HPC

HPC BAS4

Application Tuning Guide

REFERENCE
86 A2 19ER 06

HPC

HPC BAS4

Application Tuning Guide

Hardware and Software

December 2007

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

REFERENCE
86 A2 19ER 06

Printed in France

Suggestions and criticisms concerning the form, content, and presentation of this book are invited. A form is provided at the end of this book for this purpose.

To order additional copies of this book or other Bull Technical Publications, you are invited to use the Ordering Form also provided at the end of this book.

Trademarks and Acknowledgements

We acknowledge the rights of the proprietors of the trademarks mentioned in this manual.

All brand names and software and hardware product names are subject to trademark and/or patent protection.

Quoting of brand and product names is for information purposes only and does not represent trademark misuse.

Preface

Scope and Objectives

The purpose of this guide is to describe how to use the tools which enable the optimization of application programs for Bull High Performance Computing (HPC) platforms consisting of NovaScale clusters.

Intended Readers

This guide is for application programmers who wish to tune and optimize their code so that it fully exploits the processing power available.

Prerequisites

The installation of all the hardware and software components of the HPC system must have been completed.

Structure

This guide is organized as follows:

Chapter 1. Looks at the *Performance Monitoring and Application Profiling Tools* to be used to identify areas where application program performance could be improved.

Chapter 2. *Coding and Compiling Optimization.* Looks at some coding tips and compiling options to help improve the performance of your application on the Bull HPC platform. Guidelines are given in order to ensure that the application program runs as efficiently as is possible.

Chapter 3. *Program Execution Optimization.* Describes how to optimize and launch your program.

Chapter 4. *Message Passing Interface Optimization.* Looks at some optimization tips for the Message Passing Interface(MPI).

Chapter 5. *Lustre File System Optimization.* Describes how the Lustre (CFS) parallel file system should be optimized.

Appendix A Describes *Tuning LibElan4 Variables* for **Quadrics** systems.

Appendix B Describes *Amdahl's Law*.

Appendix C Describes the *CPUSET Virtual Filesytem*.

References

- Bull HPC BAS4 *Installation and Configuration Guide* (86 A2 28ER) – for information on how to install Bull's HPC software on Bull's platforms.
- Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) – for information on configuring and managing Bull's HPC clusters. Also provides information on monitoring and managing resources for the whole system.
- Bull HPC BAS4 *Maintenance Guide* (86 A2 46ER) – for information on maintenance tasks and troubleshooting.
- Bull HPC BAS4 *User's Guide* (86 A2 29ER) – for information on using the applications, compilers, libraries and debugging tools provided with Bull's HPC platforms.
- The *Software Release Bulletin* (SRB) provides release-specific information and details of restrictions resulting from known problems.
- Bull *Voltaire Switches Documentation CD* (86 A2 79ET) for clusters which include **Voltaire** Adapters and Switches.
- NovaScale 40xx and NovaScale 5xxx & 6xxx documentation.
- NovaScale Master documentation.
- *Intel® Itanium® 2 processor Reference Manual for Software Development and Optimization.*
- *Intel® Introduction to Microarchitectural Optimization for Itanium® 2 Processors Reference Manual.*

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/

https://mail.clusterfs.com/wikis/lustre/LustreDocumentation

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. For example: `<node_name>`

Table of Contents

List of Figures

Chapter 1. Performance Monitoring and Application Tools

This chapter looks at the Performance Monitoring and Application Profiling Tools to be used to identify areas where application program performance could be improved.

The following topics are described:

- 1.1 *Tools for Optimizing HPC Performance*
- 1.2 *System Monitoring Tools*
- 1.3 *Ganglia Cluster Performance Monitoring*
- 1.4 *IOstat*
- 1.5 *Perfmon*
- 1.6 *Perfware*
- 1.7 *mpianalyser and profilecomm*
- 1.8 *PAPI*
- 1.9 *Profiling Programs – HPCToolkit*
- 1.10 *Itanium® 2 Processor Events and Intel® Trace Tools*

1.1 Tools for Optimizing HPC Performance

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

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

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

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

The recommended approach is to use a command that shows the evolution of the program live. **Top** does this giving a typology of the program. If there is a requirement to log information during the whole execution of the program then a tool such as **perfware** can be used.

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

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

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

The following figure indicates the performance tools which are best adapted to the different stages of the optimization process.

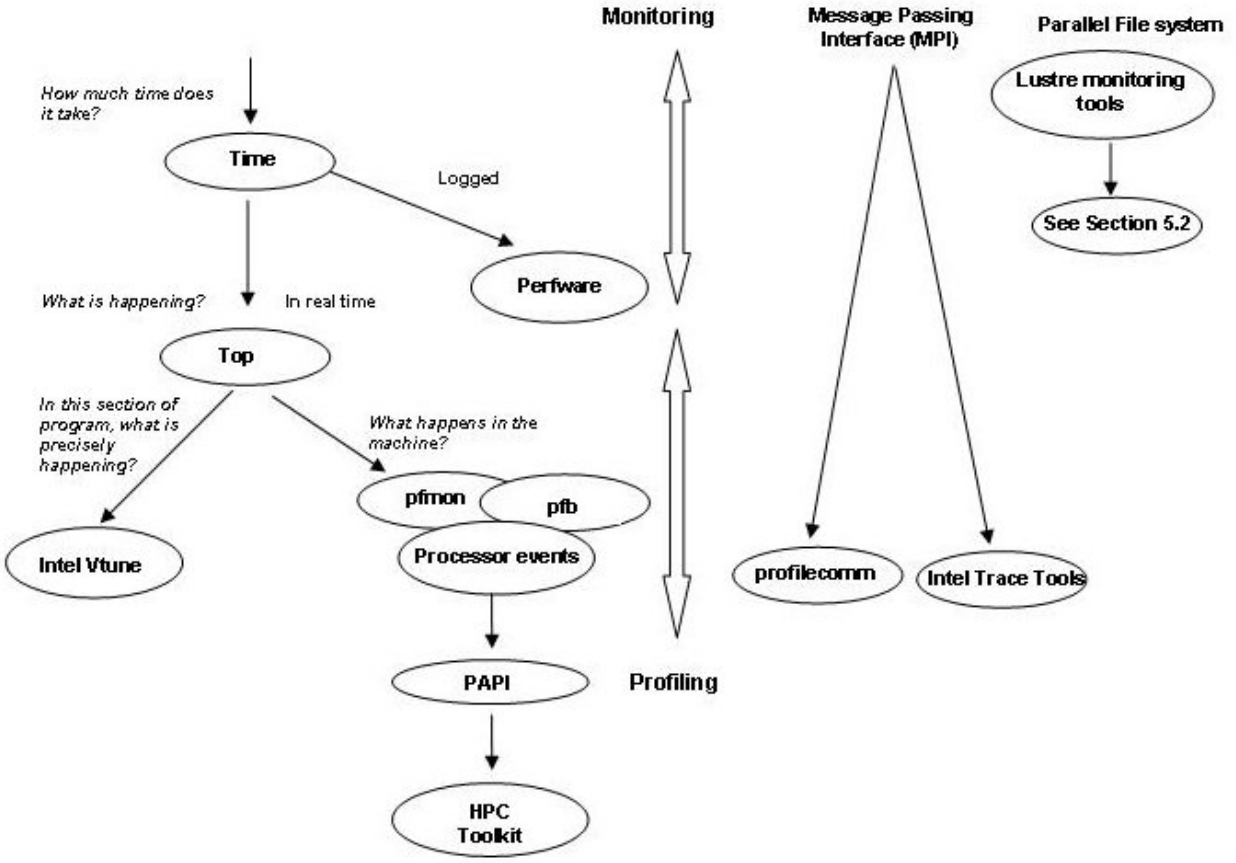

Figure 1-1. Diagram to show performance tools for different parts of the system

Note:
Intel® Trace Tools (Trace Analyzer and Trace Collector) and **Intel® Vtune™ Peformance Analyzer** are proprietary software available from **Intel**

The tools referred to in this chapter should be used in the sequence indicated above to determine where the performance of the application could be improved.

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

There are two major kinds of tools:

1. Performance monitoring tools like **pfmon** and **top**. **pfmon** provides an access to the performance registers of Itanium 2 located in the Performance Monitoring Unit. The version of **top** included in the **BAS4** delivery is an extension of the standard version of top.

2. Profiling tools using the **PAPI** library.

Intel Vtune are used to perform post mortem analysis of the output after the application has completed its execution and they cannot be used during run-time. **PAPI**, on the other hand, may be used to read event counters during run-time.

HPC Toolkit, an open source tool based on **PAPI**, is included in the BAS4 delivery. This profiles the application in a similar way as **Intel Vtune.**

1.2 System Monitoring Tools

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.** The **NovaScale 3005 Series** 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.

1.2.1 Time

The first determinant to find is the run-time for a specific operation; this will be used as a yardstick in the optimization process. Different benchmark operations, similar to those defined in the call to tender, can be used.

The **time** command is used to measure the duration of execution for a particular operation. The execution time is reported in terms of user CPU time, system CPU time, and real time.

The **etime** function is used to give the time of execution for a particular part of the application program.

1.2.2 Top

The second tool to use is **top**. This is a Linux command which provides global information in real time about the behavior of the system including:

- CPU and memory usage. Bull has added additional code to the tool which provides access to information concerning I/Os, storage devices and network usage.
- A list of the processes launched.
- Machine time used by each process in descending order (those that consume most time are displayed first). This information means that the processes that need most resources can be detected easily, and the operation of the system monitored.
- The localization and migration of applications.

Top should be launched with the application running in parallel.

An overview of what the **top** command can provide follows.

With **pfbd** running in the background NovaScale will provide essential information concerning memory localization rates and memory bandwidth use for each QBB. **pfbd** is included in the Bull Linux kernel. Only one occurrence of **pfbd** can run on a single node, but several users can use **top** in parallel. This is why these commands are separated.

There is a wide range of parameters available, refer to the man page for this tool which is activated by hitting the 'h' key and then enter.

In a **QBB** there are up to 4 CPUs (t2) which communicate with memory and the I/O interface through the Scalable Node Controller (**SNC**) – a specific chip for memory access.

The following figures show a NovaScale QBB and some of the different architectures which are possible for **NovaScale 5xxx\6xxx Series** platforms:

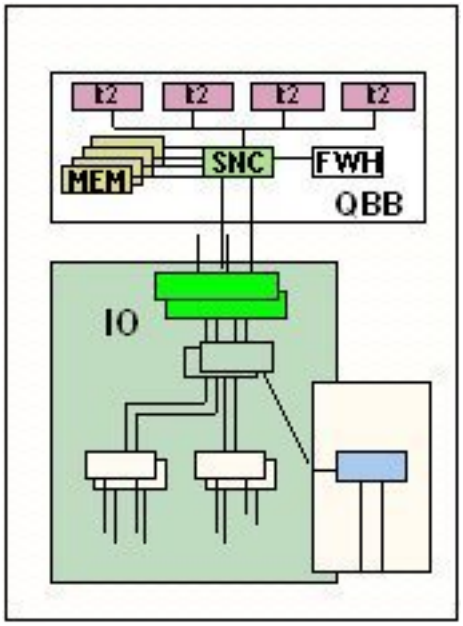

Figure 1-2. Diagram showing the layout for one QBB

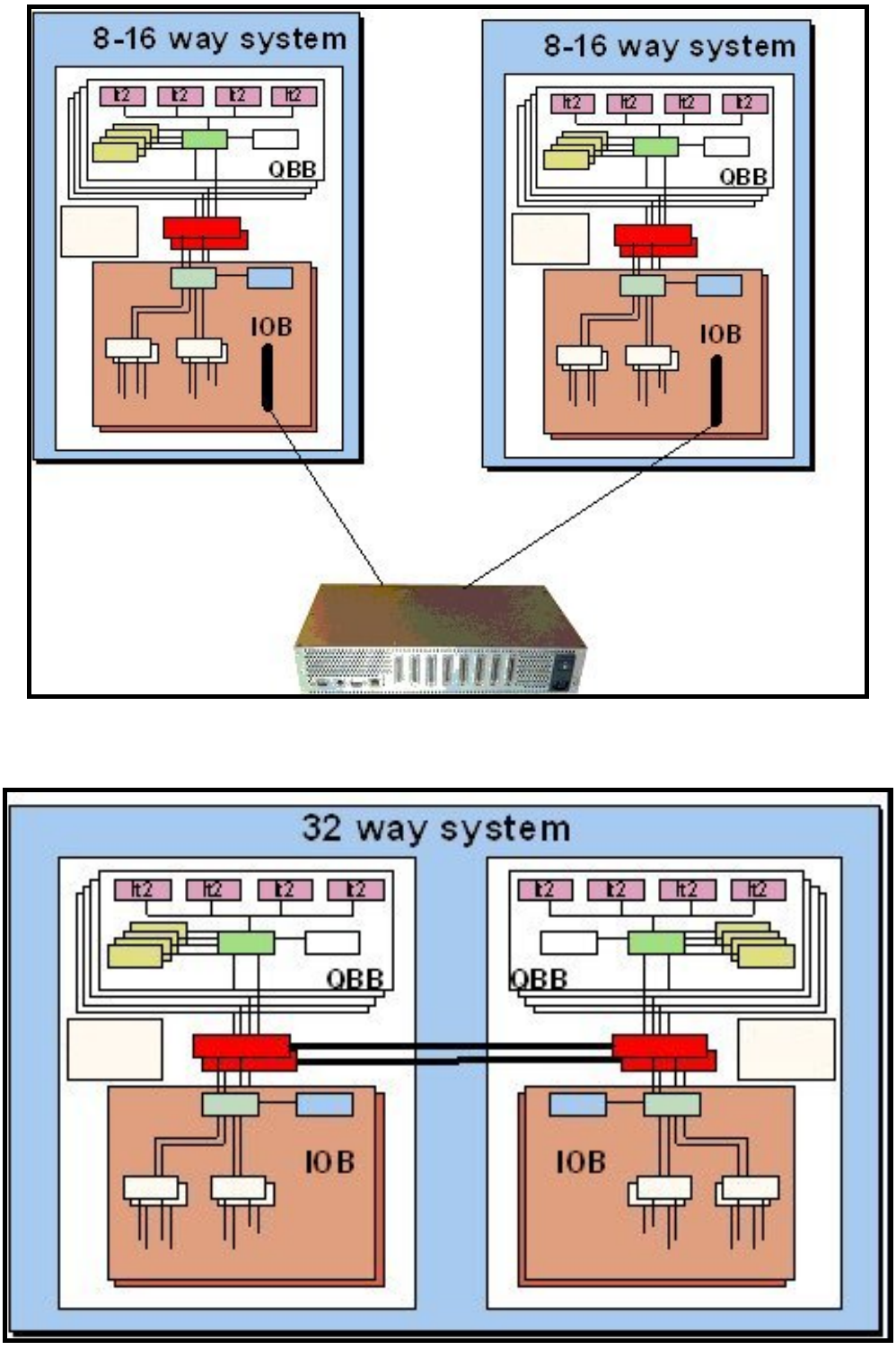

Figure 1-3. Two different systems

1.2.3 Using Top

top features many different parameters to be used for the monitoring of resources.

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

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

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

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

Meaning of the CPU percentages:

us User - regular user apps
sy Syst - system (kernel usage)
ni Nice - nice user apps (low-priority tasks)
id Idle
wa Wait - waiting for IO to complete
hi Hard interrupt (IRQ) handlers
si Soft interrupt (mainly network stacks) handlers

top options

General options:

h Displays help
d Defines the time between two measurements
x Displays disk activity
E Displays network activity
C Configures how disk and network activities are displayed

These commands display the option chosen at the top of the screen. This display is toggled on and off using the letter indicated:

W Logs the top configuration
l Displays the **load average**
t Displays the **Tasks**
m Displays the **Memory**
1 Displays the load by CPU, or globally for all

Options for processes:

L Displays the processes
P Sorts the processes according to the CPU load
M Sorts the processes according to the memory use
R Reverses the sort
f Selects the columns to be displayed

F Selects the columns to be sorted
u Selects the processes for a user
or n Defines the maximum number of processes to be displayed

To save the results produced by **top** use the batch option (**-b**). Redirect the output into a file and set the options as parameters. For example:

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

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

1.2.4 Specific Top Options for NovaScale

2 CPU load by QBB or Node Controller

N Activates the memory measurements of the **SNC\NC**, which have been collected by the **pfbd** tool (see **Pfb**, **Pfbd** and **Pfbd_activity** in section 1.5.3. & 1.5.4.).

SNC\NC measurements are made using a kernel patch, in mono access, which allows only one active instance of **pfbd** for each machine.

The NovaScale options **N** and **2** have to be used interactively when **top** is already running. To use these options when running batch mode, firstly run **top** in interactive mode, enable the **N** and **2** options, save the configuration with the **W** option, then run top in batch mode.

The following screen shot shows an example of **top** output, using **N** and **2** options, on an idle machine on which **pfbd** is running.

```
top – 16:41:37 up 1 day, 31 min, 14 users, load average: 0.08, 0.14, 0.11
Tasks : 306 total, 1 running, 305 sleeping, 0 stopped, 0 zombie
QBB0  :    0.6% us,   0.4% sy,   0.0% ni, 599.0% id,  0.0% wa,   0.0% hi,
0.0% si
QBB1  :    0.0% us,   0.0% sy,   0.0% ni, 600.0% id,  0.0% wa,   0.0% hi,
0.0% si
QBB2  :    0.1% us,   0.0% sy,   0.0% ni, 599.0% id,  0.0% wa,   0.0% hi,
0.0% si
QBB3  :    0.0% us,   0.0% sy,   0.0% ni, 600.0% id,  0.0% wa,   0.0% hi,
0.0% si
Mem:  50193856k total,  5378304k used, 44815552k free, 1643264k buffers
Swap: 73735040k total,     0k used, 73735040k free, 2441856k cached
Memory localisation rate |  Memory bandwidth use MB/s
QBB0  :       92.8%      |     0.6%
QBB1  :       38.6%      |     0.1%
QBB2  :       36.9%      |     0.1%
QBB3  :       47.2%      |     0.1%
```

QBB utilization may reach 400% if 4 CPUs are configured in the QBB. This value actually corresponds to the sum of percentages of all the CPUs.

Note:

The Memory localization rate is a key performance indicator to look at here. For the best performance of the platform the rate should be as close as is possible to 100% (>80%) for each QBB when there are high memory bandwidth rates, during program execution.

1.3 Ganglia Cluster Performance Monitoring

Ganglia is a scalable distributed monitoring system for high-performance computing systems such as those used in Bull HPC systems. It is based on a hierarchical design targeted at federations of clusters. It leverages widely used technologies including **XML** for data representation, **XDR** for compact, portable data transport, and **RRDtool (Round Robin Database tool)** for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve a very low per-node overhead and high concurrency.

NovaScale Master – HPC Edition uses a GUI to display the Ganglia data for the hardware system. This can be used to monitor the performance of the systems and detect any variations within it.

To start the Ganglia graphic interface use the following URL:

Erreur ! Référence de lien hypertexte non valide.

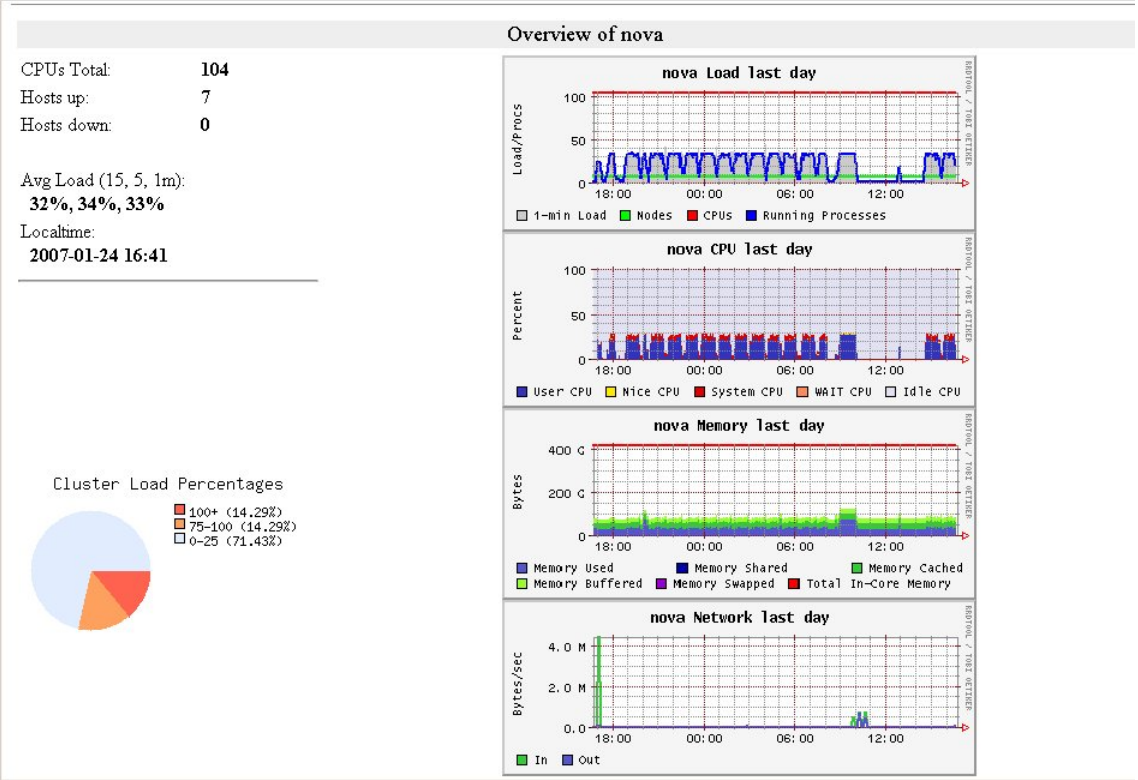

Figure 1-4. Ganglia overview of a Cluster

For more information on the cluster views which are available under NovaScale Master and how the hardware performance of the HPC system can be monitored, refer to Chapter 5 of the *HPC BAS4 Administrator's Guide* and Chapter 8 of the *Bull NovaScale Master Administrator's Guide*.

The parameters which enable the calculation of the performance of the cluster are collected on all nodes by **Ganglia**. The results may be viewed within **NovaScale Master - HPC Edition** by clicking on the Group Performance or Global Performance button.

Different categories of data are collected, including the following:

- Processors.
- Memory.
- Disks.
- Network (admin).
- Interconnect.
- Lustre (for systems which use the Lustre file system).

1.3.1 Group Performance Global View

This view displays diagrams of difference performance metrics for a selected set of nodes. Each diagram shows the evolution of the metric concerned over a user-defined period of time.

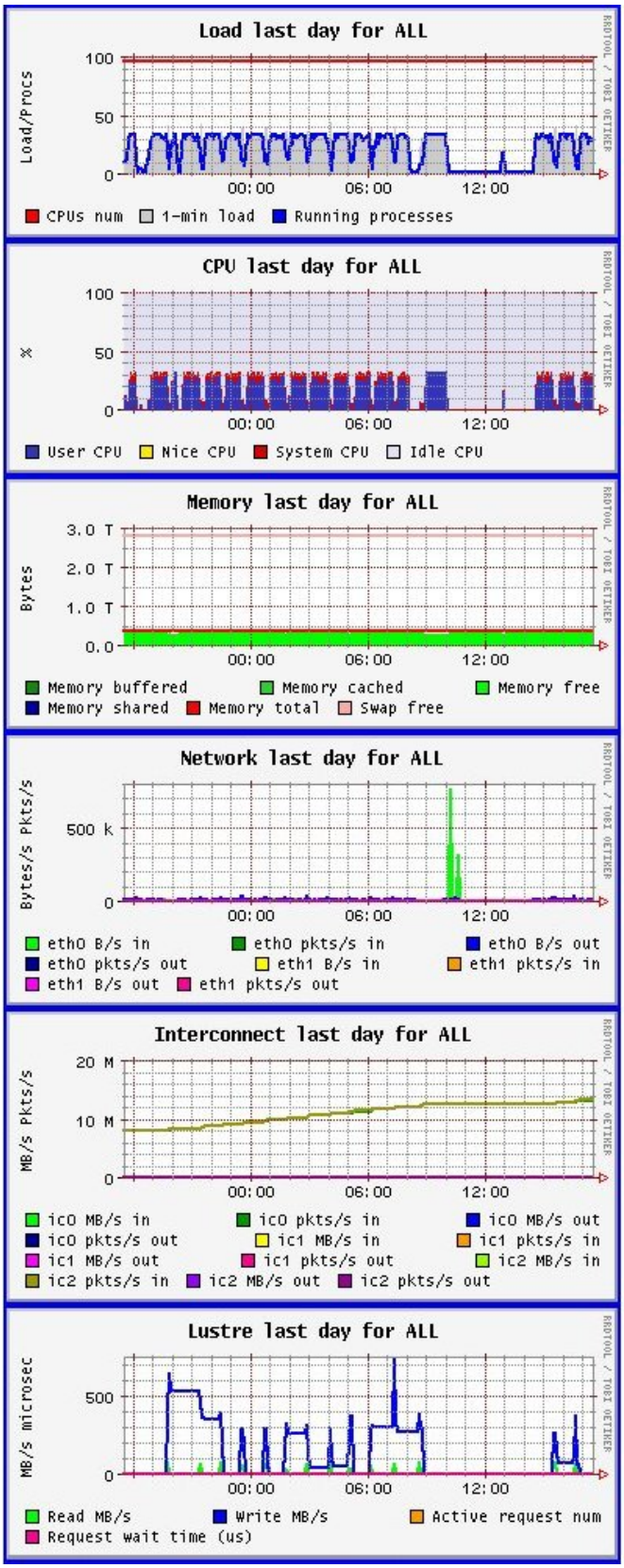

Figure 1-5. Ganglia Group Performance Global view

Clicking on a diagram will display graphs with more detailed information.

