Best Practice Guide - IBM Power

Jeroen Engelberts, SARA
Walter Lioen, SARA
Huub Stoffers, SARA
Mirko Cestari, CINECA
Silvia Giuliani, CINECA
Elda Rossi, CINECA
Jorge Rodriguez, BSC
David Vicente, BSC
Guillaume Collet, CEA

19-06-2012
Table of Contents

1. Introduction ........................................................................................................... 4

2. System Architecture and Configuration ............................................................. 5
   2.1. Processor architecture ................................................................................... 5
   2.2. Building Blocks ............................................................................................ 5
   2.3. Memory Architecture ................................................................................... 6
   2.4. Networks and Node Interconnect .................................................................. 6
   2.5. I/O subsystem ............................................................................................... 6
       2.5.1. I/O subsystem on Huygens .................................................................. 6
       2.5.1.1. Components of the Huygens I/O subsystem .................................. 6
       2.5.2. I/O subsystem on SP6 at CINECA ...................................................... 7
       2.5.2.1. Components of the SP6 I/O subsystem ............................................ 7
   2.6. File systems on Huygens .............................................................................. 8
       2.6.1. GPFS file systems (home, scratch and project space) ......................... 9
       2.6.2. NFS file system (archive) ..................................................................... 9
       2.6.3. Home file systems ............................................................................. 9
       2.6.4. Scratch file systems .......................................................................... 10
       2.6.5. Project space ..................................................................................... 11
       2.6.6. Archive ............................................................................................... 12
   2.7. File systems on SP6 at CINECA .................................................................. 13
       2.7.1. System services .................................................................................... 13
       2.7.2. Reserved file systems ......................................................................... 13
       2.7.3. /sp6: permanent, backuped space ....................................................... 14
       2.7.4. /shared/data: permanent, not backuped space ................................... 14
       2.7.5. /gpfs/scratch: temporary space ......................................................... 15
       2.7.6. Project space ..................................................................................... 15
       2.7.7. Archive ............................................................................................... 15
       2.7.7.1. STK SL8500: A new generation robotic magnetic tape library .... 16

3. System Access ...................................................................................................... 17

4. Production Environment ....................................................................................... 18
   4.1. CINECA’s policy for access and resource management on HPC machines .... 18
       4.1.1. Access (username) ............................................................................. 18
       4.1.2. Budget (Account) .............................................................................. 18
       4.1.3. On how to select among different Accounts ....................................... 18
       4.1.4. Account life: running out of budget ..................................................... 19
       4.1.5. Account set up ................................................................................... 19
       4.1.6. New budgeting rules ........................................................................... 19
       4.1.7. The CINECA’s database for HPC users and projects ......................... 20
   4.2. Accounting on Huygens .............................................................................. 20
       4.2.1. Getting budget and accounting information ...................................... 20

5. Programming Environment / Basic Porting ......................................................... 20
   5.1. Available compilers ..................................................................................... 21
   5.2. Available (vendor optimized) numerical libraries ....................................... 24
   5.3. MPI ............................................................................................................ 26
   5.4. OpenMP .................................................................................................... 28
   5.5. MPI-OpenMP hybrid parallelism ................................................................. 28
   5.6. Batch system / job command language ....................................................... 29

6. Performance Analysis ......................................................................................... 34
   6.1. Available Performance Analysis Tools ........................................................ 34
   6.2. Hints for interpreting results ....................................................................... 37

7. Tuning Applications ............................................................................................. 41
   7.1. Advanced / aggressive compiler flags ....................................................... 42
   7.2. Advanced MPI usage .................................................................................. 44
       7.2.1. Tuning / environment variables ......................................................... 44
       7.2.2. Mapping tasks on node topology ....................................................... 46
       7.2.3. Task affinity ...................................................................................... 46
1. Introduction

The IBM Power6 architecture has family resemblances in many respects to the IBM Power7 system which is currently under development and is candidate architecture for PRACE Tier-0 systems. During the PRACE preparatory phase project an IBM Power6 system - “Huygens”, the current national supercomputer for Dutch academia - was one of the prototype systems. “SP6” is the other IBM Power6 system run by CINECA, acting as the current national supercomputer for Italian academia. “Huygens” and “SP6”, are both PRACE Tier-1 systems.

At BSC in Barcelona a PowerPC based super computer called MareNostrum [http://www.bsc.es/marenostrum-support-services/marenostrum-system-architecture] has been operated since 2005. Although this guide is mainly focussing on the two Power6 based systems in Italy and the Netherlands, many of the tips and tricks apply to MareNostrum, not it the least because these tips and tips are co-written by the people from BSC.

