

```
help [modulefile...]
```

Print the usage of each sub-command. If an argument is given, print the Module specific help information for the *modulefile*.

```
load modulefile [modulefile...]
```

```
add modulefile [modulefile...]
```

Load *modulefile* into the shell environment.

```
unload modulefile [modulefile...]
```

```
rm modulefile [modulefile...]
```

Remove *modulefile* from the shell environment.

```
switch modulefile1 modulefile2
```

```
swap modulefile1 modulefile2
```

Switch loaded *modulefile*1 with *modulefile*2.

```
display modulefile [modulefile...]
```

```
show modulefile [modulefile...]
```

Display information about a *modulefile*. The display sub-command will list the full path of the *modulefile* and all (or most) of the environment changes the *modulefile* will make if loaded. It will not display any environment changes found within conditional statements.

```
list
```

List loaded modules.

```
avail [path...]
```

List all available *modulefiles* in the current MODULEPATH . All directories in the MODULEPATH are recursively searched for files containing the *modulefile* magic cookie. If an argument is given, then each directory in the MODULEPATH is searched for *modulefiles* whose pathname match the argument. Multiple versions of an application can be supported by creating a subdirectory for the application containing *modulefiles* for each version.

```
use directory [directory...]
```

```
use [-a|--append] directory [directory...]
```

Prepend directory to the MODULEPATH environment variable. The --append flag will append the directory to MODULEPATH .

```
unuse directory [directory...]
```

Remove directory from the MODULEPATH environment variable.

```
update
```

Attempt to reload all loaded *modulefiles*. The environment will be reconfigured to match the saved **${HOME }/.modulesbeginenv** and the *modulefiles* will be reloaded. **update** will only change the environment variables that the *modulefiles* set.

```
clear
```

Force the Modules Package to believe that no modules are currently loaded.

```
purge
```

Unload all loaded *modulefiles*.

```
whatis [modulefile [modulefile...]]
```

Display the *modulefile* information set up by the module-whatis commands inside the specified *modulefiles*. If no *modulefiles* are specified all whatis information lines will be shown.

```
apropos string
```

```
keyword string
```

Seeks through the **whatis** information of all *modulefiles* for the specified string. All module **whatis** information matching the string search will be displayed.

```
initadd modulefile [modulefile...]
```

Add *modulefile* to the shell's initialization file in the user's home directory. The startup files checked are **.cshrc**, .**login**, and **.csh**_variables for the C Shell; .profile for the Bourne and Korn Shells; .**bashrc**, **.bash_env**, and **.bash_profile** for the GNU Bourne Again Shell; **.zshrc**, **.zshenv**, and **.zlogin** for zsh. The .modules file is checked for all shells. If a 'module load' line is found in any of these files, the *modulefile*(s) is(are) appended to any existing list of *modulefiles*. The 'module load' line must be located in at least one of the files listed above for any of the 'init' sub-

commands to work properly. If the 'module load' line is found in multiple shell initialization files, all of the lines are changed.

```
initprepend modulefile [modulefile...]
```

Does the same as **initadd** but prepends the given modules to the beginning of the list**. initrm** *modulefile* [*modulefile*...] Remove *modulefile* from the shell's initialization files.

```
initswitch modulefile1 modulefile2
```

Switch *modulefile*1 with *modulefile*2 in the shell's initialization files.

```
initlist
```

List all of the *modulefiles* loaded from the shell's initialization file.

```
initclear
```

Clear all of the *modulefiles* from the shell's initialization files.

7.7.7 Environment

Environment variables are unset when unloading a *modulefile*. Thus, it is possible to load a *modulefile* and then unload it without having the environment variables return to their prior state.

MODULESHOME

The location of the master Modules package file directory containing module command initialization scripts, the executable program modulecmd, and a directory containing a collection of master *modulefiles*.

MODULEPATH

The path that the module command searches when looking for *modulefiles*. Typically, it is set to the master *modulefiles* directory, ${MODULESHOME}/*modulefiles*, by the initialization script. MODULEPATH can be set using 'module use' or by the module initialization script to search group or personal *modulefile* directories before or after the master *modulefile* directory.

LOADEDMODULES

A colon separated list of all loaded *modulefiles*.

LOADED_*MODULEFILES*

A colon separated list of the full pathname for all loaded *modulefiles*.

MODULESBEGINENV

The filename of the file containing the initialization environment snapshot.

Files

/opt

The **MODULESHOME** directory:

${MODULESHOME}/etc/rc

The system-wide modules rc file. The location of this file can be changed using the MODULERCFILE environment variable as described above.

${HOME}/.modulerc

The user specific modules rc file.

${MODULESHOME}/*modulefiles*

The directory for system-wide *modulefiles*. The location of the directory can be changed using the MODULEPATH environment variable as described above.

${MODULESHOME}/bin/modulecmd

The *modulefile* interpreter that is executed upon each invocation of module.

${MODULESHOME}/init/shellname

The Modules package initialization file sourced into the user's environment.

${MODULESHOME}/init/.modulespath

The initial search path setup for module files. This file is read by all shell init files.

${MODULEPATH}/.moduleavailcache

File containing the cached list of all *modulefiles* for each directory in the MODULEPATH (only when the avail cache is enabled).

${MODULEPATH}/.moduleavailcachedir

File containing the names and modification times for all sub-directories with an avail cache.

${HOME}/.modulesbeginenv

A snapshot of the user's environment taken at Module initialization. This information is used by the module update sub-command.

7.8 Modulefile

modulefile - files containing **Tcl** code for the Modules package

modulefiles are written in the Tool Command Language, Tcl(3) and are interpreted by the **modulecmd** program via the module(1) user interface**. modulefiles** can be loaded, unloaded, or switched on-the-fly while the user is working.

A modulefile begins with the magic cookie, '#%Module'. A version number may be placed after this string. The version number is useful as the format of **modulefiles** may change. If a version number doesn't exist, then **modulecmd** will assume the modulefile is compatible with the latest version. The current version for **modulefiles** is 1.0. Files without the magic cookie will not be interpreted by **modulecmd.**

Each modulefile contains the changes to a user's environment needed to access an application. Tcl is a simple programming language which allows modulefiles to be arbitrarily complex, depending upon the application's and the modulefile writer's needs. If support for extended tcl (tclX) has been configured for your installation of modules, you may also use all the extended commands provided by tclX. **modulefiles** can be used to implement site policies regarding the access and use of applications.

A typical **modulefiles** is a simple bit of code that sets or adds entries to the PATH, MANPATH, or other environment variables. **Tcl** has conditional statements that are evaluated when the modulefile is loaded. This is very effective for managing path or environment changes due to different OS releases or architectures. The user environment information is encapsulated into a single modulefile kept in a central location. The same modulefile is used by every user on any machine. So, from the user's perspective, starting an application is exactly the same regardless of the machine or platform they are on.

modulefiles also hide the notion of different types of shells. From the user's perspective, changing the environment for one shell looks exactly the same as changing the environment for another shell. This is useful for new or novice users and eliminates the need for statements such as "if you're using the C Shell do this ..., otherwise if you're using the Bourne shell do this ..." Announcing and accessing new software is uniform and independent of the user's shell. From the modulefile writer's perspective, this means one set of information will take care of all types of shell.

8. Batch Management (TORQUE)

8.1 Overview

TORQUE is a resource manager providing control over batch jobs and distributed compute nodes. TORQUE uses a queue mechanism for job execution, which works according to preconfigured priority criteria.

The user command interface for TORQUE can be use to:

- Submit a job
- Display the state and characteristics of a job
- Cancel a job
- Change the characteristics of a waiting or running job. Note that for a running job, only the limits and the output files can be changed.
- Stop or resume a job.
- Manage more than 5000 active or waiting jobs.

For more information refer to the following Web site:
http://www.clusterresources.com/products/torque/ .

The main features of TORQUE are.

Job Priority

Users can specify the priority of their jobs.

Job-Interdependency

TORQUE enables the user to define a wide range of interdependencies between batch jobs. Such dependencies include: execution order, synchronization, and execution dependent on the success or failure of another specified other job.

Automatic File Staging

TORQUE provides users with the ability to specify files that need to be copied onto the execution host before the job runs, and those that need to be copied off after the job completes. The job will be scheduled to run only after the required files have been successfully transferred.

Single or Multiple Queue Support

TORQUE can be configured with as many queues as you wish. However, TORQUE is not limited to queue-based scheduling, and therefore you can run TORQUE with a single queue.

Multiple Scheduling Algorithms

With TORQUE you can select the standard *first-in, first-out* scheduling, or a more sophisticated scheduling algorithm.

8.2 TORQUE Commands

Following is a list of the most common TORQUE commands.

Command	Description
momctl	Manage/diagnose MOM (node execution) daemon
pbsdsh	Launch tasks within a parallel job
pbsnodes	View/modify batch status of compute nodes
qdel	Delete/cancel batch jobs
qhold	Hold batch jobs
qmgr	Manage policies and other batch configurations
qrls	Release batch job holds
qrun	Start a batch job
qsub	Submit jobs
qterm	Shutdown pbs server daemon

qsub Command

Following is a short description of the qsub command. See the **qsub** man page for more details:

```
qsub - submit pbs job
```

SYNOPSIS

qsub [-a date_time] [-A account_string] [-c interval] [-C directive_prefix] [-e path] [-h] [-I] [-j join] [-k keep] [-l resource_list] [-m mail_options] [-M user_list] [-N name] [-o path] [-p priority] [-q destination] [-r c] [-S path_list] [-u user_list] [-v variable_list] [-V] [-W additional_attributes] [-z] [script]

DESCRIPTION

To create a job is to submit an executable script to a batch server. The batch server will be the default server unless the **-q** option is specified. See discussion of PBS_DEFAULT under Environment Variables below. Typically, the script is a shell script which will be executed by a command shell such as sh or csh.