1.3.2 Detailed Cluster Performance View

This view displays the global performance diagram for each type of metrics and the diagrams for the ten first nodes which are sorted in ascending or descending order of the metric value.

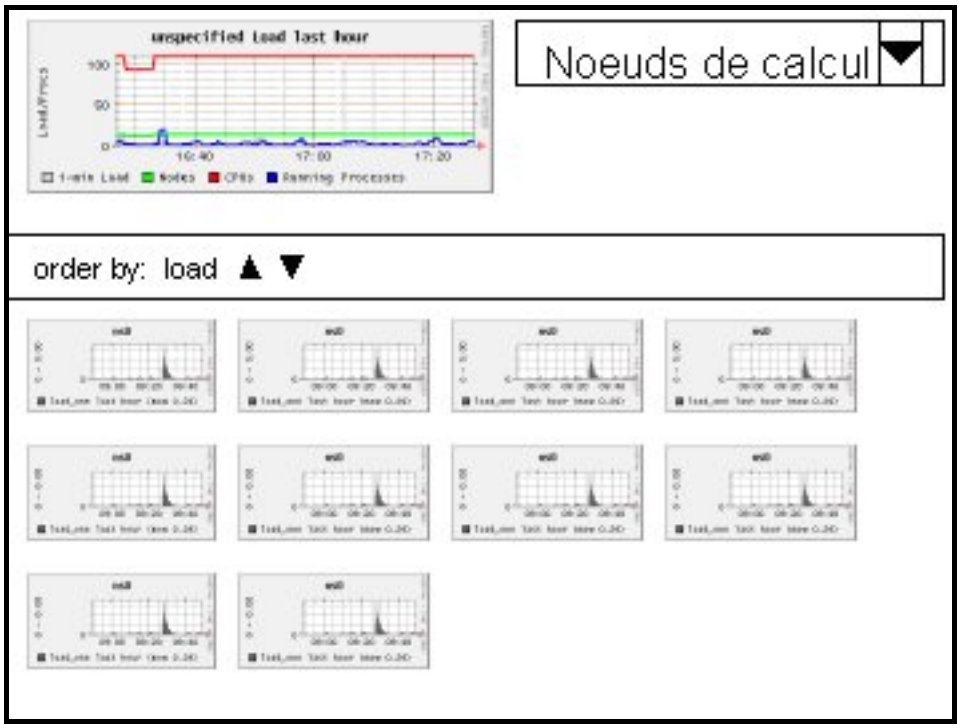

Figure 1-6. Ganglia detailed performance view

Using the monitoring data it is possible to identify areas where performance is being lost and as a result to make changes to the system or the application, and then verify the changes to see if the performance has improved.

1.4 IOstat

The **iostat** Linux command is used for monitoring system input/output device loading by observing the time the devices are active in relation to their average transfer rates. The **iostat** command generates reports that can be used to change a system's configuration to better balance the input/output load between physical disks.

Performance problems may be the result of too many files being repeatedly opened, read and written to, and then closed. This type of problem is indicated by increasing seek times and may be identified using **iostat.**

The first report generated by the **iostat** command provides statistics for the time elapsed since the system was first booted. Each subsequent report covers the period of time since the previous report. The interval parameter stipulates the time period in seconds for each report.

The count parameter may be used with the interval parameter. These determine the number of reports generated and the time period for each report. If the interval parameter is used without the count parameter, the **iostat** command generates reports continuously.

All I/O statistics are collected each time the **iostat** command runs. The report consists of a CPU header row followed by a row of CPU statistics. On multiprocessor systems, CPU statistics are calculated system-wide as averages among all processors. A device header row is displayed followed by a line of statistics for each device that is configured.

The **iostat** command generates two types of reports, the CPU Utilization report and the Device Utilization report.

- On multiprocessor systems the CPU Utilization Report provides the CPU values which are global averages for all processors.
- The Device Utilization Report provides statistics either by physical device or by partition.

Eamples

```
iostat -x 2 4
```

displays four reports of extended statistics at two second intervals.

```
iostat -x hda hdb 2 6
```

displays six reports of extended statistics at two second intervals for devices **hda** and **hdb**.

For more information on the formats of the reports and the commands which are available refer to the man page for **iostat**, alternatively look at <http://linuxcommand.org/>

1.5 Perfmon

With the **perfmon** driver running as part of the Linux kernel access to the CPU counters is obtained. **pfmon** is the command level tool which enables the user to look at the information which perfmon records.

1.5.1 Pfmon, Pfmond and Pfmond_activity

The **pfmon** command may be executed on a program. It cannot take its descendants into account. It may return from 1 to 12 known events or counters according to the processor in use, for example for Madison processors 4 events are monitored whilst for Montecito processors 12 events are monitored.

pfmon is quite simple to use. It is run with the executable to be surveyed specified. The hardware events for each execution of an application can then be examined, either for the whole of the execution run time or for a specific period of time, using the system wide option.

pfmond is a Bull utility which runs in the background as a daemon and monitors up to12 hardware events for a period of time. The hardware events will be monitored in groups, 2 groups of 4 events on Madison processors and 1 group of up to 12 events on Montecito. Care has to be taken in configuring the groups, as some events are not compatible. The only way to determine compatibility is by trial and error, if there is an incompatibility in the events chosen than an error message will be displayed.

Pfmond_activity presents **pfmond** information in different ways, for example either by time or by CPU cycle. The aim is to follow in real time what is happening for hardware events when an application is running. This is normally used for larger, more time consuming applications where the programmer has a good knowledge of the functions of their application.

1.5.2 Using pfmon

Pfmon is a good tool to obtain a figure quickly for a particular operation or process of a program, without any programming. For example, to get the number of floating point operations in the **test2** program, run:

```
pfmon -e FP_OPS_RETIRED test2
```

The output will be similar to that shown below:

```
2 421 914 081 FP_OPS_RETIRED
```

It is possible to capture data for up to 4 events using the same command and to obtain detailed output using the **with-header** option. For example:

```
pfmon --with-header -k -u
-e L2D_MISSES,L2D_REFERENCES,L3_MISSES,IA64_INST_RETIRED
--outfile=result_pfmon/pfmon_miss_16.2081 runspec ……..
```

The output will be similar to that shown below:

```
#
# date: Fri Feb 21 02:16:14 2003
#
# hostname: chircane
#
# kernel version: Linux 2.5.59 #2 SMP Wed Feb 12 06:17:00 PST 2003
#
# pfmon version: 2.0
# kernel perfmon version: 1.3
#
#
#
# page size: 16384 bytes
# CLK_TCK: 1024 ticks/second
# CPU configured: 4
# CPU online: 4
# physical memory: 4189208576
# physical memory available: 2908225536
#
# host CPUs:  4-way 997MHz Itanium 2 (McKinley, B3)
#       PAL_A: 0.7.31
#       PAL_B: 0.7.36
#       Cache levels: 3 Unique caches: 4
#       L1D:    16384 bytes, line  64 bytes, load_lat   1, store_lat   3
#       L1I:    16384 bytes, line  64 bytes, load_lat   1, store_lat   0
#       L2 :   262144 bytes, line 128 bytes, load_lat   5, store_lat   7
#       L3 :  3145728 bytes, line 128 bytes, load_lat  12, store_lat   7
#
#
# captured events:
#       PMD4: L2_REFERENCES, kernel+user level(s)
#       PMD5: L2_MISSES, kernel+user level(s)
#       PMD6: L3_MISSES, kernel+user level(s)
#       PMD7: IA64_INST_RETIRED, kernel+user level(s)
#
# monitoring mode: per-process
#
#
# instruction sets:
#       PMD4: L2_REFERENCES, ia32/ia64
#       PMD5: L2_MISSES, ia32/ia64
#       PMD6: L3_MISSES, ia32/ia64
#       PMD7: IA64_INST_RETIRED, ia32/ia64
#
#
# command: pfmon --with-header -k -u -e
L2_MISSES,L2_REFERENCES,L3_MISSES,IA64_INST_RETIRED --
outfile=result_pfmon/pfmon_miss_16.2081 runspec -c il420-linux-v7 --
iterations=1 --tune=base -I --noreportable --rate --users 16 applu
#
#
#
                  4234016 L2_MISSES
                 69799519 L2_REFERENCES
                   527948 L3_MISSES
                772924788 IA64_INST_RETIRED
```

Notes:

- The **–follow** option is useful to follow all the descendants of a process.
- When using **pfmon** with MPI processes each process is treated separately and a global result for the processes is not aggregated.

1.5.3 Using pfmond and pfmond_activity

pfmond runs as a daemon and monitors the hardware events specified (or the default ones) for the sample period. Events are specified using either the **–e** or **––event** options in the format **–e <ev1,ev2, etc.>**, and the sample period using either the **-t** or **––sample_time** options in the format **–t <secs>**

The results obtained for the sample period are aggregated with those results obtained previously, and then written into files named **cpu01, cpu02, cpu03**, (one file per CPU) within the file specified by the **–d** option or in the default file which is **/var/run/pfmond**.

Pfmond command example

```
pfmond -e IA64_INST_RETIRED,FP_OPS_RETIRED,BE_RSE_BUBBLE_ALL
```

pfmond_activity displays the instant average of all the occurrences for each hardware event that is monitored by **pfmond**. The command to launch **pfmond_activity** is shown below:

```
pfmond_activity
```

An example of output for this command is shown below:

```
Pfmond_activity : measure at 10:28:14
       | IA64_INST_RETIRED_THIS |    FP_OPS_ |    BE_RSE_
       |                        |     RETIRED| BUBBLE_ALL
       |------------------------|------------|------------
Cpu  0 |             4807346k/s |       6k/s |      11k/s
Cpu  1 |             4807543k/s |       7k/s |      12k/s
Cpu  2 |             4807957k/s |       4k/s |       9k/s
Cpu  3 |             4808000k/s |      25k/s |       8k/s
Cpu  4 |             4808268k/s |       4k/s |       7k/s
Cpu  5 |             4808214k/s |       4k/s |       8k/s
```

1.5.4 Pfb and Pfbd

The **perfbull** driver is complementary to **perfmon** but instead of accessing the CPU counters it measure the counters for the **SNC** (Scalability Node Controller) NovaScale registers **ISPS** and **BSPS**.

Pfb is a command level tool included in **top** which enables the information which **perfbull** records to be looked at. **Pfb** is an embedded tool which allows other hardware events to be examined.

1.5.5 Pfbd_activity

pfbd_activity presents **pfbd** information in different ways, for example either by time or by CPU cycle. The aim is to follow in real time what is happening for hardware events when an application is running.

pfbd_activity reads the hardware monitoring results provided by **pfbd** in the file **/var/run/pfbd/stats** and displays the instant average per second for the memory localization and/or the use of the memory bandwidth, for each QBB.

The memory localization rate is the ratio of the QBB memory access for the QBB processors to the memory access from the processors of the other QBBs. This rate is only significant if there is a lot of memory access. In this case, the higher the rate is, better it is.

The availability of each statistic depends on the **pfbd** mode. **Pfbd** has to be launched before using **pfbd_activity**.

OPTIONS:

-h, help	Displays a short help text.
-t, --sample_time=secs	Set the refreshing display period.
-d, --dir=directory	Use the given directory to find the stat file instead of /var/run/pfbd.
-D,--debug	Set debug mode to ON.
-B,--batch	Set batch mode to ON: this output is more adapted to be redirected into a file and parsed in post treatment.
-c,--check	Check if a **pfbd** session is running.


See the **pfbd_activity** man page for more information.

1.6 Perfware

Unlike **top**, which records information in real time, **perfware** collects information and logs it for a defined period of time so that a statistical analysis can be made later. The information may also be represented graphically.

perfware is a tool used to monitor a system non-intrusively whilst it is in operation. It can be used to collect cluster information remotely.

perfware is used interactively through a command line interface. For information on the options available refer to the **perfware_collector**.

1.6.1 Perfware Data Collection

Data collection is defined by the following parameters - starting time and length of time between the collections. Collections may be performed either locally or remotely and may be started either interactively or automatically.

- In interactive mode run the **collector** command and answer the questions as they are displayed. The sequence of questions displayed is shown below.

```
perfware_collector
```

```
Start Now or Later ? [N/L] ...N
Save directory path is /home/perfware ? [Y/N] ...
Sampling interval (seconds) ? [1-3600][60] ...
Collection time (minutes) ? [1-10080][1440] ...
Collection will record 1440 samples over 1 day 0 h 0 min 0 sec
Do you agree ?[Y/N] ...
```

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

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

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

```
perfware_collector -R [-l] [-N|-M|-L date] [-i interval] [-d duration] [-s
savedir] [-r remote_system] [–p remote_savedir]
```

1.6.2 Analyzing Data in Interactive Mode

To start the data analysis in interactive mode you must be in the directory used for the collection and then call the **parser** command. Follow the instructions as they are displayed.

```
cd /tmp/perfware_nameserver_date_hour
```

```
perfware_parser
```

```
*********************************************************************
*                              PerfWare Parser                      *
*********************************************************************

Start Statistics Generation

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

If you answer N the following screen is displayed:

```
Set your collect directory path :
```

Once the directory path has been set the following screen appears:

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

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

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

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

If you choose a specific day from those listed, 11 for example, the data collected information for this day is analyzed and the result is logged in the directory indicated below:

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

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

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

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

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

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

1.6.3 Analyzing Data in Automatic Mode

To start the analyses in automatic mode use the following **parser** command:

Perfware_parser [-A] [-l limit] [-c collectpath] [-d day] [-r arg_cpu] [-m max_cpu]

parser options:

-A	Starts **parser** with the default values.
-c collectpath	Specifies the collection directory path. Default value is . or */tmp/perfware_nameserver_date_hour*
-d day	Specifies the day to parse, in one of these formats: DD or * or #: -d * parses all the days and the plots generated are stored in separate directories. -d # parses all the days as if they were only one day (00) and the plots are stored in one directory. Default value is *.
-l limit	Specifies the minimum percentage of process activity to be parsed. Default value is 1.0 (1%).

Note:

Any option can be used with **–A**, in order to change the default values.

See:

For more information on the parser options refer to the **perfware_parser** man page.

1.6.4 Generating Graphical Data Displays

In order to generate a graphical display of the data collected use the following command in the parser directory which you are using.

```
txt2ps.pl gra11/cpu.txt
```

The number 11 in the gra11 reference refers to the collection day as specified earlier. The command above will generate 3 files, **gra11/CPU_0.ps**, **gra11/CPU_0_arg_ms.xml** and **gra11/CPU_0_arg_ms.ps**.

The following command has to be used to generate the graph for the first ps file.

```
epsbb2png.pl gra11/CPU_0.ps
```

The system command **display** is used to display the graph which has been generated. A graph similar to the one on the next page will be generated to show the Global CPU occupation.

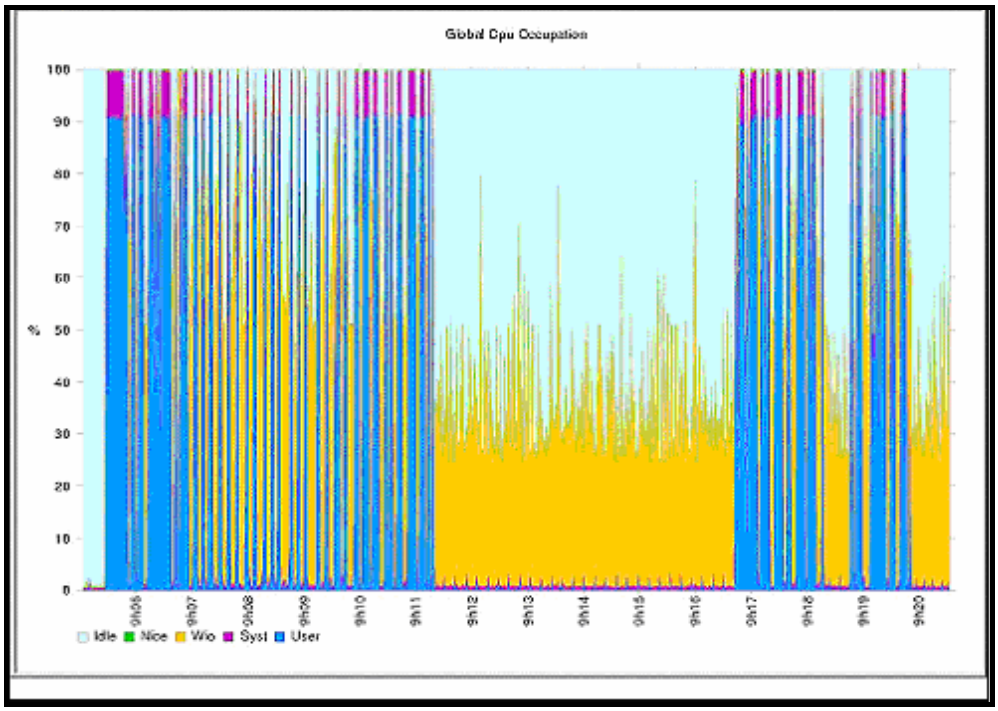

Figure 1-7. Perfware graph to show global CPU occupation

1.7 mpianalyser and profilecomm

mpianalyser is an integrated framework which uses the **PMPI** interface to analyze the behaviour of MPI programs.

Profilecomm is a part of **mpianalyser** and is dedicated to MPI application profiling. It has been designed to be:

- Light: it uses few resources and so does not slow down the application.
- Easy to run: it is used to characterize the MPI communications in a program. Communication matrices are constructed with it. **Profilecomm** is a "post-mortem" tool, which does not allow on-line monitoring.

Data is collected as long as the program is running. At the end of the program, data is written into a file for future analysis.

readpfc is a tool with a command line interface which handles the data that has been collected. Its main uses are the following:

- To display the data collected.
- To export communication matrices in a format that can be used by other applications.

Data collected

The **profilecomm** module provides the following information:

- Communication matrices
- Execution time
- Table of calls of MPI functions
- Message size histograms
- Topology of the execution environment.

1.7.1 Communication Matrices

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

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

The volume only indicates the payload of the messages.

In order to compute the standard deviation of messages size, two other matrices are collected. They contain the sum of squared messages sizes for **point-to-point** and for collective communications.

In order to get more precise information about messages sizes, each numeric matrix can be split into several matrices according to the size of the messages. The number of partitions and the size limits can be defined through the **PFC_PARTITIONS** environment variable. In a point to point communication, the sender and receiver of each message is clearly identified, this results in a well defined position in the communication matrix.

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

Execution Time

The measured execution time is the maximum time interval between the calls to **MPI_Init** and **MPI_Finalize** for all the processes. By default the processes are synchronized during the measurements. However, if necessary, the synchronization may be by-passed using an option of the **profilecomm** library.

Call Table

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

Histograms

Profilecomm collects two messages size histograms, one for point-to-point and one for collective communications. Each histogram contains the number of messages for sizes 0, 1 to 9, 10 to 99, 100 to 999, ..., 10^8 to 10^9-1 and bigger than 10^9 bytes.

Topology of the Execution Environment

The **profilecomm** module registers the topology of the execution environment, so that the machine and the CPU on which each process is running can be identified, and above all the intra- and inter-machine communications made visible.

1.7.2 Profilecomm Data Collection

When using **profilecomm** there are 2 separate operations – data collection, and then its analysis.

Using Profilecomm

To be profiled by **profilecomm**, an application must be linked with the **MPI Analyser** library.

profilecomm is disabled by default, to enable it, set the following environment variable:

```
export MPIANALYSER_PROFILECOMM=1
```

When the application finishes, the results of the data collection are written to a file (**mpiprofile.pfc** by default). By default this file is saved in a format specific to **profilecomm**, but it is possible to save it in a text format. The **readpfc** command enables **.pfc** files to be read and analysed.

1.7.3 Profilecomm Options

Different options may be specified for **profilecomm** using the **MPIPROFILE_OPTIONS** environment variable.

For example:

```
Export MPIPROFILE_OPTIONS="-f foo.pfc"
```

Some of the options that modify the behavior of **profilecomm** when saving the results in a file are below:

-f *file*, -filename *file*
Saves the result in the **file** file instead of the default file (**mpiprofile.txt** for text format files and **mpiprofile.pfc** for **profilecomm** binary format files).

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

-b, -bin
Saves the results in a **profilecomm** binary format file. This is the default format. The **readpfc** command is required to work with these files.

-s, -sync
Synchronizes the processes during the time measurements. This option is set by default.

-ns, -nosync
Doesn't synchronize the processes during the time measurements.

-v32, -volumic32
Use 32 bit volumic matrices. This can save memory when profiling application with a large number of processes. A process must not send more than 4GBs of data to another process.

-v64, -volumic64
Use 64 bits volumic matrices. This is the default behavior. It allows the profiling of processes which exchanges more than 4GBs of data.

Examples

To profile the **foo** program and save the results of the data collection in the default file **mpiprofile.pfc**:

```
$ MPIPROFILE=profilecomm prun –p my_partion –N 1 -n 4./foo
```

To save the results of the data collection in the `foo.pfc` file:

```
$ MPIPROFILE=profilecomm MPIPROFILE_OPTIONS="-f foo.pfc" prun –p my_partion –N 1 -
n 4./foo
```

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

```
$ MPIPROFILE=profilecomm MPIPROFILE_OPTIONS="-t -f foo.txt" prun –p my_partion –N
1 -n 4./foo
```

1.7.4 Messages Size Partitions

Profilecomm allows the numeric matrices to be split according to the size of the messages. This feature is activated by setting the PFC_PARTITIONS environment variable. By default, there is only one partition, i.e. the numeric matrices are not split.

The **PFC_PARTITIONS** environment variable must be of the form **[partitions:] [limits]** in which **partitions** is the number of partitions and **limits** is a comma separated list of sorted numbers representing the size limits in bytes.

If **limits** is not set, **profilecomm** uses the built-in default limits for the requested partition number.

Example 1 - 3 partitions using the default limits (1000, 1000000):

```
$ export PFC_PARTITIONS="3:"
```

Example 2 - 3 partitions using user defined limits (in this case, the partition number can be safely omitted):

```
$ export PFC_PARTITIONS="3:500,1000"
```

Or

```
$ export PFC_PARTITIONS="500,1000"
```

1.7.5 Profilecomm Data Analysis

To analyze data collected with **profilecomm** the **readpfc** command and other tools, including spreadsheets, can be used. The main features of **readpfc** are the following:

- Displaying the data contained in **profilecomm** files.
- Exporting communication matrices in a standard file format.

readpfc syntax

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

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

Readpfc output

The main feature of **readpfc** is to display the information contained in the seven different sections of a **profilecomm** file. These are:

- Header
- Point to point

- Collective
- Call table
- Histograms
- Statistics
- Topology.

Header Section:

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

- **Elapsed Time** – indicates the length of the data collection.
- **World size** - indicates the number of processes.
- **Number of partitions** – indicates the number of partitions.
- **Partitions limits** – indicates the list of size limits for the messages partitions (only used if there are several partitions).

The other fields are less interesting for the final users but are used internally by **readpfc**.

Example:

```
Header:
 Version: 2
 Flags: little-endian
 Header size: 40 bytes
 Elapsed time: 9303 us
 World size: 4
 Number of partitions: 3
 Partitions limits:  1000  1000000
 num_intsz: 4 bytes (32 bits)
 num_volsz: 8 bytes (64 bits)
```

1.7.6 Point to Point Communications

- For point to point communication matrices, use the following. The number of communication messages is displayed first, then the volume. If either the
- **--numeric-only** or **--volumic-only** options are used then only one matrix is displayed accordingly.

Example:

```
Point to point:
 numeric (number of messages)
      0   1.1k      0      0 |   1.1k
   1.1k      0      0      0 |   1.1k
      0      0      0   1.1k |   1.1k
      0      0   1.1k      0 |   1.1k

 volumic (Bytes)
      0 818.8k      0      0 | 818.8k
 818.8k      0      0      0 | 818.8k
      0      0      0 818.8k | 818.8k
      0      0 818.8k      0 | 818.8k
```

If the file contains several partitions and the -J/--split option is set then this command displays as many numeric matrices as there are partitions. Example:

```
Point to point:
 numeric (number of messages)
 0 <= msg size < 1000
      0    800      0      0 |    800
    800      0      0      0 |    800
      0      0      0    800 |    800
      0      0    800      0 |    800

 1000 <= msg size < 1000000
      0    300      0      0 |    300
    300      0      0      0 |    300
      0      0      0    300 |    300
      0      0    300      0 |    300

 1000000 <= msg size
      0      0      0      0 |      0
      0      0      0      0 |      0
      0      0      0      0 |      0
      0      0      0      0 |      0

 volumic (Bytes)
      0 818.8k      0      0 | 818.8k
 818.8k      0      0      0 | 818.8k
      0      0      0 818.8k | 818.8k
      0      0 818.8k      0 | 818.8k
```

If the -r/--rate option is set then the messages rate and data rate matrices are shown instead of communications matrices. These rates are the average rates for all execution times not the instantaneous rates. Example:

```
Point to point:
 message rate (msg/s)
      0 118.2k      0      0 | 118.2k
 118.2k      0      0      0 | 118.2k
      0      0      0 118.2k | 118.2k
      0      0 118.2k      0 | 118.2k

 data rate (Bytes/s)
      0 88.01M      0      0 | 88.01M
 88.01M      0      0      0 | 88.01M
      0      0      0 88.01M | 88.01M
      0      0 88.01M      0 | 88.01M
```

1.7.7 Collective Section

- The collective section is equivalent to the point to point section for collective communication matrices. Example:

```
Collective:
 numeric (number of messages)
      0    102    202    102 |    406
    102      0      0    100 |    202
    202      0      0      0 |    202
    102    100      0      0 |    202

 volumic (Bytes)
      0 409.6k 421.6k 409.6k | 1.241M
 12.04k      0      0    12k | 24.04k
 421.6k      0      0      0 | 421.6k
```

```
                12.04k 409.6k      0      0 | 421.6k
```

1.7.8 Call table section

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

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

1.7.9 Histograms Section

This section contains the message sizes histograms. It shows the number of messages whose size is zero, between 1 and 9, between 10 and 99, ..., between 10^8 and 10^9-1 and greater than 10^9.

Example:

```
Histograms of msg sizes
size   pt2pt   coll  total
   0       0      0      0
   1     800      6    806
  10    1.2k      6 1.206k
 100    1.2k    500   1.7k
1000    1.2k    500   1.7k
104        0      0      0
105        0      0      0
106        0      0      0
107        0      0      0
108        0      0      0
109        0      0      0
```

1.7.10 Statistics Section

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

- The *General statistics* section contains statistics for the whole application.
- The *Per process average* section contains average per process.
- The *Messages sizes partitions* section displays the distribution of messages among the partitions. This section is only present if there are several partitions.
- For each statistic we distinguish point to point communications from collective communications.

Example:

```
General statistics:
Total time: 0.009303s  (0:00:00.009303)
                       pt2pt |       coll |      total
Messages count  |       4400 |       1012 |       5412
Volume          |   3.2752MB |  2.10822MB |  5.38342MB
Avg message size|      744B |  2.08322kB |      995B
Std deviation   |     1216.4 |     1989.1 |     1488.4
Variation coef. |     1.6341 |    0.95481 |     1.4963
Frequency msg/s |   472.966k |   108.782k |   581.748k
Throughput B/s  | 352.06MB/s | 226.62MB/s | 578.68MB/s

Per process average:
                       pt2pt |       coll |      total
Messages count  |       1100 |        253 |       1353
Volume          |    818.8kB |  527.054kB |  1.34585MB
Frequency msg/s |   118.241k |   27.1955k |   145.437k
Throughput B/s  | 88.015MB/s | 56.654MB/s | 144.67MB/s

Messages sizes partitions:
                                |   pt2pt count |    coll count  |   total
count
         0 <= sz < 1000         |  3.2e+03  73% |  5.1e+02  51% |  3.7e+03  69%
      1000 <= sz < 1000000      |  1.2e+03  27% |    5e+02  49% |  1.7e+03  31%
   1000000 <= sz                |         0   0% |         0   0% |         0   0%
```

The message sizes partitions should be examined first.

Where:

Total time Total execution time between MPI_Init and MPI_Finalize.

Messages count Number of sent messages.

Volume Volume of sent messages (bytes).

Avg message size Average size of messages (bytes).

Std deviation Standard deviation of messages size.

Variation coef. Variation coefficient of messages size.

Frequency msg/s	Average frequency of messages (messages per second).
Throughput B/s	Average throughput for sent messages (bytes per second).

1.7.11 Topology Section

This section shows the distribution of processes on nodes and processors. This distribution is displayed in two different ways.

- First, for each process the node and the CPU in the node where it is running and secondly, the list of running processes for each node.

Example- 8 processes running on 2 nodes.

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

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

1.7.12 Display Options

The following options allow different information to be displayed:

-a, --all
Displays all the information. Equivalent to **–ghimst**.

-c, --collective
Displays collective communication matrices.

-g, --topology
Displays the topology of execution environment.

-h, --header
Displays header of the **profilecomm** file.

-i, --histograms
Displays messages size histograms.

-j, --joined
Displays entire numerics matrices (i.e. not split). This is the default.

-J, --splitted
Display numerics matrices split according to messages size.

-m, --matrix, --matrices
Displays communication matrix (matrices). Equivalent to **–cp**.

-n, --numeric-only
Does not display volume matrices. This option cannot be used simultaneously with the **-v/--volumic-only** option.

-p, --p2p, --pt2pt
Displays point to point communication matrices.

-r, --rate, --throughput
Displays messages rate and data rate matrices instead of communications matrices.

-s, --statistics
Computes and displays some statistics regarding MPI communications.

-S, --scalable
Displays all scalable information; this means all information whose size is independent of number of processes. Useful when there is a great number of processes. Equivalent to **histT.**

--square-matrices
Displays the matrices containing the sum of the squared sizes of messages. These matrices are used for standard deviation computation and are useless for final users. This option is mainly provided for debugging purposes.

-t, --calltable
Displays the call table.

-T, --ct-total-only
Displays only the *Total* column of the call table. By default **readpfc** displays also one column for each process.

-v, --volumic-only
Does not display numeric matrices. This option cannot be used simultaneously with **-n/--numeric-only** option.

1.7.13 Exporting a Matrix or an Histogram

The communication matrices and the histograms can be exported in different formats which can be understood by other software programs, for example spreadsheets. Three formats are available: **CSV** (Comma Separated Values), **MatrixMarket** (not available for histogram exports) and **gnuplo**t.

It is also possible to have a graphical display of the matrix or the histogram, which is better for matrices with a large number of elements. Obviously, it is also possible to include the graphics in a report. Seven graphic formats are available: PostScript, Encapsulated PostScript, SVG, xfig, EPSLaTeX, PSLaTeX and PSTeX. All these formats are vectorial, which means the dimensions of the graphics can be modified if necessary.

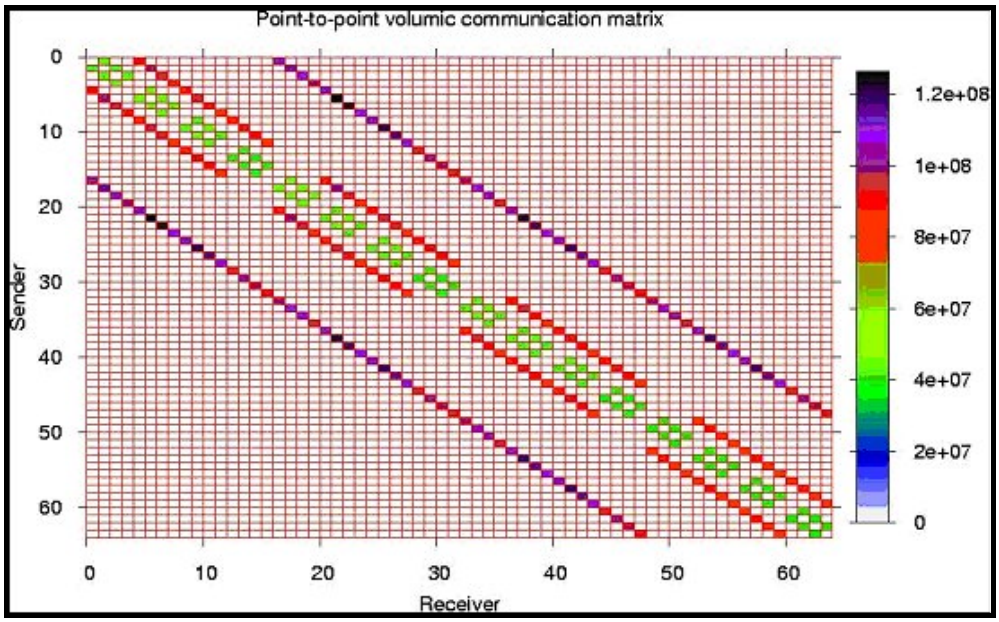

Figure 1-8. An example of a communication matrix

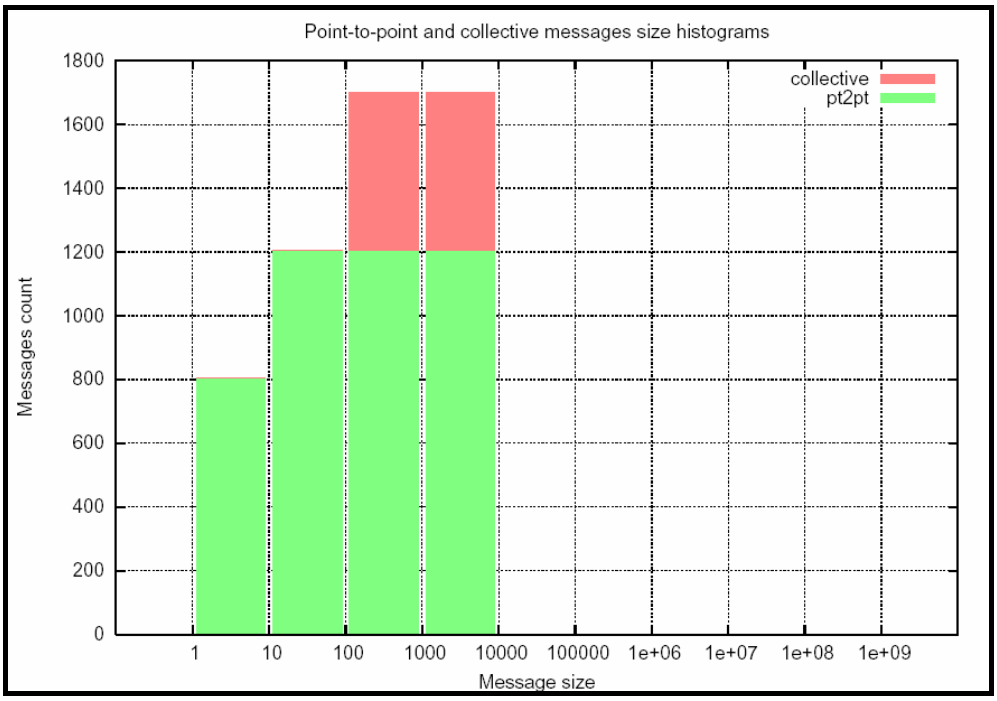

Figure 1-9. An example of a histogram

The following options may be used when exporting matrices:

--csv-separator sep	Modifies CSV delimiter. Default delimiter is comma ",". Some software programs prefer a semicolon ";".
-f format, --format *format*	Chooses export format. Default format is CSV (Comma Separated Values).
help	lists available export formats
csv	export in CSV format

mm, market, MatrixMarket	export in MatrixMarket format
gp, gnuplot	export in a format used by pfcplot so that a graphical display of the matrix can be produced
ps, postscript	export in PostScript format
eps	export in Encapsulated PostScript format
svg	export in Scalable Vector Graphics format
fig, xfig	export in xfig format
epslatex	export in LaTex and Encapsulated PostScript format
pslatex	export in LaTex format and PostScript inline
pstex	export in Tex format and PostScript inline

The available values are the following:

Important:

When using **epslatex** two files are written: **xxx.tex** and **xx.eps**. The filename indicated in the –o option is the name of the Latex file.

--logscale[=base]

Uses a logarithmic color scale. Default value for logarithm basis is 10; this basis can be modified using the **base** argument. This option is only relevant when exporting in a graphical format.

--nogrid

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

-o file, --output *file*

Specifies the file name for an export file. The default filenames are **out.csv**, **out.mm, out.dat, out.ps, out.svg, out.fig** or **out.tex**, according to export format. This option is only available with the –x option.

--palette *pal*

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

--title *title*

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

-x object, --*export object*

Exports a communication matrix or histogram specified by the ***object*** argument. Values for **object** are the following:

help	List of available matrices and histograms
pn[.part], np[.part]	Point-to-point numeric communication matrix. The optional item part is the partition number for split matrices. If part is not set, the entire matrix (i.e.the sum of the split matrices) is exported
pv, vp	Point to point volumic communication matrix
cn[.part], nc[.part]	Collective numeric communication matrix
cv, vc	Collective volumic communication matrix
ph, hp	Point-to-point messages size histogram
ch, hc	Collective messages size histogram
th, ht	Total messages size histogram (collective and point-to-point)
ah, ha	Both point-to-point and collective messages size histograms (all histograms)

Other options

-H, --help, --usage
Displays help messages

-q, --quiet
Does not display help warning messages (error messages continue to be displayed).

-V, --version
Displays program version.

Examples

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

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

This will give information similar to that below

```
Header:
 Version: 2
 Flags: little-endian
 Header size: 40 bytes
 Elapsed time: 9303 us
 World size: 4
 Number of partitions: 3
 Partitions limits:  1000  1000000
 num_intsz: 4 bytes (32 bits)
 num_volsz: 8 bytes (64 bits)
[...]
Topology:
4 process on 1 hosts
process hostid  cpuid
      0      0      0
```

```
      1      0      1
      2      0      2
      3      0      3

host  processes
   0  0 1 2 3
```

- To display a point to point numerical communication matrix:

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

```
Point to point:
 numeric (number of messages)
      0   1.1k      0      0 |   1.1k
   1.1k      0      0      0 |   1.1k
      0      0      0   1.1k |   1.1k
      0      0   1.1k      0 |   1.1k
```

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

```
$ readpfc –x cv foo.pfc
```

```
Warning: No output file specified, write to default (out.csv).
```

```
$ ls out.csv
```

```
out.csv
```

- To export the first part (small messages) of point to point numerical communication matrices in PostScript format in the foo.ps file:

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

```
foo.ps
```

1.7.14 pfcplot, histplot and gnuplot

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

For more details enter:

```
man pfcplot
```

Users who have particular requirements can invoke **gnuplot** directly. To do this the matrix must be exported with **gnuplot** format or with CSV format, choosing space as the separator.

Important:

Due to the limitations of **gnuplot**, one null line and one null column are added to the exported matrix in **gnuplot** format.

Histplot is the equivalent of **pfcplot** for histograms. Like **pfcplot**, it can be used directly by users but it is not user-friendly. More details are available from the man page:

```
man histplot
```

1.8 PAPI

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

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

PAPI supplies two interfaces:

- A high-level interface, for simple measurements,
- A low-level interface, programmable, adaptable to specific machines and linking the measurements.

PAPI with the **pfmon** driver running in the background provides more detailed information about the operation of the Itanium® 2 processors.

PAPI should only be used by specialists interested in optimizing scientific programs. These specialists can focus on code sequences using PAPI functions.

Top and **PAPI** are all open source tools.

1.8.1 Using PAPI

The PAPI library offers a set of functions for the programmer. The specific release for IA64 Itanium II manages the same counters as PAPI. Therefore, its operation consists firstly in adding elements inside the program.

Following are described the operations for the two interface levels, high-level and low-level.

- High level manages a set of classical counters called **preset.**
- Low level manages all **pfmon** counters.

1.8.2 High-level PAPI Interface

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

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

Note:

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

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

```
#include <papi.h>
```

```
#define NUM_FLOPS 10000
#define NUM_EVENTS 1

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

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

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

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

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

do_flops(NUM_FLOPS);

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

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

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

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

Possible Output:

```
After reading the counters: 441027

After adding the counters: 891959

After stopping the counters: 443994
```

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

1.8.3 Low-level PAPI Interface

The low-level API manages hardware events in user-defined groups called **Event Sets**. It is particularly well designed for experienced application programmers and tool developers who need fine-grained measurements and control of the PAPI interface. Unlike the high-level interface, it allows both PAPI preset and native event measurements.

The low-level API features the possibility of getting information about the executable and the hardware, and to set options for multiplexing and overflow handling. Compared with high-level API, the low-level API increases efficiency and functionality.

An Event Set is a user-defined group of hardware events (preset or native) which, all together, provide meaningful information. The users specify the events to be added to the Event Set and attributes such as the counting domain (user or kernel), whether or not the events are to be multiplexed, and whether the Event Set is to be used for overflow or profiling. PAPI manages other Event Set settings such as the low-level hardware registers to use, the most recently read counter values and the Event Set state (running / not running).

Following is a simple code example using the low-level API. It applies the same technique as the high-level example.

```
#include <papi.h>
#include <stdio.h>

#define NUM_FLOPS 10000

main()
{
int retval, EventSet=PAPI_NULL;
long_long values[1];

/* Initialize the PAPI library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT) {
  fprintf(stderr, "PAPI library init error!\n");
  exit(1);
}

/* Create the Event Set */
if (PAPI_create_eventset(&EventSet) != PAPI_OK)
    handle_error(1);

/* Add Total Instructions Executed to our Event Set */
if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK)
    handle_error(1);
```

```
/* Start counting events in the Event Set */
if (PAPI_start(EventSet) != PAPI_OK)
    handle_error(1);

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

/* Read the counting events in the Event Set */
if (PAPI_read(EventSet, values) != PAPI_OK)
    handle_error(1);

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

/* Reset the counting events in the Event Set */
if (PAPI_reset(EventSet) != PAPI_OK)
  handle_error(1);

do_flops(NUM_FLOPS);

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

do_flops(NUM_FLOPS);

/* Stop the counting of events in the Event Set */
if (PAPI_stop(EventSet, values) != PAPI_OK)
    handle_error(1);

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

Possible output:

```
After reading the counters: 440973

After adding the counters: 882256

After stopping the counters: 443913
```

Note that `PAPI_reset` is called to reset the counters, because `PAPI_read` does not reset the counters. This lets the second value (after adding the counters) to be approximately twice as large as the first value (after reading the counters).

For more details, please refer to PAPI man and documentation, which are installed with the product in **/usr/share** directory.

1.9 Profiling Programs – HPCToolkit

HPCToolkit provides a set of profiling tools that help you to improve the performance of the system. These tools perform profiling operations on the executables and display information in a user-friendly way.

The main advantage of HPCToolkit over other profiling tools is that you do not need to include profiling options and to re-compile the executable.

Note:
In this section, the term "executable" refers to a Linux program file, in ELF (Executable and Linking Format) format.

Prerequisites:

- The executable must contain debugging information (if not, there will be no correspondence between counters and code at source line level)
- The executable must be dynamically linked because HPCToolkit overloads the default initialization functions to call PAPI.
- The executable must not use ANSI libstdc++. (The constructor being static with the current libstdc++ at the present time, using HPCToolkit with such an executable produces a SIGSEGV).

1.9.1 HPCToolkit Tools

HPCToolkit includes the following tools:

hpcrun profiles the execution of an executable, by statistically measuring the hardware counters.

hpcprof interprets the profile files produced by **hpcrun**, and associates them with the source code.

bloop analyzes the executables to determine their structure. Its goal is to search for execution loops and to identify the corresponding source code.

hpcview generates high level metrics from raw profiling data and correlates it with logical source code abstractions. By default, it generates an Experiment database (ExperimentXML format) for use with **hpcviewer**.

hpcviewer is a graphical user interface used to view the information gathered by **hpcview** easily and in particular the links between the source code and the performance.

1.9.2 Display Counters

The **hpcrun** tool uses the hardware counters as parameters. To know which counters are available for your configuration, use the **papi_avail** command:

```
papi_avail
```

```
Available events and hardware information.

-------------------------------------------------------------------------
Vendor string and code   : GenuineIntel (1)
Model string and code    : 32 (1)
CPU Revision : 0.000000
CPU Megahertz: 1600.000122
CPU's in this Node : 6
Nodes in this System: 1
Total CPU's  : 6
Number Hardware Counters : 12
Max Multiplex Counters   : 32

-------------------------------------------------------------------------
```

The information displayed below corresponds to fields in the **PAPI_event_info_t** structure.

