HPC

HPC BAS4

User's Guide

REFERENCE
86 A2 29ER 07

HPC

HPC BAS4

User's Guide

Hardware and Software

July 2007

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
86 A2 29ER 07

Trademarks and Acknowledgements

Preface

Scope and Objectives

The purpose of this guide is to describe the tools and libraries available as part of the **Bull Advanced Server** (BAS) delivery which allow the development and testing of application programs on the **Bull High Performance Computing (HPC)** clusters. In addition various open source and proprietary tools are described.

Intended Readers

This guide is for users and developers of HPC applications.

Prerequisites

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

See the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) for more details.

Structure

This guide is organized as follows:

Chapter 1.	*Introduction to the HPC Environment.* Provides a general introduction to Bull's HPC software and hardware environments.

Two types of programming libraries are used when developing programs for the HPC environment: Parallel libraries and Mathematical libraries. These are described in the chapters 2 and 3:

Chapter 2.	*Parallel Libraries.* Describes the Message Passing Interface (MPI) libraries to be used when parallel programming.
Chapter 3.	*Scientific Libraries.* Describes the scientific libraries and scientific functions delivered with the Bull HPC BAS delivery and how these should be invoked. Some of Intel's proprietary libraries are also described.
Chapter 4.	*Compilers.* Describes the compilers available and how to use them.
Chapter 5.	*The User's Environment.* Describes the user's environment on Bull HPC clusters, how the clusters are accessed and the use of the file systems. A description of Modules follows. These can be used to change and compare environments.

Chapter 6.	*Launching an Application.* Different ways of launching cluster resources for an application(s) are explained. The TORQUE batch manager used for batch jobs is described.
Chapter 7.	*Debugging Tools.* Describes Debugging Tools.

For details on system monitoring and application performance optimizations refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER).

Appendix A	A *Troubleshooting guide* which enables you to diagnose some common problems.

Glossary and Acronyms
Provides a Glossary and lists the Acronyms used in the manual.

Bibliography

- Bull HPC BAS4 *Installation and Configuration Guide* (86 A2 28ER)
- Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER)
- Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER)
- Bull HPC BAS4 *Maintenance Guide* (86 A2 46ER)
- The Bull HPC BAS4 *Software Release Bulletin* (SRB) provides release-specific information and details of restrictions resulting from known problems.
- Bull *Voltaire Switches Documentation CD* (86 A2 79ET 00)
- NovaScale 40xx Series and NovaScale 5xxx & 6xxx Series documentation
- NovaScale Master documentation
- Intel® Itanium® 2 *Processor Reference Manual for Software Development and Optimization*

Web Links

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

http://www.quadrics.com

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

http://www.linuxhpc.org/

Highlighting

- Commands entered by the user are in a frame in "Courier" font. Example:

```
mkdir /var/lib/newdir
```

- Commands, files, directories and other items whose names are predefined by the system are in "Bold". Example:
  The **/etc/sysconfig/dump** file.

- Text and messages displayed by the system to illustrate explanations are in "Courier New" font. Example:
  `BIOS Intel`
- Text for values to be entered in by the user is in "`Courier New`". Example:
  `COM1`
- *Italics* Identifies referenced publications, chapters, sections, figures, and tables.
- < > identifies parameters to be supplied by the user. Example:
  `<node_name>`

Warning

A Warning notice indicates an action that could cause damage to a program, device, system, or data.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction to the HPC Environment

The term HPC (High Performance Computing) describes the development of large scientific applications and programs, which require a powerful computation facility that can process enormous amounts of data to give highly precise results.

Bull Advanced Server (BAS) is a software suite that is used to operate and manage a Bull HPC cluster. **Bull HPC** clusters are based on Bull NovaScale platforms connected either by a high-speed interconnect **QsNet**^{II} network from **Quadrics** or using **InfiniBand** stacks with **Voltaire** switches or with a **Gigabit Ethernet** network. **BAS** includes both Bull proprietary and Open Source software, which provides the infrastructure for optimal interconnect performance.

The Bull HPC cluster includes an administrative network based on a 10/100 Mbit or a Gigabit Ethernet network, and a separate console management network.

Its key features are:

- Excellent reliability, resulting from the experience Bull has in designing large systems and their associated features.
- Lower-cost machines, due to the use of **Intel**® CPUs and industry-standard components.
- Modular systems, with strong scalability through the use of Bull's **FAME** scalability switch.
- The **Itanium**® **2** processor featuring the Explicitly Parallel Instruction Computing (EPIC) architecture so that parallelism of instructions is achieved through the compiler, rather than in the hardware of the CPU. Itanium systems offer particularly strong floating-point performance as a result.

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

This manual describes these software components, and explains how to work within the BAS environment.

See the Bull HPC BAS4 *Application Tuning Guide (86 A2 19ER)* for more information on performance optimization tools.

1.1 Hardware Configuration

The cluster architecture and system node localisation may differ from one configuration to another.

A typical cluster infrastructure consists of **Compute Nodes** for intensive calculation and **Service Nodes** for management, storage and software development services.

Compute Nodes are optimized for code execution; limited daemons run on them. Usually, these nodes are not used to write to disk, but instead transfer data to Service Nodes.

Service Node(s) include the following node types:

- **Input/Output Node(s)** that are optimised to read data from/write data to disk sub-systems efficiently.
- A **Management Node** to administrate, manage and exploit the cluster.
- **Login Node(s)** to access the cluster, and to submit jobs. If not already dedicated to this purpose, a Service Node can be configured to manage the login activity.
- **Other node(s)** may support services, or act as a server for both parallel and distributed file-systems.

Different networks, dedicated to particular functions, are used:

- **High speed interconnect** switches 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. The Administrative Network is based on an Ethernet 10/100 Mbit or a Gigabit Ethernet switch.
- **A Backbone** to link the HPC system and the external world.

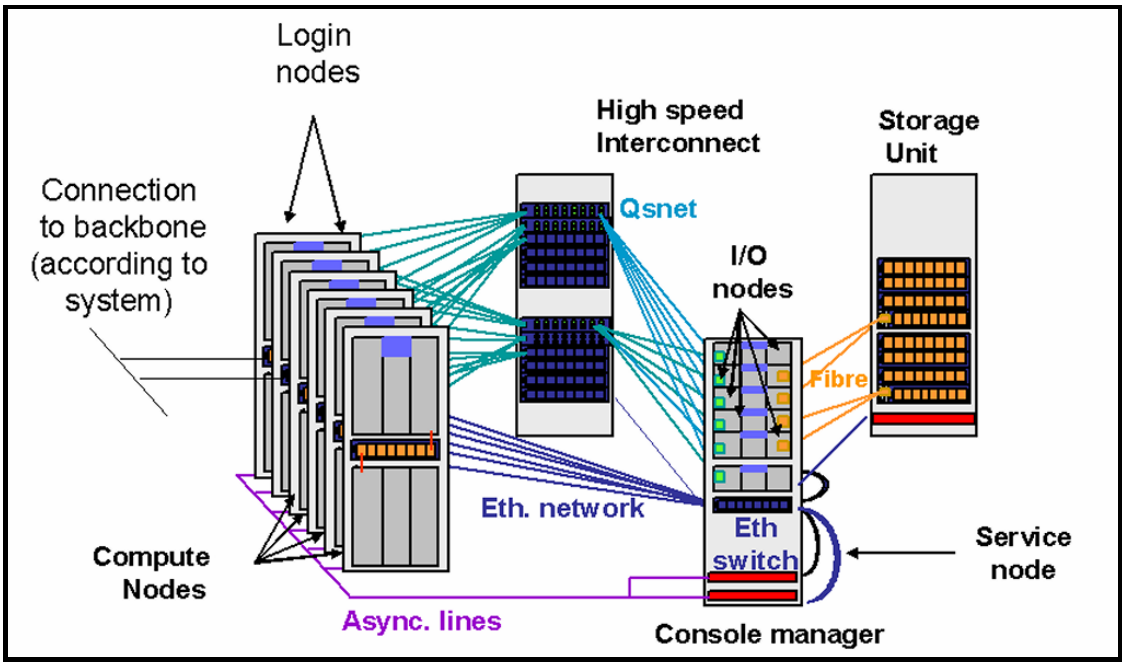


Figure 1-1. Typical architecture for HPC systems

Console Network

For **NovaScale 3045 Series** platforms the **Console Network** uses the **Digiboard PortServer TS8/16** as a console concentrator, or **Serial Over LAN** technology. Using a serial network the administrator can run and log console commands, such as starting, stopping, and booting nodes from the Management Node via the **ConMan** console management facility, or through the use of **IPMI_tools**. Conman uses the telnet protocol over Ethernet and serial lines, and **IPMI_tools** uses the **SOL** to access NovaScale 3045's **BMC**.

Some of the key requirements of Linux large-scale production clusters are the need to provide a high level of reliability using the High Availability mechanism, the need to continue to operate when part of the system fails, and the need to be able to make repairs to the system whilst it is in productive use. With its partners Bull has implemented redundancy and recovery mechanism at different levels - node, interconnect, Service Node, disk sub-system, and file system - to provide customers with cluster high availability, and easy serviceability.

Interconnection

Bull HPC cluster provides high-speed connectivity for parallel applications. Supported switches are Gigabit Ethernet, Quadrics QsNet^{II} or Voltaire switches. All nodes of the cluster are directly attached to the interconnect using at least one adapter per node. For Quadrics based systems, the adapter is a Quadrics Elan4 Network Adapter. For Gigabit Ethernet-based systems, network connections use separate adapters that are available as options to the servers.

InfiniBand interconnects combine open standards with high bandwidth, low latency performance. For InfiniBand networks the interconnection generally uses Voltaire® devices including:

- HCA 400 Ex-D Double Date Rate (DDR) Host Channel Adapters which provide up to 20 Gbs per second bandwidth exploiting the PCI-Express 8x slot.
- Voltaire® ISR 9024 Grid Switches to route up to 24 InfiniBand DDR ports per switch.
- Voltaire® ISR 9096 and 9288 Grid Directors are used to scale up clusters that include 400 Ex-D HCAs and ISR 9024 switches. The ISR 9096 and ISR 9288 Grid Directors house 96 and 288 InfiniBand DDR ports, respectively, and can be used to build extremely large clusters featuring thousands of nodes.

Storage Systems

Each node must have an internal disk drive. Optionally, some nodes may be attached to external storage systems which provide more capacity and RAID protection. External storage is mandatory for nodes which operate with the High Availability mechanism. High Availability ensures mutual takeover in the event of a problem. Typical storage systems are Bull StoreWay FDA, or DataDirect Networks S2A appliances, a storage system designed for HPC systems. Both local and external storage systems are monitored using NovaScale Master – HPC Edition.

For NovaScale 5xxx and 4xxx series platforms the supported fibre HBAs are PCI-X, LP11000 (1 port) and LP11002 (2 ports). For NovaScale 3045 series platform the supported fibre HBAs are PCI-e, LPe1150 (1 port) and LPe11002 (2 ports).

1.2 Software Configuration

1.2.1 Operating System and Installation

BAS is based on a standard Linux distribution, combined with a number of Open Source applications that exploit the best from the Open Systems community. This combined with technology from Bull and its partners, results in a powerful, complete solution for the development, execution, and management of parallel and serial applications simultaneously.

Its key features are:

- Strong manageability, through Bull's systems management suite that is linked to state-of-the-art workload management software.
- High-bandwidth, low-latency interconnect networks.
- Scalable high performance file systems, both distributed and parallel.

All cluster nodes use the same Linux distribution. Parallel commands are provided to supply users and system administrators with single-system attributes, these make it easier to manage and to use cluster resources.

Software installation is carried out by first creating an image on a node, loading this image onto the Management Node, and then distributing it to the other nodes using the **Image Building and Deployment** (KSIS) utility. This distribution is performed via the administration network.

1.2.2 Cluster Management Tools

System administrators perform most of their administrative tasks from a node designated as the **Management Node**, which is also used to start booting clusters. The Management Node also hosts daemons for system management. On the Management Node all the system configuration data is stored in a SQL based database, entitled **ClusterDB**.

To simplify the use and management of Bull Itanium server clusters, Bull has created the **Bull extension for Cluster Management**, which allows the entire HPC cluster to be managed, and used, as easily as a single server. Cluster Management includes **ClusterDB** and the following tools:

- Hardware management tools **PAM** and **NScommands** which provide immediate insight into the status and configuration of the system. These can be used to operate, monitor, and configure the server. PAM commands are available only on NovaScale 5xxx & 6xxx platforms. **NScommands** are available on all platforms including NovaScale 3045.
- Monitoring technologies for data-collection including the **Platform Administration Processor** (PAP) for NovaScale 5xxx and NovaScale 6xxx platforms or **Baseboard Management Controller** (BMC) for the NS40xx and NS3045 platforms. **NS Master - HPC edition** is used to manage alerts and provides monitoring utilities which are checked via a web browser.

- Software installation is done by first creating an image on a node, loading this image onto a **Service Node** and then distributing it to the other nodes using the **KSIS** utility. This distribution is performed over the Administration Network.
- The administrator uses **NovaScale Master- HPC Edition** to collect statistics which are based on data regarding usage and health information (hardware parameters). The data is collected from individual nodes by scalable software which may then be used to report and display performance figures at a cluster level.
- The administrator uses **Ganglia** to report and display performance figures at the cluster level.
- **NSDoctor** is a node tester used to analyze any software dysfunctions following a node's exclusion by **RMS**.
- **Postbootchecker** informs the service node that a node is booting and obtains boot time information concerning the CPUs and memory of the node. If this is different from the information already in the database, **postbootchecker** will inform the administrator.
- 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, an administrator may occasionally have to perform actions on all nodes, or on a subset of nodes, of the system. This capability is provided by the global parallel shell utility, **pdsh**.
- **mkCDrec** or **mkDVDrec** give the ability to save and restore service nodes on CD or DVD media.
- **Storage devices management** (monitoring, configuration, deployment, GUI, etc.)

1.2.3 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 and execute their applications. Applications can be executed on other cluster nodes apart from the user login system. For development, the environment consists of:

- Standard Linux tools such as **GCC** (a collection of free compilers that can compile C/C++ and FORTRAN), **GDB Gnu Debugger**, and other third-party tools including the Intel FORTRAN Compiler, the Intel C Compiler and Intel Debugger **IDB**.
- Optimized parallel libraries that are part of the **BAS** software suite. These libraries include the **MPI_Bull2** message-passing library. **Bull_MPI2** is fully integrated with the **SLURM** resource manager. **Bull_MPI2** complies with the MPI1 and 2 standards and is a high performance, high quality native implementation. **Bull_MPI2** exploits shared memory for intra-node communication and **MDM** (MPI Data Mover) technology for inter-node communication. It includes a trace and profiling tool, enabling data to be tracked.
- **Modules** software provides a means for predefining and changing environments. Each one includes a compiler, a debugger and library releases which are compatible with each other. So it is easy to invoke one given environment in order to perform tests and then compare the results with other environments.

1.2.4 Job Operation

For job execution and workload management BAS Software provides an integrated resource management, scheduling and job launch mechanism based on the **Resource Management System (RMS)** from **Quadrics** OR using **SLURM,** an Open Source resource manager.

The resource manager 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 and **SLURM** provide the means to rearrange 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 (for example, memory, or number of CPUs) to execute the jobs. 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. See *Chapter 6* for more information.

SLURM provides three key functions. Firstly, it allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for certain period of time so that they can do their work. Secondly, it provides a framework for starting, executing, and monitoring work (typically a parallel job) on a set of allocated nodes. Finally, it arbitrates when there are conflicting requests for resources by managing a queue of pending work.

1.2.5 Data and Files

Application file I/O operations may be performed using locally mounted storage devices, or alternatively, on remote storage devices using **NFS** or **Lustre** (CFS) file systems for high performance and high availability. By using a separate interconnect for administration and I/O operations, 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.

1.2.6 Exploiting the System

It is essential that users spend time familiarizing themselves with the architecture. The use of resource management tools such as **CPUSET** means that cluster operations can be configured to run in parallel or on separate partitions according to the exigencies of the job. Similarly, wherever possible, you need to optimize your code so that the parallel processing possibilities of the EPIC architecture are fully utilized.

See the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER) for more information on these subjects.

Chapter 2. Parallel Libraries

This chapter describes the following topics:

- 2.1 *Overview of Parallel Libraries*
- 2.2 *MPI_Bull 1.6.x*
- 2.3 *MPIBull2_x*
- 2.4 *Third party MPI libraries*
- 2.5 *Profiling with*

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

2.1.1 MPI Versions

Bull provides different MPI libraries 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 in a Quadrics interconnected cluster environment or on a single node. **MPI_Bull** is split 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 which is called when the program is running.
- The second generation MPI library is **MPIBull2**. This library enables dynamic communication with different device libraries; including Quadrics interconnects, InfiniBand **(IB)** interconnects, socket Ethernet/IB/EIB devices or single machine devices. See the MPIBull2 documentation for more information
- Third party MPI libraries are also available. MPICH_Ethernet is provided to allow applications to run in an Ethernet environment instead of the Quadrics interconnect environment. Bull also enables the use of LAM MPI and of Parallel Virtual Machine (PVM) – see sections 2.4.2 and 2.4.3.

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 for information in French.

2.1.2 The MPI Data Mover module (MDM)

With a standard MPI library in intra-machine communication, the sender copies data into a shared memory buffer which the receiver then copies into their own memory space. Therefore, two copies are required, as illustrated below. This system of transfer is used by the protocol **short** for messages which are under 32Kbs in size.

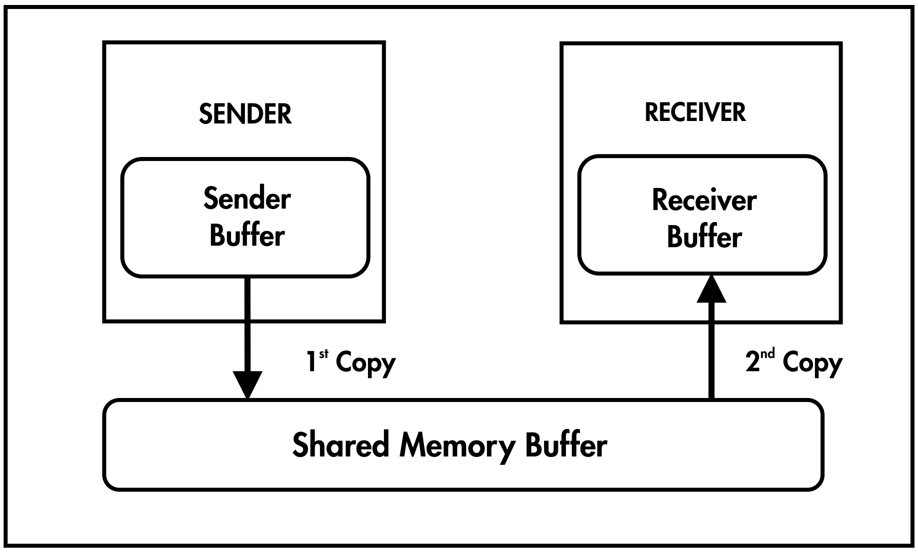


Figure 2-1. MPI Data Mover module – short protocol

With MPI_Bull and MPIBull2, the MDM module enables the use of only one copy, by directly copying the source buffer into the destination one, as illustrated below:

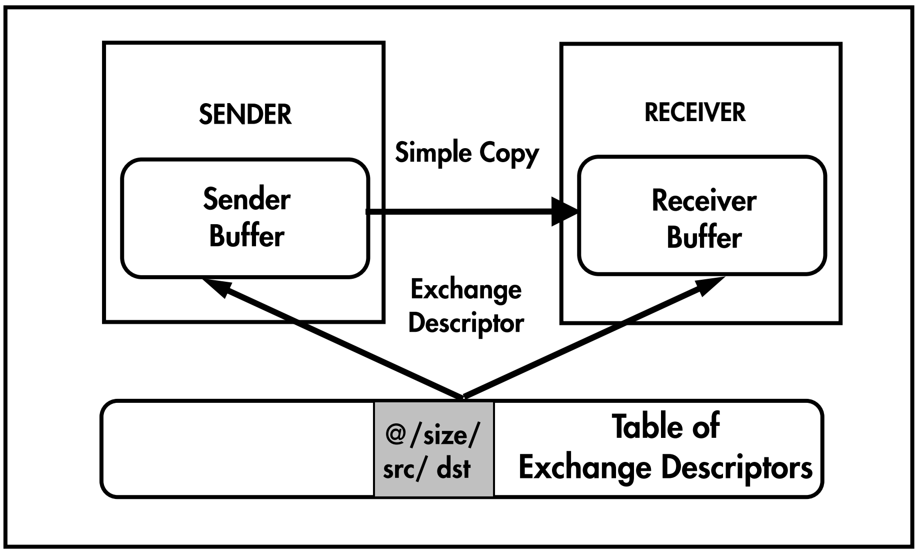


Figure 2-2. MPI Data Mover module – one copy

For messages which are bigger than a determined threshold in size the MDM module is used automatically by MPI_Bull and MPIBull2.

The **MDM** module was created from the **zcopy** module and allows the transmission of long-lasting communications (Memory window allocation) which are used in one-sided communications (MPI-2 reference).

The **MDM** module works in the same way for both **MPI_Bull** version 1.6.x and **MPIBull2** version 2.1.0-x.

2.1.2.1 Using the MDM Module

The **MDM** module being an integral part of **MPI_Bull** and **MPIBull2** has no specific options. However it includes a trace and profiling tool, enabling data to be tracked.

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

```
$ cat /proc/mdm/trace/CPU0
ITC      UID    DESCRIPTION
==================================================
```

You may also watch, in real time, the events occurring for the process whose rank is r and for the application whose MPI jobkey is p, use **/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}
```

2.2 MPI_Bull 1.6.x

Important
MPI_Bull is not supported on **InfiniBand** software stacks.

Bull has perfected a MPI 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 to the MPI -1 standard (refer to *MPI: A Message-Passing Interface Standard*, dated May 5, 1994). http://www-unix.mcs.anl.gov/mpi

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

The BAS4 **MPI_Bull** libraries have been compiled with compilers which correspond to the compilers referred to in chapter 4. This ensures compiler compatibility for any application which uses the BAS4 **MPI_Bull** libraries and these Intel compilers.

MPI_Bull has been developed from the **MPICH** 1.2.6 Open Source library to which several improvements have been made:

- **Mapping of processes on processors**, in order to inhibit scheduler actions (the ideal operation is one process per 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, thanks to 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 type of functionality.
- **Introduction of a MPI_Bull profiler called Profilecomm** to allow the profiling of applications using MPI_Bull.
- **Integration of Quadrics** Elan library.
- **Integration of mono library** to allow the running of applications on a single node.

Note:
The following restriction applies to the MPI I/O library delivered with **MPI_BULL**.

All nonblocking MPI I/O functions use an **MPIO_Request** object instead of the usual **MPI_Request** object. Accordingly, two functions, **MPIO_Test** and **MPIO_Wait**, are provided to test and wait on these **MPIO_Request** objects. These have the same semantics as **MPI_Test** and **MPI_Wait**, as shown below:

```
int MPIO_Test(MPIO_Request *request, int *flag, MPI_Status*status)
int MPIO_Wait(MPIO_Request *request, MPI_Status *status)
```

The usual functions, for example, **MPI_Test**, **MPI_Wait**, **MPI_Testany**, and so forth, will not work for nonblocking I/O.

This restriction does not exist for the **MPIBULL2** library.

Note:

For more information about **MPICH 1.2.6** Open Source library, please refer to:
http://www-unix.mcs.anl.gov/mpi/mpich/

2.2.1 MPI_Bull environments

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

Example: the following command compiles the **appli.c** code using the MPI library:

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

To produce a dynamically link object, you must set the environment variable LD_LIBRARY_PATH as follows:

```
$ export LD_LIBRARY_PATH=/usr/lib/mpishared:$LD_LIBRARY_PATH
```

Then compile the **appli.c** code using the **–shlib** option:

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

What follows explains how to launch the application either using **Quadrics** interconnects or in a single node environment.

2.2.2 MPI_Bull and the Quadrics Interconnect cluster environment

A parallel application which uses **MPI_Bull** Message Passing Interface is launched with the **RMS prun** supplied by Quadrics.

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

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

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

```
 $ prun -h.
```

For more information about **RMS** use and configuration, please refer to the Quadrics documentation. For more information about launching an application refer to chapter 6 in this manual.

2.2.3 Bull Mono libraries for the Single Node Environment

In the single node environment, used for testing a program for example, a parallel application may be launched with the **mprun** command which is included as part of the **Bull Mono Libraries**. These libraries are included in the Bull delivery which is provided for Symmetric Multi Processing on a single node.

mprun provides the following options:

-tv	Enable **Totalview** ™ Debugger support
-O	Allows resources to be over-committed. Set this flag to run more than one process per CPU.
-I	Allocate CPUs immediately or fail. By default, **mprun** holds until resources become available.
-C	Use cyclic cpusets.
-n <nprocs>	Specifies the number <nprocs> of MPI processes to start.

For example, the following command launches the **appli.exe** application on 4 processes (-**n** flag).

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

-c <cpus>	Specifies the number of CPUs required per process (default 1).
-M <rank>	Bind the process of rank <rank> to a master cpuset rather than to a given CPU. Mainly used in master/slave programs.
-Y <type>	Specify the MPI busy wait strategy: Allowed values are **s** for sched_yield(), **l** for select() and **n** for none.
-i <mode>	Specifies how standard input is redirected. See **mprun** man page for more details.
-o <mode>	Specifies how standard output is redirected. See **mprun** man page for more details.
-e <mode>	Specifies how standard error is redirected. See **mprun** man page for more details.

By default, when running a parallel program, **mprun** forwards standard input to process **0**.

2.3 MPIBull2_x

MPIBull2_1.0-x is a new MPI library which conforms to the MPI-2 standard.

2.3.1 Quick Start for MPIBull2_x

MPIBULL2 is installed in the **/opt/mpi/mpibull2-x** directory. The environmental variables MPI_HOME, PATH, LD_LIBRARY_PATH, MAN_PATH, PYTHON_PATH will need to be set or updated. These variables should not be set by the user. Use the **setenv_mpibull2** environment setting file, which may be sourced from the **/install_path/share** directory by a user or added to the profile for all users by the administrator.

2.3.2 MPIBull2 Compilers

The MPIBull2 library has been compiled with the latest Intel compilers, which, according to Bull's test farms, are the fastest ones available for the IA64 architecture. Bull uses Intel **Icc** and **Ifort** compilers to compile the MPI libraries. It is possible for the user to use their own compilers to compile their applications for example **gcc**, however see the note below.

Important:

It will be necessary to use Intel's runtime libraries when executing the application in order to exploit the EPIC architecture of the Itanium ® 2 processors. It may be necessary to consult the administrator in the event of any compatibility or compiling problems.

In order to check the configuration and the compilers used to compile the MPI libraries the following command may be used.