Options for the **qsub** command allow the specification of attributes which affect the behavior of the job.

The **qsub** command will pass on certain environment variables in the Variable_List attribute of the job. These variables will be available to the job. The value for the following variables will be taken from the environment of the **qsub** command: HOME, LANG, LOGNAME, PATH, MAIL, SHELL, and TZ. These values will be assigned to a new name which is the current name prefixed with the string "PBS_O_". For example, the job will have access to an environment variable named PBS_O_HOME which have the value of the variable HOME in the **qsub** command environment.

In addition to the above, the following environment variables will be available to the batch job.

PBS_O_HOST

The name of the host on which the qsub command is running.

PBS_O_QUEUE

The name of the original queue to which the job was submitted.

PBS_O_WORKDIR

The absolute path of the current working directory of the **qsub** command.

PBS_ENVIRONMENT

Set to PBS_BATCH to indicate the job is a batch job, or to PBS_INTERACTIVE to indicate the job is a PBS interactive job, see -I option.

PBS_JOBID

The job identifier assigned to the job by the batch system.

PBS_JOBNAME

The job name supplied by the user.

PBS_NODEFILE

The name of the file containing the list of nodes assigned to the job (for parallel **and cluster systems).**

PBS_QUEUE

The name of the queue from which the job is executed.

9. Performance Tools

This chapter describes the performance tools for optimizing your applications. For information related to the performances of the cluster itself, please refer to *HPC Administrator's Guide*.

9.1 Steps for Optimizing HPC Performance

What will interest the user who wants to improve performance, is to optimize the use of all the resources of the system and to track down any possible bottlenecks resulting from the development, compilation and execution of individual parts, or from the whole application program. Various tools are available to help the user in these tasks.

The first measurement step is to determine the run-time for a specific aspect of the program. The **time** command is used to measure this.

Secondly, it is important to examine the factors which determined this time. Which resources were used and for how long? Can saturated resources be identified or alternatively those which are underutilized.

To do this different methods exist, according to the type of program that is being analyzed and also according to the objectives of the user. For example, is the goal to optimize the behavior of the machine for a given program (benchmark), or to improve the operation of the program itself on a particular machine or network?

The recommended approach is to use a command that shows the evolution of the program live. You can use the **top** or **top_pfb** commands, which give a typology of the program. But, if you consider that it is important to log this information all along the program, you need a tool such as **PerfWare**.

If there is no Input/Output problem, then the quality of algorithms should be analyzed using a profiling approach which focuses on the parts of the program which consume most system resources.

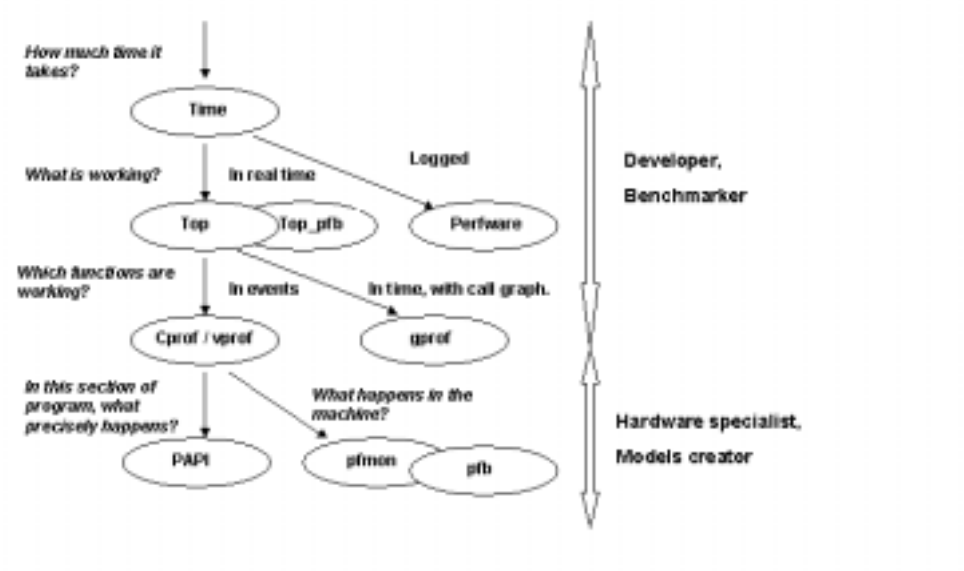
Gprof provides full information about the time taken including that used by sub-programs and functions. Using **cprof** or **vprof** it is possible to report precise

information about multiple hardware events: "miss" rate, "stall" rate, number of floating point operations, and so on.

If a program uses **MPI** (Message Passing Interface) code, each process can be analyzed separately.

If your objective is to optimize a program, the level of details provided by these tools is generally enough. On the other hand, if you need more information about the machine, you will have to run more hardware-oriented tools, which allow getting good metrics. These tools are **pfmon**, which can be used either on a full program or on a full system and **PAPI**, which is used on a section of source code.

The following figure helps to choose which tool is best adapted to your needs.



If the need is to focus on the performance of the parallel applications that use the **MPI** (Message Passing Interface) then either **Profilecomm** or proprietary software such as **Intel Trace Analyzer / Collector tools** can be used. These tools display trace information graphically.

9.2 System Monitoring Tools

9.2.1 top

top is a Unix command providing different types of information, in real time, about the processes execution and the use of a node.

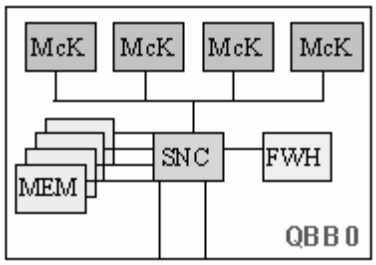
The **top** command can provide:

- A list of launched processes.
- Machine time spent by the processes, in decreasing order (those that consume most are displayed first). This information allows the processes that use most resources to be detected, and the life of the system to be observed.
- Used memory, swap percentage used and so on.

9.2.2 top_pfb

top_pfb is a variant of **top**, dedicated to the NovaScale architecture. It provides supplemental information about the memory bandwidth and the memory localization rate for each QBB (Quad Brick Block, quadri-processors card). It is based on the SNC (Scalability Node Controller), which is a specific chip for memory access.

The following figure shows a NovaScale QBB:



top_pfb does not replace **top**, but completes it.

9.2.3 PerfWare

Unlike **top**, which records information in real time, **PerfWare** collects information and logs it during a defined period of time, in order to create statistics and graphs later.

PerfWare is a tool intended to control the systems in production and therefore is as less intrusive as possible. It is well designed for collecting information on clusters, remotely.

9.3 Application Profiling Tools

9.3.1 gprof

gprof is a tool supplied with GNU compilers. It enables sequential and parallel programs to be traced. **gprof** helps to establish the time used by each function, in percentage and in seconds. It also provides the number of calls for each function. **gprof** produces a graphic of the calls, which describes "who calls what", and how many times.

9.3.2 cprof

cprof is designed to search the sections of programs that consume some specific events: the interruption of profiling (such as **gprof** does), but also high-level hardware events, as managed by PAPI.

9.3.3 profilecomm

profilecomm is a profiling tool dedicated to MPI applications. It has been designed to be:

- light: it uses few resources, in order to not slow down the application.
- easy to run: its objective is to characterize the MPI communications in a program. It allows communication matrixes to be built. Profilecomm is a “post-mortem” tool, which does not allow on-line monitoring.

Data is collected as long as the program is running.

This profiling tool includes:

- An **mpiprofile** interface
- A **profilecomm** library
- **The readpfc** command.

mpiprofile is a generic interface for profiling, added to the **MPIBull** library. It allows dynamic loading of profiling libraries such as the **profilecomm** library

profilecomm is a library designed for profiling MPI communications. It uses mpiprofile. Data collected is written in a file for future analysis.

readpfc is a tool (command line interface), which allows collected data to be handled. Its main features are the following:

- Displaying collected data
- Exporting communication matrix in a format that can be used by other applications.

Profiled Functions

The **profilecomm** library profiles only the communication functions that issue messages. The library ignores the functions that exclusively receive information. The profiled functions are the following ones:

`MPI_Allgather`, `MPI_Allgatherv`, `MPI_Allreduce`, `MPI_Alltoall`, `MPI_Alltoallv`, `MPI_Bcast`, `MPI_Bsend`, `MPI_Gather`, `MPI_Gatherv`, `MPI_Ibsend`, `MPI_Irsend`, `MPI_Isend`, `MPI_Issend`, `MPI_Reduce`, `MPI_Reduce_scatter`, `MPI_Rsend`, `MPI_Scan`, `MPI_Scatter`, `MPI_Scatterv`, `MPI_Send`, `MPI_Sendrecv`, `MPI_Sendrecv_replace`, `MPI_Ssend` and `MPI_Start`.

Some not typically communication functions may call communication functions to make their treatment For example, `MPI_Comm_split` calls `MPI_Allreduce`. In this case, these hidden calls are registered in the communication matrixes and in the call table. But when a profiled function calls another function, the call is not registered, so that the message in not counted several times. Generally the number and the size of the hidden messages is not significant.

Collected Data

The **profilecomm** library collects the following information:

- communication matrixes,
- execution time,
- table of calls of MPI functions,
- topology of the execution environment.