```
Name          Code        Avail    Deriv Description (Note)
PAPI_TOT_CYC  0x8000003b  Yes      No    Total cycles
PAPI_L1_DCM0  x80000000   Yes      No    Level1 data cache  misses
PAPI_L1_ICM0  x80000001   Yes      No    Level 1  instruction cache misses
PAPI_L2_DCM0  x80000002   Yes      Yes   Level 2 data cache misses
...
PAPI_FSQ_INS  0x80000064  No       No    Floating point square root instructions
PAPI_FNV_INS  0x80000065  No       No    Floating point inverse instructions
PAPI_FP_OPS   0x80000066  Yes      No    Floating point operations

-------------------------------------------------------------------------
```

The following counters are particularly interesting: `PAPI_TOT_CYC` (number of CPU cycles) and `PAPI_FP_OPS (number of floating point operations)`.

To display more details use the **papi_avail -d** command.

1.9.3 Using HPCToolkit

Important:
It is necessary to run all the above tools before the results can be gathered and analyzed.

1.9.3.1 Analyzing the executable code (bloop)

Syntax:

```
bloop executable_name > executable_name.psxml
```

Output:

bloop generates an XML file, whose type is PGM (program), specifying the structure of the program, and which can be used by **hpcview** or **hpcquick**.

1.9.3.2 Retrieving the execution information (hpcrun)

Syntax:

```
hpcrun -e event1[:period1] -e event2[:period2] . . . executable_name
```

–e eventx Specify the counter names.

periodx Retrieve a counter during a specific time period.

Note:
Some counters are not compatible. To resolve this problem, specify a period of time for each counter using the **:period** parameter. When this option is specified **hpcrun** retrieves each counter in sequence, thus avoiding conflicts.

Output:

hpcrun generates a file named `executable_name.eventx-etc.hostname.pid.tid`. This file contains all the counter values in the form of a histogram.

Examples:

```
hpcrun -e IA64_INST_RETIRED -e L3_MISSES -e PAPI_TOT_CYC /bin/ls
```

To retrieve the `IA64_INST_RETIRED` counter for 3000 events, enter:

```
hpcrun -e IA64_INST_RETIRED:3000 -e L3_MISSES -e PAPI_TOT_CYC /bin/ls
```

1.9.3.3 Analyzing the execution (hpcprof)

Syntax:

```
hpcprof [options] executable_name executable_name.event1-
etc.hostname.pid.tid
```

`executable_name.event1-etc.hostname.pid.tid` is the name of the file generated by the **hpcrun** command.

Options:

-e, --everything Show all information

-f, --files Show all files

-r, --funcs Show all functions

-l, --lines Show all lines

–p, --profile Specify that the output is issued in the form of a "profile" file, which can be used by **hpcview**. By default the output is in ASCII.

Example 1:

```
hpcprof executable executable.B.L3_MISSES-etc.hostname.pid.tid -f
```

```
L3_MISSES:32767 - L3 Misses (70 samples)
PAPI_TOT_CYC:32767 - Total cycles (126483 samples)

File Summary:
64.3%  44.7%
<</root/NPB3.2.1/NPB3.2-SER/bin/bt.B>>/root/NPB3.2.1/NPB3.2-SER/BT/exact_rhs.f
0.0%  40.3%
<</root/NPB3.2.1/NPB3.2-SER/bin/bt.B>>/root/NPB3.2.1/NPB3.2-
SER/BT/exact_solution.f
35.7%  12.5%
<</root/NPB3.2.1/NPB3.2-SER/bin/bt.B>>/root/NPB3.2.1/NPB3.2-SER/BT/initialize.f
0.0%   2.2%
<</lib/libgcc_s-3.4.6-20060404.so.1>>../../gcc/config/ia64/lib1funcs.asm
0.0%   0.2% <</root/NPB3.2.1/NPB3.2-SER/bin/bt.B>><unknown>
```

In this example the utilization time is specified for each file because the **-f** option was used. Use the other options to see alternative information.

Example 2:

To obtain the results in the form of a "profile" file, use the **–p** option, as follows:

```
hpcprof executable executable.B.L3_MISSES-etc.hostname.pid.tid -e
-p > profile
```

1.9.3.4 Creating a HPCVIEW configuration file (hpcquick)

The HPCVIEW configuration file is required by the **hpcview** tool. It describes which data should be examined, the type of performance data to be calculated from these results and how they should be displayed. An HPCVIEW configuration file can easily be created using the **hpcquick** tool.

Syntax:

```
hpcquick [options] [ -I dir1 dir2 ... dirN ] [ -S struct1 struct2 ...
structJ ] [ -G group1 group2 ... groupK ] -P prof1 prof2 ... profM
```

-I dir1 dir2 ... Specify the directories related to the source of the executable analyzed.

-S struct1 struct2 ... Specify the PGM output files generated by **bloop** (**.psxml**).

-P prof1 prof2 ... Specify the profile files containing the counter values analyzed by **hpcprof**.

-G group1 group2 ... Specify the group files that can be as well for the main program and/or any or all of the shared libraries used by the program.

-n This option generates the configuration file, which is used by **hpcview**.

Example:

```
hpcquick -I ../BT/ -S resbloop -P bt.B.L3_MISSES-etc.dedea.21173.0x52b5 -n
```

```
Canonicalizing performance data...
hpcprof -p bt bt.B.L3_MISSES-etc.dedea.21173.0x52b5 >
bt.B.L3_MISSES-etc.dedea.21173.0x52b5.hpcquick.pxml
* Collecting metrics from performance data...
* Generating hpcview configuration file...
  Adding source path: '../BT/'
  Adding structure file: 'resbloop'
  Adding metric 'L3_MISSES' from
  'bt.B.L3_MISSES-etc.dedea.21173.0x52b5.hpcquick.pxml'
  Adding metric 'PAPI_TOT_CYC' from
  'bt.B.L3_MISSES-etc.dedea.21173.0x52b5.hpcquick.pxml'
* Sending command to create browsable database to log...
  hpcview -o experiment-db-21368 hpcquick.xml
* Created files: 'hpcquick.xml', 'hpcquick.log'
```

Note:
The HPCVIEW configuration file is editable, in terms of the data you want to analyze and the results you want to display.

1.9.3.5 Collecting the results (hpcview)

hpcview generates high level metrics from raw profiling data and correlates it with logical source code abstractions. By default, it generates an Experiment database file (ExperimentXML format) to be used with hpcviewer.

Syntax:

```
hpcview [options] <Config_File> [<Profile_File1 . . .>]
```

Profile_File1 . . . Name of profile files generated by **hpcprof**.

-x File_Name Name of the XML file (PGM type) generated by **bloop** from the executable's analysis.

Config_File Name of the HPCVIEW configuration file describing which data should be examined, the type of performance data to be calculated from these results and how they should be displayed.

General options:

-v, --verbose [<n>] Verbose mode; generate progress messages to stderr (standard error output) at verbosity level <n>. Default: 1.
Use n=2 to debug path replacement if metric and program structure is not properly matched.

-V, --version Print version information.

-h, --help Print the command's help.

Output options:

-o <db-path>, --db <db-path>, --output <db-path>
Specify Experiment database name <db-path>.
Default: /experiment-db.

--src [yes|no], --source [yes|no]
Whether to copy source code files into the Experiment database. By default, **hpcprof** copies source files with performance metrics and that can be reached by PATH/REPLACE statements, resulting in a self-contained dataset that does not rely on an external source code repository. Note that if copying is prevented, the database is no longer self-contained.

Output formats:

Select different output formats and optionally specify the output filename <fname> (located within the Experiment databse). The output is sparse in the sense that it ignores program areas without profiling information. (Set <fname> to '-' to write to stdout.)

-x [<fname>], --experiment [<fname>]
ExperimentXML format. Default: experiment.xml.
Note: to disable, set <fname> to 'no'.

--csv [<fname>] Comma-separated-value format. Default: experiment.csv. (Flat scope tree; Loop level.) (Useful for downstream external tools.)

--tsv [<fname>] Tab-separated-value format. Default: experiment.tsv. (Flat scope tree; line level.) (Useful for downstream external tools.)

Example:

```
hpcview hpcquick.xml bt.B.L3_MISSES-etc.dedea.21173.0x52b5.hpcquick.pxml
```

1.9.3.6 Viewing the results (hpcviewer)

The **hpcviewer** tool displays the counters values for each code line (Figure below). **hpcviewer** uses the XML file generated by **hpcquick** or **hpcview**.

The following figure shows an example (Figure 1-10).

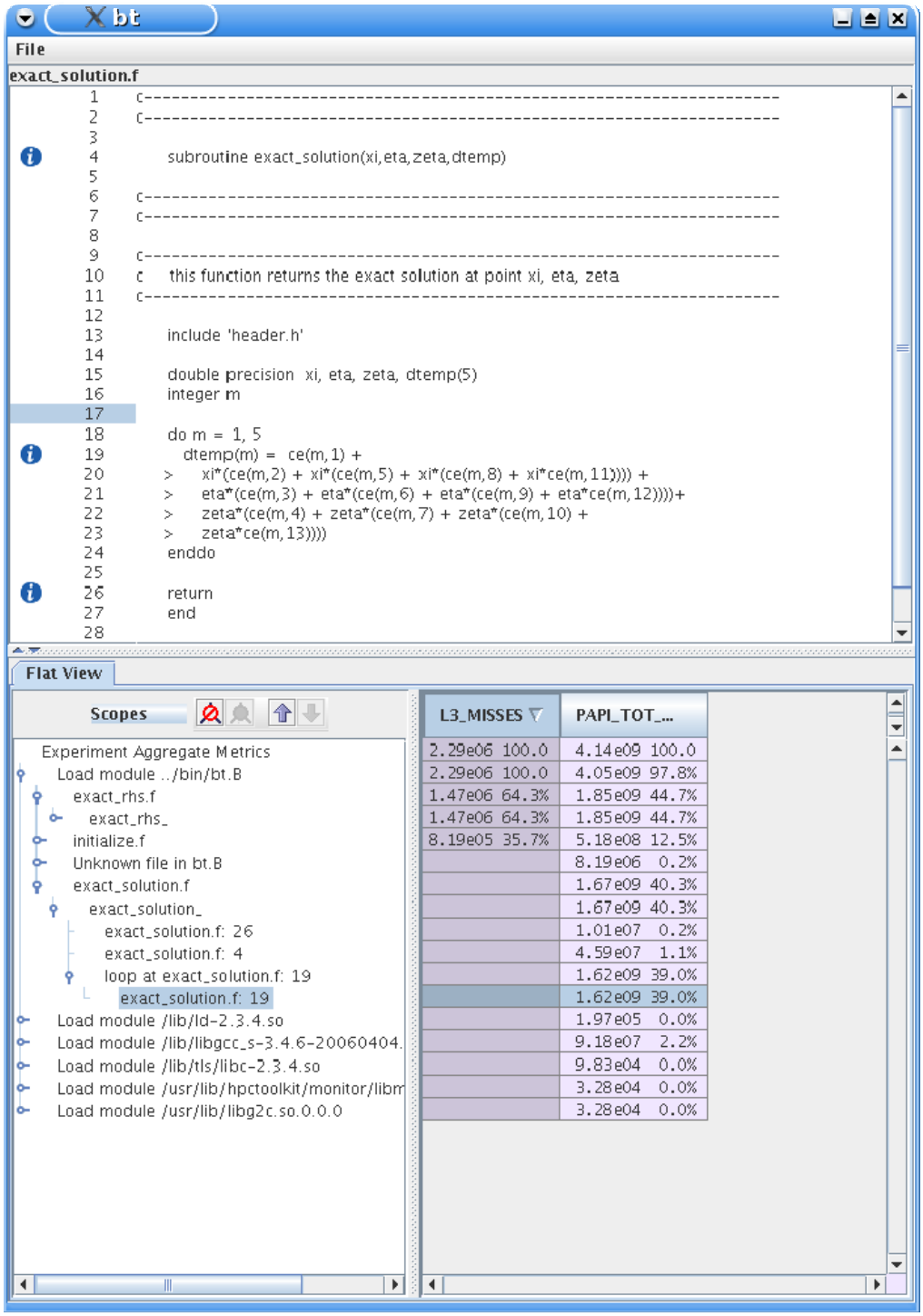

Figure 1-10. View of the counter values, using hpcviewer

1.9.4 More Information

For more information about HPC Toolkit go to:

http://hipersoft.cs.rice.edu/HPCToolkit/documentation.html

1.10 Itanium® 2 Processor Events and Intel® Trace Tools

In order to go further in the optimization of the application program the tools in this section should be used to obtain more precise hardware information. The closer the monitoring is to the hardware level, the easier it is to see what is happening.

Measurements collected at the hardware level can be used to:

- See what really happens in the machine (for CPU, cache, bus etc),
- Identify where performance is lost,
- Identify possible causes of any blocks,
- Model changes for performance if parameters such as frequency and cache are adjusted.

This information is obviously of interest to both hardware designers and software application developers. Using this information the application developer can then attempt to improve loop processing, for instance, and better optimize resources, for example, the use of cache memory.

Itanium® 2 processor monitors around 490 events.

The events are described in the chapter 11 of the *Intel® Itanium® 2 Processor reference Manual for Software Development and Optimization*.

To make the most of these events a good knowledge of the internal mechanisms of the machine is required. It is recommended to get help from an expert.

The most notable events are the following ones:

- CPU events:
  - CPU_CYCLE
  - IA64_INST_RETIRED
  - FP_OPS_RETIRED
  - NOPS_RETIRED.
- Cache misses:
  - L3_MISSES.
- Bus occupation (6.4 GB/s, 200MHz):
  - BUS_ALL_ANY
  - BUS_DATA_CYCLE.
- Blocks of pipelines (mainly due to delay in data transfer):
  - BACK_END_BUBBLE_ALL.
- Translation Lookaside Buffers (TLB) misses. If a memory reference causes a TLB miss then its data dependencies may lead to a pipeline stall and subsequently a serious impact on performance.

1.10.1 Intel® Trace Collector and Analyzer

Intel® Trace Collector is a low-overhead tracing library that performs event-based tracing in applications. The collected trace data can be analyzed for performance hotspots and bottlenecks. Intel® Trace Collector is thread-safe and integrates with C++ multi-threaded processes that use MPI. Binary instrumentation and fail-safe mode features are also included.

Intel Trace Collector is used to:

- Monitor MPI and routine entry/exit points automatically. Application source code may be monitored by using the standard API.
- Trace synchronous actions of parallel applications and calculate statistics for these for user specified time intervals and processes.
- Cache trace data in memory. As a consequence of the scalability options provided the runtime overhead is reduced.
- Monitor application runtime or collect statistics from counters. It can calculate statistics for runtime function calls, messages, and collective operations and store them without event data.
- Capture application statistics displays with trace data.
- Thread-safe trace multi-threaded MPI applications.
- Monitor C++ function entry/exit points through binary instrumentation of executables. This helps to provide a detailed analysis of the code and application runtime.
- Trace other programming models and libraries using extended APIs.
- Record event trace data in a Structured Trace Format (STF). This is scalable

Intel Trace Analyzer is used to:

Monitor the Communication Layer for parallel applications: Intel® Trace Analyzer monitors application activities gathered by Intel® Trace Collector through the use of graphical displays. The level of detail can be adjusted; consequently performance hotspots and bottlenecks can be identified and analyzed.

Monitor Parallelism: Actively view message-passing for parallel applications over a period of time and gauge the efficiency of the synchronicity.

Analyze data at different levels: The trace data can be examined and displayed at different levels of abstraction - cluster, node, and process.

Analyze program subroutines: Execution statistics for subroutines, including call-tree comparisons, over different time periods may be analyzed. It is then possible to make comparisons between different program runs in order to see if overall performance has been improved as a consequence of any changes made to the program.

Identify any communication hotspots.

Compare modifications to the program: Hotspots or bottlenecks in the trace can be looked at in more detail and identified for comparison with other executions to enable program optimization. This can be done globally or on individual process basis.

Intel® Trace Collector is proprietary software and has to be bought directly from Intel.

1.10.2 Intel® VTune™ Performance Analyzer for Linux

Intel® VTune™ Performance Analyzer is used to:

Profile software performance: Intel® VTune™ Performance Analyzer uses interrupt-based sampling to represent the application program's performance with a low overhead. It is also possible to take CPU snapshots in order to identify problems such as cache misses.

Identify critical parts of the application: Executables are monitored and broken down at the binary level so that the function calls can be analyzed. The functions which took the most time to process or were blocked may be identified through the use of call graphs.

Events and processes are sampled over a time period and then may be analyzed at different levels - operating system process, thread, module executable, function/method, individual line of source code, or individual machine/assembly language instructions - to identify specific bottlenecks.

Analyze memory intensive applications. These include applications that run on kernels that support "Hugemem" (over 4 Gigabytes).

Look at instructions at machine level for post mortem purposes. Intel® VTune™ Performance Analyzer provides the means to look at machine instructions. These are annotated and identify any instruction latencies or stalls. Possible changes to the application which may improve performance are highlighted.

The graphical user interface for the Linux Vtunes driver has to be used on a Windows® PC which is connected to the system.

Intel® Performance Analyzer is proprietary software and has to be bought directly from Intel.

Visit the Intel® web site for more details. http://www.intel.com/

Chapter 2. Coding and Compiling Optimization

This chapter looks at some coding tips and compiling options to help improve the performance of your application on the Bull HPC platform. Guidelines are given in order to ensure that the application program runs as efficiently as is possible.

The following topics are described:

- 2.1 *Application Code Optimization*
- 2.2 *Compiler Optimization Options*

2.1 Application Code Optimization

Application code optimization is hotly debated and an enormous amount of material has been written on the subject. Some of the guidelines produced are common sense regarding the use of good programming technique. The parallel processing capability means that more than ever your code must be tidily organized and streamlined. Also, of course, the structure and requirements of each application is different, bringing with it its own constraints and limitations.

Sometimes the simplest change to your application can produce the biggest gains in resource use. At all times a scientific approach must be taken with all optimizations measured and verified against existing values.

This chapter contains some general programming guidelines and pointers to ensure that the compilation is as efficient as is possible.

Throughout are tips and pieces of advice resulting from the experience of Bull's High Performance Computing Benchmarking and Software team.

2.1.1 Software Pipelining

Software Pipelining is the technique of scheduling instructions across several iterations of a loop. Consequently, there is a reduction in pipeline stalls for sequential pipelined machines. Also, it helps to exploit instruction level parallelism on superscalar machines so that iterations are overlaid, which means that an iteration starts before the previous iteration is completed. It is transparent to the application and is particularly useful for integer additions and floating point multiplications.

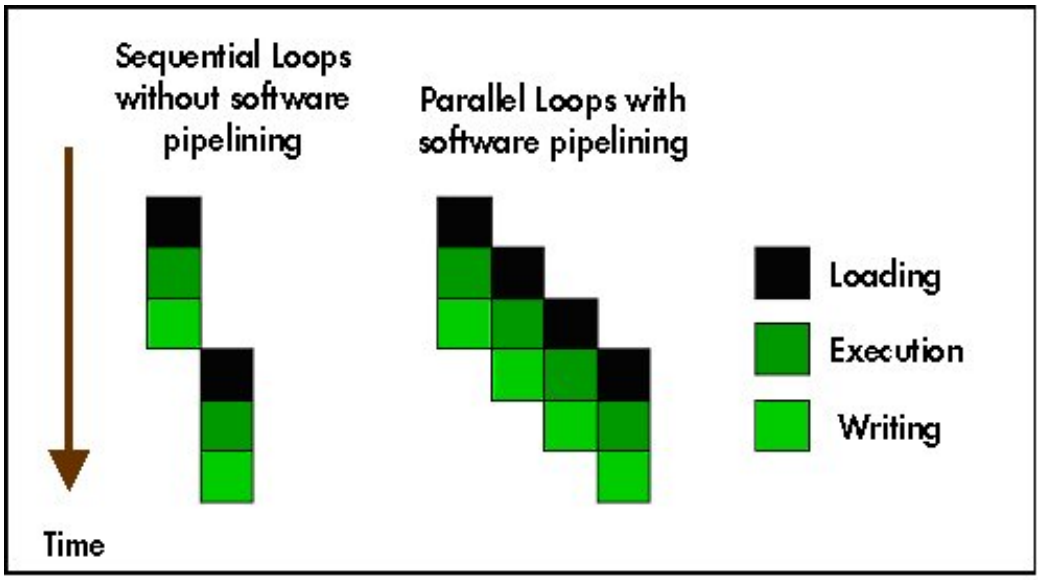

Figure 2-1 Software pipelining

The instruction-level parallelism in a software pipeline is limited by:

- Resource Constraints.
- Dependence Constraints particularly loop carried dependences between iterations, which arise when the same register is used across several iterations or the same memory location (memory aliasing) is used across several iterations.

2.1.2 Alias Usage

Aliasing is when a pointer points to the same memory zone across several iterations. Thus it is possible to increase the optimization level for the compiler as long as the developer can ensure that there are not two pointers using the same memory zone. In this case the FORTRAN and C compiler option **-fno-alias** is used to restrict alias usage.

2.1.3 Improving Loops

Loops are very powerful programming devices which in a few lines can result in a high amount of date processing and optimization. Some, if not all, of the basic loop structures – switching, partitioning, factoring, hoisting, fusion, distribution and unrolling – will be part of most programmers' repertoire. Obviously, these optimizations have to be used carefully, with a good knowledge of the application, to ensure that all data dependencies are respected.

Loops automatically allow for parallelism in terms of program scheduling and structure. They also enable the programmer to identify code parallelizing possibilities which may not have been obvious initially.

Array Loop Optimizations

Some optimizations for arranging arrays in memory are as follows:

- C Arrange as a series of lines

- Fortran Arrange as a series of columns

It is essential that data which is placed within one memory location is streamed smoothly, and the data flow for a particular object which is placed in the same memory location is not broken. Again the following options can be used:

- C Internal loop for columns
- Fortran Internal loop for lines

1. Switching, if possible, within loops is useful to align the access to arrays with their position in memory.

```
      do i = 1, N
         do j = 1, N
            A(i,j) = 1/B(i,j)
         end do
   end do
   do j = 1, N
      do i = 1, N
         A(i,j) = 1/B(i,j)
      end do
   end do
```

2. The partitioning of loops allows their granularity to be adapted to the memory hierarchy. The computation is done by blocs which are not necessarily aligned. This works well when all the loops may be switched.

```
   do i = 1, N
   do j = 1, N
         A(i,j) = 1/B(i,j)
         end do
   end do
```

```
   do jj = 1, N, sj
   do ii = 1, N, si
      do j = jj, jj+sj-1
         do i = ii,ii+si-1
               A(i,j) = 1/B(i,j)
               end do
            end do
         end do
   end do
```

3. Fusion combines loops within in the same cycle, thus eliminating the need for temporary arrays. Distribution makes it possible to build parallel loops.

```
      do i = 1, N
         A(i) = ...
   end do
   do i = 1, N
      B(i) = ... A(i) ...
   end do
```

```
do i = 1, N
             A(i) = ...
             B(i) = ... A(i) ...
end do
```

4. Scalars can be increased to remove any dependences resulting from the memory re-use.

```
   do i = 1, N
   T=f(i)
   A(i) = A(i)+T*T
   end do
```

```
   do  i = 1, N
   T[i]=f(i)
        A(i) = A(i)+T[i]*T[i]
   end do
```

Loop Peeling

Loop peeling is a traditional optimization which is used for loops with a low number of iterations. It acts to explicitly extract the first iterations from the loop in order to avoid having to have them returned to the loop, which may result in a high overhead for a low number of iterations. This approach is particularly appropriate for Intel® Itanium® 2 platforms which use the software pipeline intensively - up to 10 levels of operation.

2.1.4 C++ Programming Hints

The following hints originate from Intel's programming tutorial:

- Use the `const` modifier as much as is possible.
- Use local variables rather than global or static variables e.g.

```
int limit;                    int limit;
int function()                   int function()
{                             {
for (i=0; i<limit...)            int my_limit = limit;
}                                for (i=0; i<my_limit...)
                              }
```

- Use static variables rather than global ones e.g.

```
int flag;                          static int flag;
/* flag used only in this file */  /* flag used only in this file */
```

- Use procedures like warning(), error(), exception(), assert() and err().
- Use inline functionality for functions which are used a lot or are small in size.
- Use for or while loops instead of do while loops.
- Use **int** data types for arrays instead of unsigned **int** data types.
- Let the compiler handle prefetching except in the case when there is a problem. In this case the PREFETCH directive is used.

2.1.5 Memory Tips

- Minimize the use of the pointers.
- Use addresses based on the arrays rather than pointers.

```
int *src = src_array;
int *dst = dest_array;
for (i=0; i<10; i++)          for (i=0; i<10; i++)
{                             {
*dst++ = *src++;              dest_array[i] = src_array[i];
}                             }
```

- Use the **restrict** keyword for better control.

2.1.6 Application code performance impedances

The following points may be counter-productive in terms of application performance:

- Reusing the same code for unrelated computations.
- Unnecessary branching and procedure calls.
- Optimizing by hand, for example, loop unrolling and prefetching.
- Writing functions in assembly code.
- Dead code and empty function calls.
- Using the # pragma pack directive and the unaligned keyword. These can lead to misalignment.

2.1.7 Interprocedural Optimization (IPO)

Application performance for programs which contain a lot of small and frequently used functions can be improved considerably using IPO. IPO reduces the number of branches in code, reduces overhead calls through inlining functions and performs interprocedural memory analysis in order to keep critical date in registers across function boundaries.

Keep the following points in mind:

- Uses static variables and static functions, and avoid assigning function addresses or variable addresses to global variables.

  Unless the compiler can detect the whole program, it has no knowledge about the overall use of global variables, external functions, or static variable and static functions whose addresses are taken and assigned to a global variable or function pointer.
- If IPO does not inline automatically, uses the `inline` keyword in C++, and `_inline` in C.
- Avoid passing pointers into a function as a parameter and then assigning them to a global variable. The code below hinders IPO. x is a global variable and p is a pointer.