```
install_path/share/doc/compilers_version
```

MPI applications should be compiled using the MPIBull2 MPI wrapper to compilers:

C programs:	mpicc your-code.c	
C++ programs:	mpiCC your-code.cc	
	or	
	mpic++ your-code.cc	(for case-insensitive file systems)
F77 programs:	mpif77 your-code.f	
F90 programs:	mpif90 your-code.f90	

Wrappers to compilers simply add various command line flags and invoke a back-end compiler; they are not compilers in themselves.

The command below is used to override the compiler type used by the wrapper. **–cc**, **-fc -**, and **cxx** and used for C, Fortran and C++ wrappers.

```
mpi_user >>> mpicc -cc=gcc prog.c -o prog
```

When using **MPIBull2** compilation tools for the first time, the **MPIBull2** system is not aware of the user's preferences. **MPIBull2** compilation tools will use the default preferences: linking with a dynamic device, using the **osock** device (a socket with ethernet, etc.). Check that everything is OK for the program by using **mpibull2-device -c**. If necessary the user can give indications to the compiler by using the **-sd** options, or by using the MPIBULL2_COMM_DRIVER and MPIBULL2_LINK_STRATEGY environment variables. However, the communication device can be changed whenever needed by using the **mpibull2-device** tool.

2.3.3 Running MPIBull2

The MPI application requires a launching system in order to spawn the processes onto the cluster. Depending on the communication protocol used, different Resource Managers may be used.

1. If using **Quadrics** devices, users will need to submit their jobs with **RMS**, using the **prun** command. This supposes that **RMS** is installed and is fully functional. The optimal device to use in a Quadrics environment is **elanbull2**. For users with multiple clusters, and with process management heterogeneity, a launching abstraction tool is provided, **mpibull2-launch**.
2. As an option Bull provides the Bull provides the SLURM sub-system processes using the **InfiniBand** communication protocols. For more information see section 6.3.1 in this manual which describes the SRUN command.
3. When running a single machine the device to use is **oshm**. First launch **mpd** and then run your application with **mpiexec**.

Note:
When launching jobs within only one node, the **oshm** device has a better performance than **osock**.

4. On an Ethernet cluster which uses the **osock** device, the **MPD** system can be used. Use **mpdboot** to create a launching ring on the cluster, or use the **mpd** command in the background on a single machine, as shown below.

Create the hosts list

```
mpi_user >>> export cluster_machines="host1 host2 host3 host4"
```

Create the file used to store host information

```
mpi_user >>> for i in $cluster_machines; do echo "$i" >> machinefiles; done
```

Boot the MPD system on all the hosts

```
mpi_user >>> mpdboot -n $(cat machinefiles | wc -l) -f machinefiles
```

Check if everything is OK

```
mpi_user >>> mpdtrace
```

Run the application or try hostname

```
mpi_user >>> mpiexec -n 4 ./your_application
```

Please refer to the *Process Management* section to obtain more details.

2.3.4 MPIBull2_1.0.x features

MPIBull2_1.0.x includes the following features:

- It only has to be compiled once, supports the NovaScale architecture and is compatible with the more powerful interconnects.
- It is designed so that both development and testing times are reduced and it delivers high performance on IA64 NovaScale architectures
- Fully compatible with **MPICH2 MPI** libraries. Just set the library path to get all the **MPIBull2** features
- Supports both MPI 1.2 and MPI 2 standard functionalities including
  - Dynamic processes (**osock** only)
  - One-sided communications
  - Extended collectives
  - Thread safety (see the *Thread-Safety* Section below)
  - ROMIO including the latest patches developed by Bull
- Multi-device functionality: delivers high performance with an accelerated multi-device support layer for fast interconnects. The library supports:
  - Sockets-based messaging (for Ethernet, SDP, SCI and EIP)
  - Hybrid shared memory-based messaging for shared memory
  - **Quadrics** network drivers (**qxelan**, **elanbull2**)
  - InfiniBand architecture multirails driver Gen2
- Easy Runtime Selection: makes it easy and cost-effective to support multiple platforms. With MPIBull2 Library, both users and developers can select drivers at runtime easily, without modifying the application code. The application is built once and works for all interconnects supported by Bull.
- Ensures that the applications achieve a high performance with a high degree of interoperability with standard tools and architectures.
- Common features for all devices:
  - Mapping of processes on processors, in order to inhibit scheduler actions, and the use of CPUSET to place processes.
  - **FUTEX** (Fast User mode muTEX) mechanism in user mode
  - Optimization of data copy between processes located on the same machine using **KDM** functionalities.

2.3.5 Using MPIBull2

2.3.5.1 MPIBull2 Linking Strategies

Designed to reduce development and testing time, MPIBull2 presents two linking strategies to the users.

Firstly, the user can chose to build his application and link dynamically, leaving the choice of the MPI driver until later, according to which resources are available. For instance, if a small Ethernet cluster is the only resource available, the User compiles and links dynamically, using an **osock** driver, whilst waiting for access to a bigger cluster via a different Quadrics interconnect and which uses the elanbull2 driver at runtime.

Secondly, the User might want to use an out-of-the-box application, designed for a specified MPI device. Bull provides the combination of MPI Core and all its supported devices which enable the static libraries to be linked by the User's application.

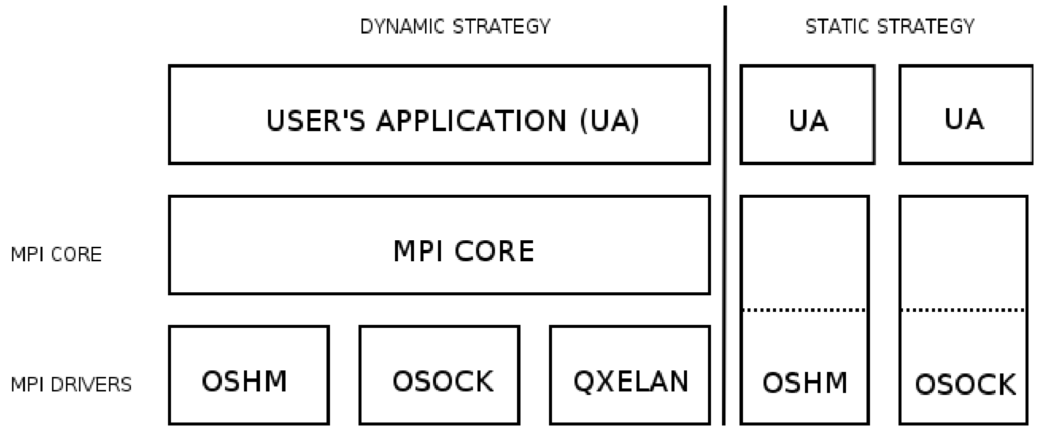


Figure 2-3. MPIBull2 Linking Strategies

2.3.5.2 Thread-safety

If the application needs an MPI Library which provides MPI_THREAD_MULTIPLE thread-safety level, then choose a device which supports TSafety and select a ***_ts device.** Use the **mpibull2-device** commands.

Note:

Thread-safety within the MPI Library requires data locking. Linking with such a library may impact performance. A loss of around 10 to 30% has been observed on microbenchmarks

Not all MPI Drivers are delivered with a thread-safe version. Devices known to support MPI_THREAD_MULTIPLE include **osock**, **oshm** and **elanbull2**.

2.3.5.3 MPIBull2 libraries

Bull provides the user with different MPI libraries. The user then selects which are to be used according to which stage of the development process they are at. Each user keeps their preferences in their **$HOME/.mpibull2** directory. The device library path is stored there; the device choice is kept in the environment, with the **MPIBULL2_COMM_DRIVER** and the **LD_LIBRARY_PATH** pointing to the right library. This preference is used when linking dynamically, which means all users are independent of each other. The preferences can be modified using **CLI** tools. A GUI can be built on top of BULL's MPI tools.

The MPI_HOME does not need to be set by the user. To use **MPIBull2** easily, the user may source the **setenv_mpibull2** file (in the **./install_path/share/ directory**), or add the **MPIBull2.modules** module file to the cluster (module file placed in the **./install_path/share/modules/ directory**). It is advisable to consul the administrator for more information on using modules. The administrator should give all users the option of loading **MPIBull2**. If for some reason it is not possible, or takes too long you may then decide to manage your personal modules in a directory of your own, adding your directory with the **module use -a** command. See the example below:

```
mpi_user >>>  module use -a /install_path/share/modules/
mpi_user >>>  module av
---------------------------- /opt/modules/modulefiles--------
autoconf/2.53                 configuration/22
intel_cc/8.1.024              intel_cc/9.1.017
intel_fc/9.0.025              intel_mkl/7.2.1
....
---------------------------- /install_path/share/modules/------
MPIBull2.modules
mpi_user >>>  module load MPIBull2.modules
```

2.3.5.4 Process management

Depending on which options have been shipped, the MPIBull2 suite comes with additional software that may be used to manage processes from beginning to end. All packages include the **MPD** system.

For **Quadrics** users, **RMS** daemons are available for use with various associated program launching tools, including **prun**, on the nodes.

For **Infiniband** users, **SLURM** can be used as well as the **MPD** process launcher. **SLURM** has to be linked with **SLURM PMI** compatible software. For instance, according to where the SLURM software is installed, a line similar to the following should be added somewhere, for example, in the file **/etc/profile.d/mpibull2***.

export MPIBULL2_PRELIBS="-lpmi"

Using an **osock** socket MPI device is easy with **RMS**, but this can also be used with the **MPICH2 MPD** ring system. The **MPD** system is a component which deploys a daemon ring on a user-by-user basis. This ring waits for the application when it is started. Refer to the **ANL** documentation and website for up to date information. In order to keep some interoperability between all process managers, the MPIBull2-launch command should be used. This tool allows users to retain their commands regardless of the process manager launcher that is being used (chose from those which are supported).

2.3.5.5 MPD ring

MPI Process Daemons (MPD) run on all nodes in a ring like structure and may be used in order to manage the launch of the different processes. Alternatively, if available, **Quadrics RMS** or **SLURM** may be used. **MPIBull2** library is **PMI** compliant which means it can interact with any other **PMI PM**, specifically, **MPD** and **RMS**. This software has been developed by **ANL**. In order to set up the system the **MPD** ring must firstly be knitted, by following the procedure below:

- At the $HOME prompt edit the **.mpd.conf** file by adding something like **MPD_SECRETWORD=your_password** and `chmod 600` to the file.
- Create a boot sequence file. Any type of file may be used. The **MPD** system will by default use the **mpd.hosts** file in your **$HOME** directory if a specific file is not specified in the boot sequence. This contains a list of hosts, separated by carriage returns. Semi-colons can be added to the host to specify the number of CPUS for the host, for example.

```
host1:4
host2:8
```

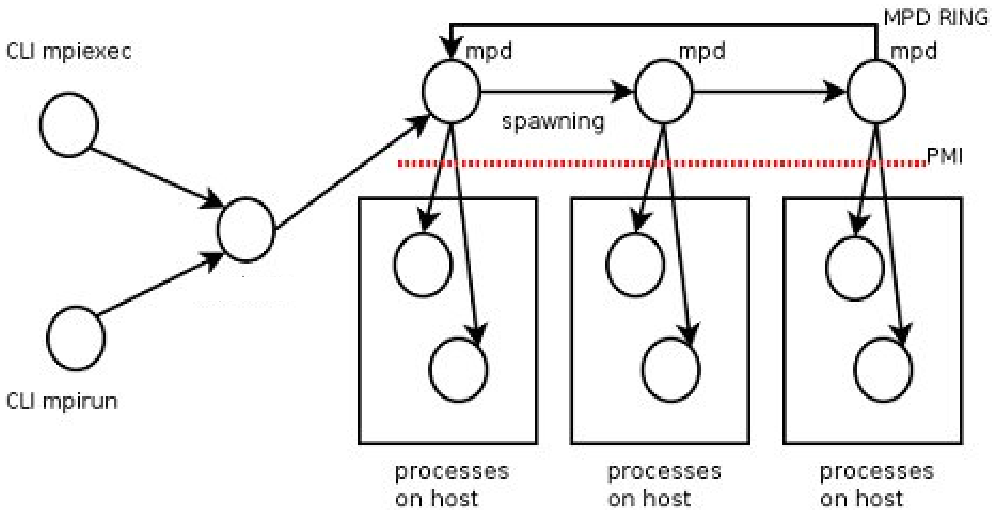


Figure 2-4. MPD ring

- Boot the ring by using the **mpdboot** command, and specify the number of hosts to be included in the ring.

```
mpdboot -n 2 -f myhosts_file
```

Check that the ring is functioning correctly by using the **mpdtrace** or **mpdringtest** commands. If everything is okay, then jobs may be run on the cluster.

2.3.6 MPIBull2 Tools

2.3.6.1 MPIBull2-devices

This is tool which may be used to change the user's preferences. It can also be used to disable a library. For example, if the program has already been compiled and the intention is to use dynamic MPI Devices which have already been linked with the MPI Core, then it is now possible to specify a particular runtime device with this tool. The following options are available with **MPIBULL2-devices**

-dl	Provides list of drivers. This is also supported by MPI wrappers.
-dlv	Provides list of drivers with versions of the drivers.

```
mpi_user >>> mpibull2-devices -dl
MPIBULL2 Communication Devices :
+ Original Devices :
*oshm        : Shared Memory device, to be used on a single machine
[static][dynamic]
*osock       : Socket protocol (can be used over IPoIB, SDP, SCI...)
[static][dynamic]
*qxelan      : Quadrics Elan3/4 device driver [static][dynamic]
******
```

-c Obtains details of the user's configuration.

```
mpi_user >>> mpibull2-devices -c
MPIBULL2 home : /install_path
User prefs    :
  \__ Directory                 : /home_nfs/mpi_user/.MPIBull2/
  \__ Custom devices            : /home_nfs/mpi_user/.MPIBull2//site_libs
  \__ MPI Core flavor           : Standard / Error detection on
  \__ MPI Communication Driver  : oshm (Shared Memory device, to be used on a
single machine) [static][dynamic]
```

-d=xxx Sets the specified communication device driver, for example **qxelan** for **Quadrics** interconnects.

```
mpi_user >>> mpibull2-devices -d=qxelan
```

2.3.6.2 mpibull2-launch

This is a meta-launcher which connects to whatever process manager is specified by the user. It is used to ensure compatibility between different process manager launchers, and also to allow users to specify their custom key bindings.

The purpose of **mpibull2-launch** is to help users to retain their launching commands even when changing the process manager, for instance when moving from **Quadrics RMS** to **SLURM**. **Mpibull2-launch** also interprets user's special keybindings, in order to allow the user to retain their preferences, regardless of the cluster and the **MPI** library. This means that the user's scripts will not need changing, except for those environment variables which are required.

The **mpibull2-launch** tool provides default keybindings. The user can check them using the **--metahelp** option. If the user wishes to check some of the **CPM** (Cluster Process Manager) special commands, they should use **--options** with the **CPM** launch name command (e.g. **--options srun**)

Some tool commands and 'device' functionalities rely on the implementation of the **MPI** components. This simple tool maps keybindings to the underlying **CPM**. Therefore, a unique command can be used to launch a job on a different CPM, using the same syntax. **mpibull2-launch** system takes in account the fact that a user might want to choose their own keybindings. A template file, named **keylayout.tmp1**, may be found in the tools rpm which may be used to construct individual keybinding preferences.

Launching a job on a cluster using mpibull2-launch

By default for Quadrics devices, there is nothing to do. Otherwise, export **MPIBULL2_LAUNCHER=prun** or specify it in the command line with **–prun**.

For a **SLURM CPM** use a command similar to the one below and set **MPIBULL2_LAUNCHER=srun** to make this command compatible with the **SLURM CPM**.

```
mpibull2-launch -n 16 -N 2 -ptest ./job
```

Example for a user who wants to use the Y key for the partition

```
PM Partition to use+Y:+partition:
```

The user should edit a file using the format found in the example template, and then add custom bindings using the **–custom_keybindings** option. The + sign is used to separate the fields. The first field is the name of the command, the second the short option, with a colon if an argument is needed, and the third field is the long option.

2.3.6.3 mpiexec

This is a launcher which connects to the MPD ring.

2.3.6.4 mpirun

This is a launcher which connects to the MPD ring.

2.3.6.5 mpicc, mpiCC, mpicxx, mpif77 and mpif90

These are all compiler wrappers and are available, for C, C++, Fortran 77 and Fortran 90 languages. These allow the user to concentrate on developing the application without having to think about the internal mechanics of MPI. The man page files provide more details about wrappers.

When using compiling tools, they need to know which communication device and a linking strategy they should use. The compiling tools parse as long as some of the following conditions have been met:

- The device and linking strategy has been specified in the command line using the **-sd** options.
- The environment variables **DEF_MPIDEV**, **DEF_MPIDEV_LINK** (required to ensure compatibility), **MPIBULL2_COMM_DRIVER**, and **MPIBULL2_LINK_STRATEGY** have been set.
- The preferences have already been set up; the tools will use the device they find in the environment using the **MPIBULL2-devices** tool.
- The tools takes the system default, using dynamic socket device.

Note:
One can obtain better performance using the **–fast/-static** options to link statically with one of the dependent libraries using the commands below.

```
mpicc -static prog.c
mpicc -fast prog.c
```

2.3.7 MPIBull2 – Example of use

2.3.7.1 Setting up the devices

When compiling an application the user may wish to keep those makefiles and build files which have already been generated. **BULL** has taken this into account. The code and build files can be kept as they are. All the user needs to do is to set up a few variables or use the **MPIBULL2-devices** tool.

During the installation process, the **/etc/profile.d/mpibull2.sh** file will have been modified by the System Administrator according to the user's needs. This file determines the default settings (by default the rpm sets the **osock** socket/TCP/IP driver). It is possible to override these settings by using environment variables – this is practical as it avoids modifying makefiles - or by using the tools options. For example, the user can statically link their application against a static driver as shown below. The default linking is dynamic, and this enables drive modification during runtime. Linking statically, as shown below, overrides the user's preferences but does not change them.

```
mpi_user >>> mpicc -sd=qxelan prog.c -o prog
mpicc : Linking statically MPI library with device (qxelan)
```

The following environment variables may also be used

MPIBULL2_COMM_DRIVER	Specifies the default device to be linked against
MPIBULL2_LINK_STRATEGY	Specifies the link strategy (the default is dynamic) (required to ensure compatibility)
MPIBULL2_MPITOOLS_VERBOSE	Provides information when building (the default is verbose off)

```
mpi_user >>> export DEF_MPIDEV=qxelan
mpi_user >>> export MPIBULL2_MPITOOLS_VERBOSE=1
mpi_user >>> mpicc prog.c -o prog
mpicc : Using environment MPI variable specifications
mpicc : Linking dynamically MPI library with device (qxelan)
```

2.3.7.2 Submitting a job

If a user wants to submit a job, then according to the process management system, they can use **MPIEXEC**, **MPIRUN**, **PRUN**, **MPRUN**, **SRUN or MPIBULL2-LAUNCH** to launch the processes on the cluster (the online man pages gives details of all the options for these launchers)

2.3.7.3 Debugging

Parallel gdb

With the **mpiexec** launching tool it is possible to add the Gnu DeBugger in the global options by using **-gdb**. All the **gdb** outputs are then aggregated, indicating when there are differences between processes. The **-gdb** option is very useful as it helps to pinpoint faulty code very quickly without the need of intervention by external software.

Refer to the **gdb** man page for more details about the options which are available.

Totalview

Totalview is a proprietary software application and is not included in the BAS distribution. See chapter 7 for more details.

It is possible to submit jobs using **Quadrics RMS** software using a command in a format similar to the one below or via MPD.

```
totalview prun -a <args> ./prog <progs_args>
```

Alternatively, it is possible to use MPI process daemons (**MPD**) and to synchronize **Totalview** with the processes running on the MPD ring.

```
mpiexec -tv <args> ./prog <progs_args>
```

2.4 Third party MPI libraries

2.4.1 MPICH_Ethernet

Bull supplies **MPICH_Ethernet** (version 1.2.6), this is to be used with Ethernet interconnects.

Modify the file **/opt/mpi/mpich_ethernet-1.2.6/share/machines.LINUX** in order to set the host name of the corresponding interface (Administration Network or Dedicated Network) and the number of processors for each machine. For example:

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

The program which uses **MPICH_Ethernet** must be compiled using the appropriate wrapping tool, for example **mpicc**, **mpif77**, etc. Launch the program with the following command 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* for the device ch_p4 which is available from http://www-unix.mcs.anl.gov/mpi/mpich/

2.4.2 LAM MPI

Bull delivers **LAM version -7.0.6-5** on the Bull Linux AS4 V5.1 DVD. However, this is not installed automatically.

For more information on **LAM/MPI** and to download the latest source files, please refer to www.lam-mpi.org .

2.4.3 Parallel Virtual Machine

Bull delivers **PVM version -3.4.4-21** on the Bull Linux AS4 V5.1 DVD. This is installed on Compute and Login Nodes.

For more information on **PVM** and to download the latest source files, please refer to www.csm.ornl.gov/pvm/pvm_home.html

2.5 Profiling with mpianalyser

mpianalyser is a profiling tool, developed by Bull for its own **MPI_Bull** implementation. **mpianalyser** includes **profilecom**, a non-intrusive tool that allows the display of data which has been logged from counters when the application runs.

For more information on **profilecomm** and how it should be used refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER)

Chapter 3. Scientific Libraries

This chapter describes the following topics:

3.1 Overview

Scientific Libraries are sets of tested, validated and optimized functions which spare users the need to develop such subprograms themselves.

The advantages of these scientific libraries are:

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

Delivery

The scientific libraries **BLACS**, **SCALAPACK**, **FFTW**, **Blocksolve95**, **SuperLU**, **PETSC** use **MPI** (Message Passing Interface). They are delivered in different environmental versions according to the implementation to be used. There are currently two implementations:

- **MPIBull** for single nodes or clusters using the Quadrics interconnect
- **MPICH_Ethernet** for clusters using Gigabit Ethernet interconnect

3.2 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**.

The Intel Cluster **MKL** is a fully thread-safe library.

An installation notice is provided by Bull with the library delivery.

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

To use it, the environment has to be set by updating the **LD_LIBRARY_PATH** variable:

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

Example for **MKL** 7.2:

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

3.3 Intel Cluster Math Kernel Library

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

The Intel Cluster MKL is a fully thread-safe library and provides **C** and **Fortran** interfaces.

An installation notice is provided by Bull with the library delivery.

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

3.4 BLAS

BLAS stands for Basic Linear Algebra Subprograms.

This library contains linear algebraic operations that include matrixes and vectors. Its functions are separated 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.

3.5 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 and so on. It can be compiled on top of **MPI** or **PVM** systems.

BLACS uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs.

3.5.1 Using BLACS

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

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

```
$ ls /usr/lib/blacs/blacs*
libblacsCinit_MPI-LINUX-0.a
libblacsF77init_MPI-LINUX-0.a
libblacs_MPI-LINUX-0.a

$
```

3.5.2 Installing and Compiling BLACS

Install the rpm File

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

It is installed using the following commands:

- To use it with MPIBull

```
$rpm -ivh --relocate /opt/blacs/ blacs-mpi_bull<release_nb>.rpm
```

- To use it with MPICH_Ethernet.

```
$rpm -ivh --relocate /opt/blacs/ blacs-mpich_ethernet<release_nb>.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 sourcing **iccvars.sh** and **ifortvars.sh**, these are available after the compilers have been installed.

```
make mpi   what=clean

make mpi
```

Compile the Tester

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

Tests

The tester is launched by 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:
The tester must be launched with at least 4 processes.