Communication Matrixes

The **profilecomm** library collects apart the point to point communications and the collective communications. It also collects the number of messages and the volume that the sender and receiver have exchanged. Finally, the library builds 4 communication matrixes:

- communication matrix of the number of point to point messages
- communication matrix of the volume (in bytes) of point to point messages
- communication matrix of the number of collective messages
- communication matrix of the volume(in bytes) of collective messages

The messages are those sent by the sender and receiver. The line number corresponds to the sender rank and the column number corresponds to the receiver rank. The volume indicates only the useful part of the messages.

In a point to point communication, the sender and receiver of each message is clearly identified, this means a well defined position in the communication matrix.

In a collective communication, the initial sender(s) and final receiver(s) are identified, but the path of the message is unknown. The **profilecomm** library disregards the real path of the messages. A collective communication is shown as a set of messages sent directly by the initial sender(s) to the final receiver(s).

The following table indicates the sender and receiver process for each collective communication function.

Function	Sender	Receiver
MPI_Allgather	All processes	All the processes of the communicator.
MPI_Allgatherv	All processes	All the processes of the communicator.
MPI_Allreduce	All processes	All the processes of the communicator except itself for the null rank process. The null rank process for the others.
MPI_Alltoall	All processes	All the processes of the communicator.
MPI_Alltoallv	All processes	All the processes of the communicator.
MPI_Bcast	Root	All the processes of the communicator except the root process.
MPI_Gather	All processes	The root process.
MPI_Gatherv	All processes	The root process.
MPI_Reduce	All processes except root	The root process.
MPI_Reduce_scatter	All processes	All the processes of the communicator except itself for the null rank process. The null rank process for the others.
MPI_Scan	All processes	Itself and the next rank process, if it exists.
MPI_Scatter	Root	All the processes of the communicator.
MPI_Scatterv	Root	All the processes of the communicator.

Execution Time

The measured execution time is the maximum of time interval between the calls to **MPI_Init** and **MPI_Finalize** on all the processes. By default the processes are synchronized during the measurements. You can by-pass the synchronization using an option of the **profilecomm** library.

Call Table

The call table contains the number of calls of each profiled function for each process. For the collective communications, since a call generates an unknown number of messages, the values indicated in the call table don’t correspond to the number of messages.

Special cases:

- When the receiver of a message is the **MPI_PROC_NULL** task, no information is registered neither in the call table nor in the communication matrixes.
- A call to **MPI_Start** increments the call table only in the case of a persistent *send*.

Topology of the Execution Environment

The **profilecomm** library registers the topology of the execution environment, which enables to know on which machine and CPU each process is running, and above all to identify the intra- and inter-machine communications.

The library collects the number of machines on which the profiled application is running (`N`) an the list of **hostid, cpuid** couples, where **hostid** is the machine identifier (value between 0 and N-1) and **cpuid** is the number of the CPU assigned to a process by cpuset. This value is significant only if the processes are assigned only to one CPU, which is the default case with **MPI_Bull**. In the other case, **cpuid** is set to `-1`.

9.3.4 Hardware Information Collection Tools

To understand and evaluate the following items, it is necessary to collect measurements at the hardware level:

- What really happens in the machine (for CPU, cache, bus, etc.)
- In which parts performance is lost
- What are the causes of the possible blocks
- What would be the performance using different parameters such as frequency, cache ... (modeling).

This information interests the hardware designers greatly; it is also useful to software developers, who can improve the loop processing so that resources (cache memory for example) are best used.

9.3.5 Events known from Itanium® 2

Itanium® 2 processors record around 490 events. The events meaning is described in the following manual: *Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization*. To make the most of these events a good knowledge of the internal mechanisms of the machine is required. It is recommended to get help from an expert.

The most notable events are the following ones:

- CPU events:

  - CPU_CYCLE
  - IA64_INST_RETIRED
  - FP_OPS_RETIRED
  - NOPS_RETIRED
- Cache miss:
  - L3_MISSES
- Bus occupation (6.4 GB/s, 200MHz):
  - BUS_ALL_ANY
  - BUS_DATA_CYCLE
- Blocks of pipelines (mainly due to waiting for data):
  - BACK_END_BUBBLE_ALL

9.3.6 pfmon

pfmon is the tool that allows you to work with the hardware counters at user command level.

The **pfmon** command may be executed on a program, taking into account, its descendants. It returns from 1 to 4 known events or counters.

pfmon is quite easy to use. Just run **pfmon** specifying the executable you want to survey. You can also study the system as a whole for an interval of time, using the **system wide** option.

9.3.7 Pfb

Pfb is a complementary tool of **perfmon** developed by Bull to access some NovaScale registers: SNC ISPS and BSPS.

Pfb is included in **top_pfb**.

9.3.8 PAPI

PAPI (Performance API) is used for the following purposes:

- To provide a solid foundation for cross-platform performance analysis tools
- To present a set of standard definitions for performance metrics on all platforms
- To provide a standard API among users, vendors and academics.

PAPI supplies two interfaces:

- A high-level interface, for simple measurements
- A low-level interface, programmable, adapting the specificity of the machine and linking the measures.

PAPI should only be used by specialists interested in the optimization of scientific programs. These specialists can focus on code sequences that they can investigate with PAPI functions.

9.4 Using performance tools

9.4.1 Using Top

top is a powerful tool with many parameters, allowing you to control the use of resources (CPU, memory, disks, network, localization and migration of applications). It provides a global view instantly and in real time.

Bull has improved **top** to take into account I/Os, disks and network resources.

top is the first tool you should use to quickly analyze how the resources of a machine are working. **top** has to be launched while the program to measure is running in parallel.

Below is an example of **top -LEX** output:

```
top - 18:08:48 up 1 day,  4:43, 10 users,  load average: 1.63, 0.82, 0.64
Tasks: 127 total,   3 running, 124 sleeping,   0 stopped,   0 zombie
 Cpu0 : 76.1% us,  9.5% sy,  0.0% ni, 14.4% id,  0.0% wa,  0.0% hi,  0.0% si
 Cpu1 : 20.5% us,  2.7% sy,  0.0% ni, 76.7% id,  0.1% wa,  0.0% hi,  0.0% si
 Cpu2 : 57.6% us,  3.6% sy,  0.0% ni, 38.7% id,  0.0% wa,  0.0% hi,  0.0% si
 Cpu3 : 27.5% us,  2.8% sy,  0.0% ni, 69.7% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:   8258080k total,  4273216k used,  3984864k free,   740544k buffers
Swap:  2047968k total,        0k used,  2047968k free,  2247376k cached

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sdc               2.71         0.00        43.31          0        144
sda               2.41         0.00       146.76          0        488

Interface:    Bytes_rcv    packets_rcv    Bytes_trans    packets_trans
eth0           44966270         226057       39055911           104466
lo                15960            291          15960              291
```

Meaning of the CPU percentages:

us User - regular user apps
sy Syst - system (general kernel stuff)
ni Nice - nice user apps (low-priority tasks)
id Idle

wa Wait - waiting for IO to complete
hi hard interrupt (IRQ) handlers
si soft interrupt (network stack, mostly) handlers

Main options for top:

- general options:

  h displays help
  d defines the time between two measurements
  x displays disk activity
  E displays network activity
  C configures how disk and network activities are displayed

- options for the abstract part (from top to bottom); these options reverse the current state of display:

  W logs the top configuration
  l displays or not the **load average** part
  t displays or not the **Tasks** part
  m displays or not the **Memory** part
  1 displays the load by CPU, or globally for all

- options for the processes part:

  L displays or not the processes part
  P sorts the processes according to the CPU load
  M sorts the processes according to the memory use
  R reverses the sort
  f selects the columns to be displayed
  F selects the columns to be sorted
  u selects the processes for a user
  # or **n** defines the maximum number of processes to be displayed

To keep the top results, you can use the batch option (**-b** option). Just redirect the output in a file, and set the options as parameters; For example:

```
top -b [-p pid] [-u user] ...
```

The configuration can also be supplied in a **.toprc** file, initialized in interactive mode and used in batch mode.

9.4.2 Using top_pfb

The **top_pfb** command is a double of **top**, specific for NovaScale. **top_pfb** includes all **top** functions and provides supplementary information about:

- CPU activity by QBB (quadri-processor block)

- Memory localization, or (exclusive choice) use of the memory bandwidth.

Only one instance of **top_pfb** can run at a time.

top_pfb specific options:

2	CPU load by QBB
N	activates the memory measurements of the SNC chip and waits for a supplemental information (exclusive choice): **l** memory localization **u** use of the memory bandwidth

The SNC measurements are performed thanks to a kernel patch, in mono access, which allows only one active instance of **top_pfb** by machine.

The following figure shows an example of **top_pfb** output, using **N** and **l** options, on an idle machine.

```
Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie
QBB0 :  0.2% us,  0.2% sy,  0.0% ni, 399.7% id,  0.0% wa,  0.0% hi,  0.0% si
QBB1 :  0.0% us,  0.0% sy,  0.0% ni, 400.0% id,  0.0% wa,  0.0% hi,  0.0% si
QBB2 :  0.0% us,  0.0% sy,  0.0% ni, 400.0% id,  0.0% wa,  0.0% hi,  0.0% si
QBB3 :  0.0% us,  0.0% sy,  0.0% ni, 400.0% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:  66692384k total,  2277728k used, 64414656k free,    28128k buffers
Swap:  2097120k total,        0k used,  2097120k free,   476272k cached
total of localization rate: 75.4%
localization rate per QBB:
   QBB0: 97.8%    QBB1: 68.4%    QBB2: 68.6%    QBB3: 65.3%

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
```

Remarks:

- Though it may be surprising that the QBB utilization can reach 400% if 4 CPUs are configured in the QBB. This value actually corresponds to the sum of percentages of all CPUs.
- The localization rate is useful to observe if the number of memory accesses is high. For good performance, you should also survey that the load is well balanced on all QBBs.