Figure 1. IBM Power6 575 System cabinets

This best practice guide provides information about IBM Power architecture machines in order to enable users of these systems to achieve good performance of their applications. The guide covers a wide range of topics, from the detailed description of the hardware, through information about the basic production environment, to information about porting and submitting jobs and tools and strategies for analysing and improving the performance of applications. Since this guide deals with a class of machines rather than with a single Tier-0 system, there is some room for variation. In fact, particularly at the levels of node interconnect technology and topology and the I/O subsystem there is considerable freedom of choice in design, and quite a few trade offs can be made in various implementation details. Hence the networks and node interconnects, and I/O subsystems of existing installations are quite system specific - rather than architecture specific. Consequently these are described in this guide at a system specific level of detail.
2. System Architecture and Configuration

2.1. Processor architecture

Power6 processors have 2 cores each running at a clock speed of 4.7 GHz inside a water-cooled Power 575 cabinet structure. Each core has 128 KB L1 cache consisting of 64 KB instruction cache and 64 KB data cache. Every core has semi-private 4 MB unified L2 cache, where the cache is assigned to a specific core, but the other has fast access to it. The two cores on a single processor share an off chip L3 cache of 32 MB. Access speed to this cache is 80 GB/s. Furthermore, each core has two floating point units. Each floating point unit is able to create a 64-bit fused multiply-add result per clock cycle. This leads to 4 flops per cycle per core. With this number the theoretical performance of Huygens (SARA) comes to 4 flops/core x 4.7 Ghz x 32 cores/node x 108 nodes = 65 Teraflops.

Power6 processors support Simultaneous Multithreading (SMT) [http://publib.boulder.ibm.com/infocenter/chresctr/vxrx/index.jsp?topic=%2Fcom.ibm.cluster.essl43.guideref.doc%2Fam501_f0106.html]. SMT is a processor technology that allows two separate instruction streams (threads) to run concurrently on the same physical processor, improving overall throughput. To the operating system, each hardware thread is treated as an independent logical processor. This means that, in that case, there are four logical cores, referred to as CPUs in LoadLeveler and Linux, on one processor.

Concerning running binaries

Sometimes confusion arises when users find out that the installed operating system on Huygens is “Linux”. They copy and try to run “Linux” binaries compiled on their own PC. It should be noted that, since the instruction set for processors is different, binaries can not just be copied from a PC and run on Power6.

Endianness

Another typical difference with regular PCs, which sometimes leads to confusion, is the endianness of the Power6 processor. In contrast with Intel and AMD processors as found in most PCs, the Power6 is big endian.

Most numbers in computer memory (or registers on the CPU), consist of multiple bytes. On big endian machines the first byte in memory is the most significant one (MSB). Consecutive bytes are less significant. This is also referred to as MSB-LSB. On little endian, the order is reversed, ie LSB-MSB.

As an example, in English it is common to write one hundred twenty three as “123”. This is, in read direction, MSB-LSB or big endian. In little endian notation, one hundred twenty three would become “321”.

This means that special care has to be taken with the transfer of binary data files from one type of machine to another. A typical case is when a user prepares (binary) input files on a local PC, which are transferred to the supercomputer for a large-scale calculation or, the other way around, when a user transfers binary output files produced on a HPC platform to its local PC for post processing or visualization. Several tools exist to circumvent this problem. Firstly, users can use ASCII coded (or human readable) files. Secondly, there exist a number of I/O libraries, like NetCDF and HDF, which ensure portability of binary data across different systems and take care of endianness under the hood.

2.2. Building Blocks

Both Huygens (SARA) and SP6 (CINECA) are based on IBM Power 575 Supercomputing nodes. The nodes are equipped with a valve system to enable the use of deionized water for cooling the high frequency processors. At the bottom of each cabinet, a heat exchanger is located to drain the excess heat.
Each node runs its own instance of the operating system. All nodes are equipped with 16 Power6 processors, or 32 physical cores. With SMT switched on, there are 64 logical cores (virtual cpus). Since each node runs its own operating system, they can be rebooted or repaired independently from the others, resulting in higher availability of the overall.

2.3. Memory Architecture

Nodes of Huygens (SARA) and SP6 (CINECA) are equipped with 128 GB of memory. Additionally, on Huygens at SARA 18 nodes contain 256 GB memory. It should be noted that only the 32 physical cores within each node have direct access to the memory. When more memory would be needed for a single task, programs should be able (or be enabled) to use some kind of communication mechanism, for example the common Message Passing Interface (MPI), or sockets. Memory inside a single node is not uniform and therefore Non-uniform memory architecture (NUMA) effects occur.

Inside a node there are four Quads. Each Quad has four processors, or Power6 chips with two cores. Each processor is called a dual chip module (DCM). A Quad has local memory, to which all cores inside it have fast access. The four Quads are all directly connected to each other at a bandwidth of 80 Gbit/s. The inter-Quad links are used to reach the memory modules that are not local to a core but within a node. It can be advantageous to assign certain tasks (MPI tasks, OpenMP threads) to certain cores. In Section 7 this will be explained in more detail.