3.6 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.

3.7 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 is not written for a parallel architecture.

This library is included in the Intel MKL package.

3.8 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 above all used for eigenvalue problems and factorizations (LU, Cholesky and QR). Matrices are distributed using BLACS.

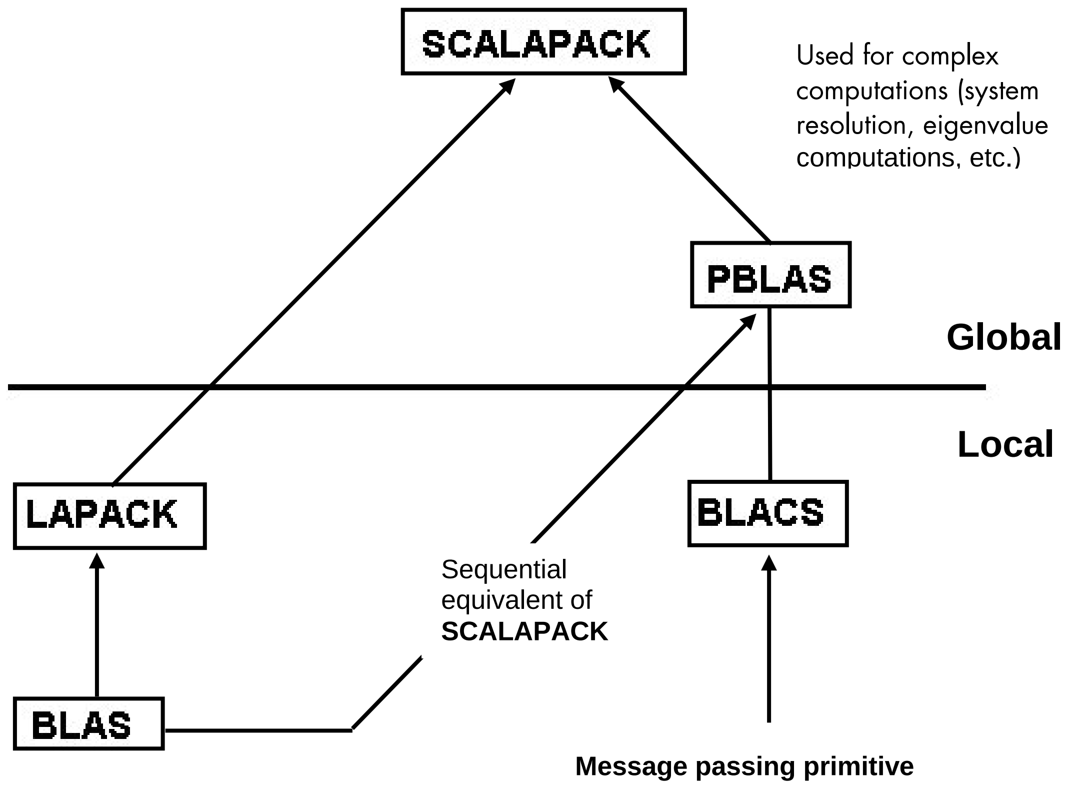

SCALAPACK
Used for complex computations (system resolution, eigenvalue computations, etc.)
PBLAS
Global
Local
LAPACK
BLACS
Sequential equivalent of SCALAPACK
BLAS
Message passing primitive

Figure 3-1. Interdependence of the different mathematical libraries

3.8.1 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.

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

3.8.2 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 commands:

- To use it with MPIBull

```
$rpm -ivh --relocate /opt/scalapack/ scalapack-mpi_bull<release_nb>.rpm
```

- To use it with MPICH_Ethernet

```
$rpm -ivh --relocate /opt/scalapack/ scalapack-mpich_ethernet<release_nb>.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 sourcing **iccvars.sh** and **ifortvars.sh**, these are available when the compilers have been 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 by 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 resulting in a total of 70 different test programs.

The variable **TOTMEM** type **PARAMETER** can be modified according to the memory available for the system test(s).

The following shell scripts execute all the tests:

For 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
```

For 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* and *xdgsep* will end with an address error, when the library is compiled with the -O3, -O2 or -O1 options. When the library is compiled entirely with -O0, all the tests except *xzsep* will run.

3.9 Blocksolve95

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

Blocksolve95 uses MPI and thus it is delivered in two releases, corresponding to the two available MPIs.

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

3.9.1 Installing and Compiling Blocksolve95

Blocksolve95 is available on the HPC CD.

Use the following commands to install BlockSolve:

To use it with MPIBull.

```
$rpm -ivh --relocate /opt/blocksolve95/ BlockSolve95-mpi_bull<release_nb>.rpm
```

To use it with MPICH_Ethernet.

```
$rpm -ivh --relocate /opt/blocksolve95/ BlockSolve95-
mpich_ethernet_bull<release_nb>.rpm
```

3.10 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:
The interfaces for some functions have been modified between versions 2.0 and 3.0 and are incompatible!

- SuperLU_SMP for shared memory systems (Version 1.0, September 1999), which use POSIX threads.
- SuperLU_DIST for distributed memory systems using MPI (Version 2.0, March 2003, the latest update dates back to October 15, 2003).

3.10.1 SuperLU Serial

Using 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**.

Installing and Compiling SuperLU Serial

SUPERLU is available on the HPC CD.

To install **SUPERLU-SEQ**, use the following command:

```
$rpm -ivh --relocate /opt/SuperLU-SEQ/ SuperLU-SEQ<release_nb>.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 sourcing **iccvars.sh** and **ifortvars.sh** which are available when 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.

Then 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 in its standard input.

A shell script which 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
$
```

3.10.2 SuperLU_SMP

Using SuperLU_SMP

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

Installing and Compiling SuperLU_SMP

SUPERLU_SMP is available on the HPC CD.

To install SUPERLU_SMP use the following command:

```
$rpm -ivh --relocate /opt/SuperLU-SMP/ SuperLU-SMP<release_nb>.rpm
```

Examples

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

A shell script is shown which starts all the programs:

```
#! /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
```

3.10.3 SuperLU_DIST

SuperLU_DIST uses **MPI** and thus it is delivered in two releases, corresponding to the two available MPIs.

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

Installing and Compiling SuperLU_DIST.

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

SuperLU-DIST is available on the HPC CD.

To install **SuperLU_DIST** use the following commands:

- To use it with **MPIBull**:

```
$rpm -ivh --relocate /opt/SuperLU_DIST/ SuperLU_DIST-mpi_bull<release_nb>.rpm
```

- To use it with **MPICH_Ethernet**:

```
$rpm -ivh --relocate /opt/SuperLU_DIST/ SuperLU_DIST-
mpich_ethernet<release_nb>.rpm
```

Update the Environment

Set the compilers environment sourcing **iccvars.sh** and **ifortvars.sh** which are available when 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

A shell script that executes all the examples in the folder **EXAMPLE** is shown on the next page:

```
#! /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 **MPIBull** 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
```

3.11 FFTW

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

FFTW uses MPI and thus it is delivered in two releases, corresponding to the two available MPI. The library is located in the directories **/usr/lib/fftw**, **/usr/include** and **/usr/doc/**.

3.11.1 Installing and Compiling FFTW

Download

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

FFTW is available on the HPC CD.

Version

Please note the following comment regarding 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.

To install FFTW use the following commands:

- To use it with MPIBull

```
$rpm -ivh --relocate /opt/fftw/ FFTW-mpi_bull<release_nb>.rpm
```

- To use it with MPICH_Ethernet

```
$rpm -ivh --relocate /opt/fftw/ FFTW-mpich_ethernet<release_nb>.rpm
```

Update the Environment

Set the compilers environment sourcing iccvars.sh and ifortvars.sh which are available when these 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
```

3.12 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 thus it is delivered in two releases, corresponding to the two available MPIs. The library is located in the directory **/usr/lib**.

3.12.1 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.

To install **PETSc** use the following commands:

- To use it with **MPIBull**:

```
$rpm -ivh --relocate opt/PETSc/ PETSc-mpi_bull<release_nb>.rpm
```

- To use it with **MPICH_Ethernet**

```
$rpm -ivh --relocate /opt/PETSc/ PETSc-mpich_ethernet<release_nb>.rpm
```

Update the Environment

Set the compilers environment sourcing **iccvars.sh** and **ifortvars.sh** available which are available when these 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 test 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.

3.13 NETCDF

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

NetCDF is an interface for array-oriented data access and is 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.

Chapter 4. Compilers

This chapter describes the following topics:

4.1 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 instructions to be executed in parallel. The parallelism has to be detected and exploited at compiler level. Bull therefore recommends the use of Intel® C/C++ and Intel® Fortran compilers.

GNU compilers are also available. However, these compilers are unable to exploit the **EPIC** instruction set and also any program which uses **MPI_Bull** cannot be compiled\linked with **GNU** products. For **MPI_Bull** programs it is essential that Intel compilers are used.

4.2 Intel Fortran Compiler

The current version of the Intel® Fortran 95 compiler is version 9.

The main features of this compiler are:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Data preloading
- 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 also be used with this compiler. It also supports big endian encoded files. Finally, this compiler allows the execution of applications which combine 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.

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

Different versions of the compiler may be installed to ensure compatibility with the compiler version used to compile the libraries and applications on your system.

Note:
It may be necessary to contact the System Administrator to ascertain the location of the compilers on your system. The paths shown in the examples below may vary.

To specify a particular environment use the command below.

```
source /opt/intel/fc/<package_id>/bin/ifortvars.sh
```

For example:

- To use version 9.0.031 of the Fortran compiler:

```
source /opt/intel/fc/9.0.031/bin/ifortvars.sh
```

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

```
 ifort --version
```

- To obtain the documentation of the compiler:

```
/opt/intel/fc/9.0.031/doc
```

Remember that if you are using **MPI_Bull** then a compiler version has to be used which is compatible with the compiler originally used to compile the MPI library.

4.3 Intel C/C++ Compiler

The current version of the Intel C/C++ compiler is version 9.

The main features of this compiler are:

- Optimization of throughput of floating point instructions
- Optimization of inter-process calls
- Data preloading
- 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 C/C++ compilers. **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 the Intel web site www.intel.com.

Different versions of the compiler may be installed to ensure compatibility with the compiler version used to compile the libraries and applications on your system.

Note:
It may be necessary to contact the System Administrator to ascertain the location of the compilers on your system. The paths shown in the examples below may vary.

To specify a particular environment use the command below:

```
source /opt/intel/cc/<package_id>/bin/iccvars.sh
```

For example:

- To use version 9.0.037 of the C/C++ compiler:

```
source /opt/intel/cc/9.0.037/bin/iccvars.sh
```

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

```
 icc --version
```

- To obtain the documentation of the compiler:

```
/opt/intel/cc/9.0.037/doc
```

Remember that if you are using **MPI_Bull** then a compiler version has to be used which is compatible with the compiler originally used to compile the MPI library.

4.4 Intel Compiler Licenses

Three types of Intel ® compiler licenses are available:

- **Single User:** allows one user to operate the product on multiple computers as long as only one copy is in use at any given time.
- **Node-Locked:** locked to a node, allows any user who has access to this node to operate the product concurrently with other users, limited to the number of licenses purchased.
- **Floating:** locked to a network, allows any user who has access to the network server to operate the product concurrently with other users, limited to the number of licenses purchased.

The node-locked and floating licenses are managed by **FlexLM** from **Macrovision.**

License installation, and **FlexLM** configuration, may differ according to your compiler, the license type, the number of licenses purchased, and the period of support for your product. Please check the Bull Product Designation document delivered with your compiler and follow the instructions contained therein.

4.5 Intel Math Kernel Library Licenses

Intel Math Kernel Library licenses are required for each compile node on which you compile with **MKL**. However, the runtime libraries which are used on the compute nodes do not require a license fee.

4.6 GNU Compilers

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

Chapter 5. The User's Environment

This chapter describes how to access the HPC environment, how to use file systems, and how to use the modules package to switch and compare environments:

5.1 Cluster Access and Security

5.1.1 Connecting to HPC

Typical connection and use of HPC for a user are as follows:

- The user logs on to the HPC platform either through Service Nodes or through the Login Node when the configuration includes these special Login Node(s). Once logged on to a node the user can launch their jobs.
- Compilation is possible on all the nodes which have compilers installed. The best approach is that compilers reside on Login Nodes, so that they do not interfere with performance on the compute nodes.

5.1.2 Using ssh (Secure Shell)

The **ssh** command is used to access a cluster node.

Syntax:

```
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) can also be used as a command to log onto a remote machine and to execute commands on it. It replaces **rlogin** and **rsh**, and provides 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 verify his/her identity, using the appropriate protocol, before being granted access to the remote machine.

5.2 Global File Systems: NFS / Lustre

Two major kinds of file systems are generally used in a HPC environment:
NFS (distributed file system) and LUSTRE (parallel file system).

Lustre is an Open Source product under a GPL License. **Lustre** is specially designed for the needs of high performance systems with a large data bandwidth. This design means that Lustre is able to take full advantage of QSNET^{II} high flow, weak latency interconnect networks so that metadata and data is transferred efficiently.

Using Lustre

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

QSNET^{II} networks allow a flow of 900 MB/s for one rail (link). This flow may be restricted by the flow of the disk bay and depends upon the Input/Output typology.

Lustre is usually used as follows:

- Each user has a private directory under **/home_nfs.**
- Each user has a private directory under the **Lustre** file system. Generally data under a **Lustre** file system is not saved, and can be deleted by the cluster's administrator whenever he needs to. If you want to save your data, you have to copy it using **NFS**.
- The **Lustre** File System is mounted following the user's request on the specified computation nodes.
- Two possible ways of running an application on a HPC system are:
  - The code is within the **Lustre** system (it must have been copied from NFS before launch) and the results are generated under **Lustre**.
  - The code is within NFS and the results are generated within **Lustre** (output files must be defined for the application in **/mnt/lustre/user's directory**).

For example:

To copy a **NFS** file into a **Lustre** file system using **prun**, enter:

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

For details about Lustre's administration and operation refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

For information about optimizing the file system refer to the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER).

5.3 Environment Modules

Environment modules provide a great way to customize your shell environment easily, particularly on the fly.

For instance an environment can consist of one set of compatible products including a defined release of a FORTRAN compiler, a C compiler, a debugger and mathematical libraries. In this way you can easily reproduce trial conditions, 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 for the **eval** command. **Modulefiles** is a kind of flat database which uses files.

In UNIX a child process cannot modify its parent environment.
So how does Modules do this? 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 provides some 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 the **eval** command.

This is the only way that a child process can 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.

The **module** command is only used in the development environment and not in other environments such as that for administration node.

More details are available at `http://modules.sourceforge.net/`

5.3.1 An example of Modules Use

The following command gives the list of available modules on this cluster.

```
module avail

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

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

---------------------- /opt/modules/modulefiles ----------------------
oscar-modules/1.0.3 (default)
```

Modules available for the user are listed under the line /opt/modules/modulefiles.

To load a module the command is:

```
module load module_name
```

To verify the loaded modules list the command is:

```
module list
```

Using the `avail` command it is possible that some modules will be marked *(default)*:

```
module avail
```

These modules are those which have been loaded without the user specifying a module version number. For example the following commands are the same:

```
module load configuration

module load configuration/2
```

Three configurations have been created. These configurations are modules which load other modules automatically.

For example the number 2 configuration 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

Configuration/1	intel_fc –version 8.0.046 intel_cc –version 8.0.066 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/2	intel_fc –version 8.0.049 intel_cc –version 8.0.071 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/3	intel_fc –version 8.0.061 intel_cc –version 8.0.071 intel_db –version 8.1.3 intel_mkl –version 7.0.017
Configuration/4	intel_fc –version 8.0.019 intel_cc –version 8.0.022 intel_db –version 8.1.3 intel_mkl –version 7.0.017

Table 5-1. Examples of different module configurations

The use of the **load** command in the module configuration context changes the user "prompt" adding the configuration name.

The **module unload** command unloads one module.

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

```
module purge
```

By design two "configuration" modules can not be loaded simultaneously. The loading of a "configuration" module unloads the previous one.

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

5.3.2 Setting Up the Shell RC Files

Here's 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** fire 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

As 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 if they are not listed in the appropriate environment variables.

Look in **./etc/global** for some sample **rc** files (**csh.login, profile**). These will define most of the variables you are likely to need. (These files will not be tailored to all platforms. The existing ones will be targeted at GNU/Linux systems.) None of the environment variables are directly needed for a correct Modules environment.
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 you specified with the --with-etc-path=<path> option to the configure script). These files will be sourced from the users' .login and .profile .

Edit these files if you want certain modules to be automatically loaded by all users.
As shown, it only loads "null'' which does absolutely nothing.

Skeleton Shell RC ("Dot'') Files

The skeleton files provide a "default'' environment for new users when they are added to your system, this can be used if you do not have the time to set them up individually. The files are usually placed in **/etc/skel** (or wherever you specified with the --with-skel-path=<path> option to the configuration script), and contains 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 with the "**-m**'' option added to the "**useradd**'' command. A set of sample "dot'' files are located in **./etc/skel**. Copy everything but the .*.in and CVS files and directories to the skeleton directory. Edit and tailor for your system.

If you have a pre-existing set of skeleton files, then make sure the following minimum set exists: **.cshrc, .login, .kshenv, .profile**. These can be automatically updated with the command:

env HOME=/etc/skel/usr/local/Modules/default/bin/add.modules.

Inspect the new "dot" files and if they are 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 prepending the appropriate commands to initialize the Modules environment.

The user can re-run this script and it will find and remember what modules they initially loaded and then strip out the previous module initialization and restore it with an upgraded one.

If the user lacks a necessary "dot" file, 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 to execute this command for each valid user.

5.4 Module Files

Once the above steps have been performed, then it is important to have module files in each of the modulefiles directories. For example, the following module files will be installed:

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

If you don't have your own module files in /usr/local/Modules/modulefiles then copy "null'' to that directory. On some systems an empty modulefiles directory will cause a core dump, whilst on other systems there will be no problem. Use /usr/local/Modules/default/modulefiles/modules as a template for creating your own module files.

For more information run:

```
module load modules
```

You will then have ready 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, /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 does "module load netscape'', then the default netscape/451c will be used. The default can be instantly changed by editing the .version file to point to a different module file in that directory. If no .version file exists then Modules will just use the last module in the alphabetical ordered directory listing as the default.

5.4.1 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 into the best locations. Using NFS is a convenient way to distribute modules, databases and other tools on all cluster nodes. This is one way to stop using /usr/local/bin as a catch-all dump for every miscellaneous binary, especially the ones that do not get used too often.

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

We unload the sources with **tar -xzf foobar-1.2.3.tar.gz** which creates a directory in the current working directory (/tmp) named ./foobar-1.2.3.
Use the **cd** command to this directory and run the configure script.

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

This should configure the source files to place all necessary files in that location. Continue the build, for example use:

```
make
make check
make install
```

The binaries, libraries, man pages, and info pages are now placed in /usr/local/pkg/foobar/1.2.3.

In order to create the "**root**", load the **modules** module with the command

```
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. Be sure to keep your original sources somewhere.

5.4.2 Upgrading via the Modules Command

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 should not appear with future versions.

Make sure you have the new version with "module --version". If it seems stable enough, then advertise it to your more adventurous users. Once you are satisfied that it appears to work adequately well, then go into **/usr/local/Modules** 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 have not loaded a specific module command version.

5.5 The Module Command

Synopsis

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

The **Module** command provides a user interface to the Modules package. The Modules package provides for the dynamic modification 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 to 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.

5.5.1 modulefiles

modulefiles are the 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** will be 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 permits **modulefiles** to be arbitrarily complex, depending on the needs of the application and the modulefile writer. If support for extended tcl (tclX) has been configured for your installation of modules, you may use all the extended commands provided by tclX, too. **modulefiles** can be used to implement site policies regarding the access and use of applications.

A typical **modulefiles** file 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 all users independent of the 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 shells.

5.5.2 Modules 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 function, creates Modules environment variables, and saves a snapshot of the environment 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 all supported by **modulecmd**. In addition, python and perl "shells" are supported which writes the environment changes to stdout as python or perl code.

5.5.3 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 ");
```

5.5.4 Modulecmd Startup

Upon invocation modulecmd sources **rc** files which contain global, user and *modulefile* specific setups. These files are interpreted as modulefiles.

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

1. Global RC file as specified by ${MODULERCFILE } or ${MODULESHOME }/etc/rc

2. User specific module RC file ${HOME }/.modulerc

3. All .module rc and .version files found during modulefile searches.

5.5.5 Module Command Line Switches

The module command accepts command line switches as its first parameter. These may be used to control output format of all information displayed and the module behavior in the 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 on 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 (e.g. **rsh** (1)) and remote execution commands (e.g. **rdist**) get confused if there is output on **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

This is a 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

5.5.6 Module Sub-Commands

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

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

Load **modulefile** into the shell environment.

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

Remove *modulefile* from the shell environment.

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

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

```
switch modulefile1 modulefile2
swap modulefile1 modulefile2
```

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 when loaded. (It will not display any environment changes found within conditional statements).

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

List loaded modules.

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

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

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

Remove directory from the MODULEPATH environment variable.

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

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

```
update
```

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

```
clear
```

Unload all loaded *modulefiles*.

```
purge
```

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

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

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

```
apropos string
keyword string
```

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.

```
initadd 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.

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

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

```
initswitch modulefile1 modulefile2
```

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

```
initlist
```

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

```
initclear
```

5.5.7 Modules Environment Variables

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:

This is 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:

This is 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 gets executed upon each invocation of a 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.

Chapter 6. Launching an Application

This chapter describes the following topics:

6.1 Launching the Application without a Batch Manager

There are different ways of launching the application on Bull HPC platforms, without using a batch manager. These vary according to platform and application type. Refer to the table on the next page for information on the different possibilities that are available.

A second step is to ensure that once launched the execution is fully optimized. The tools and commands to be used to do this are also indicated. It is possible that the system administrator may have to intervene in order to allocate the resources for the application.

For more information on where to find these tools and how to use them, refer to the rest of this chapter and the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER). For more information on the commands for the **pdsh** shell utility, refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

Note:

For more information on **mprun**, used in a single node parallel environment, and **mpibull2-launch**, a meta-launcher which helps users retain their launching commands when changing MPI environments and process managers, refer to chapter 2 of this manual.

Platform	Application		Launching tool	Execution optimization
SMP/NUMA only	Serial		Bull Linux kernel	pexec [prog_name]
	Parallel/OpenMP			pplace [prog_name] numactl [prog_name] numactl [pplace] [prog_name]
	Parallel/MPI		mprun mpiexec	N/A
Clusters with Ethernet interconnects	Non-MPI		pdsh	pplace [prog_name] numactl [prog_name] pdsh numactl [pplace] [prog_name]
	MPI		mpiexec	N/A
Clusters with Quadrics QSNet\RMS interconnects	Serial		rmsexec	rmsexec [numactl] [prog_name]
	Parallel	OpenMP on one node	allocate then numactl on the node	prun [pplace] [prog_name]
		MPI	prun	N/A
		Hybrid (MPI + OpenMP)	prun	prun [pplace] [prog_name]
Clusters with IB/SLURM Voltaire interconnects	Serial		N/A	srun options [--cpu_bind] [--mem_bind]
	Parallel	OpenMP on one node	srun –A then numactl on the node	
		MPI	srun	
		Hybrid (MPI + OpenMP)	srun	

Table 6-1. Launching tools for different platforms

Note:
There are memory access differences for the different hardware architectures covered by this manual. **NovaScale 5xxx/6xx0 Series** platforms use the Quad Brick Board (**QBB**) hardware architecture with Non Uniform Memory Access (**NUMA**). Symmetric Multiprocessing (**SMP**) is used for **NovaScale 4xx0 Series**. **NovaScale 3005 Series** platforms have a very low **NUMA** factor which is disabled by default.

In **SMP** platforms the memory access time is stable for all processors, and the Quad Brick Board hardware model is not used. The term **QBB** for these platforms refers to the set of sockets which are attached to the Scalable Node Controller (**SNC**) on the system board for **NovaScale 4xx0** platforms, and to the Node Controller (**NC**) on the system board for **NovaScale 3005** platforms. This means that 1 QBB, which may include 1-4 single processors, is possible for the **NovaScale 4xx0** platforms, whilst for the **NovaScale 3005 Series** 2 QBBs are possible, each of which may house 1-2 dual core sockets.

6.1.1 NUMACTL

Numactl is dedicated to **single-NUMA** systems. The granularity is restricted to the QBB level for each node. The following example shows a node with 16 CPUs.

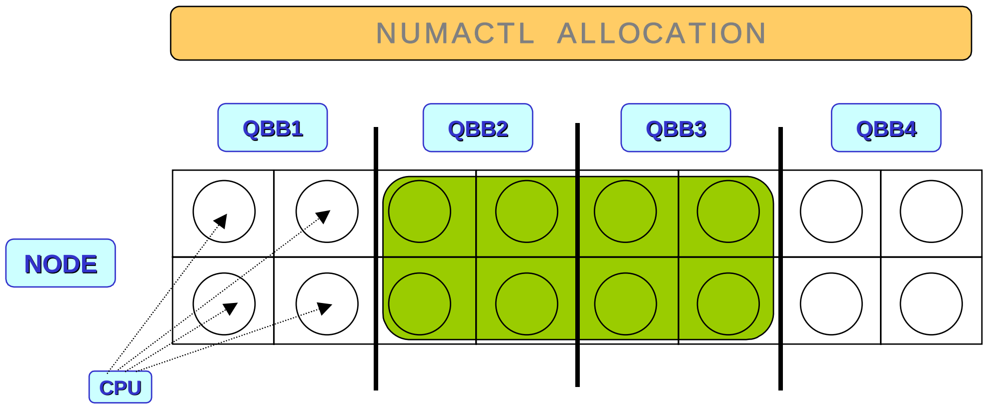


Figure 6-1. Numactl QBB application

Numactl is able to define an execution area for an application, in this example QBB2 and QBB3 (2 * 4 CPUs) are allocated.

6.1.1.1 Using Libnuma and Numactl

Important:

The scope of the **numactl** command is a mono numa configuration, with 1 to 8 QBBs that is to say, one node for a HPC cluster.

In the following paragraph concerning the numactl command, "node" means a QBB in the numa configuration.

Libnuma is a library that offers a simple programming interface to the Symmetric Multi Processing NUMA policy supported by the Linux kernel. In a **NUMA** architecture, memory areas have different latencies or bandwidths according to which CPUs they are accessed from.

Available policies are page interleaving, preferred node allocation, local allocation, allocation only on specific nodes. The binding of threads to specific nodes is also possible. All policies exist per thread and are inherited by children.

For setting global policy per process it is easiest to run **Libnuma** using the **numactl** utility. This library can be used for a more fine grained policy inside an application. Outside the application the policy applies to all the memory of the process, whereas inside you can use it for each memory zone.

The granularity level of allocation for **numactl** is the node i.e. a QBB.

All numa memory allocation policies only take effect when pages are actually faulted into the address space of a process by accessing them. The **numa_alloc_*** functions take care of this automatically.

Before any other calls in this library can be used **numa_available** must be called. When it returns a negative value all other functions in this library are undefined.

numactl runs processes with a specific **NUMA** scheduling or memory placement policy. The policy is set for a command and inherited by all of its children. In addition **numactl** can set a persistent policy for shared memory segments or files.

The most common policy settings are:

--interleave=nodes, -i nodes

Sets an memory interleave policy. Memory will be allocated using a round robin algorithm on nodes. When memory cannot be allocated on the current interleave, the target falls back to other nodes.

--membind=nodes, -m nodes

Only allocates memory from the specified nodes. Allocation will fail when there is not enough memory available on these nodes.

--cpubind=nodes, -c nodes

Only executes process on the CPUs of the nodes specified.

Note:

It is possible that this command may conflict with the usage of **CPUSETS**. As an alternative it is suggested that you use **numactl** to set your **mempolicy** set rather than **CPUSETS** and/or **taskset** or **sched_affinity** for CPU bindings.

--localalloc, -l

Always allocates locally on the current node.

Example

To run a program which allocates memory using a round robin allocation on 4 nodes of a 16 CPU NovaScale server, enter:

```
numactl -i0,1,2,3 program_name
```

For more information refer to the **numa** man pages.

Libnuma comes under the GNU Lesser General Public License, v2.1.

6.1.2 The PTOOLS and CPUSET Package

The **Ptools package** includes the **pexec** and the **pcreate** commands which can be used to create and to execute cpusets, and also to allocate resources inside an HPC node. The minimum granularity level is the CPU within a QBB. In the following example we have:

CPUSET 1 with 2 CPUs on QBB1, 2 CPUs on QBB2 and 2 CPUs on QBB3,

CPUSET 2 with 2 CPUs on QBB2 and 2 CPUs on QBB3,

CPUSET 3 with 4 CPUs on QBB4.

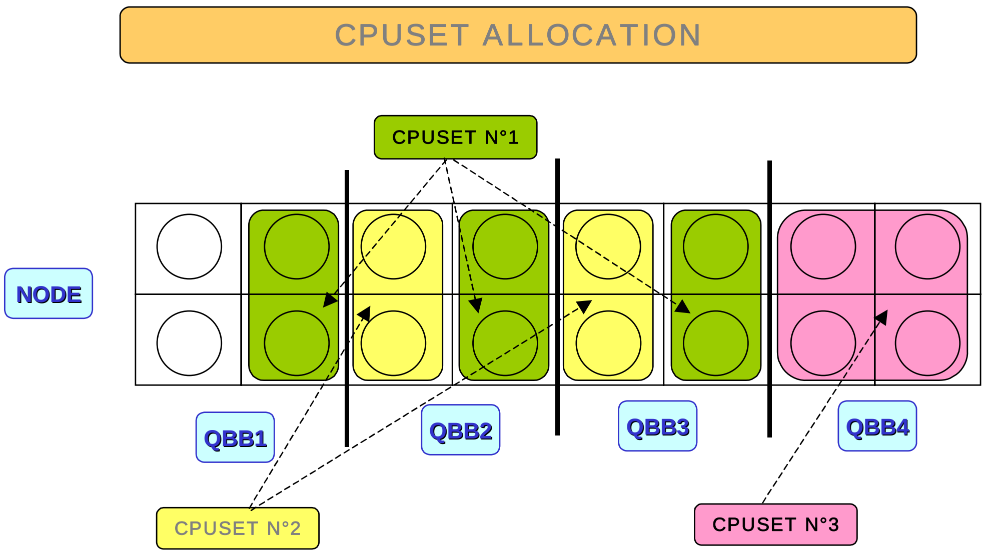


Figure 6-2. CPUSET allocation

6.1.2.1 Using Ptools and CPUSET

CPUSET is a feature of the Bull Linux kernel, which lets you define execution areas inside a multiprocessor system. The execution of each program will be limited to these predefined areas. These execution areas are called **cpusets**.

Cpusets can form a nested hierarchy meaning that **cpusets** can be created inside a **cpuset.**

Cpusets are used:

- To offer some kind of partitioning for multiprocessor systems.
- To ensure the highest performance for the execution of an application, especially on systems with a complex topology such as NUMA systems.

Cpusets also changes the way you map processes on specific processors. When a task uses the **sched_setaffinity** system call, the list of processors specified for this system call is interpreted to be used *inside* the **cpuset** in which the application is running. For example, if an application running inside a **cpuset** with processors 4, 5, 6 and 7 wants to bind one of its processes to the processor 0, the process will actually be bound to processor 4. This feature allows you to run several applications at the same time, and to finely control which processors their tasks are running on.

Bull provides the **ptools** suite to create and run **cpusets**.

ptools consists of the following:

pcreate	To create cpusets.
pexec	To create a cpuset and run an application inside it. The cpuset is destroyed when the application is completed.
passign	To move a task inside a cpuset.
pdestroy	To destroy a cpuset.
pls	To list existing cpusets.
pplace	To finely tune the binding of threads and processes for an application. See the Bull HPC BAS4 *Application Tuning Guide* (86 A2 19ER) for more details.

When a **cpuset** is created, a list of processors must be chosen. Several flags can also be set for each **cpuset**:

strict	Also called **cpu_exclusive**. This **cpuset** will not share its processors with other **cpusets** that have the same parent **cpuset**.
autoclean	To automatically remove a **cpuset** from a system and to free its resources when it becomes unused. That is to say when all the applications running inside the cpuset are finished.

Example

pexec –np <nb_cpus> --strict <my_app>

```
[root@nsadmin root]# pexec -np 2 --strict ./myapp

Created /dev/cpuset/cpuset0

Myapp running..
```

For more information refer to the installed man pages of **pexec**, **pcreate** and **passign**.

6.2 Quadrics Resource Management System

The Quadrics Resource Management System **(RMS)** includes the **prun** command to define partitions and to run jobs in a HPC cluster. One partition can lie across several nodes, as is the case for the "PARALLEL partition N°1" in the following figure:

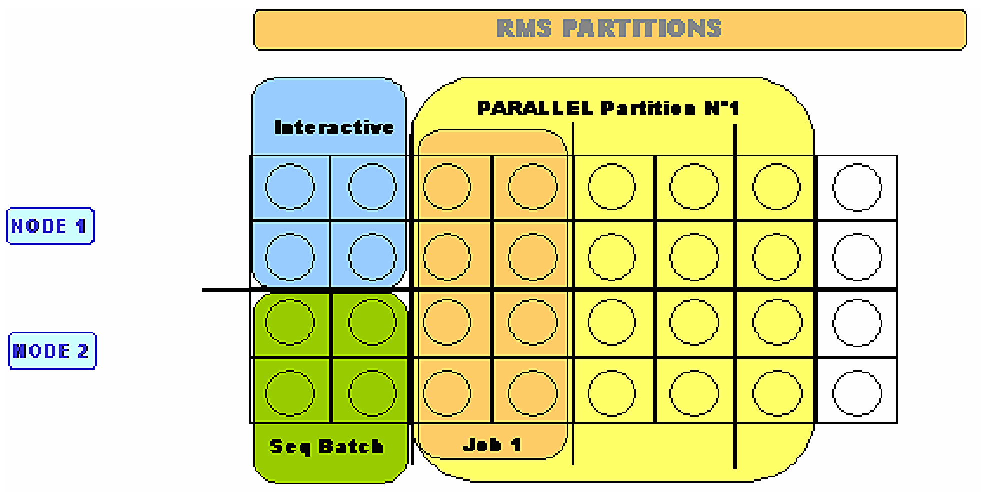


Figure 6-3. RMS Partitions

6.2.1 Using Quadrics RMS

The key to achieving high-levels of performance for a large-scale parallel application is to dedicate specific resources (CPUs, memory, network bandwidth and local I/O capability) to its execution. Quadrics RMS enables a system administrator to efficiently manage these resources to achieve maximum performance. Nodes can be configured into mutually exclusive sets known as partitions; these may each provide a specific system service. For example, your system could have an interactive partition for conventional UNIX processes and program development, a sequential batch partition, and a parallel partition running the RMS gang scheduler. Free cycles on the interactive partition could be used by sequential batch jobs running from a low priority queue. Plus the system may be configured to allow certain users to run high-priority interactive jobs during working hours.

Parallel programs under Quadrics **RMS** are managed by a controlling process **prun** and have application processes distributed over the nodes of a partition. Each process is executed by dedicated CPUs. You choose how many are required for each process, and how they are distributed over multi-CPU nodes.

The administrator of an RMS system controls how the nodes are configured into partitions, how they change, and who can access each partition and the level of resources that they use.

RMS also provides accounting facilities.

The user commands required to launch an application with **RMS** are as follows:

prun

The **prun** program loads and runs parallel programs. It can also run multiple copies of a sequential program.

rmsexec

The **rmsexec** program runs a sequential program on a lightly loaded node.

6.2.2 Prun

The main options for **prun** are as follows:

-n **<procs>**	Specifies the number of processes required
-p **<partitions>**	Specifies the partition on which to run the program
-**Rrails=<nbrails>**	Gives the number of rails to use
-**N<nodes>**	Specifies the number of nodes required
-**B<base>**	Specifies the first node to use
-**s**	Prints statistics as the job exits
-**t**	Prefix output with the process number
-**o<output_file.txt>**	Redirects output to **output_file.txt**
-**e<err_file.txt>**	Redirects errors to err_file.txt

6.2.3 Rmsexec

The **rmsexec** program provides a mechanism for running sequential programs on lightly loaded nodes with free memory or low CPU usage. It locates a suitable node and then runs the program on it. The user can select a node from a specific partition (of type login or general) with the **-p** option. Without the **-p** option **rmsexec** uses the default load-balancing partition (specified with the **lbal**-partition attribute in the attributes table). In addition, the hostname of the node can be specified explicitly. The request will fail if this node is not available according to the access rights of the user. System administrators may select any node.

Note:

This load balancing service may not be available on all types of partitions.

The main options for **rmsexec** are the following ones:

```
rmsexec [-hv] [-p partition] [-s stat] [hostname] program [args ...]
```

Use the **-h** option to get a list of the available options and valid arguments.

Selecting a Node

rmsexec restricts its search for a lightly loaded node to the partitions you are entitled to use (as defined by the system administrator). You can restrict the search still further by naming a particular partition with the **-p** option, as shown in the following example:

```
$ rmsexec -p parallel myseqprog
```

You can also request a processor on a specific node. The following example requests the node atlas2:

```
$ rmsexec atlas2 myseqprog
```

6.2.4 rinfo

rinfo is a **RMS** command used on HPC platforms with Quadrics Interconnects and which provides you with a global overview of the partitions defined by **RMS** on a cluster including the number of CPUs and machines within it. **rinfo** will also indicate the number of CPUs used when an application is executed within a partition and the state of affairs for the active applications.
This command can also be used to obtain further information on the topology of the cluster.

Example:

```
$ rinfo
```

```
MACHINE     CONFIGURATION
nsad        day

PARTITION   CPUS   STATUS     TIME   TIMELIMIT     NODES
root        28                                     ns[13-15]
nsad
part1         0/8    running  1:00:07:02             ns[13-14]
part2         ??/0   down     --:--
```

In the example above the cluster consists of 28 processors and 3 nodes: ns 13, ns 14 and ns 15. The first RMS partition is shown as ‘`part1`’ and consists of 2 nodes (ns13 and ns14) and 8 CPUs and its status is ‘running’ which means that it can be used.

The second partition is ‘`part2`’ and its status is ‘down’, with no nodes allocated, which means that it cannot be used.

6.2.5 More RMS Information

For more information, see Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) or refer to the **RMS** *User's Guide* and the Quadrics web site at http://www.quadrics.com

See the *"RMS Reference Manual"* at http://www.quadrics.com for details about other RMS commands.

6.3 SLURM Resource Management Utilities

As a cluster resource manager, SLURM has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates conflicting requests for resources by managing a queue of pending work.

Users interact with **SLURM** through various command line utilities:

- **SRUN** for submitting a job for execution and optionally controlling it interactively.
- **SBCAST** to transmit a file to all nodes running a job.
- **SCANCEL** for terminating a pending or running job.
- **SQUEUE** for monitoring job queues.
- **SINFO** for monitoring partition and overall system state.
- **SACCT** displays data for all jobs and job steps in the SLURM accounting log.
- **Global Accounting API** for merging the data from the **LSF** accounting file and the SLURM accounting file into a single record.

Note:
See the HPC BAS4 *Application Tuning Guide* for information on the Consumable Resource Scheduling Policy using the CPU Consumable Resource node allocation plug-in.

6.3.1 SRUN

SRUN submits jobs to run under **SLURM** management. **SRUN** can:

- Submit a batch job and then terminate
- Submit an interactive job and then persist to shepherd the job as it runs
- Allocate resources to a shell and then spawn that shell for use in running subordinate jobs.

SLURM associates every set of parallel tasks ("*job steps*") with the **SRUN** instance that initiated that set, and SRUN provides comprehensive control over node choice and I/O redirection for the parallel job.

6.3.1.1 SRUN Roles and Modes

SRUN executes tasks ("*jobs*") in parallel on multiple compute nodes at the same time (on machines where SLURM manages the resources). **SRUN** options allow the User to both:

- Specify the parallel environment for job(s), such as the number of nodes used, node partition, distribution of processes among nodes, and total time.
- Control the behavior of a parallel job as it runs, such as by redirecting or labeling its output, sending it signals, or specifying its reporting verbosity.

Because it performs several different roles, SRUN can be used in four distinct ways or "**modes**". These modes are described in the following table.

Mode	Description
INTERACTIVE	The simplest way to use SRUN is to distribute execution of a serial program (such as a UNIX utility) across a specified number or range of compute nodes. For example, `srun -N 8 cp ~/data1 /var/tmp/data1` copies (CP) file data1 from a common home directory to local disk space on each of eight compute nodes. SRUN allows relevant environment variables to be set on its own execute line. In interactive mode, SRUN submits job to the local SLURM job controller, then initiates all processes on the specified nodes and blocks until the requested resources become available. Many control options are available to change the details of this general pattern.
BATCH	SRUN can also directly submit complex scripts to the job queue(s) managed by SLURM for later execution, when needed resources become available and when no higher priority jobs are pending. For example, `srun -N 16 -b myscript.sh` uses the -b option of SRUN to place myscript.sh into the queue to later run on 16 nodes. Scripts in turn normally contain either MPI programs or other *simple* invocations of SRUN itself (as shown above). Thus, the -b option of SRUN supports basic, local-batch service.
ALLOCATE	The SRUN "allocate" mode can be used to combine the job complexity of scripts with the immediacy of interactive execution. For example, `srun -A -N 4 myscript.sh` uses the SRUN (uppercase) -A option to allocate specified resources (in this case, four nodes), spawn a subshell with access to those resources, and then run multiple subsequent jobs using *simple* SRUN commands within the specified script (here, myscript.sh) that the subshell immediately starts to execute.
ATTACH	To monitor or intervene in an already running SRUN job, either batch (started with -b) or interactive ("allocated", started with -A), execute SRUN again and "attach"(-a, lowercase) to that job. For example, `srun -a 6543 -j` forwards the standard output and error messages from the running job with SLURM ID 6543 to the attaching SRUN to reveal the job's current status, and (with -j, lowercase) also "joins" the job so that you can send it signals as if this SRUN had initiated the job. Omit -j for read-only attachments. Because you are attaching to a running job whose resources have already been allocated, SRUN's resource-allocation options (such as -N) are incompatible with -a.

Table 6-2. SRUN Modes

6.3.1.2 SRUN SIGNAL HANDLING

Signals sent to SRUN are automatically forwarded to the tasks that SRUN controls, with a few special cases. SRUN handles the sequence CTRL-C in different ways, depending on how many it receives in one second:

```
CTRL-Cs within one second
-------------------------
First    reports the state of all tasks
         associated with SRUN.
Second   sends SIGINT signal to all
         associated SRUN tasks.
Third    terminates the job at once,
         without waiting for remote
         tasks to exit.
```

6.3.1.3 SRUN Run-Mode Options

This section explains the ***mutually exclusive*** SRUN options that enable its different run modes. Each option has a one-character (UNIX) and a longer (Linux) alternative syntax.

NAME

SRUN - run parallel jobs

SYNOPSIS

```
srun [OPTIONS...] executable [args...]
srun --batch [OPTIONS...] job_script
srun --allocate [OPTIONS...] [job_script]
srun --attach=jobid
```

DESCRIPTION

Allocate resources and optionally initiate parallel jobs on clusters managed by SLURM.

6.3.1.4 Parallel Run Options

`-n, --ntasks=ntasks`

Specify the number of processes to run. Request that SRUN allocate ntasks processes. The default is one process per node, but note that the **-c** parameter will change this default.

`-c, --cpus-per-task=ncpus`

Request that ncpus be allocated per process. This may be useful if the job is multithreaded and requires more than one CPU per task for optimal performance. The default is one CPU per process. If **-c** is specified without **-n** as many tasks will be allocated per node as possible while satisfying the **-c** restriction.

`-N, --nodes=minnodes[-maxnodes]`

Request that a minimum of minnodes nodes be allocated to this job. The scheduler may decide to launch the job on more than minnodes nodes. A limit on the maximum node count may be specified with maxnodes (e.g. "--nodes=2-4"). The minimum and maximum node count may be the same to specify a specific number of nodes (e.g. "--nodes=2-2" will ask for two and ONLY two nodes). The partition's node limits supersede those of the job. If a job's node limits are completely outside of the range permitted for its associated partition, the job will be left in a PENDING state. Note that the environment variable SLURM_NNODES will be set to the count of nodes actually allocated to the job. See the ENVIRONMENT VARIABLES section for more information. If **-N** is not specified, the default behavior is to allocate enough nodes to satisfy the requirements of the -n and -c options.

`-r, --relative=n`

Run a job step relative to node n of the current allocation. This option may be used to spread several job steps out among the nodes of the current job. If -r is used, the current job step will begin at node n of the allocated nodelist, where the first node is considered node 0. The **-r** option is not permitted along with **-w** or **-x**, and will be silently ignored when not running within a prior allocation (i.e. when SLURM_JOBID is not set). The default for n is 0. If the value of **--nodes** exceeds the number of nodes identified with the **--relative** option, a warning message will be printed and the **--relative** option will take precedence.

`-p, --partition=partition`

Request resources from partition "partition." The SLURM administrator creates the partitions, and also identifies one of those partitions as the default.

`-P, --dependency=jobid`

Defer initiation of this job until the specified jobid has completed execution. Many jobs can share the same dependency and these jobs may belong to different users. The value may be changed after job submission using the SCONTROL command.

`--nice[=adjustment]`

Run the job with an adjusted scheduling priority. With no adjustment value, the scheduling priority is decreased by 100. The adjustment range is from -10000 (highest priority) to 10000 (lowest priority). Only privileged users can specify a negative adjustment. Note that this option is presently ignored if SchedulerType=sched/maui.

`--multi-prog`

Run a job with different programs and different arguments for each task. In this case, the executable program specified is actually a configuration file specifying the executable and the arguments for each task. See MULTIPLE PROGRAM CONFIGURATION below for details about the configuration file contents.

`--begin=time`

Defer initiation of this job until the specified time. It accepts times of the form HH:MM:SS to run a job at a specific time of day (seconds are optional). (If that time is already past, the next day is assumed.) It is also possible to specify midnight, noon, or teatime (4pm) and have a time-of-day appended with AM or PM, for running in the morning or the evening. Additionally, it is possible to specify the day on which the job will be run, by giving a date in the form month-name day with an optional year, or giving a date of the form MMDDYY, MM/DD/YY, or DD.MM.YY. Another option is to give times like now + count time-units, where the time-units can be minutes, hours, days, or weeks and SLURM can be told to run the job today with the keyword today, or to run the job tomorrow with the keyword tomorrow. The value may be changed after job submission using the SCONTROL command.

`-U, --account=account`

Change resource use by this job to specified account. The account is an arbitrary string. The account may be changed after job submission using the SCONTROL command.

`-t, --time=minutes`

Establish a time limit to terminate the job after the specified number of minutes. If the job's time limit exceeds the partition's time limit, the job will be left in a PENDING state. The default value is the partition's time limit. When the time limit is reached, the job's processes are sent SIGTERM followed by SIGKILL. The interval between signals is specified by the SLURM configuration parameter KillWait. Time limit of 0 minutes indicates that an infinite timelimit should be used.

`-D, --chdir=path`

Have the remote processes do a **chdir** to path before beginning execution. The default is to chdir to the current working directory of the SRUN process.

`-I, --immediate`

Exit if resources are not immediately available. By default, **--immediate** is off, and SRUN will block until resources become available.

`-k, --no-kill`

Do not automatically terminate a job if one of the nodes it has been allocated fails. This option is only recognized on a job allocation, not for the submission of individual job steps. The job will assume all responsibilities for fault-tolerance. The active job step (MPI job) will almost certainly suffer a fatal error, but subsequent job steps may be run if this option is specified. The default action is to terminate the job upon node failure. Note that -batch jobs will be re-queued if a node failure occurs in the process of initiating it.

`-K, --kill-on-bad-exit`

Terminate a job if any task exits with a non-zero exit code.

`-s, --share`

The job can share nodes with other running jobs. This may result in faster job initiation and higher system utilization, but lower application performance.

`-O, --overcommit`

Overcommit resources. Normally, SRUN will not allocate more than one process per CPU. Specifying --overcommit explicitly allows more than one process per CPU. However, no more than MAX_TASKS_PER_NODE tasks are permitted to execute per node.

`-T, --threads=nthreads`

Request that SRUN use nthreads to initiate and control the parallel job. The default value is the smallest of 10 or the number of nodes allocated.

`-l, --label`

Prefix task number to lines of stdout/err. Normally, stdout and stderr from remote tasks are line-buffered directly to the stdout and stderr of SRUN. The --label option will prefix lines of output with the remote task id.

`-u, --unbuffered`

Do not line buffer stdout from remote tasks. This option cannot be used with -label.

`-m, --distribution=(block|cyclic|hostfile)`

Specify an alternate distribution method for remote processes.

`block`

The block method of distribution will allocate processes in-order to the CPUs on a node. If the number of processes exceeds the number of CPUs on all of the nodes in the allocation then all nodes will be utilized. For example, consider an allocation of three nodes each with two CPUs. A four-process block distribution request will distribute those processes to the nodes with processes one and two on the first node, process three on the second node, and process four on the third node. Block distribution is the default behavior if the number of tasks exceeds the number of nodes requested.

`cyclic`

The cyclic method distributes processes in a round-robin fashion across the allocated nodes. That is, process one will be allocated to the first node, process two to the second, and so on. This is the default behavior if the number of tasks is not larger than the number of nodes requested.

`hostfile`

The hostfile method of distribution will allocate processes in the order in which they are listed in the file designated by the environment variable SLURM_HOSTFILE. If this variable is listed, it will override any other method specified. If not set, the method will default to block.

`-J, --job-name=jobname`

Specify a name for the job. The specified name will appear along with the job id number when querying running jobs on the system. The default is the supplied executable program’s name.

`--mpi=mpi_type`

Identify the type of MPI to be used. This may result in unique initiation procedures.

`list`

Lists available MPI types from which to choose.

`lam`

Initiates one `lamb` process per node and establishes necessary environment variables for LAM/MPI.

`mpich`-gm

For use with Myrinet.

`mvapich`

For use with Infiniband.

`none`

No special MPI processing. This is the default and works with many other versions of MPI.

`--ctrl-comm-ifhn=addr`

Specify the address or hostname to be used for PMI communications only (task communication and synchronization primitives for MPCIH2). The default is hostname (response from getnodename function). Use of this is required if a DNS lookup cannot be performed on the hostname or if that address is blocked from the compute nodes.

`--jobid=id`

Initiate a job step under an already allocated job with job id id. Using this option will cause SRUN to behave exactly as if the SLURM_JOBID environment variable were set.

`-o, --output=mode`

Specify the mode for stdout redirection. By default, in interactive mode, SRUN collects stdout from all tasks, and line buffers this output to the attached terminal. With --output stdout may be redirected to a file, to one file per task, or to /dev/null. If the specified file already exists, it will be overwritten. If --error is not also specified on the command line, both stdout and stderr will be directed to the file specified by --output.

`-i, --input=mode`

Specify how stdin is to be redirected. By default, SRUN redirects stdin from the terminal to all tasks.

`-e, --error=mode`

Specify how stderr is to be redirected. By default in interactive mode, SRUN redirects stderr to the same file as stdout, if one is specified. The --error option is provided to allow stdout and stderr to be redirected to different locations. If the specified file already exists, it will be overwritten.

`-b, --batch`

Submit in "batch mode." SRUN will make a copy of the executable file (a script) and submit the request for execution when resources are available. SRUN will terminate after the request has been submitted. The executable file will run on the first node allocated to the job and must contain SRUN commands to initiate parallel tasks. stdin will be redirected from /dev/null, stdout and stderr will be redirected to a file (the default is jobname.out or jobid.out in the current working directory - see -o for other IO options).

Note that if the SLURM daemons are cold-started, jobid values will be reused. Plan accordingly to avoid over-writing output and error files. The executable must be specified using either a fully-qualified pathname, or its pathname will be treated as relative to the current working directory. The search path will not be used to locate the file. The executable will be interpreted by the users default shell unless the file begins with "#!" followed by the fully-qualified pathname of a valid shell. Note that batch jobs will be re-queued if a node fails while it is being initiated.

SRUN command-line options can also be inserted into the script by prefacing the option with #SLURM. Multiple options can be on one line or multiple lines. i.e.