9.4.3 PerfWare Operation

PerfWare can be used in numerous ways, either through a command interface, a web interface or in interactive mode. Graphics are generated nearly exclusively in web mode.

Comprehensive user's guide and installation and configuration guide are in **/usr/share/doc/perfware-n.m.p** where n.m.p is the **perfware** release number.

Data collection:

Data collection is defined by parameters such as starting time, period, interval between two collections. The collect can be performed locally or remotely. It can be started either interactively or automatically.

A **perfware_status_show** tool allows you to know the collection status.

- In interactive mode, run the **collector** command and let you guide by the questions that are displayed:

```
collector
Start Now or Later ? [N/L] ...N
```

```
Now it is : 19 / 05 / 2004  08 : 36
************* Please check your system date/time is correct *************
Starting Year ? [2004-2005] [2004]      ... 2004
Starting Month ? [01-12] [05]   ... 05
Starting Day ? [01-31] [19]     ... 19
Starting Hour ? [00-23] [08]    ... 10
Starting Minut ? [0-59] [36]    ... 00
Collect will start : 19 / 05 / 2004  10 : 00
Do you agree ? [Y/N] ... Y
```

```
Save directory path is /tmp ? [Y/N] ...
```

```
Sampling interval (seconds) ? [1-3600][60] ...
```

```
Collection time (minutes) ? [1-10080][1440] ...
```

```
Collection will record 1440 samples over 1 day 0 h 0 min 0 sec
```

```
Do you agree ?[Y/N] ...
```

- In automatic mode, enter this command for a local collection:

```
collector [-A] [-l] [-N|-M|-L date] [-i interval] [-d duration] [-s savedir]
```

- In automatic mode, enter this command for a remote collection:

```
collector -R [-l] [-N|-M|-L date] [-i interval] [-d duration] [-s savedir] -
r r_system
```

Analyzing data in interactive mode (Parser):

To start the analysis in interactive mode, you must be in the directory used for the collect, then call the **parser** command and follow the instructions:

```
cd /tmp/perfware_nameserver_date_hour
parser
*****************************************************************************
**
*                                PerfWare Parser
*
*****************************************************************************
**


Start Statistics Generation

        Collect directory path is /tmp/perfware_nameserver_date_hour ? [Y/N]
...
```

If you answer N, the following screen is displayed:

```
Set your collect directory path :


         The following days have been found in
/tmp/perfware_nameserver_date_hour/collect :

           Day : 11     Nb points : 1440
           Day : 12     Nb points : 1440
           Day : *      To parse all days in separate directory
           Day : #      To parse all days in the same directory

        Which day do you want ? [*] ...

Collection directory : /tmp/perfware_nameserver_date_hour/collect
Collection day       : 11
Graphic directory    : /tmp/perfware_nameserver_date_hour/gra11
GENERATING STATS GRAPHICS ...
GENERATING PROCESS GRAPHICS ...
```

If you choose a specific day among those offered (11 for example), the collected information for this day is analyzed and the result is logged in a directory indicated below:

```
Collection directory : /tmp/perfware_nameserver_date_hour/collect
Collection day       : 11
Graphic directory    : /tmp/perfware_nameserver_date_hour/gra11
GENERATING STATS GRAPHICS ...
GENERATING PROCESS GRAPHICS ...
```

If you choose all days (*), all days are analyzed, and the results are logged in several directories, one for each day, as indicated below:

```
Collection directory : /tmp/perfware_nameserver_date_hour/collect
Collection day       : 11
Graphic directory    : /tmp/perfware_nameserver_date_hour/gra11
GENERATING STATS GRAPHICS ...
GENERATING PROCESS GRAPHICS ...

Collection directory : /tmp/perfware_nameserver_date_hour/collect
Collection day       : 12
Graphic directory    : /tmp/perfware_nameserver_date_hour/gra12
GENERATING STATS GRAPHICS ...
GENERATING PROCESS GRAPHICS ...
```

If you choose #, all days are analyzed as if they were only one day and the result is logged in a directory indicated below:

```
Collection directory : /tmp/perfware_nameserver_date_hour/collect
Collection day       : 00
Graphic directory    : /tmp/perfware_nameserver_date_hour/gra00
GENERATING STATS GRAPHICS ...
GENERATING PROCESS GRAPHICS ...
```

Analyzing data in automatic mode:

To start the analysis in automatic mode, call the **parser** command as follows:

```
parser [-A] [-c dir_col] [-d day] [-l limit]
```

parser options:

-A Starts **parser** with the default values.

-c *dir_col* Specifies the collection directory path.
Default value is . or /tmp/perfware_*nameserver_date_hour*

-d *day* Specifies the day to parse, in one of these formats: DD or * or #:

- -d * parses all the days and the plots generated are stored in separate directories.
- -d # parses all the days as if it was only one day (00) and the plots are stored in one directory.

Default value is *.

-l *limit* Specifies the minimum percentage of activity for the process to be parsed. Default value is 1.0 (1%).

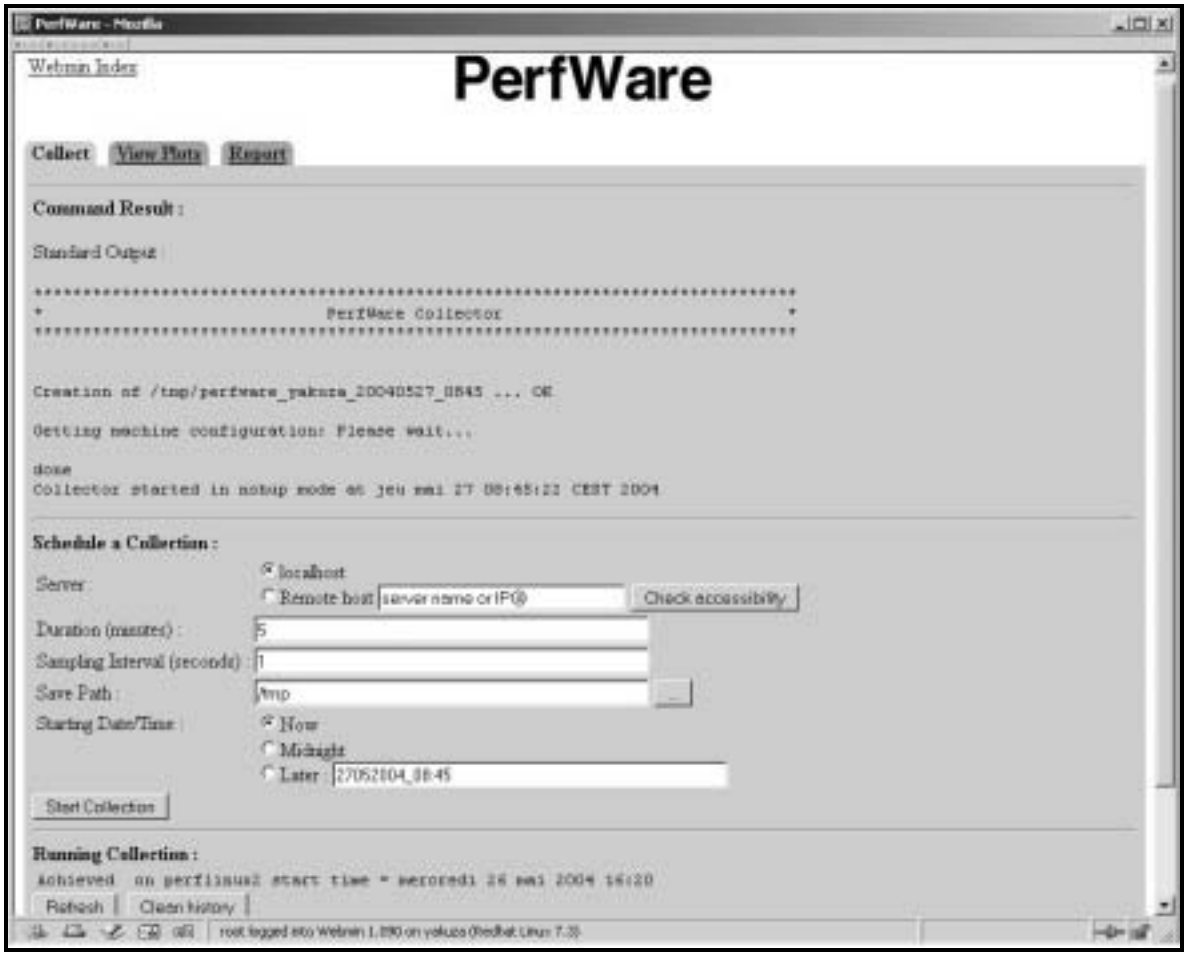
Notes:
- Any option can be used with **–A**, in order to change default values.
- If the collection directory path and the day are provided no question will be asked.