2.4. Networks and Node Interconnect

Both Huygens and SP6 are equipped with an Infiniband interconnect. Each node has 2 Infiniband adapters with 4 ports each. The 8 ports on each node are connected with cables to 8 separate Infiniband switches, each with a throughput of 20 Gbit/s, leading to an aggregated interconnect speed of 160 Gbit/s.

2.5. I/O subsystem

The I/O subsystem (both on Huygens and SP6) consists of a collection of IBM GPFS file systems. GPFS (General Parallel File System) is a parallel distributed file system technology. A GPFS environment is usually setup by connecting small groups of dedicated servers to a number storage boxes in a way that provides both load sharing and fault tolerance through redundancy. The servers subsequently serve GPFS network storage devices (NSDs), or logical storage units (LUNs), which constitute the building blocks of a distributed file system, to GPFS clients over a network infrastructure. The SP6 does use a setup with dedicated NSD servers, but the Huygens I/O subsystem however deviates from this more familiar model.

2.5.1. I/O subsystem on Huygens

The Huygens I/O subsystem does not use a layer of dedicated NSD servers at all. Instead, all -108- compute nodes are directly coupled to all storage boxes - as if they all were NSD servers. Well, in fact they are - but only to the GPFS client on the same node.

For a cluster with 108 Power6 575 supercomputing nodes the setup without the need for an auxiliary cluster of dedicated I/O servers has proven to be a cost effective way of providing very good I/O performance. But at the same time it is clear that this scheme cannot be scaled to clusters with a much larger number of nodes.

2.5.1.1. Components of the Huygens I/O subsystem

Schematically we can distinguish three layers in the Huygens I/O subsystem setup:

1. Within each of the 108 P6-575 compute nodes themselves, there are dual ported fibre channel (FC) adapters. Each port provides a 4 Gbit/s link to a SAN switch. The majority, of “standard compute nodes” (SCNs), have two adapters and hence four FC links. A minority, of “memory compute nodes” (MCNs), have four adapters and hence eight FC links.
2. The FC links go to four SAN switches. The SCNs have a single FC link to each of the four switches. The MCNs have two FC links to each switch. Other ports on the switches are for FC links that connect to the storage controllers of IBM DS4700 storage boxes.

3. In total there are 90 IBM DS4700 storage boxes in the Huygens SAN environment. Each of them has two dual ported storage controllers. Each port provides a 4 Gbit/s FC link to a SAN switch. Each port is linked to one of the SAN switches. Although all 90 storage boxes use the same type of disks, not all of them have the same amount of disks and disk expansion units. Some boxes are configured more towards maximum performance and others more towards maximum storage capacity.

Figure 2 provides a schematic overview of the three interconnected layers, with the storage boxes at the top and the compute nodes (SCNs, MCNs) at the bottom.

**Figure 2. Interconnected components of the Huygens I/O subsystem**

---

2.5.2. I/O subsystem on SP6 at CINECA

The CINECA I/O subsystem is made of 28 NSD server nodes directly connected to the physical disks via a FC (fibre optic connection) network. All the other SP6 execution nodes can share data with them using the Infiniband network.

In particular we have:

- 24 NSD servers serving the scratch filesystem (/scratch)
- 2 NSD servers serving the production filesystems (/cineca, /prod, /meteo and /home)
- 2 NSD servers serving the Deisa/CINECA filesystem (/deisa/cne)

**2.5.2.1. Components of the SP6 I/O subsystem**

the SP6 I/O sub system is organised in a layered schema:

- Each of the 28 NSD servers has got two FC fibre links, for a max speed of 8 Gb/s
- the 24 NSD servers of the /scratch file system are connected to six storage servers “DCS9900”, called DCS1, DCS2, ... DCS6, containing the physical disks. Each of them features a eight ports
controller for 8 Gb/s FC connections. Each DCS9000 is connected to exactly 4 NSD servers (see Figure 3).

- The other 4 NSD servers are connected to SAN switches through 2 Gb/s FC cards. The switches, in turn, are connected to the DCS9000 storage server (see Figure 4).

**Figure 3. Interconnected components of the SP6 scratch filesystem**

**Figure 4. Interconnected components of the SP6 home filesystem**

### 2.6. File systems on Huygens

Each Huygens node has access to node local (system disk) file systems containing the kernel and associated operating system tooling, as well as a large part of the software that is available to users. Apart from the software they provide, these file systems are not of practical interest to users. The
performance, capacity, and other characteristics of the node local file systems are not described by this guide. A few small GPFS file systems are exclusively used for system services and for providing additional software and thus are “read only” and of little interest for users too. The file systems that are described in more detail by this guide are the ones that are writable for users and indeed are intended for storing user data. On Huygens, four classes of file systems for user files are available: home file systems, scratch file systems, project space, and archive file systems.