```
#SLURM -N 2 -n 2
#SLURM --mpi=lam
```

This is running the script on 2 nodes, with 2 procs with mpi type lam. All command-line options are able to be set inside the script with the exception of the mode (which has already been set to run a batch script since the running mode is batch). Options on the command line take precedence over options in the batch script, which in turn take precedence over existing environment variables.

`-v, --verbose`

Verbose operation. Using the -v multiple times will further increase the verbosity of SRUN. By default, only errors will be displayed.

`-d, --slurmd-debug=level`

Specify a debug level for SLURMD. “level” may be an integer value between 0 [quiet, only errors are displayed] and 4 [verbose operation]. The SLURMD debug information is copied to the stderr of the job. By default, only errors are displayed.

`-W, --wait=seconds`

Specify how long to wait after the first task terminates before terminating all remaining tasks. A value of 0 indicates an unlimited wait (a warning will be issued after 60 seconds). The default value is set by the WaitTime parameter in the SLURM configuration file (see slurm.conf). This option can be useful to insure that a job is terminated in a timely fashion in the event that one or more tasks terminate prematurely.

`-q, --quit-on-interrupt`

Quit immediately on single SIGINT (Ctrl-C). Use of this option disables the status feature normally available when SRUN receives a single Ctrl-C and causes SRUN to instead immediately terminate the running job.

`-X, --disable-status`

Disable the display of task status when SRUN receives a single SIGINT (Ctrl-C). Instead, immediately forward the SIGINT to the running job. A second Ctrl-C in one second will forcibly terminate the job and SRUN will immediately exit. May also be set via the environment variable SLURM_DISABLE_STATUS.

`-Q, --quiet`

Quiet operation. Suppress informational messages. Errors will still be displayed.

`--mail-type=type`

Notify user by email when certain event types occur. Valid type values are BEGIN, END, FAIL, ALL (any state change). The user to be notified is indicated with --mail-user.

`--mail-user=user`

User to receive email notification of state changes as defined by --mail-type. The default value is the submitting user.

`--uid=user`

Attempt to submit and/or run a job as user instead of the invoking user id. The invoking user's credentials will be used to check access permissions for the target partition. User root may use this option to run jobs as a normal user in a RootOnly partition for example. If run as root, SRUN will drop its permissions to the uid specified after node allocation is successful. "user" may be the user name or numerical user ID.

`--gid=group`

If SRUN is run as root, and the --gid option is used, submit the job with group's group access permissions. group may be the group name or the numerical group ID.

`--core=type`

Adjust corefile format for parallel job. If possible, SRUN will set up the environment for the job such that a corefile format other than full core dumps is enabled. If run with type = "list", SRUN will print a list of supported corefile format types to stdout and exit.

`--propagate[=rlimits]`

Allows users to specify which of the modifiable (soft) resource limits to propagate to the compute nodes and apply to their jobs. If rlimits is not specified, then all resource limits will be propagated.

`--prolog=executable`

SRUN will run executable just before launching the job step. The command line arguments for executable will be the command and arguments of the job step. If executable is "none", then no prolog will be run. This parameter overrides the SrunProlog parameter in slurm.conf.

`--epilog=executable`

SRUN will run executable just after the job step completes. The command line arguments for executable will be the command and arguments of the job step. If executable is "none", then no epilog will be run. This parameter overrides the SrunEpilog parameter in slurm.conf.

`--task-prolog=executable`

The SLURMD daemon will run executable just before launching each task. This will be executed after any TaskProlog parameter in slurm.conf is executed. Besides the normal environment variables, this has SLURM_TASK_PID available to identify the process ID of the task being started. Standard output from this program of the form "export NAME=value" will be used to set environment variables for the task being spawned.

`--task-epilog=executable`

The SLURMD daemon will run executable just after each task terminates. This will be before any TaskEpilog parameter in slurm.conf is executed. This is meant to be a very short-lived program. If it fails to terminate within a few seconds, it will be killed along with any descendant processes.

6.3.1.5 Allocate Options

`-A, --allocate`

Allocate resources and spawn a shell. When --allocate is specified to SRUN, no remote tasks are started. Instead a subshell is started that has access to the allocated resources. Multiple jobs can then be run on the same CPUs from within this subshell. See Allocate Mode below.

`--no-shell`

Immediately exit after allocating resources instead of spawning a shell when used with the -A, --allocate option.

6.3.1.6 Attaching To Running Job

`-a, --attach=id`

This option will attach SRUN to a running job with job id = id. Provided that the calling user has access to that running job, stdout and stderr will be redirected to the current session (assuming that the tasks' stdout and stderr are not connected directly to files). stdin is not connected to the remote tasks, and signals are not forwarded unless the --join parameter is also specified.

`-j, --join`

Used in conjunction with --attach to specify that stdin should also be connected to the remote tasks (assuming that the remote tasks' stdin are not directly connected to files), and signals sent to SRUN will be forwarded to the remote tasks.

6.3.1.7 Constraint Options

The following options all put constraints on the nodes that may be considered for the job:

`--mincpus=n`

Specify minimum number of CPUs per node.

`--mem=MB`

Specify a minimum amount of real memory.

`--tmp=MB`

Specify a minimum amount of temporary disk space.

`-C, --constraint=list`

Specify a list of constraints. The constraints are features that have been assigned to the nodes by the SLURM administrator. The list of constraints may include multiple features separated by commas, in which case all nodes must have all listed features (i.e. the features are ANDed together). Alternately, the features may be separated by a vertical bar (|), in which case all nodes must have at least one of the listed features (i.e. the features are ORed together). If no nodes have the requested features, then the SLURM job manager will reject the job.

`--contiguous`
Demand a contiguous range of nodes. The default is "yes". Specify --contiguous=no if a contiguous range of nodes is not a constraint.

`-w, --nodelist=host1,host2,... or filename`
Request a specific list of hosts. The job will contain at least these hosts. The list may be specified as a comma-separated list of hosts, a range of hosts (host[1-5,7,...] for example), or a filename. The host list will be assumed to be a filename if it contains a "/" character.

`-x, --exclude=host1,host2,... or filename`
Request that a specific list of hosts not be included in the resources allocated to this job. The host list will be assumed to be a filename if it contains a "/"character.

6.3.1.8 Affinity/Multi-core Options with task/affinity or task/numa plug-in

These options are used when the **task/affinity** or **task/numa plug-in** is enabled.

`--cpu_bind=[{quiet,verbose},]type`
Bind tasks to CPUs

- `q[uiet],`
  quietly bind before task runs (default)
- `v[erbose],`
  verbosely report binding before task runs
- `no[ne]`
  do not bind tasks to CPUs (default)
- `rank`
  bind by task rank
- `map_cpu:<list>`
  bind by mapping CPU IDs to tasks as specified where <list> is <cpuid1>,<cpuid2>,...<cpuidN>. CPU IDs are interpreted as decimal values unless they are preceded with "x" in which case they are interpreted as hexadecimal values.
- `mask_cpu:<list>`
  bind by setting CPU masks on tasks as specified where <list> is <mask1>,<mask2>,...<maskN>. CPU masks are always interpreted as hexadecimal values but can be preceded with an optional "x".

To have SLURM always report on the selected CPU binding for all SRUN commands executed in a shell, enable verbose mode separately from the command line with:

```
setenv SLURM_CPU_BIND verbose
```

SLURM_CPU_BIND will not propagate into the tasks environment (binding by default only affects the first SRUN). To propagate --cpu_bind to successive SRUN commands, first do the following in each task:

```
setenv SLURM_CPU_BIND \
${SLURM_CPU_BIND_VERBOSE},${SLURM_CPU_BIND_TYPE}${SLURM_CPU_BIND_LIST}
```

6.3.1.9 Affinity/Multi-core Options with task/affinity and NUMA memory functions

These options are used when the task/affinity plug-in is enabled and the NUMA memory functions are available

`--mem_bind=[{quiet,verbose},]type`
Bind tasks to memory. Note that the resolution of CPU and memory binding may differ on some platforms. For example, CPU binding may be performed at the level of the cores within a processor while memory binding will be performed at the level of nodes, where the definition of "nodes" may differ from system to system. The use of a type other than "none" or "local" is not recommended. For greater control, try running a simple test code with the options "--cpu_bind=verbose,none --mem_bind=verbose,none" to determine the specific configuration.

`q[uiet],`
quietly bind before task runs (default)

`v[erbose],`
verbosely report binding before task runs

`no[ne]`
do not bind tasks to memory (default)

`rank`
bind by task rank (not recommended)

`local`
Use memory local to the processor in use

`map_mem:<list>`
bind by mapping a node’s memory to tasks as specified where <list> is <cpuid1>,<cpuid2>,...<cpuidN>. CPU IDs are interpreted as decimal values unless they are preceded with “x”, in which case they are interpreted as hexadecimal values (not recommended).

`mask_mem:<list>`
bind by setting memory masks on tasks as specified where <list> is <mask1>,<mask2>,...<maskN>. Memory masks are always interpreted as hexadecimal values but can be preceded with an optional “x” (not recommended).

To have SLURM always report on the selected memory binding for all SRUN commands executed in a shell, enable verbose mode separately from the command line with:

```
setenv SLURM_MEM_BIND verbose
```

SLURM_MEM_BIND will not propagate into the tasks environment (binding by default only affects the first SRUN). To propagate --mem_bind to successive SRUN commands, first do the following in each task:

```
  setenv SLURM_MEM_BIND \
${SLURM_MEM_BIND_VERBOSE},${SLURM_MEM_BIND_TYPE}${SLURM_MEM_BIND_LIST}
```

See the ENVIRONMENT VARIABLES section for a more detailed description of the individual SLURM_CPU_BIND* and SLURM_MEM_BIND* variables.

`--network=type`
Specify the communication protocol to be used. The interpretation of type is system dependent.

6.3.1.10 Affinity/Multi-Core Options with UseCPUSETS parameter

The **UseCPUSETS** option modifies the Affinity/Multi-core operations to use the CPUsets facility in Linux instead of the **scheduler affinity** calls in the **task/affinity** plug-in. Job step initialization checks the **cpu_bind** and **mem_bind** parameters from SRUN, constructs a set of CPUs and memory, and creates a CPUset with these parameters. The name of the CPUset is **slurm** suffixed by **jobid** and **local task id**, e.g., **slurm47_1**. Each task on a given compute node is assigned to its own CPUset, which constrains the job to execute only on the CPUs and Memory nodes contained within the CPUset.

The following rules apply to the parameters in this mode:

1. If neither **cpu_bind** nor **mem_bind** are specified, no CPUset is created and the job runs with no restrictions.

2. **Mem_bind** is ignored if **cpu_bind** is not specified, as any CPU may be used by the task.

--cpu_bind options

None	no cpuset created, any processor may be used
Rank	CPUs assigned based on job **localid** + cpus/task
Map_cpu:<list>	CPUs are taken from the specified list according to **localid** of the task multiplied by the number of **CPUs** per task
Mask_cpu:<list>	CPU masks are taken from the list according to the **localid** order for the task. This allows specific CPU assignment under the control of the job requester.

--mem_bind options

None	Cpuset includes all parent's memory nodes
Rank	Not supported, same as **None**
Local	Memory nodes assigned based on CPUs allocated to cpuset
Map_mem:<list>	Not supported, same as **Local**
Mask_mem:<list>	Selects mask from the list in **localid** order for the task. This allows specific memory node assignment under the control of the job requester.

6.3.1.11 Help options

`--help`
Show this help message

`--usage`
Display brief usage message

6.3.1.12 Other options

`-V, --version`
output version information and exit

Unless the -a (--attach) or -A (--allocate) options are specified (see Allocate mode and Attaching to jobs below), SRUN will submit the job request to the SLURM job controller, then initiate all processes on the remote nodes. If the request cannot be met immediately, SRUN will block until the resources are free to run the job. If the -I (--immediate) option is specified, SRUN will terminate if resources are not immediately available.

When initiating remote processes, SRUN will propagate the current working directory, unless --chdir=path is specified, in which case path will become the working directory for the remote processes.

The -n, -c, and -N options control how CPUs and nodes will be allocated to the job. When specifying only the number of processes to run with -n, a default of one CPU per process is allocated. By specifying the number of CPUs required per task (-c), more than one CPU may be allocated per process. If the number of nodes is specified with -N, SRUN will attempt to allocate at least the number of nodes specified.

Combinations of the above three options may be used to change how processes are distributed across nodes and CPUs. For instance, by specifying both the number of processes and number of nodes on which to run, the number of processes per node is implied. However, if the number of CPUs per process is more important then number of processes (-n) and the number of CPUs per process (-c) should be specified.

SRUN will refuse to allocate more than one process per CPU unless --overcommit (-O) is also specified.

SRUN will attempt to meet the above specifications "at a minimum." That is, if 16 nodes are requested for 32 processes, and some nodes do not have 2 CPUs, the allocation of nodes will be increased in order to meet the demand for CPUs. In other words, a minimum of 16 nodes is being requested. However, if 16 nodes are requested for 15 processes, SRUN will consider this an error, as 15 processes cannot run across 16 nodes.

6.3.1.13 I/O Redirection

By default stdout and stderr will be redirected from all tasks to the stdout and stderr of SRUN, and stdin will be redirected from the standard input of SRUN to all remote tasks. This behavior may be changed with the --output, --error, and --input (-o, -e, -i) options. Valid format specifications for these options are:

`all`
: stdout and stderr are redirected from all tasks to SRUN. stdin is broadcast to all remote tasks. (This is the default behavior.)

`none`
: stdout and stderr are not received from any task. stdin is not sent to any task (stdin is closed).

`taskid`
: stdout and/or stderr are redirected from only the task with relative id equal to taskid, where 0 <= taskid <= ntasks -1, where ntasks is the total number of tasks in the current job step. stdin is redirected from the stdin of SRUN to this same task.

`filename`
: SRUN will redirect stdout and/or stderr to the named file from all tasks. stdin will be redirected from the named file and broadcast to all tasks in the job. If the job is submitted in batch mode using the -b or --batch option, filename refers to a path on each of the nodes on which the job runs. Otherwise filename refers to a path on the host that runs SRUN. Depending on the cluster's file system layout, this may result in the output appearing in different places depending on whether the job is run in batch mode.

`format string`
: SRUN allows for a format string to be used to generate the named IO file described above. The following list of format specifiers may be used in the format string to generate a filename that will be unique to a given jobid, stepid, node, or task. In each case, the appropriate number of files are opened and associated with the corresponding tasks.

`%J`	jobid.stepid of the running job (e.g. "128.0").
`%j`	jobid of the running job.
`%s`	stepid of the running job.
`%N`	short hostname. This will create a separate IO file per node.
`%n`	Node identifier relative to current job (e.g. "0" is the first node of the running job). This will create a separate IO file per node.

`%t`	task identifier (rank) relative to current job. This will create a separate IO file per task.

A number placed between the percent character and format specifier may be used to zero-pad the result in the IO filename. This number is ignored if the format specifier corresponds to non-numeric data (%N for example).

Some examples of how the format string may be used for a four-task job step with a Job ID of 128 and step id of 0 are included below:

```
job%J.out   job128.0.out

job%4j.out   job0128.out

job%j-%2t.out job128-00.out, job128-01.out, ...
```

6.3.1.14 Allocate Mode

When the allocate option is specified (-A, --allocate) SRUN will not initiate any remote processes after acquiring resources. Instead, SRUN will spawn a subshell that has access to the acquired resources. Subsequent instances of SRUN from within this subshell will then run on these resources.

If the name of a script is specified on the command line with --allocate, the spawned shell will run the specified script. Resources allocated in this way will only be freed when the subshell terminates.

6.3.1.15 Attaching To a Running Job

Use of the **-a jobid** (or **--attach**) option allows SRUN to reattach to a running job, receive stdout and stderr from the job and forward signals to the job, just as if the current session of SRUN had started the job. (stdin, however, cannot be forwarded to the job.)

There are two ways to reattach to a running job. The default method is to attach to the current job in read-only. In this case, stdout and stderr are duplicated to the attaching SRUN, but signals are not forwarded to the remote processes (a single Ctrl-C will detach this read-only SRUN from the job). If the -j (--join) option is also specified, SRUN "joins" the running job, and is able to forward signals, connect stdin, and act, for the most part, much like the SRUN process that initiated the job.

Node and CPU selection options are not applicable when specifying --attach, and it is an error to use -n, -c, or -N in attach mode.

6.3.1.16 Environment Variables

Some **SRUN** options may be set via environment variables. These environment variables, along with their corresponding options, are listed below. (Note: command-line options will always override these settings.)

SLURM_CONF	The location of the SLURM configuration file.

SLURM_ACCOUNT	`-U, --account=account`
SLURM_CPU_BIND	`--cpu_bind=type`
SLURM_CPUS_PER_TASK	`-c, --ncpus-per-task=n`
SLURM_CORE_FORMAT	`--core=format`
SLURM_DEBUG	`-v, --verbose`
SLURMD_DEBUG	`-d, --slurmd-debug`
SLURM_DISTRIBUTION	`-m, --distribution=(block|cyclic|hostfile)`
SLURM_GEOMETRY	`-g, --geometry=X,Y,Z`
SLURM_LABELIO	`-l, --label`
SLURM_MEM_BIND	`--mem_bind=type`
SLURM_NETWORK	`--network=type`
SLURM_NNODES	`-N, --nodes=(n|min-max)`
SLURM_NO_ROTATE	`--no-rotate`
SLURM_NPROCS	`-n, --ntasks=n`
SLURM_OVERCOMMIT	`-o, --overcommit`
SLURM_PARTITION	`-p, --partition=partition`
SLURM_REMOTE_CWD	`-D, --chdir==dir`
SLURM_SRUN_COMM_IFHN	`--ctrl-comm-ifhn=addr`
SLURM_STDERRMODE	`-e, --error=mode`
SLURM_STDINMODE	`-i, --input=mode`
SLURM_STDOUTMODE	`-o, --output=mode`
SLURM_TASK_EPILOG	`--task-epilog=executable`
SLURM_TASK_PROLOG	`--task-prolog=executable`
SLURM_TIMELIMIT	`-t, --time=minutes`
SLURM_WAIT	`-W, --wait=seconds`
SLURM_DISABLE_STATUS	`-X, -disable-status`

Additionally, SRUN will set some environment variables in the environment of the executing tasks on the remote compute nodes. These environment variables are:

SLURM_CPU_BIND_VERBOSE
: `--cpu_bind verbosity (quiet, verbose).`

SLURM_CPU_BIND_TYPE
: `--cpu_bind type (none, rank, map_cpu:, mask_cpu:)`

SLURM_CPU_BIND_LIST
: `--cpu_bind map or mask list (<list of IDs or masks for this node>)`

SLURM_CPUS_ON_NODE
: Count of processors available to the job on this node

SLURM_JOBID
: Job id of the executing job

SLURM_LAUNCH_NODE_IPADDR
: IP addresses of the node from which the task launch was initiated (from which the SRUN command was run)

SLURM_LOCALID
: Node local task ID for the process within a job

SLURM_MEM_BIND_VERBOSE
: `--mem_bind verbosity (quiet, verbose).`

SLURM_MEM_BIND_TYPE
: `--mem_bind type (none, rank, map_mem:, mask_mem:)`

SLURM_MEM_BIND_LIST
: `--mem_bind map or mask list (<list of IDs or masks for this node>)`

SLURM_NNODES
: Total number of nodes in the job's resource allocation

SLURM_NODEID
: The relative node ID of the current node

SLURM_NODELIST
: List of nodes allocated to the job

SLURM_NPROCS
: Total number of processes in the current job

SLURM_PROCID
: The MPI rank (or relative process ID) of the current process

SLURM_TASKS_PER_NODE

Number of tasks to be initiated on each node. Values are comma separated and in the same order as SLURM_NODELIST. If two or more consecutive nodes are to have the same task count, that count is followed by "(x#)", where "#" is the repetition count. For example, "`SLURM_TASKS_PER_NODE=2(x3),1`" indicates that the first three nodes will each execute two tasks, and the fourth node will execute one task.

6.3.1.17 Signals and Escape Sequences

Signals sent to the SRUN command are automatically forwarded to the tasks it is controlling, with a few exceptions. The escape sequence <control-c> will report the state of all tasks associated with the SRUN command. If <control-c> is entered twice within one second, then the associated SIGINT signal will be sent to all tasks. If a third <control-c> is received, the job will be forcefully terminated without waiting for remote tasks to exit.

The escape sequence <control-z> is presently ignored. When implemented it will put the SRUN command into a mode in which various special actions may be invoked.

6.3.1.18 MPI Support

The **PMI** (Process Management Interface) is provided by MPIBull2 to launch processes on a cluster and provide services to the MPI interface. For example, a call to **pmi_get_appnum** returns the job id. This interface uses sockets to exchange messages.

In **MPIBull2**, this mechanism uses the mpd daemons running on each compute node. Daemons can exchange information and answer the **PMI** calls.

RMS and **SLURM** replace the Process Management Interface with their own implementation and their own daemons. No mpd is needed and when a PMI request is sent (for example pmi_get_appnum), a SLURM extension must answer this request.

The following scheme shows the difference between the use of PMI with and without a resource manager that allows process management.

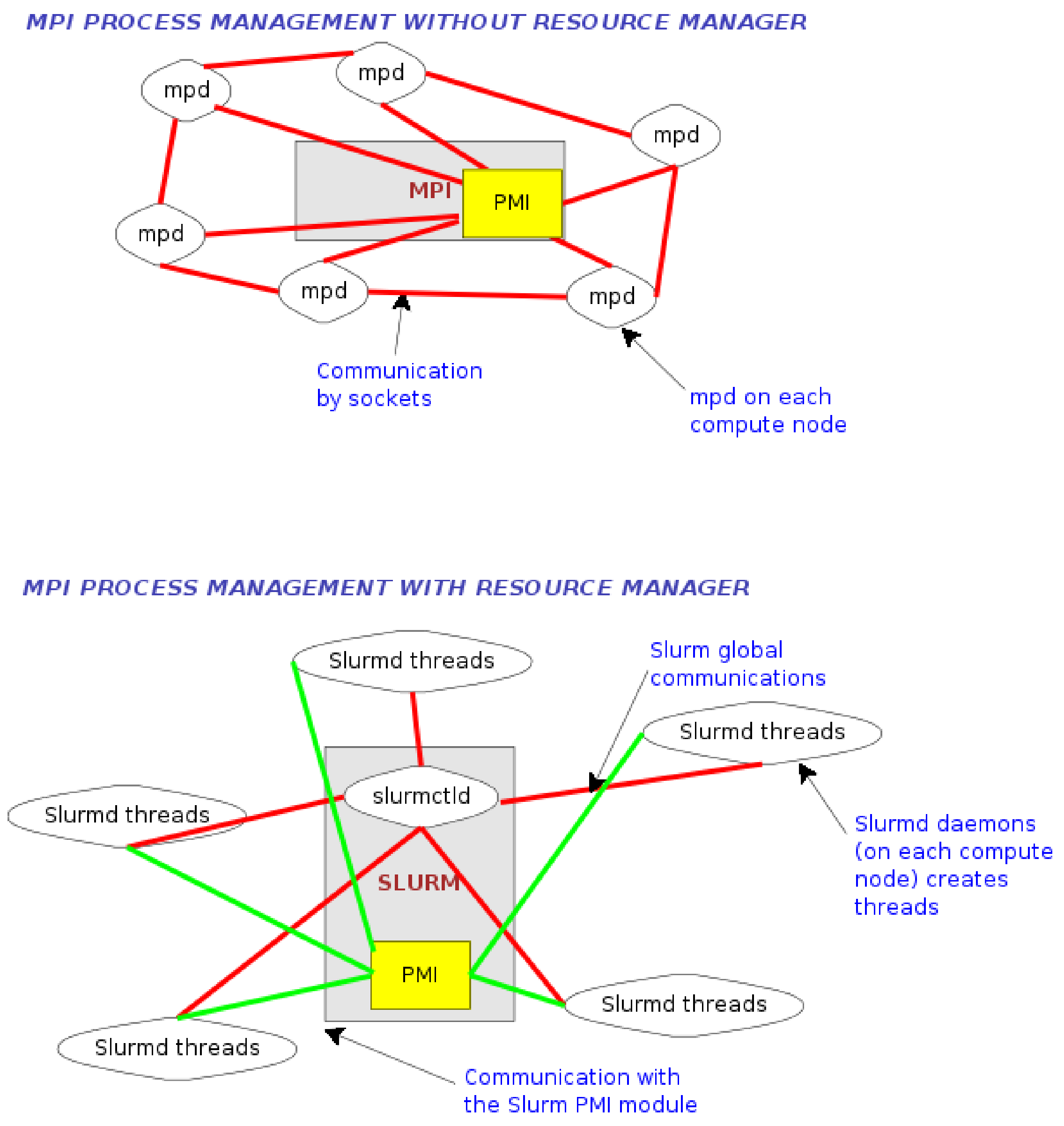

MPI PROCESS MANAGEMENT WITHOUT RESOURCE MANAGER
mpd
mpd
mpd
MPI
PMI
mpd
mpd
mpd
Communication by sockets
mpd on each compute node
MPI PROCESS MANAGEMENT WITH RESOURCE MANAGER
Slurmd threads
Slurm global communications
Slurmd threads
Slurmd threads
slurmctld
Slurmd daemons (on each compute node) creates threads
SLURM
PMI
Slurmd threads
Slurmd threads
Communication with the Slurm PMI module

Figure 6-4. MPI Process Management With and Without Resource Manager

MPIBull2 jobs can be launched directly by the **srun** command. SLURM's *none* MPI plug-in must be used to establish communications between the launched tasks. This can be accomplished either using the SLURM configuration parameter *MpiDefault=none* in **slurm.conf** or srun's *--mpi=none* option. The program must also be linked with SLURM's implementation of the PMI library so that tasks can communicate host and port information at startup. (The system administrator can add this option to the mpicc and mpif77 commands directly, so the user will not need to bother). **Do not use SLURM's MVAPICH plug-in for MPIBull2.**