9.4.4 Perfman web Interface

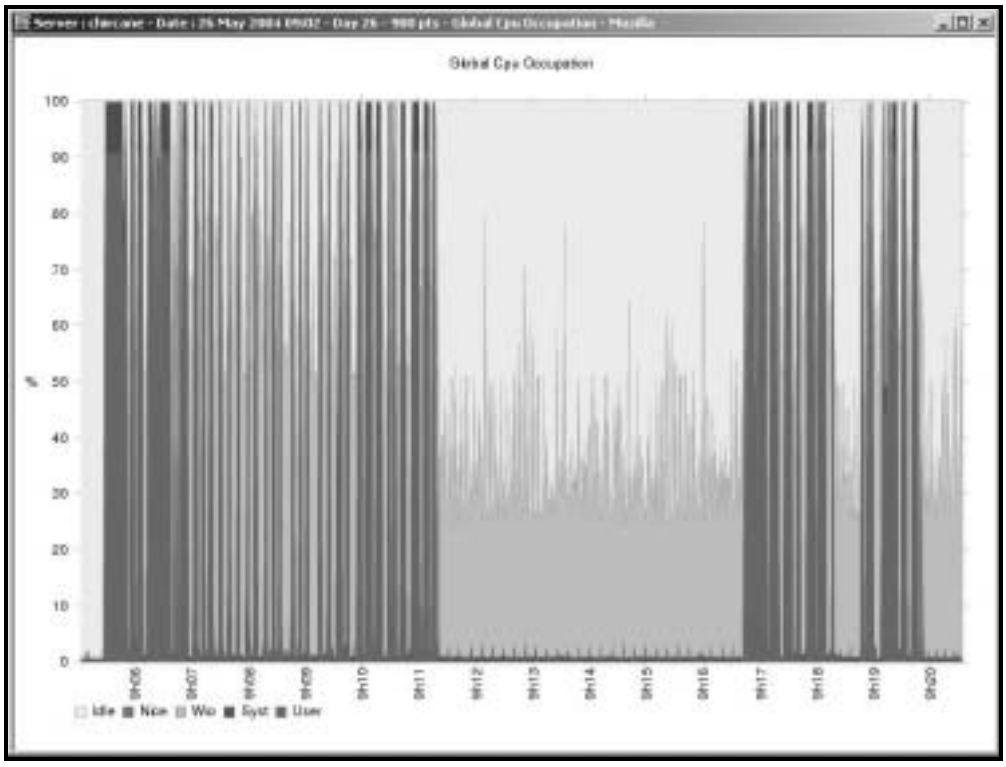
The **Webmin** tool provides the Web interface of **PerfWare**. . Webmin is installed during the normal installation process of perfware for BAS3.

Start Webmin, enter your identification and select the **Others** tab, as shown in the following figure:

Click on the **PerfWare** icon. The graphical interface of PerfWare is displayed as shown in the following screen:



Using PerfWare it is possible to see information in a graphical format. The following figure shows global CPU occupation:



For more information, please refer to the documentation included with the binary file. A man page is also installed.

9.4.5 gprof Operation

Using a profiling tool must be anticipated from the compilation step, because it requires the code to be configured. To use **gprof**, do the following:

- Compile with **–pg** option for C programs or **–p** for Fortran programs. Use also with **-g** if you want line-by-line profiling.
- Run the program:

```
./exe
```

This creates a **gmon.out** file.

- Call **gprof** to make it display the statistics collected in **gmon.out**, as in the following figure:

```
gprof exe
%   cumulative   self              self     total
   time   seconds   seconds    calls  ms/call  ms/call  name
   67.97     21.64    21.64        1 21639.65 21639.65  Tri
   31.66     31.72    10.08    15000     0.67     0.67  search_ind
    0.15     31.77     0.05        1    48.83    93.75  Init_Tab
    0.14     31.81     0.04   900000     0.00     0.00  getrand
    0.07     31.83     0.02                             __divdf3
    0.01     31.84     0.00                             main
```

9.4.6 Using cprof

cprof is a command line interface for which installation of **PAPI** is a prerequisite.

To use **cprof** on the **test.c** program, do as follows:

- Compile with **–g**.
- Link-edit with the **libvmon** and **libpapi** libraries and a program that generates automatically the profiling calls: **vmonauto_gcc.o**.
- Set **VMON** variable with the event that needs to be surveyed, and **VMON_FILE** variable with the name of the result file (**vmon.out** by defect).

  Note: It is possible to get from one to four events at the same time, setting the VMON variable with the events, as in the following example:
  `VMON = “PAPI_TOT_CYC;PAPI_BR_INS;PAPI_L3_TCM;PAPI_FP_OPS”`

- Launch the program (the file specified by **VMON_FILE** is created).
- Call **cprof** with the options you need, in order to analyze **VMON_FILE**:

cprof usage:

```
cprof [options] executable [vmon_file]
```

cprof options:

-h, --help	Displays help.
--debug	Prints debugging info.
--force	Shows data that is not accurate.
-d, --directory *dir*	Searches *dir* for source files.
-D, --recursive-directory *dir*	Searches *dir* recursively for source files.
-e, --everything	Shows all information.
-f, --files	Shows all files.
-r, --funcs	Shows all functions.
-l, --lines	Shows all lines.
-a, --annotate *file*	Annotates *file*.
-n, --number	Shows number of samples (not %).
-s, --show *thres*	Sets threshold for showing aggregate data.
-H, --html *dir*	Outputs HTML into directory *dir*.
-v, --version	Displays the version number.

Note: If the searched event is either too rare or too frequent, it is possible to change the sampling frequency, using the **VMON_FREQ** environment variable. Its default value is 100 000, which means that the number of events supplied by **cprof** must be multiplied by 100 000. For a rare event you can decrease this value.

In the following example no hardware event is specified, so the PROF event is surveyed, as **gprof** does.

```
cprof –e exe vmon.out

Columns correspond to the following events:
  PROF - SVr4 profil (32764 events)
File Summary:
 100.0% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c
Function Summary:
  68.1% Tri
  31.7% search_ind
   0.1% Init_Tab
   0.1% getrand
Line Summary:
  36.4% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:31
  31.7% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:41
  27.5% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:29
   4.2% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:32
```

In the following example, `export VMON=PAPI_L2_TCM` is used, specifying the event to be surveyed:

```
Columns correspond to the following events:
  PAPI_L2_TCM - Level 2 total cache misses (4168 events)
Function Summary:
  50.9% search_ind
  49.0% Tri
Line Summary:
  50.9% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:41
  48.8% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:31
   0.1% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:29
   0.1% /home/megyv/PROGTEST/TEST_ANALYZE/VERSION_GOOD/manip_tab.c:27
with profile annotations.
……
41  31.7%   while (*(T+i*COL)<x && i<ROW) i++;
```

9.4.7 Profilecomm data collection

The use of **profilecomm** includes 2 steps: Data collecting then its analysis.

mpiprofile Interface

To be profiled by **profilecomm**, an application must be linked to an MPI library provided with the **mpiprofile** interface, such as **MPI_Bull**.

mpiprofile lets you choose the library and the options to be used. There are 2 ways to do that: setting environment variables or using options of **MPI_Init**.

In the first method, the name of the library to load is set in the **MPIPROFILE** variable and the options are set in the **MPIPROFILE_OPTIONS** variable. Choose this method to activate the profiling when you start an application.

In the second method the name of the library to load is specified in the **mpiprofile** option of the **MPI_Init** function, and the options are specified in the **-mpiprofileoptions** option. Choose this method to set the profiling within the sources.

profilecomm Library

So long as the MPI library is compatible, using **profilecomm** requires no compilation or linkage.

To use **profilecomm** you can either:

- set the **profilecomm** argument in the **-mpiprofile** option of **MPI_Init** (providing that the arguments are transmitted to **MPI_Init**). For example:

```
$ mprun -n 4 ./foo -mpiprofile profilecomm
MPI Bull 2 v0.1 alpha (mono + one-sided), 4 (2967)
```

- or set the **MPIPROFILE** environment variable to **profilecomm**. For example:

```
$ MPIPROFILE=profilecomm mprun -n 4 ./foo
MPI Bull 2 v0.1 alpha (mono + one-sided), 4 (2979)
```

When the application finishes, the result of the collect is written in a file (`mpiprofile.pfc` by default). By default this file is saved in a format specific to profilecomm, but it is possible to save it in text format. The **readpfc** command enables to work on **.pfc** files.

Options

You can set options to define the profilecomm behavior. For that, two methods are available:

- Use the **-mpiprofileoptions** option of **MPI_Init** (providing that the arguments are transmitted to **MPI_Init**). For example:

```
$ mprun -n 4 ./foo -mpiprofile profilecomm -mpiprofileoptions "-f\ foo.pfc"
MPI Bull 2 v0.1 alpha (mono + one-sided), 4 (2967)
```

- Or use the **MPIPROFILE_OPTIONS** variable. This is the recommended method. For example:

```
$ MPIPROFILE=profilecomm MPIPROFILE_OPTIONS="-f foo.pfc" mprun -n 4 ./foo
MPI Bull 2 v0.1 alpha (mono + one-sided), 4 (2979)
```

Following are some options that modify the behavior of **profilecomm** for saving results in a file:

`-f` *`file`*, `-filename` *`file`*

Saves the result in the `file` file instead of the default file (`mpiprofile.txt` for the text format and `mpiprofile.pfc` for the profilecomm binary format).

`-t`, `-text`

Saves the result in a text format file, readable with any text editor or reader. This format is useful for debugging purpose, but it is not easily used beyond 10 processes.

-b, -bin

Saves the result in a profilecomm binary format file. This is the default format. The **readpfc** command is required to work on this file.

-s, -sync

Synchronizes the processes during the time measurements. This option is set by default.

-ns, -nosync

Doesn't synchronize the processes during the time measurements.