2.6.1. GPFS file systems (home, scratch and project space)

The home file systems, scratch file systems and project space all use IBM GPFS technology. All GPFS file systems are shared by all nodes - batch and interactive nodes alike - of the Huygens environment. Each Huygens node is directly connected to all storage units used by all GPFS file systems through the SAN infrastructure, as described in Section 2.5.1 on the “all to all” organization of the Huygens I/O subsystem. Access to (file systems)blocks on) storage units is of course regulated through communicating GPFS processes running on the nodes, but the data streams themselves run directly across a SAN switch from nodes to storage boxes.

2.6.2. NFS file system (archive)

The archive file systems available to Huygens nodes are NFS file systems that are mounted across a 1Gbit/s network infrastructure from an external server. The performance of the archive file systems is not related to the infrastructure described in section “I/O subsystem”. Because the back end used for the archive is completely independent from the GPFS and SAN infrastructure, I/O operations on the archive do not in any way influence the effective bandwidth at a node's disposal for accessing a home, scratch or project directory.

2.6.3. Home file systems

The home file systems are intended as a general repository of user resources: source codes, binaries, libraries, files with job input parameters, job outputs that are consulted on a regular bases, etc. Currently there are nine home file systems: each of them has a net storage capacity of 14 TB. All home file systems are comparable in performance. Users are more or less evenly distributed across all available home file systems. The home directory is always accessible by the pathname /home/username. In addition, in the standard Huygens user environment the shell variable $HOME set to point to a user's home directory. This is true for the interactive and batch environment alike.

A daily incremental backup service is in place, creating backups of all new and recently modified files. The backup service will skip files that are open and in the process of being modified by other processes while the backup service is running. But a file that has resided unmodified for at least 24 hours on a home file system will have a backup copy.

The decision to have multiple home file systems and to limit the capacity of an individual file system to 14 TB is mainly motivated by performance considerations concerning the backup and restore service. With the given backup and restore infrastructure a volume of 14 TB can be restored in a day, whereas a single volume of 9 times 14 TB definitely cannot.

Limits are enforced on a per user basis for data only. Currently no limits are enforced on the number of I-nodes, i.e. number of file system objects: regular files, directories, and symbolic links. The default per user data limit is 200 GB. Adequately grounded requests for an enlarged quotum are usually honoured, if they remain within reasonable bounds given the size of the home file systems.

The home file systems are readily available on all batch nodes, but users should take into account that they are configured for less demanding I/O request and somewhat smaller file sizes than the scratch file systems (see next section).

In total six DS4700 storage boxes are used to implement the nine home file systems. Contrary to other file systems, the home file systems do not have DS4700 resources that are dedicated to a single file system. The setup is as follows:
• All home file systems uniformly use six 8+p 2.2 TB LUNs.

• The stripe size on these LUNs is 1 MB and the file system block size on all home file systems is 1MB, i.e. 1048576 bytes. So, efficient I/O operations on Huygens home file systems use this block size or an exact multiple thereof.

• The six LUNs of each home file system are evenly spread over two storage boxes: three in one storage box, three in another.

Within a storage box however, despite the fact that each controller has the same number of LUNs assigned, some imbalance is introduced with respect to any particular home file system: there is no way to evenly divide three LUNs over two controllers.

Since there are 6 LUNs in each home file system and the average LUN speed that can be sustained at the level of the storage controller is 90 MB/s, the I/O bandwidth that reasonably can be expected from a single home file system is about 540 MB/s. Empirical tests in the Huygens production environment corroborate that a slightly better result is easily obtained on a single host, using multiple processes that do I/O requests of 1 MB in size, or even by a single process doing I/O requests of eight or more blocks in size.

When multiple nodes are used - thus utilizing more SAN resources on the host side to push data to or pull them from the same storage units - addressing the same home file system simultaneously in a similarly efficient manner, the total bandwidth can be pushed to about 780 MB/s. The gain in bandwidth is no doubt at the expense of other home file systems that share the controller same resources and could only be achieved because the other file systems simply were not so busy at the time of testing.

### 2.6.4. Scratch file systems

As implied by the name, the scratch file systems are intended as a “scratch space” for jobs. They are the best file systems to use for large scale and demanding I/O. So, in many cases they are also the recommended file systems to produce the end results of jobs on. If a job’s end results have to be kept for a longer time, which is quite common, they will have to be saved subsequently to a home directory or to the archive file system. Files on scratch are automatically purged once older than 14 days and no backup is made at all. So, if you accidentally overwrite or delete files there is no way to have them restored.