```
$ mpicc -L<path_to_slurm_lib> -lpmi ...
$ srun -n20 --mpi=none a.out
```

Notes:

- Some **MPIBull2** functions are not currently supported by the **PMI** library integrated with **SLURM**.
- Set the environment variable **PMI_DEBUG** to a numeric value of 1 or higher for the PMI library to print debugging information.

6.3.1.19 Multiple Program Configuration

Comments in the configuration file must have a "#" in column one. The configuration file contains the following fields separated by space:

`Task rank`
: One or more task ranks to use this configuration. Multiple values may be comma separated. Ranges may be indicated with two numbers separated with a "-". To indicate all tasks, specify a rank of "*" (in which case, this option is not recommended).

`Executable`
: The name of the program to execute. May be fully-qualified pathname if desired.

`Arguments`
: Program arguments. The expression "%t" will be replaced with the task's number. The expression "%o" will be replaced with the task's offset within this range (e.g. a configured task rank value of "1-5" would have offset values of "0-4"). Single quotes may be used to avoid having the enclosed values interpreted. This field is optional.

Example:

```
###################################################################
# srun multiple program configuration file
#
# srun -n8 -l --multi-prog silly.conf
###################################################################
4-6     hostname
1,7     echo task:%t
0,2-3    echo offset:%o

$ srun -n8 -l --multi-prog silly.conf
0: offset:0
1: task:1
2: offset:1
3: offset:2
4: linux15.llnl.gov
5: linux16.llnl.gov
6: linux17.llnl.gov
7: task:7
```

6.3.1.20 EXAMPLES

The following simple example demonstrates the execution of the command hostname over eight tasks. At least eight processors will be allocated to the job (the same as the task count). The output of each task will be preceded with its task number. (The machine "dev" in the example below has a total of two CPUs per node)

```
> srun -n8 -l hostname
0: dev0
1: dev0
2: dev1
3: dev1
4: dev2
5: dev2
6: dev3
7: dev3
```

The following example demonstrates how one might submit a script for later execution (batch mode). The script will be initiated when resources are available and no higher priority job is pending for the same partition. The script will execute on four nodes with one task per node implicitly.

```
> cat test.sh
#!/bin/sh
date
srun -l hostname

> srun -N4 -b test.sh
srun: jobid 42 submitted
```

The output of test.sh would be found in the default output file "slurm-42.out."

The SRUN -r option is used within a job script to run two job steps on disjoint nodes in the following example. The script is run using allocate mode, instead of batch mode in this case.

```
> cat test.sh
#!/bin/sh
echo $SLURM_NODELIST
srun -lN2 -r2 hostname
srun -lN2 hostname

> srun -A -N4 test.sh
dev[7-10]
0: dev9
1: dev10
0: dev7
1: dev8
```

The following script runs two job steps in parallel within an allocated set of nodes.

```
> cat test.sh
#!/bin/bash
srun -lN2 -n4 -r 2 sleep 60 &
srun -lN2 -r 0 sleep 60 &
sleep 1
squeue
squeue -s
wait

> srun -A -N4 test.sh
 JOBID PARTITION   NAME  USER ST  TIME NODES  NODELIST
 65641   batch test.sh grondo  R  0:01   4  dev[7-10]

STEPID   PARTITION    USER   TIME NODELIST
65641.0    batch   grondo   0:01 dev[7-8]
65641.1    batch   grondo   0:01 dev[9-10]
```

This example demonstrates how one executes a simple MPICH job. SRUN is used to build a list of machines (nodes) to be used by mpirun in its required format. A sample command line and the script to be executed follow.

```
> cat test.sh
#!/bin/sh
MACHINEFILE="nodes.$SLURM_JOBID"

# Generate Machinefile for mpich such that hosts are in the same
# order as if run via srun
#
srun -l /bin/hostname | sort -n | awk ‘print $2}’ > $MACHINEFILE

# Run using generated Machine file:
mpirun -np $SLURM_NPROCS -machinefile $MACHINEFILE mpi-app

rm $MACHINEFILE

> srun -AN2 -n4 test.sh
```

This simple example demonstrates the execution of different jobs on different nodes in the same SRUN. This can be done for any number of nodes or any number of jobs. The executables are placed on the nodes sited by the SLURM_NODEID environment variable, starting at 0 and going up to the number specified on the SRUN command line.

```
> cat test.sh
case $SLURM_NODEID in
 0) echo "I am running on "
  hostname ;;
 1) hostname
  echo "is where I am running" ;;
esac

> srun -N2 test.sh
dev0
is where I am running
I am running on
dev1
```

6.3.2 SBCAST

sbcast is used to copy a file to local disk on all nodes allocated to a job. This should be executed after a resource allocation has taken place and can be faster than using a single file system mounted on multiple nodes.

NAME

sbcast - transmit a file to the nodes allocated to a SLURM job.

SYNOPSIS

```
sbcast [-CfpsvV] SOURCE DEST
```

DESCRIPTION

sbcast is used to transmit a file to all nodes allocated to the **SLURM** job which is currently active. This command should only be executed within a **SLURM** batch job or within the shell spawned after the resources have been allocated to a SLURM. **SOURCE** is the name of the file on the current node. **DEST** should be the fully qualified pathname for the file copy to be created on each node. **DEST** should be on the local file system for these nodes.

Note:
Parallel file systems may provide better performance than **sbcast** can provide.

6.3.2.1 OPTIONS

`-C, --compress`
Compress the file being transmitted.

`-f, --force`
If the destination file already exists, replace it.

`-F number, --fanout=number`
Specify the fanout of messages used for file transfer. Maximum value is currently eight.

`-p, --preserve`
Preserves modification times, access times, and modes from the original file.

`-s size, --size=size`
Specify the block size used for file broadcast. The size can have a suffix of k or m for kilobytes or megabytes respectively (defaults to bytes). This size is subject to rounding and range limits in order to maintain good performance. This value may need to be set on systems with very limited memory.

`-v, --verbose`
Provide detailed event logging whilst the program is executing.

`-V, --version`
Print version information and exit.

6.3.2.2 ENVIRONMENT VARIABLES

Some **sbcast** options may be set via environment variables. These environment variables, along with their corresponding options, are listed below.

Note:
Command line options will always override these settings

SBCAST_COMPRESS	`-C, --compress`
SBCAST_FANOUT	`-F number, fB--fanout=number`
SBCAST_FORCE	`-f, --force`
SBCAST_PRESERVE	`-p, --preserve`
SBCAST_SIZE	`-s size, --size=size`

6.3.2.3 EXAMPLE

Using a batch script, transmit local file **my.prog** to **/tmp/my.proc** on the local nodes and then execute it.

```
> cat my.job
#!/bin/bash
sbcast my.prog /tmp/my.prog
srun /tmp/my.prog


> srun --nodes=8 --batch my.job
srun: jobid 12345 submitted
```

6.3.3 SQUEUE (List Jobs)

SQUEUE displays (by default) the queue of running and waiting jobs (or "*job steps*"), including the **JobId** (used for **SCANCEL**), and the nodes assigned to each running job. However, **SQUEUE** reports can be customized to cover any of 24 different job properties, sorted by the most important properties. It also displays the job ID and job name for every job currently managed by the SLURM control daemon (**SLURMCTLD**). The status and resource information for each job (such as time used so far, or a list of committed nodes) are presented in a table whose content and format details can be controlled with the **SQUEUE** options.

NAME

SQUEUE - view information about jobs located in the SLURM scheduling queue.

SYNOPSIS

```
squeue  [OPTIONS...]
```

DESCRIPTION

SQUEUE is used to view job and job step information for jobs managed by SLURM.

6.3.3.1 OPTIONS

`-a, --all`
Display information about jobs and job steps in all partitions. This causes information to be displayed about partitions that are configured as hidden and partitions that are unavailable to user's group.

`--help`
Print a help message describing all SQUEUE options.

`--hide`
Do not display information about jobs and job steps in all partitions. By default, information about partitions that are configured as hidden or are not available to the user's group will not be displayed (i.e. this is the default behavior).

`--usage`
Print a brief help message listing the SQUEUE options.

`-h, --noheader`
Do not print a header on the output.

`-i <seconds>, --iterate=<seconds>`
Repeatedly gather and report the requested information at the interval specified (in seconds). By default, prints a time stamp with the header.

`-j, --jobs`
Specify the jobs to view. This flag indicates that a comma-separated list of jobs to view follows without an equal sign (see examples). Defaults to all jobs.

`-l, --long`
Report more of the available information for the selected jobs or job steps, subject to any constraints specified.

`-n <node_name>, --node=<node_name>`
Report only on jobs allocated to the specified node. This may either be the NodeName or NodeHostname, as defined in slurm.conf in the event that they differ. A node_name of localhost is mapped to the current host name.

`-o <output_format>, --format=<output_format>`
Specify the information to be displayed.
The default format for jobs is:
`Default` "%.7i %.9P %.8j %.8u %.2t %.9M %.6D %R"
If –l or –long is specified, the default job format is:
`-l, --long` "%.7i %.9P %.8j %.8u %.8T %.9M %.9l %.6D %R"
Format strings used internally by SQUEUE when running with various options are:
`-s, --steps` "%10i %.8j %.9P %.8u %.9M %N"
The field specifications available include:

`%a`	Account associated with the job
`%b`	Time at which the job began execution
`%c`	Minimum number of CPUs (processors) per node requested by the job. This reports the value of the SRUN --mincpus option with a default value of zero.
`%C`	Number of CPUs (processors) requested to the job or job step. This reports the value of the SRUN --ntasks option with a default value of zero.
`%d`	Minimum size of temporary disk space (in MB) requested by the job
`%D`	Number of nodes allocated to the job or the minimum number of nodes required by a pending job. The actual number of nodes allocated to a pending job may exceed this number if the job specified a node range count or the cluster contains nodes with varying processor counts.
`%e`	Time at which the job ended or is expected to end (based upon its time limit)
`%E`	Job dependency. This job will not begin execution until the dependent job completes. A value of zero implies this job has no dependencies.
`%f`	Features required by the job
`%g`	Group name
`%G`	Group ID
`%h`	The nodes allocated to the job can be shared with other jobs
`%i`	Job or job step id
`%j`	Job or job step name
`%l`	Time limit of the job in days-hours:minutes:seconds. The value may be "NOT_SET" if not yet established or "UNLIMITED" for no limit.
`%m`	Minimum size of memory (in MB) requested by the job
`%M`	Time used by the job or job step in days-hours:minutes:seconds. The days and hours are printed only as needed. For job steps, this field shows the elapsed time since execution began and thus will be inaccurate for job steps that have been suspended.
`%n`	List of node names explicitly requested by the job
`%N`	List of nodes allocated to the job or job step. In the case of a COMPLETING job, the list of nodes will comprise only those nodes that have not yet been returned to service. This may result in the node count being greater than the number of listed nodes.

`%o`	`Minimum number of nodes requested by the job`
`%O`	`Are contiguous nodes requested by the job`
`%p`	Priority of the job (converted to a floating point number between 0.0 and 1.0).
`%P`	Partition of the job or job step
`%r`	The reason why a job is waiting for execution. See the JOB REASON CODES section below for more information.
`%R`	For running or completed jobs: the list of allocated nodes. For pending jobs: the reason why a job is waiting for execution is printed within parenthesis. See the JOB REASON CODES section below for more information.
`%s`	Node selection plug-in specific data. Possible data includes: Geometry requirement of resource allocation (X,Y,Z dimensions), Connection type, Permit rotation of geometry (yes or no), etc.
`%S`	Start time of the job or job step
`%t`	Job state, compact form: PD (pending), R (running), CA (cancelled), CG (completing), CD (completed), F (failed), TO (timeout), and NF (node failure). See the JOB STATE CODES section below for more information.
`%T`	Job state, extended form: PENDING, RUNNING, SUSPENDED, CANCELLED, COMPLETING, COMPLETED, FAILED, TIMEOUT, and NODE_FAIL. See the JOB STATE CODES section below for more information.
`%u`	User name
`%U`	User ID
`%x`	List of node names explicitly excluded by the job
`%.<*>`	right justification of the field
`%<Number><*>`	size of field

`-v`

Display all job information.

`-p <part_list>, --partition=<part_list>`

Specify the partitions of the jobs or steps to view. Accepts a comma-separated list of partition names.

`-s, --steps`

Specify the job steps to view. This flag indicates that a comma-separated list of job steps to view follows without an equal sign (see examples). The job step format is "job_id.step_id". The default is all job steps.

`-S <sort_list>, --sort=<sort_list>`

Specification of the order in which records should be reported. This uses the same field specification as the <output_format>. Multiple sorts may be performed by listing multiple sort fields separated by commas. The field specifications may be preceded by "+" or "-" for ascending (default) and descending order respectively. For example, a sort value of "P,U" will sort the records by partition name then by user id. The default value of sort for jobs is "P,t,-p" (increasing partition name then within a given partition by increasing node state and then decreasing priority). The default value of sort for job steps is "P,i" (increasing partition name, then within a given partition by increasing step id).

`-t <state_list>, --states=<state_list>`

Specify the states of jobs to view. Accepts a comma-separated list of state names or "all". If "all" is specified then jobs of all states will be reported. If no state is specified then pending, running, and completing jobs are reported. Valid states (in both extended and compact form) include: PENDING (PD), RUNNING (R), SUSPENDED (S), COMPLETING (CG), COMPLETED (CD), CANCELLED (CA), FAILED (F), TIMEOUT (TO), and NODE_FAIL (NF). Note that the <state_list> supplied is case insensitive ("pd" and "PD" work the same). See the JOB STATE CODES section below for more information.

`-u <user_list>, --user=<user_list>`

Specifies a comma separated list of users whose jobs or job steps are to be reported. The list can consist of user names or user id numbers.

`-v, --verbose`

Report details of SQUEUE'S actions.

`-V , --version`

Print version information and exit.

6.3.3.2 JOB REASON CODES

The following codes identify the reason why a job is waiting for execution. A job may be waiting for more than one reason, in which case only one of those reasons is displayed.

Dependency	This job is waiting for a dependent job to complete.
None	No reason is set for this job.
PartitionDown	The partition required by this job is in a DOWN state.
PartitionNodeLimit	The number of nodes required by this job is outside of its partitions current limits.
PartitionTimeLimit	The job's time limit exceeds its partition's current time limit.
Priority	One or more higher priority jobs exist for this partition.
Resources	The job is waiting for resources to become available.

6.3.3.3 JOB STATE CODES

Jobs typically pass through several states in the course of their execution. The typical states are PENDING, RUNNING, SUSPENDED, COMPLETING, and COMPLETED. An explanation of each state follows.

`CA`	`CANCELLED`	Job was explicitly cancelled by the user or system administrator. The job may or may not have been initiated.
`CD`	`COMPLETED`	Job has terminated all processes on all nodes.
`CG`	`COMPLETING`	Job is in the process of completing. Some processes on some nodes may still be active.
`F`	`FAILED`	Job terminated with non-zero exit code or other failure condition.
`NF`	`NODE_FAIL`	Job terminated due to failure of one or more allocated nodes.
`PD`	`PENDING`	Job is awaiting resource allocation.
`R`	`RUNNING`	Job currently has an allocation.
`S`	`SUSPENDED`	Job has an allocation, but execution has been suspended.
`TO`	`TIMEOUT`	Job terminated upon reaching its time limit.

6.3.3.4 ENVIRONMENT VARIABLES

Some **SQUEUE** options may be set via environment variables. These environment variables, along with their corresponding options, are listed below. (Note: Command-line options will always override these settings.)

SLURM_CONF	The location of the SLURM configuration file.
SQUEUE_ALL	`-a, --all`
SQUEUE_FORMAT	`-o <output_format>, --format=<output_format>`
SQUEUE_PARTITION	`-p <part_list>, --partition=<part_list>`
SQUEUE_SORT	`-S <sort_list>, --sort=<sort_list>`
SQUEUE_STATES	`-t <state_list>, --states=<state_list>`
SQUEUE_USERS	`-u <user_list>, --users=<user_list>`

6.3.3.5 Examples

Print the jobs scheduled in the debug partition and in the COMPLETED state in the format with six right justified digits for the job id followed by the priority with an arbitrary fields size:

```
# squeue -p debug -t COMPLETED -o "%.6i %p"
     JOBID PRIORITY
     65543 99993
     65544 99992
     65545 99991
```

Print the job steps in the debug partition sorted by user:

```
# squeue -s -p debug -S u
  STEPID        NAME  PARTITION     USER  TIME_USED  NODELIST(REASON)
 65552.1       test1      debug    alice       0:23  dev[1-4]
 65562.2     big_run      debug      bob       0:18  dev22
 65550.1      param1      debug  candice    1:43:21  dev[6-12]
```

Print information only about jobs 12345, 12346, and 12348:

```
# squeue --jobs 12345,12346,12348
 JOBID PARTITION  NAME  USER  ST TIME_USED NODES  NODELIST(REASON)
 12345     debug  job1  dave   R      0:21     4  dev[9-12]
 12346     debug  job2  dave   PD     0:00     8  (Resources)
 12348     debug  job3  ed     PD     0:00     4  (Priority)
```

Print information only about job step 65552.1:

```
# squeue --steps 65552.1
  STEPID    NAME   PARTITION    USER    TIME_USED    NODELIST(REASON)
 65552.1  test2        debug   alice        12:49    dev[1-4]
```

6.3.4 SINFO (Report Partition and Node Information)

SINFO displays a summary of status information on SLURM-managed partitions and nodes (*not* jobs). Customizable SINFO reports can cover the node count, state, and name list for a whole partition, or the CPUs, memory, disk space, or current status for individual nodes as specified. These reports can assist in planning job submittals and avoiding hardware problems. The SINFO output is a table whose content and format can be controlled using the SINFO options.

NAME

SINFO - view information about SLURM nodes and partitions.

SYNOPSIS

```
sinfo [OPTIONS...]
```

DESCRIPTION

SINFO is used to view partition and node information for a system running SLURM.

6.3.4.1 OPTIONS

```
-a, --all
```

Display information about all partitions. This causes information to be displayed about partitions that are configured as hidden and partitions that are unavailable to user's group.

`--help`
: Print a message describing all SINFO options.

`--hide`
: Do not display information about hidden partitions. By default, partitions that are configured as hidden or are not available to the user's group will not be displayed (i.e. this is the default behavior).

`--usage`
: Print a brief message listing the SINFO options.

`-d, --dead`
: If set, only report state information for non-responding (dead) nodes.

`-e, --exact`
: If set, do not group node information on multiple nodes unless their configurations to be reported are identical. Otherwise CPU count, memory size, and disk space for nodes will be listed with the minimum value followed by a "+" for nodes with the same partition and state (e.g., "250+").

`-h, --noheader`
: Do not print a header on the output.

`-i <seconds>, --iterate=<seconds>`
: Print the state on a periodic basis. Sleep for the indicated number of seconds between reports. By default, prints a time stamp with the header.

`-l, --long`
: Print more detailed information. This is ignored if the --format option is specified.

`-n <nodes>, --nodes=<nodes>`
: Print information only about the specified node(s). Multiple nodes may be comma separated or expressed using a node range expression. For example, "linux[00-07]" would indicate eight nodes, "linux00" through "linux07."

`-N, --Node`
: Print information in a node-oriented format. The default is to print information in a partition-oriented format. This is ignored if the -format option is specified.

`-o <output_format>, --format=<output_format>`
: Specify the information to be displayed using an SINFO format string. Format strings transparently used by SINFO when running with various options are:

```
Default        "%9P %5a %.10l %.5D %6t %N"

--summarize    "%9P %5a %.10l %15F %N"

--long         "%9P %5a %.10l %.8s %4r %5h %10g %.5D %11T
%N"