Examples

To make the profiling of the **foo** program and save the result of the collect in the `mpiprofile.pfc` default file, enter:

```
$ MPIPROFILE=profilecomm mprun -n 4 ./foo
```

To save the result of the collect in the `foo.pfc` file, enter:

```
$ MPIPROFILE=profilecomm MPIPROFILE_OPTIONS="-f foo.pfc" mprun -n 4 ./foo
```

To save the result of the collect in text format in the `foo.txt` file, enter:

```
$ MPIPROFILE=profilecomm MPIPROFILE_OPTIONS="-t -f foo.txt" mprun -n 4 ./foo
```

9.4.8 Profilecomm Data Analysis

To analyze data collected with profilecomm you can use the **readpfc** command and other tools like spreadsheet. The main features of **readpfc** are the following:

- Displaying data contained in **profilecomm** files
- Exporting communication matrixes in usual file format.

readpfc syntax

```
readpfc [options] [file]
```

If `file` is not specified, **readpfc** reads the default file **mpiprofile.pfc** in the current directory.

Readpfc output

The main feature of readpfc is to display information contained in the six different sections of a profilecomm file, which are:

- Header
- Point to point
- Collective
- Call table
- Statistics
- Topology.

- **Header Section:**

  Displays information contained into the header of a profilecomm file. The more interesting fields are:

  - Elapsed Time, which represents the duration of the collect
  - "world size", which shows number of processes.

  The other fields are less interesting or will be useful for profilecomm evolutions.

  Example:

```
Header:
 Version: 1
 Flags: little-endian
 Header size: 28 bytes
 Elapsed time: 80757 us
 World size: 8
 Number of partitions: 1
 num_intsz: 4 bytes (32 bits)
 num_volsz: 4 bytes (32 bits)
```

- **Point to point Section:**

  Displays point to point communication matrixes. The number of communication is displayed first, then the volume. If one of the`--numeric-only` or `--volumic-only` options is activated only one matrix is displayed.

Example:

```
Point to point:
      0   1.1k      0      0      0      0      0      0
   1.1k      0      0      0      0      0      0      0
      0      0      0   1.1k      0      0      0      0
      0      0   1.1k      0      0      0      0      0
      0      0      0      0      0   1.1k      0      0
      0      0      0      0   1.1k      0      0      0
      0      0      0      0      0      0      0   1.1k
      0      0      0      0      0      0   1.1k      0

      0 818.8k      0      0      0      0      0      0
 818.8k      0      0      0      0      0      0      0
      0      0      0 818.8k      0      0      0      0
      0      0 818.8k      0      0      0      0      0
      0      0      0      0      0 818.8k      0      0
      0      0      0      0 818.8k      0      0      0
      0      0      0      0      0      0      0 818.8k
      0      0      0      0      0      0 818.8k      0
```

- **Collective Section:**

  The collective section is equivalent to the point to point section for collective communication matrixes.

  Example:

```
Collective:
      0    100    100    100    100    100    200    100
    100      0      0      0      0      0      0    100
    200      0      0      0    100      0    100      0
      0    100    100      0      0    100      0    100
    100      0    100      0      0      0      0      0
      0      0      0    100    100      0      0      0
      0      0    100      0    100      0      0      0
      0      0      0    100      0      0    100      0

      0 409.6k 409.6k 409.6k 409.6k 409.6k 421.6k 409.6k
   12k      0      0      0      0      0      0    12k
 421.6k     0      0      0 409.6k      0 409.6k      0
      0 409.6k    12k      0      0 409.6k      0 409.6k
   12k      0    12k      0      0      0      0      0
      0      0      0    12k    12k      0      0      0
      0      0    12k      0    12k      0      0      0
      0      0      0    12k      0      0    12k      0
```

- **Call table Section:**

  This section contains the call table. If the `--ct-total-only` option is activated, only the total column is displayed..

  Example:

```
Call table:
                     0      1      2      3      4      5      6      7  Total
Allgather            0      0      0      0      0      0      0      0      0
Allgatherv           0      0      0      0      0      0      0      0      0
Allreduce            2      2      2      2      2      2      2      2     16
Alltoall             0      0      0      0      0      0      0      0      0
Alltoallv            0      0      0      0      0      0      0      0      0
Bcast              200    200    200    200    200    200    200    200   1.6k
Bsend                0      0      0      0      0      0      0      0      0
Gather               0      0      0      0      0      0      0      0      0
Gatherv              0      0      0      0      0      0      0      0      0
Ibsend               0      0      0      0      0      0      0      0      0
Irsend               0      0      0      0      0      0      0      0      0
Isend                0      0      0      0      0      0      0      0      0
Issend               0      0      0      0      0      0      0      0      0
Reduce             200    200    200    200    200    200    200    200   1.6k
Reduce_scatter       0      0      0      0      0      0      0      0      0
Rsend                0      0      0      0      0      0      0      0      0
Scan                 0      0      0      0      0      0      0      0      0
Scatter              0      0      0      0      0      0      0      0      0
Scatterv             0      0      0      0      0      0      0      0      0
Send              1.1k   1.1k   1.1k   1.1k   1.1k   1.1k   1.1k   1.1k   8.8k
Sendrecv             0      0      0      0      0      0      0      0      0
Sendrecv_replace     0      0      0      0      0      0      0      0      0
Ssend                0      0      0      0      0      0      0      0      0
Start                0      0      0      0      0      0      0      0      0
```

- **Statistics Section:**

  This section displays statistics computed by **readpfc**. These statistics are based on the information contained in the collect file. This section is divided into two sub-sections:

  – The *General statistics* section contains statistics for the whole application.
  – The *Per process average* section contains average per process.

  For each statistic we distinguish point to point communications from collective communications.

Example:

```
General statistics:
                        pt2pt |       coll |      total
Messages count  |        8800 |       2600 | 1.1400e+04
Volume          |  6.24695Mb |   5.2269Mb |  11.4738Mb
Avg message size|       744b |  2.05859kb |  1.03063kb
Throughput b/s  |  77.35Mb/s |  64.72Mb/s |  142.1Mb/s

Per process average:
                        pt2pt |       coll |      total
Messages count  |        1100 |        325 |       1425
Volume          |  799.609kb |  669.043kb |  1.43423Mb
Throughput b/s  |  9.669Mb/s |   8.09Mb/s |  17.76Mb/s
```

Where:

Messages count	Number of sent messages
Volume	Volume of sent messages
Avg message size	Average size of messages
Throughput b/s	Throughput for sent messages

- **Topology Section:**

  This section shows the distribution of the processes on nodes and processors. This information is displayed in two tables:

  – The first table displays, for each process, the node and the cpu where the process is running.
  – The second table displays, for each node, the list of running processes.

  Example: 8 processes running on 2 nodes.

```
Topology:
8 process on 2 hosts
process hostid  cpuid
      0      0      0
      1      0      1
      2      0      2
      3      0      3
      4      1      0
      5      1      1
      6      1      2
      7      1      3

host  processes
   0  0 1 2 3
   1  4 5 6 7
```

Display options

The following options allow to select information to be displayed:

-a, --all

Displays all the information. Equivalent to `-ghmst`.

-c, --collective

Displays collective communication matrixes.

-g, --topology

Displays topology of execution environment.

-h, --header

Displays header of profilecomm file.

-m, --matrix, --matrices

Displays communication matrix. Equivalent to `–cp`.

-n, --numeric-only

Does not display volume matrixes. You cannot use this option simultaneously with `-v`/`--volumic-only` option.

-p, --p2p, --pt2pt

Displays point to point communication matrixes.

-s, --statistics

Computes and displays some statistics regarding MPI communications.

-S, --scalable

Displays all scalable information; this means all information whose size is independent of number of processes. Useful when there is a great number of processes. Equivalent to `-hstT`.

-t, --calltable

Displays the call table.

-T, --ct-total-only

Displays only the *`Total`* column of the call table. By default **readpfc** displays also one column for each process.

-v, --volumic-only

Does not display numeric matrixes. You canno use this option simultaneously with `-n`/`--numeric-only` option.

Exporting a matrix

The communication matrixes can be exported in different formats that can be understood by other software like spread sheet. Three formats are available: CSV (Comma Separated Values), MatrixMarket and gnuplot.

It is also possible to have a graphic display of the matrix, which is a better representation for matrixes with a big number of elements. This also allows to include graphics in a report. Seven graphic formats are available: PostScript, Encapsulated PostScript, SVG, xfig, EPSLaTeX, PSLaTeX and PSTeX. All these formats are vectorial, which lets you modify the dimensions of the graphics if necessary.

The following options manage matrixes export:

`--csv-separator` ***`sep`***

Modifies CSV delimiter. Default delimiter is comma ",". Some software prefer semicolon ";".

`-f` ***`format`***, **`--format`** ***`format`***

Chooses export format. Default format is CSV (Comma Separated Values). Values for *`format`* are the following:

`help`	lists available export format.
`csv`	export with CSV format.
`mm`, `market`, `MatrixMarket`	export with MatrixMarket format.
`gp`, `gnuplot`	export with format used by pfcplot in order to realize graphical display of the matrix.
`ps`, `postscript`	export with PostScript format.
`eps`	export with Encapsulated PostScript format.
`svg`	export with Scalable Vector Graphics format.
`fig`, `xfig`	export with xfig format.
`epslatex`	export with LaTex and Encapsulated PostScript format.
`pslatex`	export with LaTex format and PostScript inline.
`pstex`	export with Tex format and PostScript inline.

Warning: Using `epslatex`, two files are written: xxx.tex and xx.eps. The filename indicated in the –o option is the name of the Latex file (.tex).