For historical reasons there are two scratch file systems with a combined capacity of approximately 240 TB. Data limits are enforced on both of them on a per user basis. Currently no limits are enforced on the number of I-nodes. The default per user data limit is 16 TB, i.e.: 8 TB on each of the two scratch file systems. Adequately grounded requests for enlarged quota can be honoured. However, in our experience the scratch space quota seldom are the real problem. Most users with such a requests are in fact better helped with project space, a space exempted from the automatic purge policy for files older than 14 days (see next section).

The two scratch file systems on Huygens are usually referred to as “local scratch” and “shared scratch”. The shared scratch is accessible from every node by the pathname /scratch/shared. It is a symbolic link on every node that points to the same directory on a GPFS file system. So, if you create a directory /scratch/shared/my_work on the interactive node, the same path will denote the same file system location on all other nodes. The /scratch/shared directory is writable by everyone, but only the owners of files and directories, usually the users that created them in the first place, can remove them. But the idea is not to let your jobs produce output in this directory. Rather create a subdirectory of your own under it first, and use that.

The “local scratch” file system is accessible from every node by the pathname /scratch/local. This is a symbolic link that is setup on every node to point a node specific directory. So, if you e.g. create a directory /scratch/local/my_work on five different nodes, that path will denote five distinct directories on the “local scratch” file system. Actually, the term “local scratch” is a bit of a misnomer in that the file system is in fact shared (GPFS). The default data limit of 8 TB applies on a global file system level too. The “local scratch” setup merely mimics the separate name space situation
of conventional Beowulf clusters with node local scratch file systems to which some programs, if not their users, are accustomed. In the batch environment the shell variable `TMPDIR` is set to point to a node specific and job step specific directory and the “local scratch” file system.

We do not object to users using the “local scratch” file system in a global, not node specific way. The actual mount point of the file system is the same on every node: `/gpfs/mcn1`.

Both scratch file systems are built from equally equipped and configured dedicated DS4700 storage boxes. Hence they are very comparable but not identical in performance. The “shared scratch” file system is built from 30 DS4700 boxes, and the “local scratch” file system is built from 24 DS4700 boxes. The six additional boxes add both more capacity as well as more aggregate bandwidth.

The setup is as follows:

- Both scratch file systems uniformly use 4+p 1.1 TB LUNs
- The stripe size on these LUNs is 2 MB and the file system block size on the scratch file systems is 4 MB, i.e. 4194304 bytes
- The “local scratch” file system has 96 LUNs, resulting in a capacity of approximately 105 TB and an aggregate bandwidth of approximately 21 GB/s
- The “shared scratch” file system has 120 LUNs, resulting in a capacity of approximately 132 TB and an aggregate bandwidth of approximately 26.4 GB/s

Empirical tests have shown that it takes the orchestrated action of the I/O resources of at least eighteen SCNs to actually reveal the aggregate bandwidth that the “local scratch” file system is capable of. To do the same for the “shared scratch” file system at least twenty SCNs are be needed.

Next to local and shared scratch the system disk contains conventional entry points for temporary files: directories such as `/tmp`, `/var/tmp` and `/var/lock` are present and are, in principle, writeable by any user. These are there for system operating purposes and not meant for extensive usage by the user. There is only limited space available on them and the I/O performance is much lower than on any of the scratch entry points.

### 2.6.5. Project space

At present there are two file systems available for special project space. By default they are not writable by users. Writeable directories and quota arrangements are created on an ad hoc basis, for an agreed upon limited period of time. Unlike scratch directories, special project space is exempted from routines that automatically purge files. Like with scratch directories, there is no backup service in place for project space. Users must archive files of which they need a backup copy themselves.

Both project space file systems use 8+p LUNs, but one is built from six DS4700 storage boxes with a total of 60 LUNs, resulting in a volume of 132 TB. This is the slower file system of the two. It is slower because the controller to LUN ration in 1:5, resulting in an average LUN speed of 90 MB/s. The file system block size used on this file system is 1MB. The mount point for this file system on every node is `/gpfs/prj1`.

A faster project space is implemented on 24 DS4700 boxes with a total of 96 LUNs, resulting in a volume of 211 TB. The controller to LUN ratio is 1:2, resulting in an average LUN speed of 225 MB/s. The file system block size used on this file system is 4 MB. The performance characteristics are comparable to those of the scratch file systems. In fact this file system was once intended as a separate local scratch for SCN nodes. The mount point for this file system on every node, `/gpfs/scn1`, still reflects that.

Table 1 compares both project spaces in terms of several file system characteristics with other GPFS file systems in the Huygens.

---

1The name still reflects that originally a separate “local scratch” file system for MCN nodes was planned.
Table 1. Several characteristics of GPFS file systems in the Huygens environment compared. Note that for a single home file system 2 DS4700 are used, but there resources are not exclusively dedicated to a single home file system.