--Node         "%#N %.5D %9P %6t"
```

```
--long –Node   "%#N %.5D %9P %11T %.4c %.6m %.8d %.6w %8f
%R"
```

```
--list-reasons "%35R %N"
```

```
--long --list-reasons "%50R %6t %N"
```

In the above format strings the use of "#" represents the maximum length of a node list to be printed.

The field specifications available include:

`%a`	State/availability of a partition
`%A`	Number of nodes by state in the format "allocated/idle". Do not use this with a node state option ("`%t`" or "`%T`") or the different node states will be placed on separate lines.
`%c`	Number of CPUs per node
`%C`	Number of CPUs per partition in the format "allocated/idle/total"
`%d`	Size of temporary disk space per node in megabytes
`%D`	Number of nodes
`%f`	Features associated with the nodes
`%F`	Number of nodes by state in the format "allocated/idle/other/total". Do not use this with a node state option ("`%t`" or "`%T`") or the different node states will be placed on separate lines.
`%g`	Groups which may use the nodes
`%h`	Jobs may share nodes, "yes", "no", or "force"
`%l`	Maximum time for any job in the format "days-hours:minutes:seconds"
`%m`	Size of memory per node in megabytes
`%N`	List of node names
`%P`	Partition name
`%r`	Only user root may initiate jobs, "yes" or "no"
`%R`	The reason why a node is unavailable (down, drained, or draining states)
`%s`	Maximum job size in nodes
`%t`	State of nodes, compact form
`%T`	State of nodes, extended form
`%w`	Scheduling weight of the nodes

`%.<*>` right justification of the field

`%<Number><*>` size of field

`-r, --responding`
If set, only report state information for responding nodes.

`-R, --list-reasons`
List reasons nodes are down or drained. When nodes are in these states SLURM supports the optional inclusion of a "reason" string by an administrator. This option will display the first 35 characters of the reason field and list of nodes with that reason that are, by default, down, drained, or draining. This option may be used with other node-filtering options (e.g. `-r, -d, -t, -n`), however, combinations of these options that result in a list of nodes that are not down or drained will not produce any output. When used with -l the output additionally includes the current node state.

`-s, --summarize`
List only a partition state summary with no node state details. This is ignored if the --format option is specified.

`-S <sort_list>, --sort=<sort_list>`
Specification of the order in which records should be reported. This uses the same field specification as the <output_format>. Multiple sorts may be performed by listing multiple sort fields separated by commas. The field specifications may be preceded by "`+`" or "`-`" for ascending (default) and descending order respectively. The partition field specification, "`P`", may be preceded by a "`#`" to report partitions in the same order that they appear in the SLURM configuration file, **slurm.conf**. For example, a sort value of "`+P,-m`" requests that records be printed in order of increasing partition name and within a partition by decreasing memory size. The default value of sort is "`#P,-t`" (partitions ordered as configured then decreasing node state). If the --Node option is selected, the default sort value is "`N`" (increasing node name).

`-t <states> , --states=<states>`
List nodes which have the given state(s). Multiple states may be comma separated and the comparison is case insensitive. Possible values include (case insensitive): ALLOC, ALLOCATED, COMP, COMPLETING, DOWN, DRAIN, DRAINED, DRNG, DRAINING, IDLE, UNK, and UNKNOWN. By default nodes in the specified state are reported whether they are responding or not. The --dead and --responding options may be used to filter nodes by the responding flag.

`-p <partition>, --partition=<partition>`
Print information only about the specified partition.

`-v, --verbose`
Provide detailed event logging through program execution.

`-V, --version`
Print version information and exit.

6.3.4.2 Output Field Descriptions

AVAIL
: Partition state: up or down.

CPUS
: Count of CPUs (processors) on these nodes.

CPUS (A/I)
: Count of allocated CPUs and idle CPUs per nodes.

GROUPS
: Resource allocations in this partition are restricted to the named groups. “all” indicates that all groups may use this partition.

JOB_SIZE
: Minimum and maximum node count that can be allocated to any user job. A single number indicates the minimum and maximum node count are the same. infinite is used to identify partitions without a maximum node count.

TIMELIMIT
: Maximum time limit for any user job in days-hours:minutes:seconds. “infinite” is used to identify partitions without a job time limit.

MEMORY
: Size of actual memory in megabytes on these nodes.

NODELIST
: Names of nodes associated with this configuration/partition.

NODES
: Count of nodes with this particular configuration.

NODES(A/I)
: Count of nodes with this particular configuration by node state in the form "available/idle".

NODES(A/I/O/T)
: Count of nodes with this particular configuration by node state in the form "available/idle/other/total".

PARTITION
: Name of a partition. Note that the suffix "*" identifies the default partition.

ROOT
: Is the ability to allocate resources in this partition restricted to user root, yes or no.

SHARE
: Defines whether jobs can share allocated resources. “no” indicates resources are never shared. “force” indicates resources are always available to be shared. “yes” indicates resources may be shared or not per job’s resource allocation.

STATE

State of the nodes. Possible states include: down, unknown, idle, allocated, drained, draining, completing and their abbreviated forms: down, unk, idle, alloc, drain, drng, and comp respectively. Note that the suffix "*" identifies nodes that are presently not responding.

TMP_DISK
: Size of temporary disk space in megabytes on these nodes.

6.3.4.3 Node State Codes

Node state codes are shortened as required for the field size. If the node state code is followed by "*", this indicates the node is presently not responding and will not be allocated any new work. If the node remains non-responsive, it will be placed in the DOWN state (except in the case of DRAINED, DRAINING, or COMPLETING nodes).

ALLOCATED
: The node has been allocated to one or more jobs.

ALLOCATED+
: The node is allocated to one or more active jobs plus one or more jobs are in the process of COMPLETING.

COMPLETING
: All jobs associated with this node are in the process of COMPLETING. This node state will be removed when all of the job's processes have terminated and the SLURM epilog program (if any) has terminated. See the Epilog parameter description in the slurm.conf man page for more information.

DOWN
: The node is unavailable for use. SLURM can automatically place nodes in this state if some failure occurs. System administrators may also explicitly place nodes in this state. If a node resumes normal operation, SLURM can automatically return it to service. See the **ReturnToService** and **SlurmdTimeout** parameter descriptions in the **slurm.conf** man page for more information.

DRAINED
: The node is unavailable for use per system administrator request. See the update node command in the **scontrol** man page or the **slurm.conf** man page for more information.

DRAINING
: The node is currently executing a job, but it will not be allocated to additional jobs. The node state will be changed to state DRAINED when the last job on it completes. Nodes enter this state per system administrator request. See the update node command in the scontrol man page or the slurm.conf man page for more information.

IDLE
: The node is not allocated to any jobs and is available for use.

UNKNOWN

The SLURM controller has just started and the node's state has not yet been determined.

6.3.4.4 Environment Variables

Some SINFO options may be set via environment variables. These environment variables, along with their corresponding options, are listed below. (Note: Command-line options will always override these settings.)

SLURM_CONF	The location of the SLURM configuration file.
SINFO_ALL	`-a, --all`
SINFO_FORMAT	`-o <output_format>, --format=<output_format>`
SINFO_PARTITION	`-p <partition>, --partition=<partition>`
SINFO_SORT	`-S <sort>, --sort=<sort>`

6.3.4.5 Examples

Report basic node and partition configurations:

```
> sinfo
PARTITION AVAIL TIMELIMIT NODES STATE  NODELIST
batch     up     infinite     2 alloc  adev[8-9]
batch     up     infinite     6 idle   adev[10-15]
debug*    up        30:00     8 idle   adev[0-7]
```

Report partition summary information:

```
> sinfo -s
PARTITION AVAIL TIMELIMIT NODES(A/I/O/T) NODELIST
batch     up     infinite 2/6/0/8        adev[8-15]
debug*    up        30:00 0/8/0/8        adev[0-7]
```

Report more complete information about the partition debug:

```
> sinfo --long --partition=debug
PARTITION AVAIL TIMELIMIT JOB_SIZE ROOT SHARE GROUPS NODES STATE NODELIST
debug*    up        30:00       8 no    no    all        8 idle  dev[0-7]
```

Report only those nodes that are in state DRAINED:

```
> sinfo --states=drained
PARTITION AVAIL NODES TIMELIMIT STATE  NODELIST
debug*    up        2     30:00 drain  adev[6-7]
```

Report node-oriented information with details and exact matches:

```
> sinfo -Nel
NODELIST   NODES PARTITION STATE   CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
adev[0-1]      2 debug*    idle       2   3448    38536     16 (null)   (null)
adev[2,4-7]    5 debug*    idle       2   3384    38536     16 (null)   (null)
adev3          1 debug*    idle       2   3394    38536     16 (null)   (null)
adev[8-9]      2 batch     allocated 2     246    82306     16 (null)   (null)
```

```
adev[10-15]    6 batch      idle       2     246     82306      16 (null)    (null)
```

Report only down, drained and draining nodes and their reason field:

```
> sinfo -R
REASON                              NODELIST
Memory errors                       dev[0,5]
Not Responding                      dev8
```

Report partition information that includes the number of allocated and idle CPUs.

```
> sinfo -o "%9P %5a %.5D  %.10A %.12C %N"
PARTITION AVAIL NODES  NODES(A/I)  CPUS(A/I/T) NODELIST
global*   up       28        1/26    8/208/224 linux[10-37]
bench     up       11        1/10      8/80/88 linux[10-20]
batch     up       10        0/10      0/80/80 linux[21-30]
```

6.3.5 SCANCEL (Signal/Cancel Jobs)

SCANCEL cancels a running or waiting job, or sends a specified signal to all processes on all nodes associated with a job (only job owners or their administrators can cancel jobs). SCANCEL may also be used to cancel a single job step instead of the whole job.

NAME

SCANCEL - Used to signal jobs or job steps that are under the control of SLURM.

SYNOPSIS

```
scancel [OPTIONS...] [job_id[.step_id]] [job_id[.step_id]...]
```

DESCRIPTION

SCANCEL is used to signal or cancel jobs or job steps. An arbitrary number of jobs or job steps may be signaled using job specification filters or a space-separated list of specific job and/or job step IDs. A job or job step can only be signaled by the owner of that job or user root. If an attempt is made by an unauthorized user to signal a job or job step, an error message will be printed and the job will not be signaled.

6.3.5.1 Options

`--help`
Print a help message describing all SCANCEL options.

`--usage`
Print a brief help message listing the SCANCEL options.

`-b, --batch`
Signal only the batch job shell.

`-i, --interactive`
: Interactive mode. Confirm each job_id.step_id before performing the cancel operation.

`-n, --name=job_name`
: The name of the jobs to be signaled.

`-p, --partition=partition_name`
: The name of the partition from which jobs are to be signaled.

`-q, --quiet`
: Do not report an error if the specified job is already completed. This option is incompatible with the --verbose option.

`-s, --signal=signal_name`
: The name or number of the signal to be sent. Default value is "KILL".

`-t, --state=job_state_name`
: The state of the jobs to be signaled. job_state_name may have a value of either "PENDING", "RUNNING" or "SUSPENDED".

`-u, --user=user_name`
: The name of the user whose jobs are to be signaled.

`-v, --verbose`
: Print additional logging. Using -v multiple times increases logging detail. This option is incompatible with the --quiet option.

`-V, --Version`
: Print the version number of the command.

6.3.5.2 Arguments

`job_id`
: The SLURM job ID of the job to have one or more of its steps signaled.

`step_id`
: The step ID of the job step to be signaled. If none is provided and the --batch option is not used, then all jobs steps associated with the provided job_id will be signaled.

6.3.5.3 Environment Variables

Some SCANCEL options may be set via environment variables. These environment variables, along with their corresponding options, are listed below. **Note:** Command-line options will always override these settings.

SLURM_CONF	The location of the SLURM configuration file.
SCANCEL_BATCH	`-b, --batch`
SCANCEL_INTERACTIVE	`-i, --interactive`

SCANCEL_NAME	`-n, --name=job_name`
SCANCEL_PARTITION	`-p, --partition=partition_name`
SCANCEL_STATE	`-t, --state=job_state_name`
SCANCEL_USER	`-u, --user=user_name`
SCANCEL_VERBOSE	`-v, --verbose`

Notes:

- If multiple filters are supplied (e.g. --partition and --name) only the jobs satisfying all of the filtering options will be signaled.
- If a signal value of "KILL" (the default value) is to be sent to an entire job, this will result in the job's termination and its resource allocation being released.
- Canceling a job step will not result in a job being terminated. The job must be cancelled to release a resource allocation.

6.3.5.4 Examples

Send SIGTERM to steps 1 and 3 of job 1234:

```
scancel --signal=TERM 1234.1 1234.3
```

Cancel job 1234 along with all of its steps:

```
scancel 1234
```

Cancel all pending jobs belonging to user "bob" in partition "debug":

```
scancel --state=PENDING --user=bob --partition=debug
```

6.3.6 SACCT (Accounting Data)

NAME

SACCT - displays accounting data for all jobs and job steps in the SLURM job accounting log.

SYNOPSIS

```
sacct options
```

DESCRIPTION

Accounting information for jobs invoked with SLURM is logged in the job accounting log file.

The **SACCT** command displays job accounting data stored in the job accounting log file in a variety of forms for your analysis. The SACCT command displays information about jobs, job steps, status, and exit codes by default. The output can be tailored with the use of the **--fields=** option to specify the fields to be shown.

For the root user, the SACCT command displays job accounting data for all users, although there are options to filter the output to report only the jobs from a specified user or group.

For the non-root user, the SACCT command limits the display of job accounting data to jobs that were launched with their own user identifier (UID) by default. Data for other users can be displayed with the **--all**, **--user**, or --**uid** options.

Note:

Much of the data reported by SACCT has been generated by the **wait3()** and **getrusage()** system calls. Some systems gather and report incomplete information for these calls; SACCT reports values of 0 for this missing data. See the **getrusage** man page for your system to obtain information about which data are actually available on your system.

6.3.6.1 Options

`-a , --all`

Displays the job accounting data for all jobs in the job accounting log file.

This is the default behavior when the SACCT command is executed by the root user.

`-b , --brief`

Displays a brief listing, which includes the following data:

- jobid
- status
- exitcode

This option has no effect when the ---dump option is also specified.

`-d , --dump`

Displays (dumps) the raw data records.

This option overrides the --brief and --fields= options.

The section titled "INTERPRETING THE --dump OPTION OUTPUT" describes the data output when this option is used.

`-S , --stat`

Queries the status of a job as the job is running displaying the following data:

- jobid
- vsize
- rss
- pages
- cputime
- ntasks
- status

The --jobs=job(.step) option must also be included. If no (.step) is given, the job.0 step will be received.

`-e time_spec , --expire=time_spec`

Removes job data from SLURM's current accounting log file (or the file specified with --file) for jobs that completed more than time_spec ago and appends them to the expired log file.

If time_spec is an integer value only, it is interpreted as minutes. If time_spec is an integer followed by "h", it is interpreted as a number of hours. If time_spec is an integer followed by "d", it is interpreted as number of days. For example, "--expire=14d" purges the job accounting log of all jobs that completed more than 14 days ago.

The expired log file is a file with the same name as the accounting log file, with ".expired" appended to the file name. For example, if the accounting log file is /var/log/slurmacct.log, the expired log file will be /var/log/slurmacct.log.expired.

`-F field_list , --fields=field_list`

Displays the job accounting data specified by the field_list operand, which is a comma-separated list of fields. Space characters are not allowed in the field_list.

See the --help-fields option for a list of the available fields. See the section titled "Job Accounting Fields" for a description of each field.

The job accounting data is displayed in the order specified by the field_list operand. Thus, the following two commands display the same data but in different order:

```
# sacct --fields=jobid,status
Jobid     Status
3         COMPLETED
3.0       COMPLETED

# sacct --fields=status,jobid
Status           Jobid
COMPLETED        3
COMPLETED        3.0
```

The default value for the field_list operand is "jobid,partition,process,ncpus,status,exitcode".

This option has no effect when the --dump option is also specified.

`-f file, --file=file`

Causes the SACCT command to read job accounting data from the named file instead of the current SLURM job accounting log file.

`-O , --formatted_dump`

Dumps accounting records in an easy-to-read format.

This option is provided for debugging.

`-g gid, --gid=gid`

Displays the statistics only for the jobs started with GID gid.

`-g group, --group=group`

Displays the statistics only for the jobs started by users in the group group.

`-h , --help`

Displays a general help message.

`--help-fields`

Displays a list of fields that can be specified with the --fields option.

The available fields are the following:

account	blockid	cpu	cputime	elapsed	end
exitcode	gid	group	idrss	inblock	isrss
ixrss	job	jobid	jobname	majflt	minflt
msgrcv	msgsnd	ncpus	nivcsw	nodes	nprocs
nsignals	nswap	ntasks	nvcsw	outblocks	pages
partition	rss	start	status	submit	systemcpu
uid	user	usercpu	vsize		

These fields are described in the section titled "Job Accounting Fields:"

`-j job(.step) , --jobs=job(.step)`

Displays information about the specified job(.step) or list of job(.step)s.

The job(.step) parameter is a comma-separated list of jobs.

Space characters are not permitted in this list.

The default is to display information on all jobs.

`-l, --long`

Displays a long listing, which includes the following data:

- jobid
- jobname
- partition
- vsize
- rss
- pages
- cputime
- ntasks
- ncpus
- elapsed
- status
- exitcode

`--noheader`

Prevents the display of the heading over the output. The default action is to display a header.

This option has no effect when used with the --dump option.

`-p partition_list , --partition=partition_list`

Displays information about jobs and job steps specified by the partition_list operand, which is a comma-separated list of partitions. Space characters are not allowed in the partition_list.

The default is to display information on jobs and job steps on all partitions.

`-s state_list , --state=state_list`

Selects jobs based on their current state, which can be designated with the following state designators:

r	running
s	suspended
ca	cancelled
cd	completed
pd	pending
f	failed
to	timed out
nf	node_fail

The state_list operand is a comma-separated list of these state designators. Space characters are not allowed in the state_list.

`-t , --total`

Displays only the cumulative statistics for each job.
Intermediate steps are displayed by default.

`-u uid, --uid=uid`

Displays the statistics only for the jobs started by the user whose UID is uid.

`-u user, --user=user`

Displays the statistics only for the jobs started by user user.

`--usage`

Displays a help message.

`-v , --verbose`

Reports the state of certain variables during processing.
This option is primarily used for debugging.

6.3.6.2 Job Accounting Fields

The following describes each job accounting field:

- account — user-supplied account number of the job
- cpu — sum of the system time (systemcpu) and user time (usercpu) in seconds
- cputime — minimum CPU time of any process followed by its task id along with the average of all processes running in the step
- elapsed — job's elapsed time (format : [DD-[hh:]]mm:ss) as defined by the following:

DD	days
hh	hours
mm	minutes
ss	seconds

- end — termination time of the job (format : MM/DD-hh:mm:ss) as defined by the following:

MM	month
DD	days
hh	hours
mm	minutes
ss	seconds

- exitcode The first non-zero error code returned by any job step.
- gid The group identifier of the user who ran the job.
- group The group name of the user who ran the job.
- idrss Maximum unshared data size (in KB) of any process.
- inblocks Total block input operations for all processes.
- isrss Maximum unshared stack space size (in KB) of any process.
- ixrss Maximum shared memory (in KB) of any process.
- job The SLURM job identifier of the job.
- jobid The number of the job or job step. It is in the form: job.jobstep.
- jobname The name of the job or job step.
- majflt Maximum number of major page faults for any process.
- minflt Maximum number of minor page faults (page reclaims) for any process.
- msgrcv Total number of messages received for all processes.
- msgsnd Total number of messages sent for all processes.
- ncpus Total number of CPUs allocated to the job.
- nivcsw Total number of involuntary context switches for all processes.
- nodes List of nodes allocated to the job.
- nprocs Total number of tasks in job. Identical to ntasks.
- nsignals Total number of signals received for all processes.
- nswap Maximum number of swap operations of any process.
- ntasks Total number of tasks in job.
- nvcsw Total number of voluntary context switches for all processes.
- outblocks Total block output operations for all processes.
- pages Maximum page faults of any process followed by its task id along with the average of all processes running in the step.
- partition Identifies the partition on which the job ran.
- rss Maximum resident set size of any process followed by its task id along with the average of all processes running in the step.
- start Initiation time of the job in the same format as end.
- status Displays the job status, or state. Output can be RUNNING, SUSPENDED, COMPLETED, CANCELLED, FAILED, TIMEOUT, or NODE_FAIL.
  If the job has been CANCELLED, the status will include the user ID of the user who cancelled the job.
- submit The time and date stamp (in Universal Time Coordinated, UTC) the job was submitted. The format of the output is identical to that of the end field.
- systemcpu The amount of system CPU time. The format of the output is identical to that of the elapsed field.

• uid	The user identifier of the user who ran the job.
• uid.gid	The user and group identifiers of the user who ran the job. (This field is used in record headers, and simply concatenates the uid and gid fields.)
• user	The user name of the user who ran the job.
• usercpu	The amount of user CPU time. The format of the output is identical to that of the elapsed field.
• vsize	Maximum Virtual Memory size of any process followed by its task id along with the average of all processes running in the step.

6.3.6.3 Interpreting the Dump Option

The --**dump** option of the SACCT command displays data in a horizontal list of fields depending on the record type; there are three record types: **JOB_START**, **JOB_STEP**, and **JOB_TERMINATED**. There is a subsection that describes the output for each record type.

When the data output is a job accounting field, as described in the section titled "Job Accounting Fields", only the name of the job accounting field is listed. Otherwise, additional information is provided.

Note:
The output for the JOB_STEP and JOB_TERMINATED record types presents a pair of fields for the following data: Total CPU time, Total User CPU time, and Total System CPU time. The first field of each pair is the time in seconds expressed as an integer. The second field of each pair is the fractional number of seconds multiplied by one million. Thus, a pair of fields output as "1 024315" means that the time is 1.024315 seconds. The least significant digits in the second field are truncated in formatted displays.

Output for the JOB_START Record Type

The following describes the horizontal fields output by the SACCT --dump option for the JOB_START record type.

Field#	Field
1	job
2	partition
3	The job's start time; this value is the number of non-leap seconds since the Epoch (00:00:00 UTC, January 1, 1970)
4	submitted
5	blockid
6	(Reserved)
7	JOB_START (literal string)
8	Job Record Version (1)
9	The number of fields in the record (17)
10	uid
11	gid
12	The job name

13	Batch Flag (0=no batch)
14	Relative SLURM priority
15	ncpus
16	nodes
17	account

Output for the JOB_STEP Record Type

The following describes the horizontal fields output by the SACCT --dump option for the JOB_STEP record type.

<u>Field#</u>	<u>Field</u>
1	job
2	partition
3	The job's start time; this value is the number of non-leap seconds since the Epoch (00:00:00 UTC, January 1, 1970)
4	submitted
5	blockid
6	(Reserved)
7	JOB_STEP (literal string)
8	Job Record Version (1)
9	The number of fields in the record (55)
10	jobid
11	end
12	Completion Status; the mnemonics, which may appear in uppercase or lowercase, are: CA Cancelled CD Completed successfully F Failed NF Job terminated from node failure R Running S Suspended TO Timed out
13	exitcode
14	ntasks
15	ncpus
16	Elapsed time in seconds expressed as an integer.
17	Integer portion of the Total CPU time in seconds for all processes.
18	Fractional portion of the Total CPU time for all processes expressed in microseconds.
19	Integer portion of the Total User CPU time in seconds for all processes.
20	Fractional portion of the Total User CPU time for all processes expressed in microseconds.
21	Integer portion of the Total System CPU time in seconds for all processes.
22	Fractional portion of the Total System CPU time for all processes expressed in microsecs.
23	rss
24	ixrss

25	idrss
26	isrss
27	minflt
28	majflt
29	nswap
30	inblocks
31	outblocks
32	msgsnd
33	msgrcv
34	nsignals
35	nvcsw
36	nivcsw
37	max_vsize
38	max_rss
39	max_vsize_node
40	max_vsize_task
41	ave_vsize
42	max_rss_node
43	max_rss_task
44	ave_rss
45	max_pages
46	max_pages_node
47	max_pages_task
48	ave_pages
49	min_cpu
50	min_cpu_node
51	min_cpu_task
52	ave_cpu
53	stepname
54	nodes
55	account

Output for the JOB_TERMINATED Record Type

The following describes the horizontal fields output by the SACCT --dump option for the JOB_TERMINATED (literal string) record type.

Field#	Field
1	job
2	partition
3	the jobs start time; this value is the number of non-leap seconds since the Epoch (00:00:00 UTC, January 1, 1970)
4	submitted
5	blockid
6	(Reserved)
7	JOB_TERMINATED (literal string)

8 Job Record Version (1).

9 The number of fields in the record (56).

Although thirty-eight fields are displayed by the SACCT command for the JOB_TERMINATED record, only fields 1 through 12 are recorded in the actual data file; the SACCT command aggregates the remainder.

10 the total elapsed time in seconds for the job.

11 end

12 completion Status; the mnemonics, which may appear in uppercase or lowercase, are:

`CA`	Cancelled
`CD`	Completed successfully
`F`	Failed
`NF`	Job terminated from node failure
`R`	Running
`TO`	Timed out

13 exitcode

14 ntasks

15 ncpus

16 Elapsed time in seconds expressed as an integer.

17 Integer portion of the Total CPU time in seconds for all processes.

18 Fractional portion of the Total CPU time for all processes expressed in microseconds.

19 Integer portion of the Total User CPU time in seconds for all processes.

20 Fractional portion of the Total User CPU time for all processes expressed in microseconds.

21 Integer portion of the Total System CPU time in seconds for all processes.

22 Fractional portion of the Total System CPU time for all processes expressed in microseconds.

23 rss

24 ixrss

25 idrss

26 isrss

27 minflt

28 majflt

29 nswap

30 inblocks

31 outblocks

32 msgsnd

33 msgrcv

34 nsignals

35 nvcsw

36 nivcsw

37 max_vsize

38 max_rss

39 max_vsize_node

40 max_vsize_task

41 ave_vsize

42	max_rss_node
43	max_rss_task
44	ave_rss
45	max_pages
46	max_pages_node
47	max_pages_task
48	ave_pages
49	min_cpu
50	min_cpu_node
51	min_cpu_task
52	ave_cpu
53	--
54	nodes
55	account
56	requid

6.3.6.4 Examples

The following example illustrates the default invocation of the SACCT command:

```
# sacct
Jobid    Jobname     Partition  Ncpus  Status      Exitcode
2        script01    srun           1  RUNNING     0
3        script02    srun           1  RUNNING     0
4        endscript   srun           1  RUNNING     0
4.0                  srun           1  COMPLETED   0
```

The following example shows the same job accounting information with the brief option.

```
# sacct --brief
Jobid   Status      Exitcode
2       RUNNING     0
3       RUNNING     0
4       RUNNING     0
4.0     COMPLETED   0

# sacct --total
Jobid   Jobname     Partition  Ncpus   Status    Exitcode
3       sja_init    andy           1   COMPLETED    0
4       sjaload     andy           2   COMPLETED    0
5       sja_scr1    andy           1   COMPLETED    0
6       sja_scr2    andy          18   COMPLETED    2
7       sja_scr3    andy          18   COMPLETED    0
8       sja_scr5    andy           2   COMPLETED    0
9       sja_scr7    andy          90   COMPLETED    1
10      endscript   andy         186   COMPLETED    0
```

The following example demonstrates the ability to customize the output of the SACCT command. The fields are displayed in the order designated on the command line.