--logscale[=*base*]

Uses a logarithmic color scale. Default value for logarithm basis is 10; this basis can be modified using `base` argument. This option is pertinent only with export in graphical format.

--nogrid

Does not display the grid on a graphical representation of the matrix.

-o *file*, --output *file*

Specifies the file name for export file. The default filename is `out.csv`, `out.mm`, `out.dat`, `out.ps`, `out.svg`, `out.fig` or `out.tex`, according to export format, This option is available only with `-x` option.

--palette *pal*

Uses a personalized colored palette. This option is pertinent only in the case of export in graphical format. This palette must be compatible with the `defined` function of gnuplot, for instance: `--palette '0 "white", 1 "red", 2 "black"'` or `--palette '0 "#0000ff", 1 "#ffff00", 2 "ff0000"'`

`--title` *title*

Uses a personalized title for graphical display. The default title is *Point-to-point/collective numeric/volumic communication* matrix, according to the exported matrix.

-x *matrix*, --export *matrix*

Exports communication matrix specified by the *matrix* argument. Values for *matrix* are the following:

help	list of available matrixes;
pn, *np*	Point to point numerical communication matrix ;
pv, *vp*	Point to point volumic communication matrix ;
cn, *nc*	Collective numerical communication matrix ;
cv, *vc*	Collective volumic communication matrix ;

Other options

-H, --help, --usage

Displays help message

-q, --quiet

Does not display help warning messages (error messages continue to be displayed).

-V, --version

Displays program version.

Examples

To display all information available in foo.pfc file, enter:

```
$ readpfc -a foo.pfc
```

```
Header:
 Version: 1
 Flags: little-endian
 Header size: 28 bytes
 Elapsed time: 10755 us
 World size: 4
 Number of partitions: 1
 num_intsz: 4 bytes (32 bits)
 num_volsz: 4 bytes (32 bits)
[...]

Topology:
4 process on 1 hosts
process hostid  cpuid
      0      0      0
      1      0      1
      2      0      2
      3      0      3

host  processes
   0  0 1 2 3
```

To display point to point numerical communication matrix, enter:

```
$ readpfc -pn foo.pfc
```

```
Point to point:
      0      0      0      0
      2      0      0      0
      2      0      0      0
      2      0      0      0
```

To export the collective volumic communication matrix in CSV format in the default file, enter:

```
$ readpfc –x cv foo.pfc
Warning: No output file specified, write to default (out.csv).
$ ls out.csv
out.csv
```

To export the point to point numerical communication matrix in MatrixMarket format in the `foo.mm` file, enter:

```
$ readpfc -x np -f mm -o foo.mm foo.pfc
$ ls foo.mm
foo.mm
```

pfcplot and gnuplot

The **pfcplot** script converts matrixes into graphic using **gnuplot**. It is generally used by **readpfc**, but can be used directly by the user who wants more flexibility. The matrix must be exported with the `–f gnuplot` option to be read by **pfcplot**.

For more details enter:

```
man pfcplot
```

Users who have special requirements can invoke gnuplot directly

. To do this, the matrix must be exported with gnuplot format or with CSV format, choosing space as separator.

Warning: due to limitation in gnuplot, one null line and one null column are added to the exported matrix in gnuplot format.

9.4.9 Using pfmon

pfmon is a good tool to quickly get a characteristic figure of a program, without any programming. For example, to get the number of operations in floating point in the `test2` program, run:

```
pfmon -e FP_OPS_RETIRED test2
```

The output is similar to the following one:

```
2 421 914 081 FP_OPS_RETIRED
```

It is possible to capture 4 events in the same command and to get a complete output using **with-header** option. For example:

```
pfmon --with-header -k -u
-e L2_MISSES,L2_REFERENCES,L3_MISSES,IA64_INST_RETIRED
--outfile=result_pfmon/pfmon_miss_16.2081 runspec ……..
```

The output is similar to the following one:

```
#
# date: Fri Feb 21 02:16:14 2003
#
# hostname: chircane
#
# kernel version: Linux 2.5.59 #2 SMP Wed Feb 12 06:17:00 PST 2003
#
# pfmon version: 2.0
# kernel perfmon version: 1.3
#
#
#
# page size: 16384 bytes
# CLK_TCK: 1024 ticks/second
# CPU configured: 4
# CPU online: 4
# physical memory: 4189208576
# physical memory available: 2908225536
#
# host CPUs:  4-way 997MHz Itanium 2 (McKinley, B3)
#       PAL_A: 0.7.31
#       PAL_B: 0.7.36
#       Cache levels: 3 Unique caches: 4
#       L1D:    16384 bytes, line  64 bytes, load_lat   1, store_lat   3
#       L1I:    16384 bytes, line  64 bytes, load_lat   1, store_lat   0
#       L2 :   262144 bytes, line 128 bytes, load_lat   5, store_lat   7
#       L3 :  3145728 bytes, line 128 bytes, load_lat  12, store_lat   7
#
#
# captured events:
#       PMD4: L2_REFERENCES, kernel+user level(s)
#       PMD5: L2_MISSES, kernel+user level(s)
#       PMD6: L3_MISSES, kernel+user level(s)
#       PMD7: IA64_INST_RETIRED, kernel+user level(s)
#
# monitoring mode: per-process
#
#
# instruction sets:
#       PMD4: L2_REFERENCES, ia32/ia64
```

```
#       PMD5: L2_MISSES, ia32/ia64
#       PMD6: L3_MISSES, ia32/ia64
#       PMD7: IA64_INST_RETIRED, ia32/ia64
#
#
# command: pfmon --with-header -k -u -e
L2_MISSES,L2_REFERENCES,L3_MISSES,IA64_INST_RETIRED --
outfile=result_pfmon/pfmon_miss_16.2081 runspec -c il420-linux-v7 --
iterations=1 --tune=base -I --noreportable --rate --users 16 applu
#
#
#
                  4234016 L2_MISSES
                 69799519 L2_REFERENCES
                   527948 L3_MISSES
                772924788 IA64_INST_RETIRED
```

Note: The **–follow** option is useful to follow all the descendants of a process.

9.4.10 Using PAPI

The PAPI library offers a set of functions for the programmer. The specific release for IA64 Itanium II manages the same counters than PAPI. Therefore its operation consists firstly in adding elements inside the program. There two interface levels:

- The high-level interface manages a set of classical counters called preset
- The low-level interface manages all pfmon counters.

Following are described the operations for the two interface levels.

9.4.11 PAPI high-level interface

The high-level API provides the ability to start, stop and read the counters for a specified list of events. It is well designed for programmers who need simple event measurements, using PAPI preset events.

Compared with low-level API, the high-level is easier to use and requires less setup (additional calls). However this ease of use leads to somewhat higher overhead and loss of flexibility.

Note: Earlier versions of high-level API were not thread safe. This restriction is removed with PAPI 3.

Following is a simple code example using the high-level API:

```
#include <papi.h>

#define NUM_FLOPS 10000
#define NUM_EVENTS 1

main()
{
int Events[NUM_EVENTS] = {PAPI_TOT_INS};
long_long values[NUM_EVENTS];

/* Start counting events */
if (PAPI_start_counters(Events, NUM_EVENTS) != PAPI_OK)
   handle_error(1);

/* Defined in tests/do_loops.c in the PAPI source distribution */
do_flops(NUM_FLOPS);

/* Read the counters */
if (PAPI_read_counters(values, NUM_EVENTS) != PAPI_OK)
   handle_error(1);

printf("After reading the counters: %lld\n",values[0]);

do_flops(NUM_FLOPS);

/* Add the counters */
if (PAPI_accum_counters(values, NUM_EVENTS) != PAPI_OK)
   handle_error(1);
printf("After adding the counters: %lld\n", values[0]);

/* double a,b,c; c+= a* b; 10000 times  */
do_flops(NUM_FLOPS);

/* Stop counting events */
if (PAPI_stop_counters(values, NUM_EVENTS) != PAPI_OK)
   handle_error(1);

printf("After stopping the counters: %lld\n", values[0]);
}
```

Possible Output:

```
After reading the counters: 441027
After adding the counters: 891959
After stopping the counters: 443994
```

Note that the second value (after adding the counters) is approximately twice as large as the first value (after reading the counters). This is because `PAPI_read_counters` resets and leaves the counters running, then `PAPI_accum_counters` adds the current counter value into the **values** array.

9.4.12 PAPI low-level interface

The low-level API manages hardware events in user-defined groups called **Event Sets**. It is well designed for experienced application programmers and tool developers who need fine-grained measurements and control of the PAPI interface. Unlike the high-level interface, it allows both PAPI preset and native events.

The low-level API features the possibility to get information about the executable and the hardware, and to set options for multiplexing and overflow handling. Compared with high-level API, the low-level increases efficiency and functionality.

An Event Set is a user-defined group of hardware events (preset or native) which, all together, provide meaningful information. The users specify the events to be added to the Event Set and attributes such as the counting domain (user or kernel), whether or not the events are to be multiplexed, and whether the Event Set is to be used for overflow or profiling. PAPI manages other Event Set settings such as the low-level hardware registers to use, the most recently read counter values and the Event Set state (running / not running).