<table>
<thead>
<tr>
<th>filesystem purpose</th>
<th>/gpfs mount point</th>
<th>Number of DS4700 storage boxes</th>
<th>Number of LUNs</th>
<th>LUN type</th>
<th>Block size</th>
<th>File system capacity</th>
<th>Aggregate file system bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>a single home file system</td>
<td>h0&lt;n&gt;</td>
<td>*2</td>
<td>6</td>
<td>8+p</td>
<td>1 MB</td>
<td>14 TB</td>
<td>540 MB/s</td>
</tr>
<tr>
<td>local scratch</td>
<td>mcn1</td>
<td>24</td>
<td>96</td>
<td>4+p</td>
<td>4 MB</td>
<td>105 TB</td>
<td>21 GB/s</td>
</tr>
<tr>
<td>shared scratch</td>
<td>shscr1</td>
<td>30</td>
<td>120</td>
<td>4+p</td>
<td>4 MB</td>
<td>132 TB</td>
<td>26.4 GB/s</td>
</tr>
<tr>
<td>Fast project space</td>
<td>scn1</td>
<td>24</td>
<td>96</td>
<td>8+p</td>
<td>4 MB</td>
<td>211 TB</td>
<td>21 GB/s</td>
</tr>
<tr>
<td>slower project space</td>
<td>prj1</td>
<td>6</td>
<td>60</td>
<td>8+p</td>
<td>1 MB</td>
<td>132 TB</td>
<td>5.3 GB/s</td>
</tr>
</tbody>
</table>

2.6.6. Archive

Job output results that have to be kept for some time may very well become too large to be kept on the home file system. Basically all files that are to be kept but will not be actively used for some time can be moved to the archive. Currently there are four archive file systems. They are identical in terms of capacity and performance. Automated migration policies are in place on all archive file systems. Files that have resided on an archive file system for a while will have been migrated to tape.

A daily backup service is in place. If a file is migrated to tape, an independent backup copy, residing on another tape, exists as well. The archive file systems are not exclusive to Huygens, but are shared by a wider group of systems at SARA.

No data limits are set. No I-node limits are set either, but users with sets of many small files are urged to organize such files into sets that are put in a single container file and to subsequently put only the container file in the archive. This significantly reduces the number of I-nodes within the archive file systems and thus will have a very beneficial effect on the overall performance of many routines that handle meta-data. The tar command is the most appropriate tool to create such container files. It also has options to list files in a “tar archive”, extract files, etc. Please consult the online tar(1) manual page for further information.

The archive file systems available to Huygens nodes are NFS file systems that are mounted across a 1Gbit/s network infrastructure from an external server. Users should not bother on which particular home file system his or her archive directory is located. The archive directory is always accessible by the pathname /archive/username that is implemented as a symbolic link to the actual directory.

Although multiple links to the archive server are trunked, a single I/O operation on an archive file system will only utilize one of the links and hence cannot achieve more than a bandwidth of 1Gbit/s, i.e. 128 MB/s. No file system performance data are given. In practice speeds are very much determined by the activities of the tape backend.

We tend to regard the NFS file interface as an interface for convenience. It easy to have a look “what files are where” in the archive. Better storage and retrieval performance if often achieved however by using multiple SCP copies over the network or by using the gridFTP protocol with multiple streams.
2.7. File systems on SP6 at CINECA

Each SP6 node has access to node local (node specific) file systems as well as several shared file systems. The node local file systems are not of practical interest to users and are best considered “read only” for users. The performance, capacity, and other characteristics of the node local file systems are not described by this guide.

The file systems that are of interest to the user are the three read/write data areas. These three areas can be addressed via the predefined “environment variables”:

- $HOME - Home directory
- $CINECA_DATA - Data area
- $CINECA_SCRATCH - Scratch space

You are strongly encouraged to use these environment variables instead of full paths to refer to data in your scripts and codes.

### 2.7.1. System services

A few small shared GPFS file systems are exclusively used for system services and for providing additional software and thus are “read only” (/cineca, /prod).

<table>
<thead>
<tr>
<th>File System</th>
<th>Description</th>
<th>Dimension</th>
<th>Number of NSD/LUN</th>
<th>Block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>/cineca</td>
<td>Contains the files related to system management: configurations, accounting, binary packages for the operating system, control scripts.</td>
<td>200 GB</td>
<td>2</td>
<td>16 KB</td>
</tr>
<tr>
<td>/gpfs/prod</td>
<td>Linked to /cineca/prod: contains the application software for the final users.</td>
<td>300 GB</td>
<td>2</td>
<td>16 KB</td>
</tr>
</tbody>
</table>

### 2.7.2. Reserved file systems

Other shared file systems are reserved to dedicated services or special categories of users:

<table>
<thead>
<tr>
<th>File System</th>
<th>Description</th>
<th>Dimension</th>
<th>Number of NSD/LUN</th>
<th>Block size</th>
</tr>
</thead>
<tbody>
<tr>
<td>/gpfs/meteo</td>
<td>Contains programs and data for the national weather forecast service.</td>
<td>25 TB</td>
<td>24</td>
<td>16 KB</td>
</tr>
<tr>
<td>/deisa/cne/home</td>
<td>Contains the home space reserved to Deisa users on SP6 at CINECA. This file system is based on GPFS, but is managed by a different cluster with respect to the other GPFS file systems. It is mounted using the “multicluster feature” of GPFS. This file system is mounted on all SP6 execution nodes, as well as to all HPC clusters in the DEISA network.</td>
<td>2 TB</td>
<td>2</td>
<td>16 KB</td>
</tr>
</tbody>
</table>
Block size: 1 MB

/deisa/cne/data: contains the data space reserved to Deisa users on SP6 at CINECA. This file system is based on GPFS and its organisation is similar to the previous one. It is mounted on all SP6 execution nodes, as well as to all HPC clusters in the DEISA network.

Dimension: 3 TB

Number of NSD/LUN: 4

Block size: 1 MB

The file systems that are described in more detail by this guide are the ones that are writable for users and indeed are intended for storing user data. On SP6, three different file systems are available (home, scratch, data); a new file system (project) is planned for the near future and will substitute data. Moreover an archive facility is available.

**Figure 5. User file systems of CINECA HPC systems**

2.7.3. /sp6: permanent, backuped space

This is a data area where you start after the login procedure. It is where system and user applications store their dot-files and dot-directories (.nwchemrc, .ssh,...) and where users keeps initialization files specific for the systems (.cshrc, .profile,...). This area is 2 Gbytes large and is conceived to store programs, script and small personal data. Files are never deleted from this area, moreover they are guaranteed by daily backups: if you delete or overwrite a file accidentally, you can ask the helpdesk to restore it.

/sp6: contains the home directories of the users. On this file system the “quota” of 2 GB is defined for each user.

Dimension: 2 TB

Numbers of NSD/LUN: 2

Block size: 16 KB

2.7.4. /shared/data: permanent, not backuped space

This is a very large data area, conceived to store large personal data. There is no backup on this area, but it is much larger than home. Data will be never deleted automatically. At present a quota of 200GB
is defined on this data area. Data stored here are preserved from disk failures with some level of data replication: data will be still accessible even in case of failure of some of the physical disks.

/shared/data is not mounted on the computing nodes, and it is not available to batch jobs. If you need to move large amount of data to/from this filesystem, and the interactive way is not feasible, use the “archive” class of the loadleveler, that is the only batch class that can access the file system.

This file system is based on GPFS, but is managed by a different cluster with respect to the other GPFS file systems. It is mounted using the “multicluster” feature of GPFS.

One special feature of /shared/data is that it is available on all login nodes of ALL HPC systems in CINECA, thus facilitating the sharing of data among different systems.

<table>
<thead>
<tr>
<th>/shared/data: large data area for storing personal data. A default quota of 200 Gbyte is set on this file system. It is mounted only on the login nodes of SP6, not accessible by batch jobs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension: 130 TB</td>
</tr>
<tr>
<td>Number of NSD/LUN: 64</td>
</tr>
<tr>
<td>Block size: 1 MB</td>
</tr>
</tbody>
</table>

### 2.7.5. /gpfs/scratch: temporary space

This is a temporary storage, the retention time is 30 days: files are removed daily by an automatic procedure if not accessed for more than 30 days. In the home directory of each user ($HOME) a file lists all deleted files for a given day. This area is conceived for hosting large temporary data files. It is characterized by good I/O performance when accessing large blocks of data, while it is not well suited for frequent and small I/O operations. We suggest to launch batch jobs from this filesystem, since it is mounted on execution nodes and is very large. $CINECA_SCRATCH does not have any disk quota.

<table>
<thead>
<tr>
<th>/hpfs/scratch: large filesystem optimized for large size block of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension: 550 TB</td>
</tr>
<tr>
<td>Numbers of NSD/LUN: 144</td>
</tr>
<tr>
<td>Block size: 4 MB</td>
</tr>
</tbody>
</table>

### 2.7.6. Project space

At present there is no data space assigned to projects, although it is planned for the near future. The “project space” should be assigned to each project (called “account” on the machine) with a default quota of 200 GB. Each user who takes part in the project has read/write access to the project space. The quota can be increased in agreement with the project requirements. Unlike “scratch” directories, special “project space” is exempted from routines that automatically purge files. Like with “scratch” directories, there is no backup service in place for “project space”. Users must archive files of which they need a backup copy themselves.

The “project area” is expected to substitute the “data area”. It will be created of the same disks and servers, using the same hardware devices.