```
int *x;
foo()
{
   int y;
   bar(&y);
}
bar (p)
{
   x = p
}
```

2.2 Compiler Optimization Options

One of the most important ways of generating efficient executables is to closely examine the compiler optimization options. A single set of optimization options doesn't exist. You have to find the best set of options according to the characteristics of the source code. In addition, each source file can be compiled using different options. Finally, compiler directives can be inserted into the source code in order help the compiler to optimize the program.

2.2.1 Starting Options

Before using advanced optimization options, it is advisable to generate a reference executable using the default compilation optimization options. Advanced optimization option time differences will be analyzed against this execution time.

The default optimization options for the Intel FORTRAN Compiler are the following:

-72	Sets the number of column in the source code to 72.
-O2	Level 2 optimizations (software pipelining, unrolling, inlining).
-align	Memory aligning.
-nomodule	Compilation with F90 modules located in the current directory.
-tpp2	Itanium2 optimizations.
-IPF_fp_speculationfast	Speculate on floating point operations.
-IPF_fltacc	Enables/disables optimizations that affect floating-point accuracy.
-IPF_fma	Enables/disables the combinations of floating-point multiplies and add/subtract operations into a single operation.
-Zp8	8 bytes alignment.

The default optimization options for the Intel C/C++ Compiler are the following:

-O2	Optimizations of level 2 (software pipelining, unrolling, inlining).
-Ob1	Enables inlining of functions declared with the __inline keyword.
-alias-args	Assume arguments may be aliased.
-falias	Assume aliasing in the program.
-ffnalias	Assume aliasing within functions.
-IPF_fltacc	Enables/disables optimizations that affect floating-point accuracy.

-IPF_fma	Enables/disables the combinations of floating-point multiplies and add/subtract operations into a single operation.

2.2.2 Intel C/C++ and Intel Fortran Optimization Options

Once the reference execution time is collected, more aggressive optimization options can be activated.

The following optimization commands may be activated on both C/C++ and Fortran compilers:

-O3	Level 3 optimizations (**-O2** optimizations plus more aggressive optimizations such as prefetching and loop transformations).
-ip	Enables more interprocedural optimizations for single file compilations.
-ipo	When this option is specified, the compiler performs inline function expansion for calls to functions defined in separate files.
-Qoption,f,-ip_ninl_min_stats=n	Modifies the number of inlining levels (by default this is 15)
-Qoption,f,-ip_ninl_max_total_stats=n	Modifies the number of lines added when inlining (by default n = 2000)
-static	Causes the executable to link all the libraries statically.
-fno-alias	Specifies that aliasing should not be assumed in the program.
-fno-fnalias	Specifies that aliasing should not be assumed within functions, but should be assumed across calls.
-ftz	Enables denormal results to be flushed to zero.
-IPF-fp-relaxed	Enables use of faster but slightly less accurate code sequences for math functions, such as **divide** and **sqrt**.

Loop unrolling is an optimization option whereby instructions called in multiple iterations of a loop are combined so that only a single iteration is necessary. This technique is particularly useful for parallel processing. Performance is improved as result of the reduction in the number of overhead instructions that have to be executed for a loop, which in turn reduces branching and improves cache hit rate. However, this option has to be handled carefully.

Unrolling options are:

-unroll0	Ending of unrolling
-unroll	Activation of unrolling
-unrollM	M is the maximum number of loops to be unrolled

See section 2.2.7 for the options available for the generation of optimization reports. The information in the reports can help to identify where optimizations are possible in the code.

2.2.3 Compiler Options which may Impact Performance

The following compiler options **must be avoided if possible** as they will lead to a loss in performance:

-assume dummy_aliases
This forces the compiler to assume that dummy (formal) arguments to procedures share memory locations with other dummy arguments or with variables shared through use association, host association, or common block use. These program semantics slow performance, so you should specify **-assume dummy_aliases** only for the subprograms called that depend on such aliases. The use of dummy aliases violates the FORTRAN-77 and Fortran 95/90 standards but occurs in some older programs.

-c
If you use **-c** when compiling multiple source files, also specify **-ooutputfile** to compile many source files together into one object file. Separate compilations prevent certain inter-procedural optimizations, used with multiple compiler invocations or with **-c** without the **-ooutputfile** option.

-check bounds

Generates extra code for array bounds checking at run time.

-check overflow
Generates extra code to check integer calculations for arithmetic overflow at run time. Once the program is debugged, omit this option to reduce executable program size and slightly improve run-time performance.

-fpe 0
Using this option slows program execution. It enables certain types of floating-point exception handling, which can be expensive.

-g
Generate extra symbol table information in the object file. Specifying this option also reduces the default level of optimization to **-O0** (no optimization). The **-g** option only slows your program down when no optimization level is specified, in which case **-g** turns on **-O0**, which slows the compilation down. If **-g, -O2** are specified, then the code runs at much the same speed as if **-g** were not specified.

-save
Forces the local variables to retain their values from the last invocation terminated. This may change the output of your program for floating-point values as it forces operations to be carried out in memory rather than in registers, which in turn causes more frequent rounding of your results.

-O0
Turns off optimizations. Can be used during the early stages of program development or when you use the debugger.

-vms
Controls certain VMS-related run-time defaults, including alignment. If you specify the **-vms** option, you may need to also specify the -align records option to obtain optimal run- time performance.

2.2.4 Flags and Environment Variables

-assume buffered_io with **FORT_BUFFERRED=TRUE**

-dryrun Gives non specific information regarding what has happened at the ld level

KMP_STACKSIZE Allows the stack size to be increased. This works with **ulimit**

For example with **ulimit -s 1 024 000** or with **ulimit -S -s unlimited** the following command is used.

```
export KMP_STACKSIZE=250 000
```

2.2.5 Compiler Directives for Loops

The following directives are to be specified before the loops concerned:

#pragma For C and C++ programs

[Cc*!]DIR$ For Fortran programs.

The following pragmas can be used:

[NO]SWP Modifies the compiler's heuristic for software pipelining.

```
!DIR£ SWP
do i = 1, m zResource II = 1
   if (a(i) .eq. 0) then   Recurrence II = 1
            b(i) = a(i) + 1    Minimum II = 1
else              Scheduled II = 1
   b(i) = a(i)/c(i)   Estimated GCS II = 1
   endif          Percent of Resource II needed by arithmetic ops = 100%
enddo             Percent of Resource II needed by memory ops = 100%
                  Percent of Resource II needed by floating point ops = 0%
                  Number of stages in the software pipeline = 1
```

LOOP COUNT(N) Specifies the number of loop iterations for the pragma.

DISTRIBUTE POINT May be placed inside or outside of a loop.

[NO]UNROLL, UNROLL(N) Controls loop unrolling.

[NO]PREFETCH a:b:c Controls 'prefetch' for the variable a.

IVDEP Ignores vectorial dependences.

Example for **IVDEP**: The results generated using the **opt_report** option – see section 2.2.7:

```
 do i = 1, m
   if (a(i) .eq. 0) then      Resource II = 1
   b(i) = a(i) + 1       Recurrence II = 1
else                Minimum II = 1
   b(i) = a(i)/c(i)   Last attempted II = 1
   endif         Estimated GCS II = 1
enddo
```

Modulo scheduling was successful but there was no overlap across iterations therefore the loop was not pipelined.

2.2.6 Fortran Compilers and the Intel® Itanium® 2 Architecture

A feature of the Intel® Itanium® 2 Architecture is the absence of an integer multiplier. This means that the Fortran compiler will create incorrect code when generating addresses.

Each integer multiplication is done through the use of floating point calculations. Conversions between integers and floating point numbers have an overhead, and are as resource intensive as multiplications for floating point numbers.

Floating point operations are now used more and more for address generation and not just for scientific computing.

In order to get round this problem, one solution is to extract the Fortran code and to rewrite it in C checking the address generation through the use of pointer arithmetic.

2.2.7 Options for Compiler Optimization Reports

The following options instruct the compiler to generate an optimization report:

-opt_report Instructs the compiler to generate an optimization report to **stderr.**

-opt-report-file<file> Instructs the compiler to generate an optimization report named ***<file>***.

-opt-report-level{min|med|max} Specifies the level of detail for the optimization report.

-opt-report-phase<phase> Specifies the optimizer phase **<phase>** to generate reports for. <*phase*> can be one of the following:

- **ecg_swp** : Code generator / software pipelining
- **hlo** : high level optimizer
- **ipo** : interprocedural optimizer

-opt-report-help Displays the logical names of optimizer phases available for report generation (using -opt-report-phase).

2.2.8 Analysis of Compiled Files

Detection of multiplication problems for floating point address generation:

- Look at the assembler code for the functions which are using a lot of resources. If you observe that the Assembler code contains a lot of instructions of the type XMA whereas the FORTRAN source for the function in question does not contain any, (or only a few) multiplications, then a problem can be identified.
- This type of problem may be the result of a lack of registers (instead of keeping the address within a register and then increasing it after each access, everything has to be recalculated). When inspecting the optimization report you may find a section similar to the output below:

```
Strength ReductionOptimization Report for: totalisation_()
Phase: Operator Strength Reduction - 2
Counts:
   Strength Reduction: 15
   Not reducing - Update cost too high: 5
   Not Reducing - Live time threshold exceeded: 4
   TOTAL transformations: 24

Info:
   TOTALISATION.f90(126):
   Not reducing - Update cost too high: 0
Not Reducing - Live time threshold exceeded: 30
Not Reducing - Live time threshold exceeded: 30
Not reducing - Update cost too high: 0
Not reducing - Update cost too high: 0
Not reducing - Update cost too high: 0
Not Reducing - Live time threshold exceeded: 30
Not reducing - Update cost too high: 0
Not Reducing - Live time threshold exceeded: 30
```

It is possible to see that the compiler chose to redo the calculations 9 (5+4) times (not necessarily due to addressing and multiplication problems), rather than keeping the results in the registers. This can represent a significant amount of time.

Some points which are worth considering are:

- To look at the hot functions indicated by profiling.
- Precisely measure the execution time for the resource intensive loops by adding debugging calls to the source code (it is should be enough to add **tmp=mysecond()**at the beginning of the loop and its equivalent at the end).
- Look at the optimization reports for the code analyzed above. If the cycle number indicated by the compiler is nowhere near what you observe empirically then there is a problem. Look at the following things:
  - Does the code access memory correctly? Add **prefetch** to check.
  - Are there any XMA packets present which should not be there? Examine the OSR live registers or generate the addresses manually.
  - Does the performance increase if the function is changed to C? Sometimes the code generated by the Fortran compiler is problematical.

2.2.9 Compiling Tips

Consider both the -02 and -03 options

The best compiler options are very much dependent on the nature and structure of the program. The length of the vectors involved can be crucially important. In some circumstances the aggressive –O3 optimizations may be counter-productive and generate inefficient code which does not match the expected performance in terms of time and resource use.

The less sophisticated option -O2 generates more conservative code but may have a lower overhead.

Be careful when using loop unrolling options

The loop unrolling options – see section 2.2.4 can be counter productive in terms of performance. Register usage will be increased due to the need to store more temporary variables and code size will increase following unrolling, which is particularly undesirable for embedded applications.

These costs will have to be weighed up against the benefits achieved in terms in the reduction of the number of loop iterations for the program.

Try the option -O3 -unroll0

If a binary file compiled using the -O2 option performs better than a binary compiled with the -O3 option, it is often worth considering the combination **' -O3 -unroll0'**.

The implementation of unrolling when switching from -O2 to -O3 may prove to be counter-productive – see above. However, some, if not all, of other -O3 optimization routines could be beneficial. This means that, generally speaking (depending on the program), the combination -O3 -unroll0 may be the most effective.

Look at floating-point assist faults.

Floating-point assist faults (FPAF) are a mechanism which makes it possible to treat calculations implementing denormalized numbers (floating numbers with a zero mantissa). If these cannot be handled directly by the processor, then the OS will intervene with specific functions, leading to a potentially high time penalty. To see if the application generates FPAFs, use the command dmesg which shows system messages. The messages are of the type:

```
a.out(27243): floating-point assist fault At IP 4000000000032461,
isr 0000020000000008
```

It should be noted that each line of this type may correspond to a variable number of occurrences. FPAF problems may be avoided as follows:

- By using the -ftz option, which changes denormalized numbers to zero. This is included as a default option with the -O3 option, but not with lower optimization settings.

- If FPAF problems remains after the **-ftz** option has been implemented, then they may be due to the unfruitful branching prediction attempts by the compiler. This leads to useless calculations being carried out on invalid data, including denormalized numbers. One solution may be to prevent branch prediction. This is done using the option **- IPF_fp_speculationoff.**

2.3 Other References

For more guidance on optimizing your code and compiler options refer to Intel's web site and documentation. In particular look at *Intel® Itanium® 2 Processor Reference Manual for Software Development and Optimization* manual and at the articles on *Black Belt Itanium® 2 Processor Performance* and on *Performance Modeling Using Compilers* which are available on the Intel Developer web site.

Chapter 3. Program Execution Optimization

This chapter contains a description of various ways that the execution of the program on the Bull HPC platforms can be made as smooth as possible exploiting all the computing power that is available. For information on the different platforms, application types, launching tools and specific execution optimization options refer to chapter 5 in the Bull HPC BAS4 *User's Guide*.

The following topics are described:

- 3.1 *Using Libnuma, Numactl*
- 3.2 *NUMA/Interleave mode for the NovaScale 3005 Series platform*
- 3.3 *Mprun*
- 3.4 *RMS prun*
- 3.5 *CPUSET*
- 3.6 *pplace*
- 3.7 *Tuning Performance for SLURM clusters*
- 3.8 *Bull Linux Kernel Memory Handling*
- 3.9 *Avoiding Memory Access Stalls*
- 3.10 *Fixing Unaligned Memory* Accesses
- 3.11 *Suspend to Swap Optimization*

3.1 Using Libnuma, Numactl

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.

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 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 latency or bandwidth according to which CPU they are accessed by.

Libnuma also allows the binding of threads to specific nodes. 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 pages will be allocated using the round-robin scheduling algorithm on nodes. When memory cannot be allocated on the current interleave, 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 specified nodes.

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 with round robin allocation on the 4 nodes of a 16 CPU NovaScale server, enter:

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

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

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

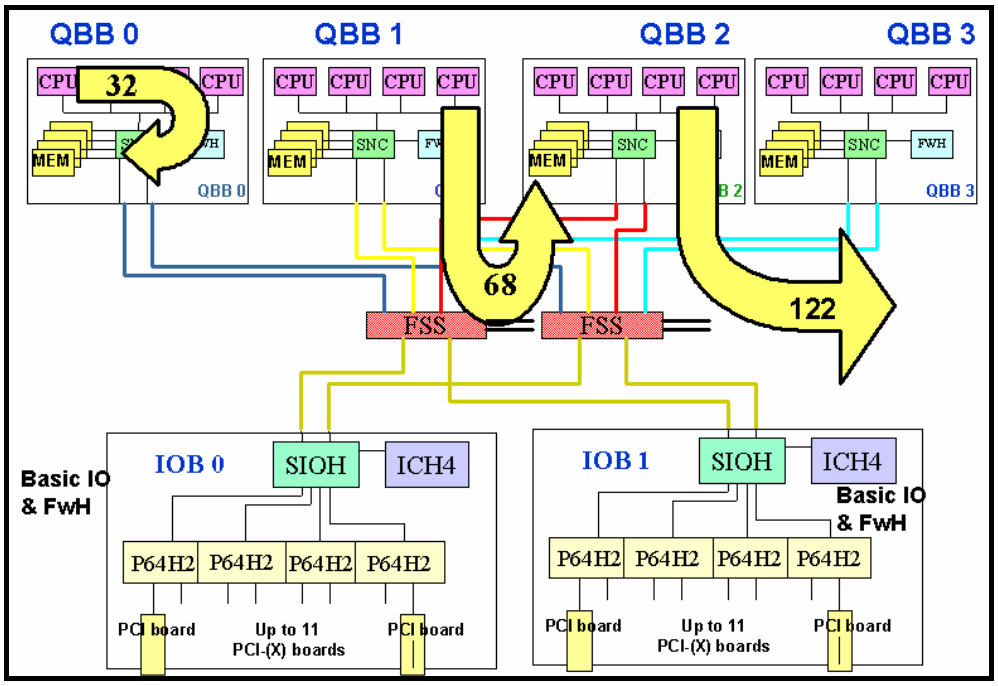

Figure 3-1 NUMA Memory access – NovaScale 5xxx\6xxx Series only

As shown in the above diagram to load data from the memory of a local QBB 32 machine cycles are required. To load from a neighboring QBB 68 machine cycles are necessary and to load from 32 way system which is connected through 2 FAME Scalability Switches (**FSS**) 122 machine cycles are necessary. If the cycle number is multiplied by approximately 4, the time necessary to load the data increases by a much higher ratio. This phenomenon is called the **NUMA_factor** and can be limited by **CPUSET** using the **memset** function, pplace or the other program launching parameters described in this chapter.

3.2 NUMA/Interleave mode for the NovaScale 3005 Series platform

The **NovaScale 3005 Series** are NUMA machines with a low NUMA factor (around 1.3). They include 2 NUMA nodes each equipped with 2 bi-core Montecito Itanium2 sockets.

The memory can be configured in two modes:

1. **NUMA:** t this provides the lowest latency when accessing the memory for the local node. This setting is best for applications that run in parallel and which have good localization when accessing memory (e.g. MPI applications).

2. **INTERLEAVE**: t this interleaves the memory on the 2 nodes, globally at cache line level. This setting configures the machine as a **SMP** machine, and provides better memory throughput and is best suited to applications that share memory, or applications where the memory localization is not certain.

The user should try both modes in order to determine which is best for complex, memory intensive applications.

Note:
The configuration changes described below apply to the hardware settings at the cache level. See the previous section for details regarding NUMA scheduling at the software level for memory pages

3.2.1 Changing the NovaScale 3005 Series Memory Settings using the EFI Boot Manager

The NovaScale 3005 Series NUMA memory setting is set using the Memory Configuration screen. This is accessed as follows t:

1. Switch on the server

2. When the **EFI Boot Manager** menu appears select the **Boot option maintenance menu** and press Enter

3. Select **System Setup** and press Enter

4. Select the **Memory** option and press Enter.

5. Enable/Disable the **NUMA** memory. The NUMA disable setting is equivalent to setting Interleave on.

6. Reboot each machine individually.

See the *NovaScale 3005 Series Installation and User's Guide* for more information on making configuration changes using the EFI Boot Manager

3.2.2 Changing the NovaScale 3005 Series Memory Settings using the syssetup command

The NovaScale 3005 Series memory settings may also be configured under Linux using the **syssetup** command t . This command sets/displays the memory mode to be used at the next boot.

The options used with the command are:

`-n | --numa` : set memory Numa mode

`-i | --interleave` : set memory Interleave mode

If no option is specified the command will display which memory configuration will be used the next time the system boots.

In addition the **syssetup** command can be used to set either CPU multithreaded or CPU monothreaded mode for the next boot. Bull **HPC** clusters should use the default CPU monothreaded mode.

Notes:

- The server should be rebooted after the memory setting has been changed.
- The **syssetup** command is specific to the **NovaScale 3005** series machines.

In a Bull HPC cluster configuration, care should be taken to ensure that the all the machines\nodes in the cluster are configured so that the same memory settings are used throughout.

3.3 Mprun

Mprun is included in **MPI_Bull_Mono** library and is used for launching parallel programs on single node platforms.

Some of the more useful command options are as follows:

-tv	Enable Totalview(tm) 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** blocks until resources become available.
-C	Use cyclic cpusets.
-n <nprocs>	Specifies the number **<nprocs>** of MPI processes to start.
-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.
-o <mode>	Specifies how standard output is redirected. Valid values for mode and their meanings are described below.
-e <mode>	Specifies how standard error is redirected. Valid values for mode and their meanings are described below.

Optimization Tips for using mprun

- The **-o overcommit** option is used when there are not enough CPUs, however, there is a risk that performance may be decreased. The potential impact may be minimized using the option **-Y <type>** which specifies the MPI busy wait strategy. There are 3 possible values, these are **s** for **sched_yield()**, **l** for **select()** and **n** for **none**. The value to be used depends on the application and they should all be tried to see which one minimizes the compute time the most.

  Example for a machine with 4 CPUs, an error message is returned if the –O option is not used:

```
[user@host] basic > mprun -n 5 -O
```

  Overcommit example with the **n** MPI busy wait strategy. All 3 possible values should be tried.

```
[user@host] basic > time mprun -n 5 -O -Y n
```

- When using the **out –o** and **err –e** options be careful not to use too many resources writing logs unless this is really necessary. As an alternative it is possible to write all the output into a single log file or to better still have 1 log file per process to minimize conflicts.

  Example command which redirects all the output into a single file:

```
[user@host] basic > mprun -n 4 -o test.log hostname
```

  Example command which redirects the output on the basis of one file per process:

```
[user@host] test > mprun -n 4 -o test.log.% hostname
```

- In situations when the master process is the one dispatching the data it is better to use the **–M <rank>** option to ensure that the input is properly dispatched to the right process. This helps to make the program more coherent and user friendly.
- The **–C** cyclic option is used to optimize the spread of memory usage across a cpuset within a QBB. This is useful for a program which requires a lot of memory.

3.4 RMS prun

The Quadrics Resource Management System (**RMS**) includes the **prun** command to run jobs in a HPC cluster. For more information on the **prun** command and using **RMS**, see chapter 6 in the Bull HPC BAS4 *User's Guide* (86 A2 29ER).

3.4.1 priority-rms and priority-job Attributes

2 new configurable attributes have been added to the **RMS** database in the **Quadrics** CDROM version 4.7 which may have a direct impact on application performance:

- **priority-rms** used to set the priority of the RMS daemons
- **priority-job** used to set the priority of the applications started by the daemons

These attributes are used to replace the priority values previously set by the **rmsmhd** daemon (included within the service directory **/etc/init.d/rms**) for sub-processes and jobs launched by the **prun** command.

The priority values for the **priority-rms** and **priority-job** attributes are set as follows:

```
rcontrol set attribute priority-rms val <-20 to +19>
rcontrol set attribute priority-job val <-20 to +19>
```

Note1:

RMS must be restarted in order that priority value changes can take effect.

Note2:

For the **priority-rms** and **priority-job** attributes negative values have the highest priority using the syntax of the Unix nice command. The priority scale ranges from -20 for the highest priority applied to the daemons or applications whilst +19 is used for the lowest priority

If the **priority-rms** and **priority-job** attributes do not exist then the priorities will default to 0.

The higher the priority given to a daemon or application then the greater access the daemon or application is given to the system resources and execution time_slices.

3.4.1.1 Daemon priority definition and performance impact

The **priority-job** and **priority-rms** attributes must be used carefully as performance problems may result if the daemon priority conflicts with other parts of a system under a heavy load.

If the **priority-job** attribute is set too high (i.e. with a low negative nice value) for the application then the RMS daemons, which are necessary for it to run, may not run smoothly leading to a loss in performance for the application.

If the **priority-job** attribute is set too low (i.e. with a high positive value) the performance will be impacted because the job is not getting enough resources.

If the **priority-rms** attribute has a higher priority than that for the Batch Manager, e.g. **LSF** or **TORQUE,** then problems can be expected in terms of communication and job scheduling. The Batch Manager should have a priority which is higher (using a lower **nice** value) than or equal to that for **RMS.**

Similarly, if RMS is running with a priority which is higher than the file system kernel threads, e.g. **Lustre**, may not have enough resources to write the data to the disk, causing thread scheduling starvation.

Note:

For optimal performance it is recommended that both **priority-rms** and **priority-job** attributes are set to 0.

3.4.2 RMS and the cpuset –support –enabled attribute

An **MPI** program runs as fast as its slowest process, therefore any problems with the process placement may result in severe performance penalties in the form of cache misses, etc. Each process must be placed correctly according to its submission requirements, which will assign **n** processors to each process.

The **RMS** attribute **cpuset –support – enabled** will ensure that processes remain bound to the CPUs that they were assigned to. For earlier versions of **RMS** (3.1.6 to 3.1.11) the **cpuset –support –enabled** is not set by default and has to be created and set manually. **RMS** versions after 3.1.11 will set this attribute automatically.

If this **RMS** attribute is not set then a process will be able to see all the processors, including those which have been allocated to other processes. The process may then migrate, leading to overloaded processors and performance problems.

Example 1

The command,

```
prun -p test -n 3 sh -c 'echo "process rank $RMS_RANK sees"; grep
processor /proc/cpuinfo; echo'
```

produces the following results:

```
process rank 0 sees
processor  : 0
processor  : 1
processor  : 2

process rank 1 sees
processor  : 0
processor  : 1
processor  : 2
```

```
process rank 2 sees
processor  : 0
processor  : 1
processor  : 2
```

Here each process sees all the CPUs and thus may use them.

Example 2

Run the command, below, to create and set the **cpuset – support –enabled** attribute.