Following is a simple code example using the low-level API. It applies the same technique as the high-level example.

```
#include <papi.h>
#include <stdio.h>

#define NUM_FLOPS 10000

main()
{
int retval, EventSet=PAPI_NULL;
long_long values[1];

/* Initialize the PAPI library */
```

```
retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT) {
  fprintf(stderr, "PAPI library init error!\n");
  exit(1);
}

/* Create the Event Set */
if (PAPI_create_eventset(&EventSet) != PAPI_OK)
    handle_error(1);

/* Add Total Instructions Executed to our Event Set */
if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);

/* Start counting events in the Event Set */
if (PAPI_start(EventSet) != PAPI_OK)
    handle_error(1);

/* Defined in tests/do_loops.c in the PAPI source distribution */
do_flops(NUM_FLOPS);

/* Read the counting events in the Event Set */
if (PAPI_read(EventSet, values) != PAPI_OK)
    handle_error(1);

printf("After reading the counters: %lld\n",values[0]);

/* Reset the counting events in the Event Set */
if (PAPI_reset(EventSet) != PAPI_OK)
  handle_error(1);

do_flops(NUM_FLOPS);

/* Add the counters in the Event Set */
if (PAPI_accum(EventSet, values) != PAPI_OK)
   handle_error(1);
printf("After adding the counters: %lld\n",values[0]);

do_flops(NUM_FLOPS);

/* Stop the counting of events in the Event Set */
if (PAPI_stop(EventSet, values) != PAPI_OK)
```

```
    handle_error(1);

printf("After stopping the counters: %lld\n",values[0]);
}
```

Possible Output:

```
After reading the counters: 440973
After adding the counters: 882256
After stopping the counters: 443913
```

Note that `PAPI_reset` is called to reset the counters, because `PAPI_read` does not reset the counters. This lets the second value (after adding the counters) to be approximately twice as large as the first value (after reading the counters).

For more details, please refer to PAPI man and documentation, which are installed with the product in **/usr/share** directory.

A. Troubleshooting

Following is the list of frequently asked questions (FAQ) for which solutions and advices are provided.

Problems when compiling and executing

- I get the message: "`error while loading shared libraries`" when a program executes.
- My parallel program cannot find the program on the other machines.
- How do I optimize compilation with the **Intel FORTRAN compiler**?
- How do I optimize compilation with the **Intel C / C++ compiler**?
- Can I run applications compiled under previous OS releases?
- I get lots of "`unaligned access`" error messages.

Problems when compiling and executing with MPICH

- I have a problem with **memory allocations** when I use MPICH.
- Problems when compiling and executing with QSNET MPI
- At runtime my program hangs when I use QSNET MPI (libelan).

OpenMP

- To run a program parallelized with OpenMP, how do I **define the number of threads** (processors) used?

I get the message: "`error while loading shared libraries`" when a program executes.

Add the path for this library to the LD_LIBRARY_PATH environment variable.

My parallel program cannot find the program on the other machines.

You must have the binaries on all machines running the benchmarks and respect the tree structure of the machine from which the benchmark is started, or use NFS.

How do I optimize compilation and debugging with the Intel fortran compiler?

For optimization, add the following compilation options:

-implicitnone	Forces the declaration of variables: If a variable is used without being declared, this triggers errors on compilation.
-w95	Removes warnings for non standard f95 instructions.
-mp	Respects IEEE standard double precision.
-unroll2	To unroll a loop: This favors vectorization and the instructions pipeline.
-ip, -ipo	Optimizes calls to a subprogram (parameter management).
-auto	Allocates the variables dynamically to the stack rather than in static storage in the memory.
-zero	Implicitly initializes variables to 0.
-ftz	flush-to-zero.
-i_dynamic	Avoids loading static libraries and therefore reduces the size of the executable.
-parallel	Parallelizes certain sequences (supplied by the par_report option).
-par_report3	Provides information about how successful the compilation has been (e.g. parallelized loops).
-openmp	Takes into account OpenMP directives.

For debugging, add the following compilation options:

-g	debugging
-fpp	pre-processing

How do I optimize compilation and debugging with the Intel C / C++ compiler?

Add the following compilation options:

-O3	Highest code optimization possible.
-mp	Respects IEEE standard double precision.
-ip, -ipo	Optimizes calls to a subprogram (parameter management).
-unroll	(to unroll a loop): This favors vectorization and the instructions pipeline.

Can I run applications compiled under previous OS releases?

Some applications that have been compiled under previous OS releases (typically ISV products produced under BAS 2 or RH EL AS 2) will not execute under BAS 3. At runtime, the following kind of message appears:

```
symbol _dl_loaded, version GLIBC_2.2 not defined in file ld-linux-ia64.so.2 with link time reference
```

In this case, two different and independent workarounds can be experimented:

1. Set the **LD_ASSUME_KERNEL** variable to a value like 2.4, 2.4.18, or 2.4.20, and then re-start the application.
2. If the previous workaround does not solve the issue, you can create a "dummy" library that declares only the missing symbols (which is often not used):

```
echo "char* _dl_loaded=0; " > dl_loaded.c
gcc -o libdlloaded.so -shared dl_loaded.c
export LD_PRELOAD=`pwd`/libdlloaded.so
```

I get lots of "`unaligned access`" error messages.

These are not errors, but warnings. The application made an unaligned access and the processor had to get help from the kernel to access the data. This message can be ignored but be aware that too many unaligned accesses can be a source of performance loss. To hide these messages, run:

```
prctl --unaligned=silent
```

To help debugging the program, you can run:

```
prctl --unaligned=signal
```

I have a problem with memory allocations when I use Ethernet MPICH.

Error message displayed during execution:

```
p3_1858: (18446744073792.328125) xx_shmalloc: returning NULL; requested
65584 bytes
p3_1858: (18446744073792.328125) p4_shmalloc returning NULL; request = 65584
bytes
You can increase the amount of memory by setting the environment variable
P4_GLOBMEMSIZE (in bytes)
```

The memory that the communication requires cannot be allocated correctly. To do this, run the following command:

```
export P4_GLOBMEMSIZE=100000000
```

At runtime my program hangs when I use QSNET MPI (libelan)

If the following error message appears:

```
ELAN_EXCEPTION @ 1: 5 (Memory exhausted)
elan_createSubGroup(): Failed to allocate global Vaddr for subgroup
```

Then try again to run your program after having set the MPI_USE_LIBELAN_SUB environment variable to zero using the following command:

```
export MPI_USE_LIBELAN_SUB=0
```

To run a program parallelized with OpenMP, how do I define the number of threads (processors) used?

Run the commands:

```
export  OMP_NUM_THREADS=2 to run the program on 2 processors
```

export OMP_NUM_THREADS=4 to run the program on 4 processors

B. Acronyms

API	Application Programmer Interface
BAS	Bull Advanced Server
BIOS	Basic Input Output System
BMC	Baseboard Management Controller
B-SPS	Bull Scalable Port Switch
CMOS	Complementary Metal Oxyde Semiconductor
EFI	Extensible Firmware Interface (Intel)
EIP	Encapsulated IP
EMP	Emergency Management Port
EPIC	Explicit Parallel Instruction set Computing
EULA	End User License Agreement (Microsoft)
FRU	Field Replaceable Unit
FSS	Fame Scalability Switch
GCC	GNU C Compiler
GNU	GNU's Not Unix
GPL	General Public License
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HDD	Hard Disk Drive
HPC	High Performance Computing
HSC	Hot Swap Controller
IDE	Integrated Device Electronics
IPMI	Intelligent Platform Management Interface
KSIS	Utility for Image Building and Deployment

KVM	Keyboard Video Mouse (allows to connect the keyboard, video and mouse either to the PAP, either to the node)
LUN	Logical Unit Number
MPI	Message Passing Interface
NFS	Network File System
NPTL	Native POSIX Thread Library
NTFS	New Technology File System (Microsoft)
NUMA	Non Uniform Memory Access
NVRAM	Non Volatile Random Access Memory
OEM	Original Equipment Manufacturer
OPK	OEM Preinstall Kit (Microsoft)
PAM	Platform Administration and Maintenance software
PAP	Platform Administration Processor
PAPI	Performance Application Programming Interface
PCI	Peripheral Component Interconnect (Intel)
PDU	Power Distribution Unit
PMB	Platform Management Board
PMU	Performance Monitoring Unit
PVFS	Parallel Virtual File System
PVM	Parallel Virtual Machine
QBB	Quad Brick Block
ROM	Read Only Memory
SDR	Sensor Data Record
SEL	System Event Log
SCSI	Small Computer System Interface
SM	System Management
SMP	Symmetric Multi Processing
SSH	Secure Shell
VGA	Video Graphic Adapter

Index

T

Vos remarques sur ce document / Technical publications remarks form

Titre / Title :	**Bull HPC BAS3 User's Guide**

N° Référence / Reference No. :	**86 A2 30EM Rev02**

Date / Dated :	**November 2005**

ERREURS DETECTEES / ERRORS IN PUBLICATION

AMELIORATIONS SUGGEREES / SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Vos remarques et suggestions seront attentivement examinées. Si vous désirez une réponse écrite, veuillez indiquer ci-après votre adresse postale complète.
Your comments will be promptly investigated by qualified personnel and action will be taken as required. If you require a written reply, furnish your complete mailing address below.

NOM / NAME : ---------- DATE : ----------

SOCIETE / COMPANY : ----------

ADRESSE / ADDRESS : ----------

Remettez cet imprimé à un responsable Bull S.A. ou envoyez-le directement à :
Please give this technical publications remarks form to your Bull S.A. representative or mail to:

Bull S.A.
CEDOC
Atelier de reprographie
357, Avenue Patton BP 20845
49008 ANGERS Cedex 01
FRANCE

BULL CEDOC
357 AVENUE PATTON
BP 20845
49008 ANGERS CEDEX 01
FRANCE

ORDER REFERENCE
86 A2 30EM Rev02