```
# sacct --fields=jobid,ncpus,ntasks,nsignals,status
Jobid  Ncpus  Ntasks  Nsignals       Status
3         2      1          0      COMPLETED
3.0       2      1          0      COMPLETED
```

```
4           2       2           0       COMPLETED
4.0         2       2           0       COMPLETED
5           2       1           0       COMPLETED
5.0         2       1           0       COMPLETED
```

6.3.7 Global Accounting API

Note:
The Global Accounting API only applies to clusters which use **SLURM** and the Load Sharing Facility (**LSF**) batch manager from **Platform Computing** together.

Both the **LSF** and **SLURM** products can produce an accounting file. The Global Accounting API offers the capability of merging the data from these two accounting files and presenting it as a single record to the program using this API.

Perform the following steps to call the Global Accounting API:

After SLURM has been installed (assumes **/usr** folder), build the Global Accounting API library by going to the **/usr/lib/slurm/bullacct** folder and executing the following command:

```
make –f makefile-lib
```

This will build the library **libcombine_acct.a**. This **makefile-lib** assumes that the SLURM product is installed in the **/usr** folder, and **LSF** is installed in **/app/slurm/lsf/6.2**. If this is not the case, the **SLURM_BASE** and **LSF_BASE** variables in the **makefile-lib** file must be modified to point to the correct location.

After the library is built, add the library **/usr/lib/slurm/bullacct/libcombine_acct.a** to the link option when building an application that will use this **API**.

In the user application program, add the following:

```
//  for new accounting record
//  assumes Slurm is installed under the opt/slurm folder

#include "/usr/lib/slurm/bullacct/combine_acct.h"

//  define file pointer for LSF and Slurm log file
FILE *lsb_acct_fg = NULL;    // file pointer for LSF accounting log file
FILE *slurm_acct_fg = NULL;  // file pointer for Slurm log file
int status, jobId;
struct CombineAcct newAcct;  // define variable for the new records

//  call cacct_init routine to open lsf and slurm log file,
//  and initialize the newAcct structure
status = cacct_init(&lsb_acct_fg, &slurm_acct_fg, &newAcct);

//  if the status returns 0 imply no error,
//    all log files are opened successfully.
//  then call get_combine_acct_info rountine to get the
//    combine accounting record.

//  the calling sequence is
//    int get_combine_acct_info(File *lsb_acct_fg,
//                              File *slurm_acct_fg,
//                              int   jobId,
  //                              CombineAcct *newAcct);
  //  where:
  //  lsb_acct_fg is the pointer to the LSF accounting log file
  //  slurm_acct_fg is the pointer to the Slurm accounting log file
  //  jobid is the job Id from the LSF accounting log file
  //  newAcct is the address of the variable to hold the new record
  //  information.

  //  This routine will use the input LSF job ID to locate the LSF accounting
  //  information in the LSF log file, then get the SLURM_JOBID and locate the
  //  SLURM accounting information in the SLURM log file.
  //  This routine will return a zero to indicate that both records are found
  //  and processed successfully, otherwise one or both records are in error
  //  and the content in the newAcct variable is undefined.
  //  For example:

  //  to get the combine acct information for a specified jobid(2010)

    jobId = 2010;
    status = get_combine_acct_info(lsb_acct_fg,
                                   slurm_acct_fg,
                                   jobId,
                                   &newAcct);

  //  to display the record call display_combine_acct_record routine.

  display_combine_acct_record(&newAcct);

  //  when finished accessing the record, the user must close the log files and
  //  the free memory used in the newAcct variable by calling cacct_wrapup
  //  routine.
  //  For example:
  //
    if (lsb_acct_fg != NULL)               // if open successfully before
      cacct_wrapup(&lsb_acct_fg, &slurm_acct_fg, &newAcct);

  //  if an extra combine account variable is needed , the user can define
  //  the new variable and call init_cacct_rec to initialize the record
  //  and call free_cacct_ptrs to free the memory used in the new variable.
  //  For example:

  //  to define variable for the new record
      struct CombineAcct otherAcct;

  //  before using the variable otherAcct do:
     init_cacct_rec(&otherAcct);

  //  when done do the following to free the memory used by the otherAcct
  //  variable.
      free_cacct_ptrs(&otherAcct);
```

The new record contains the combined accounting information as follows:

```
/* combine LSF and SLURM acct log information */
struct CombineAcct {

         /* part one is the LSF information */

    char   evenType[50];
    char   versionNumber[50];
    time_t eventTime;
    int    jobId;
    int    userId;
    long   options;
    int    numProcessors;
    time_t submitTime;
    time_t beginTime;
time_t termTime;
time_t startTime;
char   userName[MAX_LSB_NAME_LEN];
char   queue[MAX_LSB_NAME_LEN];
char   *resReq;
char   *dependCond;
char   *preExecCmd;            /* the command string to be pre_executed */
char   fromHost[MAXHOSTNAMELEN];
char   cwd[MAXFILENAMELEN];
char   inFile[MAXFILENAMELEN];
char   outFile[MAXFILENAMELEN];
char   errFile[MAXFILENAMELEN];
char   jobFile[MAXFILENAMELEN];
int    numAskedHosts;
char   **askedHosts;
int    numExecHosts;
char   **execHosts;
int    jStatus;                        /* job status */
double hostFactor;
char   jobName[MAXLINELEN];
char   command[MAXLINELEN];
struct lsfRusage LSFrusage;
char   *mailUser;                   /* user option mail string */
char   *projectName;                /* the project name for this job, used
                                       for accounting purposes */
int    exitStatus;                  /* job status */
int    maxNumProcessors;
char   *loginShell;                 /* login shell specified by user */
char   *timeEvent;
int    idx;                         /* array idx, must be 0 in JOB_NEW */
int    maxRMem;
int    maxRswap;
char   inFileSpool[MAXFILENAMELEN];  /* spool input file */
char   commandSpool[MAXFILENAMELEN]; /* spool command file */
char   *rsvId;
char   *sla;             /* The service class under which the job runs. */
int    exceptMask;
char   *additionalInfo;
int    exitInfo;
char   *warningAction;                /* warning action, SIGNAL | CHKPNT |
                                         command, NULL if unspecified */
int    warningTimePeriod;             /* warning time period in seconds,
                                         -1 if unspecified */
char   *chargedSAAP;
char   *licenseProject;               /* License Project */
int    slurmJobId;                    /* job id from slurm */

 /* part two is the SLURM info minus the duplicated infomation from LSF */

long   priority;                      /* priority */
char   partition[64];                 /* partition node */
int    gid;                           /* group ID */
int    blockId;                       /* Block ID */
int    numTasks;                      /* nproc */
double aveVsize;                      /* ave vsize */
int    maxRss;                        /* max rss */
int    maxRssTaskId;                  /* max rss task  */
double aveRss;                        /* ave rss */
int    maxPages;                      /* max pages */
int    maxpagestaskId;                /* max pages task */
double avePages;                      /* ave pages */
int    minCpu;                        /* min cpu */
int    minCpuTaskId;                  /* min cpu task */
char   stepName[NAME_SIZE];           /* step process name */
char   stepNodes[STEP_NODE_BUF_SIZE];  /* step node list */
int    maxVsizeNode;                  /* max vsize node */
int    maxRssNodeId;                  /* max rss node */
```

```
    int    maxPagesNodeId;                 /* max pages node */
    int    minCpuTimeNodeId;               /* min cpu node */
    char   *account;                       /* account number */

};
```

6.4 Launching the Application using TORQUE Batch Manager

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.

6.4.1 Configuring Passwordless Access for TORQUE

ssh keys have to be configured to create public\private keys for an ordinary user of a cluster so that passwordless access is enabled for the whole of the cluster\partition on which the application and **TORQUE** is running. Otherwise **TORQUE** will not work correctly.

This is done by using the **ssh-keygen** command.

```
ssh-keygen -trsa
```

Append this key to the list of authorized keys.

Note:
See chapters 2 and 10 in the HPC BAS4 *Administrator's Guide* for more information on configuring **ssh**

The user command interface for TORQUE can be used to:

- Submit a job
- Display the state and characteristics of a job
- Cancel a job
- Change the characteristics of a job, which is either running or waiting. 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 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 necessary. However, TORQUE is not limited to queue-based scheduling, which means it is possible to run TORQUE with a single queue.

Multiple Scheduling Algorithms

With TORQUE it is possible to specify the standard *first-in, first-out* scheduling routine or more sophisticated algorithms.

6.4.2 TORQUE Commands

Below 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

Table 6-3. TORQUE commands

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.

Chapter 7. Application Debugging Tools

7.1 Overview

There are two types of debuggers; symbolic ones and non-symbolic ones.

A **symbolic debugger** gives access to a program's source code. This means that:

- The lines of the source file can be accessed.
- The program variables can be accessed by name.

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

The following tools are described in this chapter:

- *7.2 GDB*
- *7.3 IDB*
- *7.4 TOTALVIEW*
- *7.5 MALLOC_CHECK_ - Debugging Memory Problems in C programs*
- *7.6 Dmalloc Library*
- *7.7 Electric Fence*
- *7.8 System Monitoring and Performance Tools*

7.2 GDB

GDB stands for Gnu DeBugger. It is a powerful Open-source debugger, which can be used either 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.

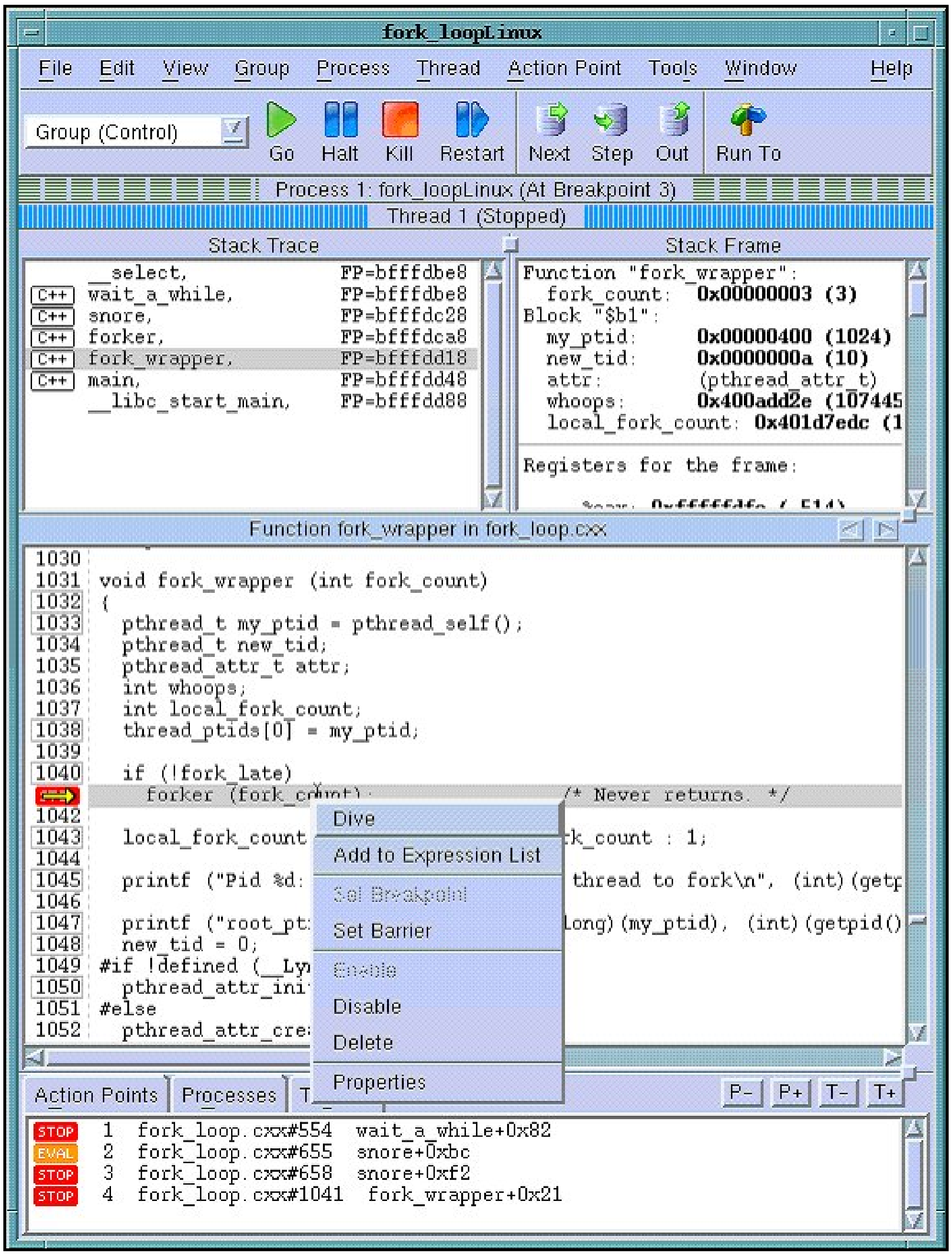
GDB supports parallel applications and threads.

GDB is published under the GNU license.

7.3 IDB

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

7.4 TOTALVIEW



```
      __select,          FP=bfffdbe8
C++   wait_a_while,      FP=bfffdbe8
C++   snore,             FP=bfffdc28
C++   forker,            FP=bfffdca8
C++   fork_wrapper,      FP=bfffdd18
C++   main,              FP=bfffdd48
      __libc_start_main, FP=bfffdd88
```

```
Function "fork_wrapper":
  fork_count:        0x00000003 (3)
Block "$b1":
  my_ptid:           0x00000400 (1024)
  new_tid:           0x0000000a (10)
  attr:              (pthread_attr_t)
  whoops:            0x400add2e (107445
  local_fork_count:  0x401d7edc (1

Registers for the frame:
```

```
void fork_wrapper (int fork_count)
{
  pthread_t my_ptid = pthread_self();
  pthread_t new_tid;
  pthread_attr_t attr;
  int whoops;
  int local_fork_count;
  thread_ptids[0] = my_ptid;

  if (!fork_late)
    forker (fork_count);                      /* Never returns. */

  local_fork_count                           k_count : 1;

  printf ("Pid %d:                           thread to fork\n", (int)(getp

  printf ("root_pt:                          long)(my_ptid), (int)(getpid()
  new_tid = 0;
#if !defined (__Ly
  pthread_attr_ini
#else
  pthread_attr_cre
```

```
STOP  1  fork_loop.cxx#554   wait_a_while+0x82
EVAL  2  fork_loop.cxx#655   snore+0xbc
STOP  3  fork_loop.cxx#658   snore+0xf2
STOP  4  fork_loop.cxx#1041  fork_wrapper+0x21
```

Figure 7-1 Totalview graphical interface – image taken from www.etnus.fr

TotalViewTM is a proprietary software application from **Etnus** and is not included with the BAS distribution. TotalviewTM is used in the same way as standard symbolic debuggers for C, C++ and Fortran (77, 90 and HPF) programs. It can also debug PVM or MPI applications. **TotalViewTM** has the advantage of being a debugger which supports multi-processes and multi-threading. It can take control of the various processes or threads of the program and make it possible for the user to visualize the evolution of the execution in the same window or in different windows. The processes may be local or remote.

It works just as well with mono-processor, SMP, clustered, distributed and MPP systems.

TotalView™ accepts new processes and threads exactly as generated by the application and regardless of the processor used for the execution. A process started up outside **TotalView**™ can also be connected to. Data tables can be filtered, displayed, and viewed in order to monitor the behavior of the program. Finally, you can descend *("call the components and details of…")* into the objects and structures of the program.

The program which needs debugging must be compiled with the option '- g', and then breakpoints should be added to the program to control its execution.

TotalView™ is an Xwindows application. Context-sensitive help provides you with basic information. You may download **TotalView**™ 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 additional information, and for copies of the documentation for **Totalview**™, please refer to http://www.etnus.com/.

7.5 MALLOC_CHECK_ - Debugging Memory Problems in C programs

When developing an application, the developer should ensure that all the buffers allocated during the run-time of the application are freed afterwards. However, even if he is vigilant, it is not unusual for memory leaks to be introduced into the code.

A simple way to detect these memory leaks is to use the environment variable **MALLOC_CHECK** __. This variable ensures that allocation routines check that each allocated buffer is freed correctly. The routines then become more 'tolerant' and allow byte overflows on both sides of blocks or for the block to be released again.

According to the value of **MALLOC_CHECK** __, when a release or allocation error appears the application behaves as follows:

- If **MALLOC_CHECK** __ is set to 1, an error message is written when exiting normally.
- If **MALLOC_CHECK** __ is set to 2, an error message is written when exiting normally and the process aborts. A core file is created. You should check that it is possible to create a core file by using the command *ulimit –c.* If not, enter the command *ulimit -c unlimited.*
- For any other value of **MALLOC_CHECK** __, the error is ignored and no message appears.

Example.c program:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 256

int main(void){

  char *buffer;

  buffer = (char *)calloc(256*sizeof(char));
  if(!buffer){
    perror(``malloc failed'');
    exit(-1);
  }

  strcpy(buffer, ``fills the buffer'');
  free(buffer);
  fprintf(stdout, ``Buffer freed for the first time'');
  free(buffer);
  fprintf(stdout,``Buffer freed for the second time'');
  return(0);

}
```

A program which is executed with the environmental variable MALLOC_CHECK __ set to 1 gives the following result:

```
$ export MALLOC_CHECK__=1

$./example

Buffer freed for the first time

Segmentation fault

$ ulimit -c 0

# The limit for the core file size must be changed to allow files
 bigger than 0 bytes to be generated

$ ulimit -c unlimited

# Allows an unlimited core file to be generated
```

A program which is executed with the environmental variable MALLOC_CHECK __ set to 2 gives the following result:

```
$ export MALLOC_CHECK__=2

$ ./example

Buffer freed for the first time

Segmentation fault (core dumped)
```

Example Program Analysis using the GDB Debugger

The core file should be analyzed to identify where the problem is (the program should be compiled with the option - G):

```
$ gdb example -c core
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
 and you are welcome to change it and/or distribute copies of it
 under certain conditions.
Type "show copying" to see the conditions. There is absolutely no
 warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-linux"...Using host libthread_db
 library "/lib/libthread_db.so.1".

Core was generated by `./example'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x40097354 in mallopt () from /lib/libc.so.6
(gdb) bt
#0  0x40097354 in mallopt () from /lib/libc.so.6
#1  0x4009615f in free () from /lib/libc.so.6
#2  0x0804852f in main () at exemple.c:18
(gdb)
```

The **bt** command is used to display the current memory stack. In this example the last line indicates the problem came from line 18 in the main function of the **example.c** file. Looking at the **example.c** program on page 7.4 we can see that line 18 corresponds to the second call to the free function which created the memory overflow.

7.6 Dmalloc Library

Dmalloc is an open source library and is included in the BAS distribution to help with application development and to ensure that memory leaks are detected quickly. This tool is complementary to the use of MALLOC_CHECK __ and is used to find memory leaks in C programs.

The debug memory allocation or dmalloc library is a memory management routine which provides powerful debugging facilities which are 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. Thus it makes it possible to obtain precise information about a memory allocation problem. Small changes must be made to the code to run it. It also provides support for the debugging of threaded programs.

This **dmalloc** library substitutes the primitive calls **malloc**, **calloc**, **realloc** and **free** with the primitives which are available in the **dmalloc** library.

The **dmalloc** User's Guide and further information is available from the site http://www.dmalloc.com

7.7 Electric Fence

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

Electric Fence is installed only on the management node.

Electric Fence helps you detect two common programming bugs:

- Software that overruns the boundaries of a **malloc()** memory allocation.
- Software that touches 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 more details about Electric Fence please refer to http://perens.com/FreeSoftware/ .

7.8 System Monitoring and Performance Tools

In the world of HPC architectures, monitoring and improving performance is an important concern in order to fully optimize calculation speeds and memory usage for these powerful machines.

For information on monitoring tools and on improving overall performance of the application program on the HPC platform refer to the Bull HPC *Application Tuning Guide* (86 A2 19ER). This manual describes system monitoring tools provided by *NovaScale Master – HPC Edition* including **time**, **top**, and **perfmon** and application profiling tools including **gprof**, **profilecomm**, the **PAPI** library and **Intel® Trace Tools**.

For information related to the performance of the cluster itself, please refer to the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER).

Appendix A. Application Troubleshooting

A list of frequently asked questions (FAQs) with solutions and advice follows:

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.
-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 BAS3 or RH EL AS 3) will not execute under BAS4. 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 tried:

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, 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 setting 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
```

Glossary and Acronyms

A

ANL
Argonne National Laboratory (MPICH2)

API
Application Programmer Interface

B

BAS
Bull Advanced Server

BIOS
Basic Input Output System

BMC
Baseboard Management Controller

B-SPS
Bull Scalable Port Switch

C

CLI
Command Line Interface

CMOS
Complementary Metal Oxide Semiconductor

D

DDN
DataDirect Networks S2A (storage system)

E

EFI
Extensible Firmware Interface (Intel)

EIP
IP over QSnet using Elan Kernel communications

EMP
Emergency Management Port

EPIC
Explicit Parallel Instruction set Computing

F

FAME
Flexible Architecture for Multiple Environments

FSS
FAME Scalability Switch. Each CSS Module is equipped with 2 Scalability Port Switches providing high speed bi–directional links between server components

FUTEX
Fast User mode muTEX

G

GCC
GNU C Compiler

GDB
Gnu Debugger

GNU
GNU's Not Unix

GPL
General Public License

GUI
Graphical User Interface

GUID
Globally Unique Identifier

H

HDD	Hard Disk Drive
HBA	Host Bus Adapter
HPC	High Performance Computing
HSC	Hot Swap Controller

I

ICC	Intel C Compiler
IDE	Integrated Device Electronics
IFORT	Intel Fortran Compiler
IPMI	Intelligent Platform Management Interface

K

KDM	Kernel Data Mover
KSIS	Utility for Image Building and Deployment
KVM	Keyboard Video Mouse (allows the connection of the keyboard, video and mouse either to the PAP or to the node)

L

LSF	Load Sharing Facility
LUN	Logical Unit Number

M

MDM	MPI Data Mover module
MPD	MPI Process Daemons
MPI	Message Passing Interface

N

NFS	Network File System
NPTL	Native POSIX Thread Library
NTFS	New Technology File System (Microsoft)
NUMA	Non Uniform Memory Access. A method of configuring a cluster of microprocessors in a multiprocessing system so that they can share memory locally, improving performance and the ability of the system to be expanded.
NVRAM	Non Volatile Random Access Memory

O

OEM	Original Equipment Manufacturer
OPK	OEM Preinstall Kit (Microsoft)

P

PAM
Platform Administration and Maintenance software

PAP
Platform Administration Processor

PAPI
Performance Application Programming Interface

PCI
Peripheral Component Interconnect (Intel)

PDU
Power Distribution Unit

PM
Process Manager

PMB
Platform Management Board

PMI
Process Management Interface

PMU
Performance Monitoring Unit

PRUN
Parallel Run (Quadrics)

PVFS
Parallel Virtual File System

PVM
Parallel Virtual Machine

Q

QBB
Quad Brick Board. The QBB is the heart of the Bull NovaScale 5xxx/6xxx Series platforms, housing 4 Itanium _ 2 processors.

R

RMS
Resource Management Service (Quadrics)

RPM
RPM Package Manager

S

SCI
Scalable Coherent Interconnect

SDR
Sensor Data Record

SDP
Sockets Direct Protocol

SEL
System Event Log

SCSI
Small Computer System Interface

SM
System Management

SMP
Symmetric Multi Processing. The processing of programs by multiple processors that share a common operating system and memory.

SNMP
The protocol governing network management and the monitoring of network devices and their functions.

SOL
Serial Over LAN

SSH
Secure Shell

U

UA

User’s Application

V

VGA

Video Graphic Adapter

Index

Technical publication remarks form

Title:	HPC BAS4 User's Guide

Reference:	86 A2 29ER 07	Date:	July 2007

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by qualified technical personnel and action will be taken as required.
If you require a written reply, please include your complete mailing address below.

NAME: ____________________ DATE: __________
COMPANY: ____________________
ADDRESS: ____________________

____

Please give this technical publication remarks form to your BULL representative or mail to:

Bull - Documentation D$^{ept.}$
1 Rue de Provence
BP 208
38432 ECHIROLLES CEDEX
FRANCE
info@frec.bull.fr

Technical publications ordering form

To order additional publications, please fill in a copy of this form and send it via mail to:

BULL CEDOC		
357 AVENUE PATTON	Phone:	+33 (0) 2 41 73 72 66
B.P.20845	FAX:	+33 (0) 2 41 73 70 66
49008 ANGERS CEDEX 01	E-Mail:	srv.Duplicopy@bull.net
FRANCE		

Reference	Designation	Qty
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
_ _ _ _ _ _ _ _ _ [_ _]		
[_ _] : The latest revision will be provided if no revision number is given.		

NAME: ____________________ DATE: __________

COMPANY: ____________________

ADDRESS: ____________________

____

PHONE: ____________________ FAX: __________

E-MAIL: ____________________

For Bull Subsidiaries:
Identification: ____________________

For Bull Affiliated Customers:
Customer Code: ____________________

For Bull Internal Customers:
Budgetary Section: ____________________

For Others: Please ask your Bull representative.

BULL CEDOC
357 AVENUE PATTON
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

REFERENCE
86 A2 29ER 07