```
rcontrol create attribute cpuset-support-enabled val 1
```

The command,

```
prun -p test -n 3 sh -c 'echo "process rank $RMS_RANK
sees"; grep processor /proc/cpuinfo; echo'
```

will then produce the following result:

```
process rank 0 sees
processor  : 0

process rank 1 sees
processor  : 0

process rank 2 sees
processor  : 0
```

This indicates that the processes can only see the processors which have been allocated to them, and are using their CPU allocation correctly. The processors for each process are numbered upwards from 0.

3.5 CPUSET

CPUSETs are lightweight objects in the linux kernel that enable users to partition their multiprocessor machine by creating execution areas. A virtualization layer has been added so it becomes possible to split a machine in terms of CPUs.

The main motivation of this patch is to give the linux kernel full administration capabilities concerning CPUs. CPUSETs are rigidly defined, and a process running inside this predefined area won't be able to run on other parts of the system.

This is useful for:

- HPC applications, especially those running on NUMA machines.
- Creating sets of CPUs on a system, and binding applications to them.
- Providing a way of creating sets of CPUs inside a set of CPUs so that a system administrator can partition a system among users, and users can further partition their partition among their applications.
- The same applies to memory; the memory used by a CPUSET can be restricted to particular nodes of a NUMA system.

These features have been implemented as a kernel patch for Linux 2.6 and as a suite of user tools.

Note:
The creation and management of CPUSETs is done via the **pexec** and **pcreate** tools included with **ptools** on the **Bull BAS4 HPC CD**. For more details, see them man pages for these tools, and refer to Chapter 6 in the HPC BAS4 User's Guide for more information.
If for some reason, it is decided not to use **ptools,** then CPUSETs can be manipulated using the CPUSET virtual filesystem – see Appendix C in this manual for more details.

3.5.1 Typical Usage of CPUSETS

- CPU-bound applications: Many applications (as it is often the case for HPC apps) used to have a "one process on one processor" policy using **sched_setaffinity()** to define this, but what if we have to run several such apps at the same time? One can do this by creating a CPUSET for each app.
- Critical applications: processors inside strict areas may not be used by other areas. Thus, a critical application may be run inside an area with the knowledge that other processes will not use it CPUs. This means that other applications will not be able to lower its reactivity. This can be done by creating a CPUSET for the critical application, and another for all the other tasks.

3.5.2 BULL CPUSETS

CPUSETS are now integrated in the **-mm** series of Linux kernels.
However these kernels lack some interesting CPUSET features that can be found in the Bull BAS suite:

- **migration**: Change on the fly the execution area of a whole set of processes (to give more resources to a critical application, for example). When you change the CPU list of a CPUSETSs, all processes that belong to this cpuset will be migrated, if necessary, to stay inside the CPU list.
- **virtualization**: translate the masks of CPUs given to **sched_setaffinity()** so they stay inside the set of CPUs. With this mechanism, processors are virtualized, for the use of **sched_setaffinity()** and /proc information. Thus, any former application using this syscall to bind processes to processors will work with virtual CPUs without any change. A new file is added in each cpuset, in the cpuset filesystem, to choose whether a cpuset is virtualized, or not.

3.6 pplace

pplace is a tool included within **ptools** which offers finer control over the binding of threads and processes of an application to individual CPUs than CPUSET.

Depending on the type of application it is possible to gain 10% improvement in performance using this tool.

It may be used when using **OPENMP** for Benchmarking. **OpenMP** is an industry-standard parallel programming model which implements a fork-join model of parallel execution.

With **OPENMP** the source thread or process is split into several parallel threads or processes. These include threads used for calculating and a monitor thread which controls the other threads. You have to be careful to only bind the calculation threads to the CPUs using **pplace** and not the monitoring thread.

SYNOPSIS

pplace -np <nb_cpus> -p <policy> [--name <process_name>] <command>

pplace will create a **cpuset**, enable the process placement policy inside this **cpuset**, and run the <command> inside this **cpuset**.

OPTIONS

-np <nb_cpus>
Specify how many CPUs the application will use. A new CPUSET, with this number of CPUs will be created.

-p <policy>
Specify the placement policy-this policy is actually a comma-separated list of per-task policies. These policies can be:

ignore this task	(nothing)
bind task on next cpu	+
bind task on specific cpu	cpu number

The last policy becomes the default policy for all the tasks which follow. For instance:

-p 0,+ will place the first task on cpu0, the second task on cpu1, the third on cpu2, and so on.

-p 0,,,+ will place the first task on cpu0, ignore the second and third tasks, place the fourth task on cpu1, the fifth on cpu2, and so on.

-p 1, will place the first task on cpu1, and will ignore all other tasks.

--name <process_name> will only consider processes with name <process_name> for the placement. Note: only the 15 first characters of the name are taken into account.

-d debug. When the command terminates, **pplace** will print detailed information about how the process placement occurred. This can help you to choose your policy.

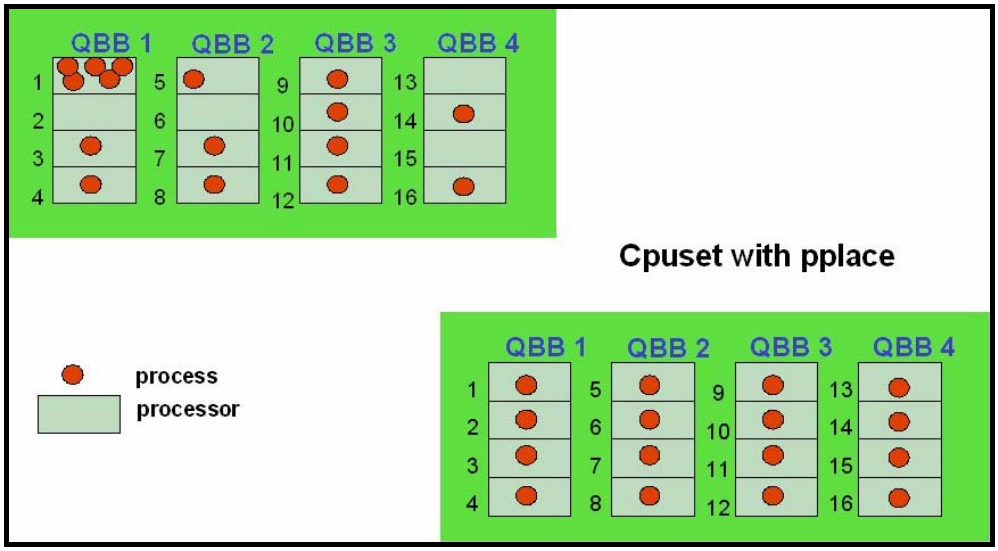

Figure 3-2 Diagram to show process/thread binding with pplace (NovaScale 5xxx\6xxx Series only)

For the application developer individual calls to CPUs can be made in the source code using the command **Sched_setaffinity** which operates in the same way as **pplace**. The advantage which **pplace** offers is that this fixing of processes and threads can be made on the binaries without modifying the source code.

When the compiler uses **OPENMP** pragmas to generate a multithreaded application it uses runtime libraries from Intel and it is not possible to add individual calls in the manner of the **Sched_setaffinity** command. In this instance it may be advantageous to use **pplace** to control the CPU allocation.

3.7 Tuning Performance for SLURM clusters

3.7.1 Configuring CPUs as a Consumable Resource in SLURM

SLURM, using the default node allocation plug-in, allocates nodes to jobs in exclusive mode, which means that even when all the resources within a node are not utilized by a given job, another job will not have access to these resources.

Nodes possess resources such as processors, memory, swap, local disk, etc. and jobs consume these resources. The **SLURM** exclusive use default policy may result in inefficient utilization of the cluster and of node resources.

SLURM provides a plug-in which supports CPUs being used as a consumable resource . Information on how to use this plug-in is described below.

3.7.1.1 The CPU Consumable Resource Node Allocation Plug-in

The CPU Consumable resource node allocation plug-in has the following characteristics:

- The consumable resource plug-in can be enabled by defining a **SelectType** in the **slurm.conf** file (e.g. *SelectType=select/cons_res*) .
- The **select/cons_res** plug-in is enabled or disabled cluster-wide.
- Partitions labeled as SHARED=Yes and SHARED=FORCE do not make sense when connected with the consumable resource support. Consumable resources support only makes sense for SHARED=No. Within the SLURM code, SHARED is set to No if the **select/cons_res** plug-in is enabled. In cases where the **select/cons_res** plug-in is not enabled, the normal SLURM behaviors are not disrupted. The only change users will see when using the **select/cons_res** plug-in, are that jobs can be co-scheduled on nodes when permitted by CPU resources. The remaining SLURM behavior, such as SRUN and switches, etc., is not affected by this plug-in. From a user's perspective, SLURM basically works the same way as when using the default node selection scheme.
- The – ***--exclusive*** switch allows users to reserve/use nodes in exclusive mode even when consumable resource is enabled. Refer to the **srun** man page for more details.
- Using the ***--overcommit*** or **-O** switch in the **select/cons_res** environment is only allowed when a user requests dedicated nodes using the *--exclusive* switch. Overcommiting CPUs in a non-dedicated environment would impact jobs that are co-located on the nodes, which is not a desirable feature.
- SLURM's default **select/linear** plug-in uses a best-fit algorithm based on number of consecutive nodes. The same node allocation approach was chosen for consistency.

Example of Node Allocation using the Consumable Resource Plug-in

The following example illustrates the different ways four jobs are allocated across a cluster using (1) SLURM's default allocation (exclusive mode) and (2) a processor as consumable resource approach.

It is important to understand that the example listed below is a contrived example and is only given here to illustrate the use of CPUs as a consumable resource. Jobs 2 and 3 call for the node count to equal the processor count. This would typically be done because one task per node requires all of the memory, disk space, etc. The bottleneck would not be processor count.

Trying to execute more than one job per node will almost certainly severely impact parallel job's performance. The biggest beneficiary of CPUs as consumable resources will be serial jobs or jobs with modest parallelism, which can effectively share resources. On a lot of systems with larger processor counts, jobs typically run one fewer tasks than the total of processors to minimize interference by the kernel and daemons.

The example cluster is composed of 4 nodes (10 CPUs in total):

- linux01 (with 2 processors),
- linux02 (with 2 processors),
- linux03 (with 2 processors), and
- linux04 (with 4 processors).

The four jobs are in the following:

- [2] srun -n 4 -N 4 sleep 120 &
- [3] srun -n 3 -N 3 sleep 120 &
- [4] srun -n 1 sleep 120 &
- [5] srun -n 3 sleep 120 &

The user launches them in the same order as listed above.

3.7.1.2 Using SLURM's Default Node Allocation (Non-shared Mode)

The four jobs have been launched and three of the jobs are now pending, waiting for resources to be allocated to them. Only Job 2 is running, and it uses one CPU on all four nodes. This means that linux01 to linux03 each have one idle CPU and linux04 has three idle CPUs.

```
# squeue
  JOBID PARTITION     NAME     USER  ST     TIME  NODES NODELIST(REASON)
      3       lsf    sleep     root  PD     0:00      3 (Resources)
      4       lsf    sleep     root  PD     0:00      1 (Resources)
      5       lsf    sleep     root  PD     0:00      1 (Resources)
      2       lsf    sleep     root   R     0:14      4 xc14n[13-16]
```

Once Job 2 is finished, Job 3 is scheduled and runs on linux01, linux02, and linux03. Job 3 is only using one CPU on each of the 3 nodes. Job 4 can be allocated onto the remaining idle node (linux04) so Job 3 and Job 4 can run concurrently on the cluster.

Job 5 must wait for idle nodes to be able to run.

```
# squeue
  JOBID PARTITION     NAME     USER  ST     TIME  NODES NODELIST(REASON)
      5       lsf    sleep     root  PD     0:00      1 (Resources)
      3       lsf    sleep     root   R     0:11      3 xc14n[13-15]
      4       lsf    sleep     root   R     0:11      1 xc14n16
```

Once Job 3 finishes, Job 5 is allocated resources and can run.

The advantage of the exclusive mode scheduling policy is that the job will get all the resources of the assigned nodes for optimal parallel performance. The drawback is that jobs can tie up a large amount of resources which it does not use and which cannot be shared with other jobs.

3.7.1.3 Using a Processor Consumable Resource Approach

The output of **SQUEUE** shows that three out of the four jobs are allocated and running. This is an increase of two running jobs over the default SLURM approach.

Job 2 is running on nodes linux01 to linux04. Job 2's allocation is the same as for the SLURM default allocation, which is that it uses one CPU on each of the four nodes. Once Job 2 is scheduled and running, nodes linux01, linux02 and linux03 still have one idle CPU each, and node linux04 has three idle CPUs. The main difference between this approach and the exclusive mode approach described above is that idle CPUs within a node can now be assigned to other jobs.

It is important to note that *assigned* does not mean *oversubscription*. The consumable resource approach tracks how much of each available resource (in our case CPUs) must be dedicated to a given job. This ensures per-node oversubscription of resources (CPUs) is prevented.

Once Job 2 is running, Job 3 is scheduled on node linux01, linux02, and Linux03 (using one CPU on each of the nodes) and Job 4 is scheduled on one of the remaining idle CPUs on Linux04.

Job 2, Job 3, and Job 4 are now running concurrently on the cluster.

```
# squeue
  JOBID PARTITION     NAME     USER  ST     TIME  NODES NODELIST(REASON)
      5       lsf    sleep     root  PD     0:00      1 (Resources)
      2       lsf    sleep     root   R     0:13      4 linux[01-04]
      3       lsf    sleep     root   R     0:09      3 linux[01-03]
      4       lsf    sleep     root   R     0:05      1 linux04

# sinfo -lNe
NODELIST  NODES PARTITION   STATE   CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
linux[01-03]  3     lsf*  allocated    2   2981        1      1   (null) none
linux04       1     lsf*  allocated    4   3813        1      1   (null) none
```

Once Job 2 finishes, Job 5, which was pending, is allocated available resources and is then running as illustrated below:

```
# squeue
  JOBID PARTITION     NAME     USER  ST     TIME  NODES NODELIST(REASON)
      3       lsf    sleep     root   R     1:58      3 linux[01-03]
      4       lsf    sleep     root   R     1:54      1 linux04
      5       lsf    sleep     root   R     0:02      3 linux[01-03]
# sinfo -lNe
NODELIST  NODES PARTITION   STATE   CPUS MEMORY TMP_DISK WEIGHT FEATURES REASON
linux[01-03]  3     lsf*  allocated    2   2981        1      1   (null) none
linux04       1     lsf*       idle    4   3813        1      1   (null) none
```

Job 3, Job 4, and Job 5 are now running concurrently on the cluster.

```
# squeue
  JOBID PARTITION     NAME     USER  ST     TIME  NODES NODELIST(REASON)
      5       lsf    sleep     root   R     1:52      3 linux[01-03]
```

Job 3 and Job 4 have finished and Job 5 is still running on nodes linux[01-03].

The advantage of the consumable resource scheduling policy is that the job throughput can increase dramatically. The overall job throughput/productivity of the cluster increases, thus reducing the amount of time users must wait for their job to complete, as well as increasing the overall efficiency of the use of the cluster. The drawback is that users do not have the entire node dedicated to their job, since they must share nodes with other jobs if they do not use all of the resources on the nodes.

The **SRUN** ***--exclusive*** switch allows users to specify that they would like their allocated nodes to run in exclusive mode. For more information, refer to **man srun**. This is beneficial for users that have **MPI/threaded/openMP** programs which will take advantage of all the CPUs within a node, but only need one MPI process per node.

3.7.2 SLURM and Large Clusters

This section contains **SLURM** administrator information specifically for clusters containing 1,024 nodes or more. Virtually all **SLURM** components have been validated (through emulation) for clusters containing up to 16,384 compute nodes. Obtaining good performance at this scale requires some tuning and this section provides some basic information with which to get started.

3.7.2.1 Node Selection Plug-in (SelectType)

While allocating individual processors within a node is great for smaller clusters, the overhead of keeping track of the individual processors and memory within each node adds significant overhead. For best scalability, it is recommended that the consumable resource plug-in (*select/cons_res*) not be used.

3.7.2.2 Job Accounting Plug-in (JobAcctType)

Job accounting relies on the **slurmstepd** daemon to periodically sample data on each compute node. This data collection will take compute cycles away from the application, inducing what is known as *system noise*. For large parallel applications, this system noise can detract from application scalability.

For optimal application performance, it is best to disable job accounting **jobacct/none**. Consider use of job completion records **JobCompType** for accounting purposes, as this entails far less overhead.

If job accounting is required, configure the sampling interval to a relatively large size (e.g. *JobAcctFrequency=300*). Some experimentation may also be required to deal with collisions on data transmission.

3.7.2.3 Node Configuration

While **SLURM** can track the amount of memory and disk space actually found on each compute node and use it for scheduling purposes, this entails extra overhead. Optimize performance by specifying the expected configuration using the available parameters (**RealMemory, Procs**, and **TmpDisk)**. If the node is found to contain fewer resources than configured, it will be marked DOWN and not be used. Additionally, set the **FastSchedule** parameter.

While **SLURM** can easily handle a heterogeneous cluster, configuring the nodes using the minimal number of lines in **slurm.conf** will make administration easier and result in better performance.

3.7.2.4 Timers

The configuration parameter **SlurmdTimeout** determines the interval at which **slurmctld** routinely communicates with **slurmd**. Communications occur at half the **SlurmdTimeout** value. The intent of this is to determine when a compute node fails and thus should not be allocated work. Longer intervals decrease system noise on Compute nodes (these requests are synchronized across the cluster, but there will be some impact on applications). For extremely large clusters, **SlurmdTimeoutl** values of 120 seconds or more are reasonable.

3.8 Bull Linux Kernel Memory Handling

By default, the Bull kernel is tuned to provide optimal performance in the HPC environment. Normally, there is nothing to modify in the kernel configuration.

Bull changed the size of TCP Hash entries within the kernel parameters to reduce them to 1Mb, because HPC machines are not used in the same ways as large servers. This is done through the following boot option: "**thash_entries=104875**".

Swap

Bull changed the way the swap works so that the kernel start swapping when there is small amount of memory left as opposed to starting swapping when there is no memory left as is the case with standard Linux behavior on work stations in the office where swapping is not an issue. This is because HPC platforms work with the memory that is available and try to avoid swapping if possible.

overcommit_memory

The virtual memory subsystem may be tuned to avoid problems with memory through the use of **overcommit_memory.** This is a value which sets the general kernel policy towards memory allocations and can be used to allocate more memory than is available on the system. This is a real issue in HPC computing because a randomly chosen application may be killed when the memory is exhausted.

The Linux kernel supports three **overcommit_memory** values:

- 0 – Heuristic for overcommit handling. Obvious overcommits of address space are refused. Used for a typical system. It ensures a seriously wild allocation fails with an error code returned. **root** is allowed to allocate slightly more memory in this mode.
- 1 – Always overcommit. The kernel allows all memory allocations, regardless of the current memory allocation state Appropriate for some scientific applications.
- 2 – Don't overcommit a strict limit. The total address space commit for the system is not permitted to exceed swap + a configurable percentage of physical RAM – set by the **overcommit_ratio value**. Depending on the percentage you use, in most situations this means a process will not be killed while accessing pages but will receive errors on memory allocation as appropriate. This is useful when large amounts of memory are allocated to forestall worst case scenarios and normally the memory is not used.

If your applications use all of the memory that they are allocated then **overcommit_memory** can lead to short performance gains but will be followed by long latencies as your applications are swapped out to disk frequently in the competition for oversubscribed RAM.

Bull linux kernel overcommit values

For the BAS4 HPC software suite Bull has set the **vercommit** policy to 2 (**vm.overcommit_memory=2**), and the overcommit percentage to 90, (**vm.overcommit_ratio= 90**).

These settings can be changed using **sysctl** or a command like:

```
# echo 2 > /proc/sys/vm/overcommit_memory
```

For more information see the kernel documentation directory.

3.9 Avoiding Memory Access Stalls

For an application to be truly optimized it is must run as fast as is possible on the system and at the same time avoid any possible memory access stalls whilst a data process is fetched from memory.

Instructions for Itanium 2 systems which are blocked because of the unavailability of instructions required from an anterior operation can lead to a chain reaction which results in the delay being compounded.

Compilers will automatically generate code optimized to exploit the parallel processing capability possible with groups of instructions sequenced correctly. In many cases the easiest thing to do is leave the compiler to handle all the parallel optimizations.

Memory access stalls are caused when the data to be accessed is not loaded beforehand into the cache. This typically occurs when the memory access is random and not sequential. Two data structures which often use random-like data access are linked lists and hash table arrays.

Linked lists

Create the nodes in pre-allocated blocks of memory rather than taking them from the heap each time. This increases data locality which means that there is a good chance that the data node is already in memory as part of a data block which has already loaded.

Hash tables

In order to prevent data access collisions when re-hashing new slots are allocated at random, with the result that the new slots are unlikely to be in the cache, which in turn leads to a memory access stall.

These stalls can be prevented as follows:

- Ensure data can be accessed sequentially. Fortran arrays should be accessed in column-major order whilst C arrays should be accessed in row-major order.
- If the data is such that it cannot be accessed sequentially then preload it using the **_lfetch** command. However, care has to be taken to ensure that when the data is preloaded it does not displace data which is in the process of being used. Otherwise, the displaced data will have to be reloaded constantly and thus impact performance.
- Do not mix floating point data with other data types when structuring data sequentially. Floating point numbers are serviced using the L2 cache whilst other items use the L1 cache. The resulting cache conflicts from mixed data type groups may seriously impact performance.

Design your code to avoid branching miss-predictions so that similar data takes similar paths. See the *Introduction to Microarchitectural Optimization for Itanium® 2 Processors Reference Manual* from Intel®.

3.10 Fixing Unaligned Memory Accesses

The ia64 architecture specifies that memory accesses are restricted by the size of the accessed data.

Some user code may generate unaligned accesses which in turn produces hardware exceptions at the processor level. These are managed by the **Linux** kernel unaligned access handler. Serious performance issues may result in a high frequency of unaligned accesses. Moreover, unaligned accesses may lead to program inconsistencies for the parts of the code which share and refer to the non-aligned data.

The unaligned accesses are usually recorded in the system error log in the form of messages similar to those below:

unaligned(<*pid*>***): unaligned access to*** <*unaligned data address*>***,***
ip=<*program address*>

The recommendation is to fix the program code to avoid these accesses.

The **Bull prctl** command modifies the OS behaviour when unaligned accesses are detected. The options below are particularly useful.

```
prctl --unaligned=signal
```

This generates a signal (SIGBUS) which makes it easier to debug the source code.

```
prctl --alignement-check=off
```

This allows unaligned accesses to be handled at the platform level, if the platform supports it, (**um.ac=0**). This solution should only be used as a last recourse for those applications where the source code is not available.

See the **prctl** man page for more information.

3.11 Suspend to Swap Optimization

The **Suspend to Swap** feature is used to suspend a job, so that a job with a higher priority can take precedence. This feature is included in the standard BAS distribution. The swap operation is a lot more efficient if the mechanism described in this section is used.

The solution consists in increasing the number of disks dedicated to swap operations, allowing I/O operations to be parallelized. To do this, more than one swap device must be configured in the **fstab** file, and all the swap devices which are configured must be given the same priority. For example:

```
/dev/sda4      none      swap      pri=10

/dev/sdb4      none      swap      pri=10
```

Using this configuration, the system accesses the swap devices in "round-robin" mode and I/O operations are parallelized.

Chapter 4. Message Passing Interface Optimization

This chapter looks at some optimization tips for the Message Passing Interface(MPI).

The following topics are described:

- 4.1 *Introduction*
- 4.2 *General Tips for MPI_Bull Usage*
- 4.3 *MPI-2 One-Sided Operations*
- 4.4 *mpibull2-params*

4.1 Introduction

The number and size of files and data objects which may be handled by the Bull BAS4 HPC supercomputer, with its **EPIC** parallel processing heart, is enormous. With a potential throughput of more than 100 Gigabits per second care must be taken in the organization of the application program exchanges so that all impedances to performance are removed.

Bull has developed a complete solution which helps to enable the full exploitation of NovaScale HPC platforms. To fully utilize the power of a cluster it is necessary to ensure that the application programs are executed in parallel and to take into account environmental factors including the part played by distributed and shared memory.

Bull's **BAS** (Bull Advanced Server) environment is dedicated to parallel programming and includes:

- **MPI** (Message Passing Interface) libraries **MPI_Bull** and **MPICH_Ethernet** – for more details on these libraries see chapter 2 in Bull HPC BAS4 *User's Guide* (86 A2 29ER).
- **PVM** (Parallel Virtual Machine – similar to MPI).
- **OpenMP** - an API dedicated to multiprocessing environments which use shared memory.
- Mathematical libraries for example **MKL.**
- Performance monitoring tools including **pfmon**, **grprof**, **PAPI.**
- The MPI profiling tool **profilecomm** is supplied as part of the **MPI_Bull** library and is used to identify hotspots or bottlenecks within the message passing for the application – see chapter 1 for more information on the data which **profilecomm** provides.
- Debugging/compilation tools.

4.1.1 MDM Optimization Tools

The Bull **MPI Data Mover Module (MDM)** which is incorporated in **MPI_Bull** library versions 1.6.3 and 2.0.9-a includes a trace tool, a profiling tool and a KDB module which may all be used for **MPI** profiling and optimization purposes.

The **trace tool** logs the most recent events for each processor. This tool which has the advantage of not influencing the behavior of the application helps to solve problems of concurrent access to data and of synchronization of processes.

The **KDB module** allows access to traces made when a crash occurs and thus enables more efficient error detection.

The **profiling tool** may be used to identify the critical parts of the application code. Its implementation is similar to that of the trace tool and involves a low loss of performance.

4.2 General Tips for MPI_Bull Usage

There follows some suggestions and points to be kept in mind when configuring the Message Passing Interface using the **MPI_Bull** libraries:

Wait until the sent buffer exchange is finished before modifying it

It is worth looking at the use of the synchronous and asynchronous exchanges when using **MPI_Bull**.

The developer may prefer to use asynchronous exchanges if he wishes to use the compute nodes for another application at the same time that the data is being exchanged for the first one.

Certain implementations of **MPI**, like **MPICH** use the device **ch_shmem**. This uses shared memory as an exchange zone and will tolerate the modification of the sending buffer before the call **MPI_WAIT** runs.

MPICH copies the buffer sent into a shared memory area when the **send** call is used and then the receiving process will copy this buffer into its own receiver buffer. Therefore two copies of the buffer are made and the buffer sent may be modified just after the **send** call without there being any consequence for the exchange.

MPI_Bull has a zero-copy mechanism provided by the **MDM** module which uses only one copy of the buffer to make the exchange. This module makes it possible for messages and data greater than 32Kb in size to be copied directly into the memory of the distant process.

However, it is necessary to pay close attention to buffer use when doing this. If an asynchronous exchange is initialized, it is absolutely prohibited to modify the buffer sent before the exchange is finished, i.e. the buffer sent should only be modified once the call **MPI_WAIT** is finished – this guarantees that the exchange is finished.

Look at the stack size for memory intensive applications

The definition of static buffers does not pose any problem for **MPI_Bull**. It is advisable, however, to look at the size of the stack for applications using a lot of memory, as it may not be large enough. A characteristic symptom of this type of problem is that the application is blocked, without an error message appearing. In this case, it is possible to modify the size of the stack by using the command:

```
$ ulimit – s unlimited
```

This command allows the stack size for the system to be modified. Care should be taken as the unlimited size extends up to the limit of the size of the hardware stack, which cannot be increased. This command can also be used to ascertain the stack size.

If possible avoid the MPI_Bsend function

The use of the **MPI_Bsend** function is possible with **MPI_Bull**, however it is not recommended. Although it can seem attractive as it excludes blockages, it uses an additional buffer (part of **MPI_Buffer_Attach**) into which the data to be sent is recopied. This use of an additional copy of the buffer can have a big impact on performance.

It is better to re-examine an algorithm which falls into deadlock and to use either synchronous or asynchronous **send** calls, rather than using a set-up with **MPI_Bsend**.

Use MPI_Sendrecv rather than MPI_Send and MPI_Recv

When implementing a parallel algorithm, it is important to keep in mind the possible simplifications that the **MPI_Bull** offers. The use of successive **sends** and then **recvs** between several processes may be an example of a complicated algorithm which could be simplified, and may even lead to errors of implementation and execution. In this case, you should verify that the send and receive calls are coordinated correctly. The use of **MPI_Sendrecv** leaves **MPI_Bull** to manage the exchange of the **send** and **recv** calls.

As far as is possible, use the **MPI_Sendrecv** call instead of successive calls to **MPI_Send** and then to **MPI_Recv**.

Use Collective operations whenever possible

It is worthwhile to bear in mind the collective operations which are possible with **MPI_Bull** and to simplify the code as much as is possible to take these into account. Often, a succession of point-to-point operations can be restructured and changed into a collective operation.

Do not use ANY_SOURCE

In accordance with the MPI-1 standard, it is possible to not specify the source for point-to-point operations, but to use the variable **ANY_SOURCE** which allows the "first source arising in the exchange" to be used.

As far as possible, it is advised to use an explicit source and not the **ANY_SOURCE** variable, which includes an additional overhead and consequently an impact on performance.

Whenever possible use intra-QBB transfers

The **MPI_Bull** library helps to minimize the overhead of data exchanges.

For messages greater than 32Kbs the **MPI_Bull** library uses the **MDM** module and the zero-copy mechanism to transfer the messages.

There is a light overhead for the zero-copy mechanism as this is done through the use of system calls but this is more efficient in terms of performance than the use of shared buffer zones.

For messages smaller than 32Kbs in size, the transfer of the data is carried out through the shared memory buffer and not through the **MDM** module –see chapter 2 in Bull HPC BAS4 *User's Guide*.

For messages smaller than 32Kbs in size a shared memory zone is created on each QBB to optimize intra-QBB data transfer. It is advisable to prioritise the use of intra-QBB transfers as these provide better performance than inter-QBB transfers.

4.3 MPI-2 One-Sided Operations

Regarding the **MPI-2 One-Sided** functionality present in **MPI_Bull**, it is important to understand the implementation choices which were made and to have an idea of how the program works so that possible improvements in performance can easily be identified.

The **MPI-2** standard stipulates that **MPI_PUT**, **MPI_GET** and **MPI_ACCUMULATE** operations should be completed before the return call of corresponding synchronization function (e.g. **MPI_WIN_POST, MPI_WIN_START, MPI_WIN_FENCE,** etc).

Accordingly, Bull chose to program the MPI library so that the data exchange is carried out immediately the **MPI_PUT**, **MPI_GET** and **MPI_ACCUMULATE** functions are called.

4.4 mpibull2-params

mpibull2-params is a tool that is used to list/modify/save/restore the environment variables that are used by the **mpibull2** library and/or by the communication device libraries (**InfiniBand**, **Quadrics**, etc.). The behaviour of the **mpibull2** MPI library may be modified using environment variable parameters to meet the specific needs of an application. The purpose of the **mpibull2-params** tool is to help **mpibull2** users to manage different sets of parameters. For example, different parameter combinations can be tested separately on a given application, in order to find the combination that is best suited to its needs. This is facilitated by the fact that **mpibull2-params** allow parameters to be set/unset dynamically.

Once a specific combination of parameters has been tested and found to be good for a particular context, they can be saved into a file by a mpibull2 user. Using the **mpibull2-params** tool, this file can then be used to restore the set of parameters, combined in exactly the same way, at a later date.

Note:
The effectiveness of a set of parameters will vary according to the application. For instance, a particular set of parameters may ensure low latency for an application, but reduce the bandwidth. By carefully defining the parameters for an application the optimum, in terms of both latency and bandwidth, may be obtained.

Note:
Some parameters are located in the **/proc** file system and only super users can modify them.

The entry point of the **mpibull2-params** tool is an internal function of the environment. This function calls an executable to manage the MPI parameter settings and to create two temporary files. According to which shell is being used, one of these two files will be used to set the environment and the two temporary files will then be removed. To update your environment automatically with this function, please source either the **$MPI_HOME/bin/setenv_mpibull2.sh** file or the **$MPI_HOME/bin/setenv_mpibull2.csh** file, according to which shell is used.

4.4.1 The mpibull2-params command

SYNOPSIS

```
mpibull2-params <operation_type> [options]
```

Actions

The following actions are possible for **mpibull2-param**s command:

-l : List the MPI parameters and their values

-f : List families of parameters

-m : Modify a MPI parameter

-d : Display all modified parameters

-s : Save the current configuration into a file

-r : Restore a configuration from a file

-h : Show help message and exit

Options

The following options and arguments are possible for the **mpibull2-params** command.

Note:
The options shown can be combined, for example, **-li** or can be listed separately, for example **–l –i.** The different option combinations for each argument are shown below.

-l [iv] [PNAME]

List current default values of all MPI parameters. Use the PNAME argument (this could be a list) to specify a precise MPI parameter name or just a part of a name. Use the **-v** (verbose) option to also display all possible values, including the default. Use the **-i** option to list all information.

Examples

```
mpibull2-params -l all shm
```

This will list all the parameters with the string 'all' or 'shm' in their name.
mpibull2-params -l | grep -e all -e shm will return the same result.

```
mpibull2-params -li all
```

This will display all information - possible values, family, purpose, etc. for each parameter name which includes the string 'all'. This command will also indicate when the current value has been returned by **getenv()** i.e. the parameter has been modified in the current environment.

```
mpibull2-params -lv rom
```

This will display current and possible values for each parameter name which includes the string 'rom'. It is practical to run this command before a parameter is modified.

-f [l[iv]] [FNAME]

This will list all the default family names. Use the FNAME argument (this could be a list) to specify a precise family name or just a part of a name. Use the **-l** option to list all parameters for the family specified. **–l**, **-v** and **-i** options are as described above.

Examples

```
mpibull2-params -f band
```

List all family names with the string 'band' in their names.

```
mpibull2-params -fl band
```

For each family name with the string 'band' inside, list all the parameters and current values.

-m [v] [PARAMETER VALUE]

Modify a MPI PARAMETER with VALUE. The exact name of the parameter should be used to modify a parameter. The parameter is set in the environment, independently of the shell syntax (**ksh/csh**) being used. The keyword 'default' should be used to restore the parameter to its original value. If necessary, the parameter can then be unset in its environment. The **-m** operator lists all the modified MPI parameters by comparing all the MPI parameters with their default values. If none of the MPI parameters have been modified then nothing is displayed. The **–m** operator is like the **-d** option. Use the **-v** option for a verbose mode.

Examples

```
mpibull2-params -m mpibull2_romio_lustre true
```

This will set the ROMIO_LUSTRE parameter in the current environment.

```
mpibull2-params -m mpibull2_romio_lustre default
```

This will unset the ROMIO_LUSTRE parameter in the environment in which it is running and returns it to its default value.

-d [v]

This will display the difference between the current and the default configurations. Displays all modified MPI parameters by comparing all MPI parameters with their default values.

-s [v] [FILE]

This will save all modified MPI parameters into FILE. It is not possible to overwrite an existing file, an error will be returned if one exists. Without any specific arguments, this file will create a file named with the date and time of the day in the current directory. This command works silently by default. Use the **-v** option to list all modified MPI parameters in a standard output.

Example

```
mpibull2-params -sv
```

This command will, for example, try to save all the MPI parameters into the file named Thu_May_10_15_50_28_2007.

Output Example

```
save the current setting :
mpibull2_mpid_xxx=1
1 parameter(s) saved.
```

-r [v] [FILE]

Restore all the MPI parameters found in FILE and set the environment. Without any arguments, this will restore all modified MPI parameters to their default value. This command works silently, in the background, by default. Use the **-v** option to list all restored parameters in a standard output.

Example

```
mpibull2-params -r
```

Restore all modified parameters to default.

-h

Displays the help page

4.4.2 Family names

The command **mpibull2-params –f** will list the parameter family names that are possible for a particular cluster environment.

The parameter families which are possible for Bull **BAS4** are listed below.

Quadrics_libElan_driver
LK_Ethernet_Core_driver
LK_IPv4_route
LK_IPv4_driver
OpenFabrics_IB_driver
Marmot_Debugging_Library
MPI_Collective_Algorithms
MPI_Errors
CH3_drivers
CH3_drivers_Shared_Memory
Execution_Environment
Elan
Elan_Hooks
Infiniband_RDMA_IMBR_mpibull2_driver
Infiniband_Gen2_mpibull2_driver
UDAPL_mpibull2_driver
IBA-VAPI_mpibull2_driver
MPIBull2_Postal_Service
MPIBull2_Romio

Run the command **mpibull2-params <fl> <family>** to see the list of individual parameters that are included in the parameter families used within your cluster environment.

Chapter 5. Lustre File System Optimization

This chapter describes how the **Lustre** (CFS) parallel file system should be optimized.

The following topics are described:

- 5.1 *Parallel File Systems - Introduction*
- 5.2 *Monitoring Lustre Performance*
- 5.3 *Lustre Optimization - Administrator*
- 5.4 *Lustre Optimization – Application Developer*
- 5.5 *Lustre Filesystem Tunable Parameters*

5.1 Parallel File Systems - Introduction

To be fully optimized large cluster needs all its storage devices, and file systems of the input/output sub-system, to work in parallel with very high I/O rates and capable of accessing many processors at once. A distributed file system such as NFS is not sufficient for the requirements of the system.

A parallel file system provides network access to a file system distributed across different disks or storage devices on multiple independent servers or I/O nodes. Real files are split into several chunks of data or stripes, each stripe being written onto a different component in a cyclic distribution manner (striping).

For a parallel file system based on a client/server model, the servers are responsible for file system functionality and the clients provide access to the file system through a "mount" procedure. This mechanism provides a consistent namespace across the cluster and is accessible via the standard Linux I/O API.

I/O operations occur in parallel across multiple nodes in the cluster simultaneously. All files are spread across multiple nodes including the I/O buses and disks, therefore I/O bottlenecks are reduced and the overall I/O performance is increased.

For large cluster configurations HPC **Bull** integrates the **Cluster File System (CFS) Lustre** parallel file system which is an open-source, object-based, Linux-based, POSIX-compliant system that offers very high performance.

There are separate sections in this chapter for the system administrator and the application developer. However, these are not exclusive, as any optimization of the Lustre file system will involve collaboration between these two. A lot of the optimizations need to be put in place when the platform is configured initially, and many aspects of application tuning cannot be done with user rights alone.

Refer to chapter 4 in the Bull HPC BAS4 *Administrator's Guide* (86 A2 30ER) for a description of the different parts of the **Lustre** file system architecture and the command syntax.

5.2 Monitoring Lustre Performance

The I/O performance of an application depends on:

1. The application itself, in particular how Input/Output data is sent to the File System
2. The performance of the system including the Linux kernel and drivers, and the Lustre file system.
3. Hardware performance including networking cards, disk arrays and storage devices.

These three aspects are interrelated and the overall performance for an application on a cluster depends on having a balance between all three. Different means exist for monitoring the performance of the file system and for identifying potential improvements.

5.2.1 Ganglia

Ganglia is a scalable, distributed, open-source monitoring tool, and is included as part of **NovaScale Master – HPC Edition**. This provides information about the cluster distribution for the **Lustre** file system and can be used for monitoring its performance in real time. This is important as the file system may be used by several different users at the same time, and if there is a perceptible drop in performance then **Ganglia** will indicate where there is uneven access to the files. Ganglia also collects **Lustre** statistics from **/proc/fs/lustre** in order to measure different aspects of file system performance.

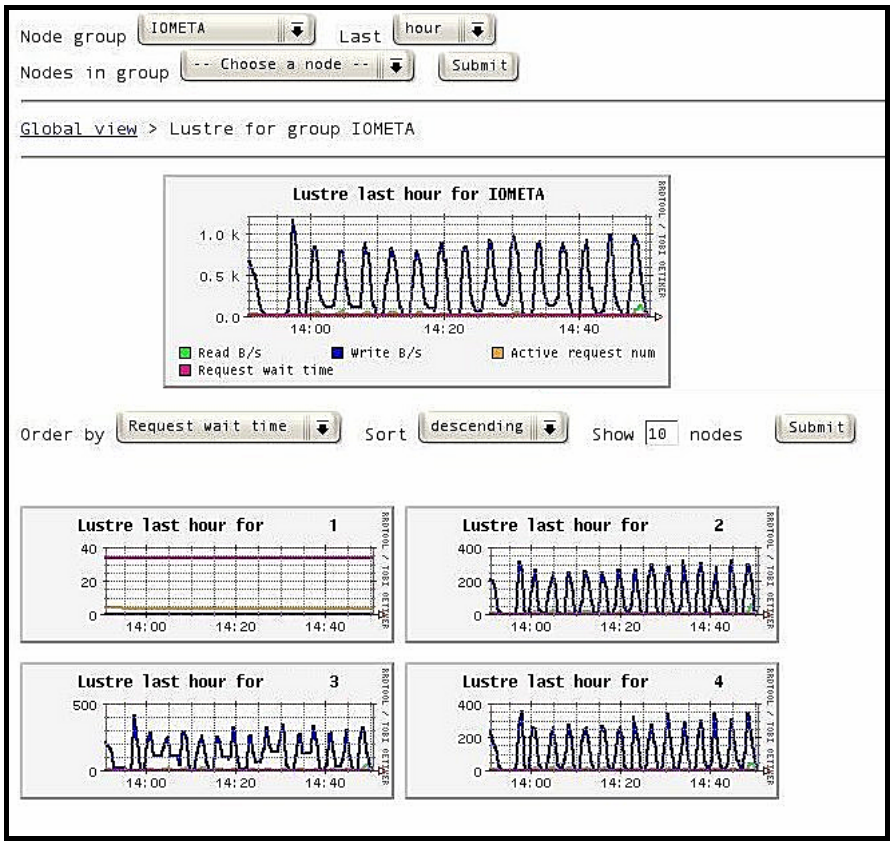

Figure 5-1 Ganglia Lustre monitoring statistics for a group of 4 machines with total accumulated values in top graph

See section 8.10 in the Bull *HPC Administrator's Guide* for more information on Ganglia

5.2.2 Lustre Statistics System

Lustre itself collects a range of statistics. These are available in files in the proc filesystem. A list of these files can be retrieved by using the command:

```
find /proc/fs/lustre –name "*stats*"
```

See the Lustre Operations Manual available on https://mail.clusterfs.com/wikis/lustre/LustreDocumentation, Chapter III -2 for more information on **LustreProc**.

5.2.3 Time

Time command – see Chapter 1 of this manual. Here are two examples with the **dd** command which is used exclusively for I/O operations.

Example 1

```
# time dd if=/dev/zero of=/tmp/testfile bs=1M count=100
100+0 records in
100+0 records out
104857600 bytes transferred in 0.738386 seconds (142009176
bytes/sec)

real    0m0.749s
user    0m0.001s
sys     0m0.476s
```

If the system time is high, as in example 1, then this is an indication that the resources being used for the application I/O are high.

Example 2

```
# time dd if=/dev/zero of=/home/testfile bs=1M count=1000
1000+0 records in
1000+0 records out
1048576000 bytes transferred in 44.987850 seconds (23307982
bytes/sec)

real    0m45.039s
user    0m0.012s
```

```
sys     0m5.262s
```

If the sum of the CPU user time and the system time is significantly lower than real time, as in example 2, then this may be an indication that, again, the resources being used for the application I/O operations are high.

5.2.4 RMS Prun

For systems which include Quadrics Interconnects the **prun** command with the **–sv** option will provide the same information as the **time** command and will also provide additional Quadrics statistics.

5.2.5 Iostat

When the host kernel has been configured to provide detailed I/O stats per partition the following information is available.

iostat can provide insight into the nature of I/O bottlenecks. It provides the nature and concurrency of requests being made of the attached storage devices.

iostat -x is invaluable for profiling the load on the storage devices which are part of a server node. The raw throughput numbers (wkB/s) combined with the requests per second (w/s) gives the average size of I/O requests to the device.

The service time indicates the amount of time it takes the device to respond to an I/O request. This sets the maximum number of requests that can be handled in turn when requests are not issued concurrently. Comparing this with the requests per second gives a measure of the amount of storage device concurrency.

Refer to the **iostat** man page for details regarding the meaning of the various columns in the output.

5.2.6 Llstat

llstat is a command which allows the examination of some of the Lustre statistics files. It 'decodes' the content by calculating statistics (min, max, mean, standard deviation) based on the contents of the file (sum and sum of squares).

See the Lustre Operations Manual available on https://mail.clusterfs.com/wikis/lustre/LustreDocumentation , Chapter III - 2.4.1 *RPC information for other OBD devices.*

5.2.7 Vmstat

The CPU use columns in the **vmstat** output file can be used to identify a node whose CPUs are completely occupied. On Metadata servers (**MDS**) and Object Storage Server (**OSS**) nodes, the I/O columns tell you how many blocks are flowing through the node's I/O subsystem. Coupled with the details of the attached storage for the node, it is then possible to determine if a subsystem in the node is the bottleneck.

Vmstat does not provide I/O block flow details for clients.

The columns that report swap activity can identify nodes that are having trouble keeping their working applications in memory.

5.2.8 Top

This tool is used to identify tasks which are using a lot of system resources. It can also be used to identify tasks which are not generating file system load, because they are using CPU or server threads which are struggling to obtain system resources on an overloaded node. See chapter 1 for more details.

5.2.9 Strace

The **strace** command intercepts and records the system calls called and received by a running process and is used to measure I/O activity at the system level.

Two options which are useful are:

-e trace=file traces activity related to system calls for file activity

-tttT gives a microsecond resolution.

These two options combined allow the measurement and evaluation of the performance of file system calls. However whilst **strace** is being used the performance of the application may be impacted. It is possible to use **strace** command for each system call. For example, use **–e trace=write** option to analyze the write performance.

5.2.10 Application Code Monitoring

It is possible to add system calls in the code during the development of the application which can then be used to measure the I/O file performance of the application itself.

Another option is that any I/O operation debugging traffic is included within the **NFS** system and not the **Lustre** system in order to minimize any additional overhead in the use of system resources.

Note:

Increasing the debug level can lead to a major performance impact. Full debugging can slow the system by as much as 90%, compared to the default settings.

Benchmarking tools are easier to work with to study performance. Once the tuning changes have been made, a general purpose benchmarking tool can then be used to check for any adverse effects.

5.3 Lustre Optimization - Administrator

The **Lustre** system administrator will need to monitor the file system to check the overall performance, and to identify any areas where there may be possible degradation in the service. See the *HPC BAS4 Administrator's Guide* for more details. **Lustre** includes tools to monitor I/O performance. These can be used to evaluate any changes that are made to the application, and also to see if, and where, performance could be improved.

The application developer and large cluster administrator will need to be clear in their definition of the needs for the application at the outset, as some of the system configuration settings made when installing and configuring the system will impact the performance of the application. Some flexibility is possible, as there are parameters which can be modified after the cluster has been configured to meet the particular requirements of the application.

The developer needs to be aware that the Input/Output algorithms, specified for the application will have a big impact on performance and some performance compromises may have to be made for different parts of the application with respect to the overall performance for the complete program.

The main bottleneck within the whole system normally is the I/O speed of the data storage devices, as this is usually the slowest part of a cluster.

Note:

It may be possible to observe superlinear speedups for the I/O throughput using **Lustre** client cache.

Raw benchmarking data, including control data should be available for the systems. The objective is to get as close to these performance figures as is possible with the application in place.

Lustre is ideal for large sequential write I/O operations as used, for example, by checkpoint/restart. Using **Lustre** it should be possible to obtain 80 to 90% of the raw I/O performance figures (Write operations generally perform better than read operations).

Attention is particularly needed when small random I/O Metadata operations are being performed. This is because the data may be unevenly distributed throughout the system.

Several points have to be kept in mind when attempting to tune the file system:

- **Lustre** is a part of a shared file system which means that it will be difficult to obtain exclusive use of Interconnects and data storage devices. For clients who need to have exclusive use of the file system, it is possible to do this by mounting it directly on the clients. This is in contrast to CPUs and memory where an application can be given exclusive use easily. Overall there are a lot of variables which can impact an application's performance.
- System caches can have a positive effect on performance if all the I/O traffic takes place in the client cache – in fact it is possible that the application bandwidth may appear greater than the disk bandwidth. System caches can also have a negative effect as **Lustre**'s **readahead** option may impact performance.

- There are a lot of data pipelines within the Lustre architecture. Two in particular have a direct impact on performance. Firstly, the network pipe between clients and OSSs, and secondly the disk pipe between the OSS software and its backend storage. **Balancing these two pipes maximizes performances**.

The **Lustre** file system stores the file striping information in extended attributes (EAs) on the MDT. If the file system has large-inode support enabled (> 128bytes), then EA information will be stored inline (fast EAs) in the extra available space.

The table below shows how much stripe data can be stored inline for various inode sizes:

Inode size (Bytes)	# of stripes stored inline
128	0 (all EA would be stored in external block)
256	3
512	13
1024	35
2048	77
4096	163

Note:

It is recommended that MDT file systems be formatted with the inode large enough for the default number of stripes per file to improve performance and storage efficiency.

One needs to keep enough free space in the MDS file system for directories and external blocks. This represents ~512 Bytes per inode.

Lustre stripes the file data across the OSTs in a round-robin fashion.

Note:

It is recommended to stripe over as few objects as possible to limit network overhead and reduce the risk of data loss when there is an OSS failure.

The stripe size must be a multiple of the page size. The smallest recommended stripe size is 512 KB because **Lustre** tries to batch I/O into 512 KB blocks on the network.

5.3.1 Stripe Tuning

Check that enough stripes are being used

It is important to remember that the peak aggregate bandwidth for I/O to a single file is restricted by the number of stripes multiplied by peak bandwidth per server. No matter how many clients try to write to that file, if it only has one stripe, all of the I/O will go to only one server.

Check that files are striped evenly over the Object Storage Targets

Lustre will create stripes on consecutive OSTs by default, so files created at one time will be optimally distributed among OSTs, assuming there are enough stripes and/or files created at that time. However, files created at different times may not have an optimal distribution among OSTs. To ascertain the file distribution, use the following command:

```
lfs getstripe
```

For more information refer to lfs man page.

If some servers appear to be receiving a disproportionate share of the I/O load check that the files are striped evenly over the OSTs.

If the I/O load is unbalanced for servers then use the lfs command to create a balanced set of files before the application starts, or if applicable, restructure the application so that Lustre striping is more efficient.

See the Lustre Operations Manual available on https://mail.clusterfs.com/wikis/lustre/LustreDocumentation, *Chapter IV-2. Striping and Other I/O Options* for more information on File Striping

5.4 Lustre Optimization – Application Developer

The main determinant on performance for the **Lustre** file system is the file size and how this is handled by the I/O devices. **POSIX** will handle the parallel distribution of the file, however, if the file is large performance may be impacted. The application developer has to decide if it is possible to chunk the program and thus gain performance.

The optimal level of performance is when the HPC platform I/O device read/write operations is as near as possible to that of the raw **Lustre** file system performance without the application running. Depending on the application program it should be possible to achieve 80% to 90% of the performance of a 'clean' HPC system.

One of the key questions to look at is to ascertain if the application performs I/O from enough client nodes to take full advantage of the aggregate bandwidth provided by the Object Storage Servers.

5.4.1 Striping Optimization for the Developer

Default striping settings are usually in the hands of the Lustre administrator who will normally use the default values. However, the application developer can also change these settings by using the **lfs** command on a per directory or on a per file basis. This controls the way parallel I/O operations are carried out for the files. Refer to the man page displayed under **lfs(1)** for more information.

Optimal striping settings depend primarily on the file size. It does not make sense to stripe a small file over several **OST**s, on the other hand it does make sense to stripe a big file over several **OST**s.

It is recommended to use the default striping settings configured by the Lustre administrator.

If the striping is to be changed, it is best to perform I/O tests with different striping configurations in order to find the best possible striping configuration.

5.4.2 POSIX File Writes

Being a high-performance distributed file system makes **Lustre** especially complex. By being POSIX compliant this complexity is simplified and no code modification is required whether a code is run on a local file system (**ext3, xfs**) or on **Lustre**. Only performance is enhanced.

There are several points to be kept in mind for file writes. Writes flow from the application that generates them to **OST**s where they are placed within the storage system. The network between the client and storage target needs to have capacity for the write traffic. It is also advisable to look for possible choke points along this path.

It should be said that these problems will only occur in extreme cases on the large cluster platform.

There must be enough write capacity for the application

If an application is to exploit a large network and disk pipes, it must generate a lot of write traffic, which can be cached on the client node and packaged into RPCs for the network.

There must be enough free memory on the node for use as a write cache. If the kernel cannot keep at least 4 MB in use for Lustre write caching, it cannot keep an optimal number of network transactions in progress at once.

There must be enough CPU capacity for the application to do the work which generates data for writing.

There must be enough storage space available

To prevent a situation in which Lustre puts application data into its cache, but then cannot write it to disk because the disk is full, Lustre clients must reserve disk space in advance. However, if it is unable to reserve this space as the OSTs are almost full, less than 2% space available, it must execute its writes synchronously with the server, instead of caching them for efficient bundling.

The degree to which this affects performance depends on how much your application would benefit from write caching. The cur_dirty_bytes file in the subdirectory of each OSC of /proc/fs/lustre/osc/ on a client records the amount of cached writes which are destined for a particular storage target.

The maximum amount of cached data per OSC is determined by the max_dirty_mb value in the same directory. This is usually 4 MBs by default. Increasing this value will allow more dirty data to be cached on a client before it needs to flush to the OST, but also increases the time needed for other clients to read or overwrite the cached data once it has been written to the OST.

Server thread availability

Write RPCs arrive at the server and are processed synchronously by kernel threads (named ll_ost_*). ps will help to identify the number of threads that are in the D state indicating that they're busy servicing a request.

Vmstat –see section 5.2.8 - can give a rough approximation of the number of threads that are blocked processing I/O requests when a node is busy servicing only I/O RPCs. The number of threads sets an upper bound on the number of I/O RPCs that can be processed concurrently, which in turn sets an upper limit for the number of I/O requests that will be serviced concurrently by the attached storage.

5.4.3 Fortran

Particular attention may be necessary for the I/O operations of Fortran as opposed to C as the Fortran run time library may modify the way the I/O operations are programmed. Please refer to the chapter 2 for more information on Fortran compiler optimizations, or to the compiler documentation from Intel, or to the manual page for the **ifort** command with particular reference to the section on environmental variables. The environmental variables **FORT_BUFFERED**, **FORT_CONVERT*** and **F_UFMTENDIAN** should be particularly looked at.

5.5 Lustre Filesystem Tunable Parameters

Warning:

Changing tunable parameters of the lustre filesystem can render the filesystem non functional. It should be done with great care on production filesystems. The use of default values is recommended.

One scenario where tuning the filesystem is beneficial is a cluster with several filesystems, some of which have clearly defined workloads. For these filesystems, the filesystem can then be tuned and optimized for this clearly defined workload. For example, if a filesystem is used only for checkpoint/restart purposes; the workload for this filesystem will probably consist of large sequential write and read I/O operations. It is then beneficial to tune the filesystem for this particular workload, particularly if the cluster has a large amount of memory.

Another example is when performing benchmarking: (temporary) changes may be applied in order to optimize benchmark throughtput.

This section describe the tunable parameters of the Lustre filesystem. For the syntax please refer Chapter 4 in the *Bull HPC BAS4 Administrator's Guide* (86 A2 30ER) or the **lustre_util(8)** manual page.

Additional information for these parameters is also available in the *Lustre Operation Manual* on https://mail.clusterfs.com/wikis/lustre/LustreDocumentation

5.5.1 Tuning Parameter Values and their Effects

max_read_ahead_mb

```
/proc/fs/lustre/llite/fs0/max_read_ahead_mb
```

This parameter defines the per-file read-ahead value for a client. Defaulting to 40MB **Read_ahead** is a two-edged sword: this can increase the read throughtput, but can be inefficient (if a file is read randomly rather than sequentially), and in turn detrimental as the memory which is wasted is not available elsewhere. The default value of 40MB is a general purpose value. It may be beneficial to increase this for sequential read workloads, whilst in other situations it may be better to disable it completely.

max_cache_mb

```
/proc/fs/lustre/llite/fs0/max_cache_mb
```

This parameter defines the maximum amount of inactive data cached by the client (the default value is ¾ of the RAM which is available).

max_dirty_mb

```
/proc/fs/lustre/osc/<...>/max_dirty_mb
```

This parameter has a value between 0 and 512MB.

This value controls the write back cache on the client per OSC. While it is beneficial to use larger values, the quantity of dirty data can become so high that, depending on the number of clients, it results in a significant amount time being needed to copy the data to disk.

max_page_per_rpc

```
/proc/fs/lustre/osc/<...>/max_page_per_rpc
```

This value should not be changed from the default value as the optimal value depends on the kernel page size.

max_rpc_in_flight

This value should not be changed from the default value as the optimal value depends on characteristics of the machine.

lru_size

```
/proc/fs/lustre/ldlm/ldlm/namespaces/<OSC name|MDC name>/lru_size
```

Increasing the default value is recommended for login nodes using **lustre** and for improving metadata performance.

debug

```
/proc/sys/lnet/debug
```

The debug level can impact the performance. This is the reason why it is disabled by default. When analyzing problems, different values may be used. The exact optimal value depends on the problem being analyzed: **full debugging** (**-1** value) can slow the filesystem noticeably and even mask the problem under diagnosis.

Note:

Increasing the debug level can lead to a major performance impact. Full debugging can slow the system by as much as 90%, compared to the default settings.

See Chapter III - 3.2 in the *Lustre Operations Manual* for more information on DDN Tuning.

5.6 More Information

For more information on tuning Lustre file systems see the latest version of the *Lustre Operations Manual* available from:

https://mail.clusterfs.com/wikis/lustre/LustreDocumentation

Appendix A. Tuning LibElan4 Variables

The Elan4 communications processor packaged in the QM500 network adapter card interfaces the nodes with the **QsNet^{II}** interconnect (also known as a switch network).

Problems can arise when the LibElan4 variables included in the Elan library used to drive the Elan4 board within the **Quadrics QsNet^{II}** Interconnect are set incorrectly.

A.1 MPI_USE_LIBELAN

MPI_USE_LIBELAN is a flag which indicates whether or not MPI should use Elan library optimizations in place of the default **MPICH** functions. It is also a bitmask, laid out as follows, allowing individual optimized MPI communication functions to be selected and, if necessary, disabled:

```
 11 10  9  8  7  6  5  4  3  2  1
+--+--+--+--+--+--+--+--+--+--+--+
| X | X | X | X | X | X | X | X | X | X |
+--+--+--+--+--+--+--+--+--+--+--+
  |  |  |  |  |  |  |  |  |  |   \ All the Quadrics optimisations
  |  |  |  |  |  |  |  |  |   \ Barrier
  |  |  |  |  |  |  |  |   \ Bcast
  |  |  |  |  |  |  |   \ Gather
  |  |  |  |  |  |   \ Allgather
  |  |  |  |  |   \ Alltoall
  |  |  |  |   \ Reduce
  |  |  |   \ Allreduce
  |  |   \ Gatherv
  |   \ Alltoallv
   \ Allgatherv
```

Normally, by default everything is active; however, if there is a specific hardware problem, these settings can be changed to disable the given feature or hardware functionality. Functionality should be restored but performance will still be slower than if there were no hardware problems and the default settings were used.

Note:

Modifying the settings for **MPI_USE_LIBELAN** should be seen as a temporary solution, to be used when there is a particular hardware or application problem which is impacting performance. Generally, it is always best to use the default LibElan4 settings to obtain maximum performance.

For example, if one wishes to disable hardware barriers the **MPI_USE_LIBELAN** bit settings which should be used are as follows:

```
 1  1  1 | 1  1  1  1 | 1  1  0  0
```

In HEX this yields:
0x 7 f c

Therefore, the following command should be used:

```
export MPI_USE_LIBELAN=0x7fc
```

Consequently a software barrier will be used.

Note:

If **Quadrics** hardware features are not used and communication operations are done in software alone then the performance will be slower.

A.2 LIBELAN_WAITTYPE

When there are more processes than CPUs available for a machine/cluster as is the case for overcommitted jobs, performance can be impacted by lock contention, which is when a process locks a critical variable so that other processes cannot access it.

Lock contention can be minimized by setting the LIBELAN_WAITTYPE variable within the LibElan4 library to YIELD as shown below.

```
export LIBELAN_WAITTYPE=YIELD
```

The variable should be set before the program is launched.

A.3 LIBELAN_TPORT_SHM_ENABLE

When using the **MPI_Bull** library, as opposed to the standard **Quadrics** MPI library, the standard Quadrics memory optimizations should be disabled as Bull's **MPI Data Mover (MDM)** mechanism performs these optimizations more efficiently. For example, these gains are most obvious if the **MPI_ANYSOURCE** variable is used and if it is receiving data.

Note:

As stated in section 4.2, it is good message passing practice not to use the ANY_SOURCE variable at all.

The **LIBELAN_TPORT_SHM_ENABLE** variable flags whether or not the Elan library can use the **Quadrics** based shared memory mechanism to exchange messages. This should be disabled as shown below:

```
export LIBELAN_TPORT_SHM_ENABLE=0
```

This setting should be made before the program is launched.

A.4 Debugging with the LIBELAN_DEBUGFLAGS variable

LIBELAN_DEBUGFLAGS is a bitmask variable used to specify which parts of the Elan libraries will include debug logging in order to try and isolate difficult MPI and Elan library communication problems. This should be used carefully when all other debugging methods have failed, as very large log files will be generated which may further reduce bandwidth and performance.

Note:

Modifying the settings for **LIBELAN_DEBUGFLAGS** should be seen as a temporary solution, to be used when there is a particular hardware or application problem which is impacting performance. Generally, it is always best to use the default LibElan4 settings to obtain maximum performance.

The bitmask settings for LIBELAN_DEBUGFLAGS are shown below:

```
#define DBG_DBG          0x00000    /* Message always displayed */
#define DBG_INIT         0x00001    /* General initialisation */
#define DBG_RX           0x00002    /* Message Reception (e.g. TPORT) */
#define DBG_TX           0x00004    /* Message Transmission (e.g. TPORT) */
#define DBG_BUF          0x00008    /* System buffering (TPORT) */
#define DBG_THRD         0x00010    /* Elan thread activity */
#define DBG_GROUP        0x00020    /* Collective operations */
#define DBG_SLEEP        0x00040    /* Blocking operations */
#define DBG_BUFGC        0x00080    /* Garbage collection of system buffers (TPORT) */
#define DBG_GALLOC       0x00100    /* Global memory allocator */
#define DBG_MULTI        0x00200    /* Multi-rail specific operations */
#define DBG_STAT         0x00400    /* Statistics collection */
#define DBG_FRAG         0x01000    /* TPORT Frag protocol - Control */
#define DBG_FRAG_RX      0x02000    /* TPORT Frag protocol - Reception */
#define DBG_FRAG_TX      0x04000    /* TPORT Frag protocol - Transmission */
#define DBG_T_STRIPE     0x08000    /* TPORT Striped protocol Tx/Rx */
#define DBG_SHM          0x10000    /* TPORT SHM protocol - Control */
#define DBG_SHM_RX       0x20000    /* TPORT SHM protocol - Reception */
#define DBG_SHM_TX       0x40000    /* TPORT SHM protocol - Transmission */
#define DBG_SHM_COLL     0x80000    /* SHM Collective operations */
#define DBG_PUTGET       0x100000   /* PUTGET operations */
#define DBG_QUEUE        0x200000   /* Queue operations */
#define DBG_LOCK         0x400000   /* Lock operations */
```

```
#define DBG_RTE         0x800000    /* Rte operations */
#define DBG_AM          0x1000000   /* Elan Active Message operations */
#define DBG_CALLOC      0x2000000   /* Cacheing allocator */
#define DBG_PGV         0x4000000   /* PGV operations */
#define DBG_CKSUM       0x10000000  /* TPORT Enable message checksums */
#define TRACE_GROUP     0x20000000  /* Traceing of group calls */
#define DBG_TRACE       0x80000000  /* Trace flag */
```

A.5 More information

For more information regarding the Elan Library Environment Variable settings and MPI refer to the *Elan Programming Manual* available from the **Quadrics** site, and also to the link below:

http://www.llnl.gov/computing/mpi/elan.html

Appendix B. Amdahl's Law

Amdahl's Law states that the proportion of the program which can run in parallel – the variable *p* – can never reach 100%:

$$Speedup(n) = \frac{1}{(p/n) + (1-p)}$$

p = parallel fraction of the program

n = number of CPUs

In addition, the benefits resulting from augmenting the processing power available for an application will diminish proportionally as a result of hardware constraints and extra message passing latency. The examples below are simple illustrations of this point.

Example 1

p = 0.5 *n* = 10 Speedup = 1.82

p = 0.5 *n* = 15 Speedup = 1.88%

% Increase in Speedup for an extra 5 CPUs = 3.3%

Example 2

p = 0.95 *n* = 10 Speedup= 6.9

p = 0.95 *n* = 15 Speedup = 8.8

% Increase in Speedup for an extra 5 CPUs = 27.5%

Therefore, the higher the value of *p*, the greater the return for any addition to processing power. This applies equally to small increases in *p*, and where the numbers of CPUs involved may be considerably higher.

A key part of any program development is to identify and remove as many dependence constraints as is possible. Generally speaking, there is more to be gained from increasing *p*, than there is to be gained from simply adding additional processing power as Amadhl's law demonstrates.

The benefits to be gained from optimizing and improving the program itself will generally outweigh benefits gained from adding to the hardware's performance.

Appendix C. The CPUSET Virtual Filesystem

With the installation of the **ptools** suite from the **BAS4 HPC CD**, the creation and management of cpusets will be done using the **pexec** and **pcreate** commands. However, if for some reason **ptools** are unavailable, **cpusets** may be managed using the CPUSET virtual filesystem.

To mount the CPUSET virtual filesystem, enter:

```
mount -t cpuset none /dev/cpuset
```

Under **/dev/cpuset** it will be possible to find a tree that corresponds to the tree of the cpusets in the system. For instance, **/dev/cpuset** is the **cpuset t**hat holds the whole system.

To create a new cpuset under **/dev/cpuset:**

```
cd /dev/cpuset
mkdir my_cpuset
```

To go to the new cpuset:

```
cd my_cpuset
```

To see the files within it:

```
ls
cpus  cpu_exclusive  mems  mem_exclusive  tasks
```

These files provide information about the state of this cpuset: the CPUs and Memory Nodes it can use, the processes that are using it, its properties. By writing to these files it is possible to modify the cpuset.

Set some flags:

```
/bin/echo 1 > cpu_exclusive
```

Add some cpus:

```
/bin/echo 0-7 > cpus
```

Attach the shell which is being used to this cpuset:

```
/bin/echo $$ > tasks
```

Create cpusets inside the cpuset by using mkdir in this directory:

```
mkdir my_sub_cs
```

To remove a cpuset:

```
rmdir my_sub_cs
```

The **rmdir** command will fail if the cpuset is in use - has cpusets inside it, or has processes attached.

Adding/removing cpus

The syntax to use when writing in the cpus or mems files in cpuset directories is as follows:

- To set cpus list to cpus 1,2,3,4:

```
/bin/echo 1-4 > cpus
```

- To set cpus list to cpus 1,2,3,4:

```
/bin/echo 1,2,3,4 > cpus
```

Setting flags

The syntax is very simple.

- To set the 'cpu_exclusive' flag:

```
/bin/echo 1 > cpu_exclusive
```

- To unset the 'cpu_exclusive' flag:

```
/bin/echo 0 > cpu_exclusive
```

Attaching Processes

```
/bin/echo PID > tasks
```

Note: it is PID, not PIDs.

It is only possible to attach ONE task at a time. If you have several tasks to attach, you have to do it one after another:

```
/bin/echo PID1 > tasks
/bin/echo PID2 > tasks
...
/bin/echo PIDn > tasks
```

Glossary and Acronyms

A

ACL
Access Control List

API
Application Programmer Interface

E

BAS
Bull Advanced Server

BIOS
Basic Input Output System

BMC
Baseboard Management Controller

C

CFS
Cluster File Systems

CGI
Common Gateway Interface.

ConMan
A management tool, based on telnet, enabling access to all the consoles of the cluster.

CMOS
Complementary Metal Oxide Semiconductor

Cron
A UNIX command for scheduling jobs to be executed sometime in the future. A cron is normally used to schedule a job that is executed periodically - for example, to send out a notice every morning. It is also a daemon process, meaning that it runs continuously, waiting for specific events to occur.

Cygwin
A Linux-like environment for Windows. The Bull cluster management tools use Cygwin to provide ssh support on a Windows system, enabling access in command mode from the Cluster management system.

D

DDN S2A
DataDirect Networks S2A

DNS
Domain Name Server. A server that retains the addresses and routing information for TCP/IP LAN users.

F

EFI
Extensible Firmware Interface (Intel)

EIP
Encapsulated IP

EPIC
Explicit Parallel Instruction set Computing

EULA
End User License Agreement (Microsoft)

G

FDA
Fibre Disk Array

FSS
Fame Scalability Switch.

FWH
Firm Ware Hub

G

Ganglia
A distributed monitoring tool used to view information associated with a node, such as CPU load, memory consumption, network load.

GCC
GNU C Compiler

GNU
GNU's Not Unix

GPL
General Public Licence

GUI
Graphical User Interface

GUID
Globally Unique Identifier

H

HBA
Host Bus Adapter.

Hyper-Threading
Hyper-Threading technology is an innovative design from Intel that enables multi-threaded software applications to process threads in parallel within each processor resulting in increased utilization of processor execution resources. To make it short, it is to place two logical processors into a single CPU die.

HPC
High Performance Computing.

HSC
Hot Swap Controller

I

IDE
Integrated Device Electronics

IPMI
Intelligent Platform Management Interface

IPO
Interprocedural Optimization

K

KDC
Key Distribution Centre.

KDE
Kool Desktop Environment.

KSIS
Utility for image building and development.

KVM
Keyboard Video Mouse (allows the connection of the keyboard, video and mouse either to the PAP or to the node)

L

LDAP
Lightweight Directory Access Protocol.

LKCD
Linux Kernel Crash Dump. A tool capturing and analyzing crash dumps.

LOV
Logical Object Volume.

LUN
Logical Unit Number

Lustre
Parallel file system managing the data shared by several nodes.

LVM
Logical Volume Manager.

M

MIB
Management Information Base.

MDS
MetaData Server.

MDT
MetaData Target.

MkCDrec
Make CD-ROM Recovery. A tool making bootable system images.

MPI
Message Passing interface.

N

Nagios
A powerful monitoring tool, used to monitor the services and resources of Bull HPC clusters.

NFS
Network File System.

NIC
Network Interface Card.

NPTL
Native POSIX Thread Library

NTFS
New Technology File System (Microsoft)

NTP
Network Time Protocol.

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

OpenSSH
Open Source implementation of the SSH protocol.

OSC
Object Storage Client.

OSS
Object Storage Server.

OST
Object Storage Targets.

P

PAM
Platform Administration & Maintenance Software.

PAP
Platform Administration Processor (Bull NovaScale platforms).

PAPI
Performance Application Programming Interface.

PCI
Peripheral Component Interconnect (Intel)

PDSH
A parallel distributed shell.

PDU
Power Distribution Unit

PMB
Platform Management Board

PMU
Performance Monitoring Unit

PVFS
Parallel Virtual File System

PVM
Parallel Virtual Machine

Q

QBB
Quad Brick Board – The QBB is the heart of the NovaScale Server, housing 4 Itanium™ 2 processors.

R

ROM
Read Only Me

RMS
Resource Management System. Manages the cluster resources.

RPC
Remote Procedure Call

RPM
RedHat Package Manager

S

SAN
Storage Area Network.

SCSI
Small Computer System Interface

SIS
System Installation Suite.

SM
System Management

SMP
Symmetric Multi Processing. The processing of programs by multiple processors that share a common operating system and memory.

SSH
Secure Shell. A protocol for creating a secure connection between two systems.

Syslog-ng
Syslog New Generation, a powerful system log manager.

T

TORQUE
Tera-scale Open-source Resource and QUEue manager. A batch manager controlling and distributing the batch jobs on compute nodes.

U

Unit
Generally it is the set of nodes linked to the same Quadrics switch. One unit contains 4 cells. (See also *Cell*).

UID
User ID

V

VNC
Virtual Network Computing. It is used to enable access to Windows systems and Windows applications from the Bull NovaScale cluster management system.

W

WWPN
World Wide Port Name- a unique identifier in a Fibre Channel SAN.

X

XFS
eXtended File System

Index

L

M

N

O

P

Q

R

S

T

U

V

Technical publication remarks form

Title:	HPC BAS4 Application Tuning Guide

Reference:	86 A2 19ER 06

Date:	December 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 Dept.
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
B.P.20845
49008 ANGERS CEDEX 01
FRANCE

Phone: +33 (0) 2 41 73 72 66
FAX: +33 (0) 2 41 73 70 66
E-Mail: srv.Duplicopy@bull.net

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 19ER 06