### 2.7.7. Archive

Job output results that have to be kept for some time may very well become too large to be kept on the home or data file system. Basically all files that are to be kept but will not be actively used for some time can be moved to the archive. Currently there are no archive file systems. Whenever a user wants to archive his data, he has to explicitly move his data to the archive server using the cart* commands.
A user can acquire as many “volumes” as he wants (they can be regarded as virtual CDs) and store data on them. When he does not need his archived data anymore, he can delete single files or even whole volumes. The most useful cart commands are:

- **cart_new:** create a new volume
- **cart_dir:** show all volumes and files
- **cart_put:** save a file in a volume
- **cart_get:** read a file from a volume
- **cart_del:** delete a file from a volume or a full volume

For more details refer to the commands short help (−? flag).

The **cart** commands can be run in interactive mode (from the login nodes) or in batch mode if more than 10 minutes are required. In this case a user has to use the archive batch queue. This is the only class allowed to access the cartridge system.

### 2.7.7.1. STK SL8500: A new generation robotic magnetic tape library

The STK SL8500 is a new generation robotic magnetic tape library dedicated to medium/long term storage. Its main feature is the independent movement of the 4 robotic arms providing a higher availability and shorter response times to mount/dismount requests. It provides direct user access by means of specialized software (Tivoli/TSM) running on highly available servers for scientific and enterprise applications and services.

- **Model:** StorageTek StreamLine 8500
- **Features:** Optical label recognition, Robotic cartridge loader
- **Magnetic Media:** STK 9940B 200GB (600GB compressed)
- **Read/Write Units:** 6*9940B Fibre Channel
- **Connectivity:** Switched Fabric SAN, Local FC HUB
- **Capacity:** 2.7 PB total (max. density media), 4500 cartridge slots
- **Performance:** up to 500 mount operation per hour
3. System Access

Logging in

Both systems, Huygens at SARA and the SP6 at CINECA are running some flavor of Unix. Command line access to such systems can be obtained with SSH. Once users have received a username and a password, they will be able to login.

Currently, three operating systems are popular amongst users, Windows, Mac OS X and Linux. To start with the latter, the SSH client is already installed with a basic installation of the OS. A second advantage is that most of these installations run on the X windows system. This comes in handy whenever a graphical user interface is required. Examples of such occasions are graphical debuggers.

Some 10 years ago, Apple decided to redesign its Mac OS to an own Unix flavor called Darwin. Like with Linux, an SSH client is already installed and X windows is available and easily installed.

For Windows users, some more preparations are necessary. By default, Windows does not come with an SSH client, nor with an X windows server. Several freeware and commercial packages are available and can be found with Google. Commonly used command line tools that can be downloaded from the web are PuTTY (free) and “SSH Tectia” (free trial version for 45 days). A free X windows server program for Windows is Xming.

File transfer

Like login in over a secure socket layer (SSL), also files are transferred over that layer. In Unix and OS X, files can be transferred using SCP. Again, on Windows there is no default SCP client. “SSH Tectia” comes with a file transfer feature, another free tool is WinSCP.

On the supercomputers, several file systems (locations) exist for storing files. Please refer to the following sections to see which is suitable for each type of file.
DEISA (http://www.deisa.eu)

Both systems described above are part of the DEISA infrastructure. This means that the supercomputers can also be accessed and controlled by other programs.

Login in can be done using the GSISSH protocol. To make use of this, a user needs a Grid certificate as means of authentication, rather than a username and a password. File transfer takes place over GSIFTP or GridFTP.

Jobs can be prepared and submitted via the special DEISA Unicore client. This is an extra layer to hide the differences in methods of controlling jobs to the user. Several batch systems to control jobs are installed on the supercomputers. The batch systems used here are described in Section 5.

4. Production Environment

Both the SP6 and Huygens have their own specific rules and setup with respect to accounts and users. In this section this information will be provided.

4.1. CINECA's policy for access and resource management on HPC machines

The policy reported here has been introduced at CINECA in July 2010 and applies to all new defined users. These new rules are presently effective only for new users. The remaining usernames will stay unaltered until their natural expiration date. Starting 2012, old active usernames will be migrated and finally standardized within the new paradigm. Everything that follows is only applicable to “new style” usernames.

4.1.1. Access (username)

Access to HPC platforms is allowed to “login” owners: this means that each user is uniquely identified by a username/password pair. Login credentials are to be considered as strictly personal, meaning that no sharing between members of the same working group is allowed. Each single user entitled with login credentials is considered to be personally responsible for any misuse that takes place.

Users will keep their login credentials as long as their work on CINECA platforms lasts: this means that personal usernames are bound to specific projects the user is involved in.

4.1.2. Budget (Account)

Relevant information related to projects, such as budget in CPU-hours, validity (start and end date), Principal investigator and collaborators, are managed within the scope of the ACCOUNT: this is defined for each single project.

One or more users, once provided with login credentials, can ask to be associated to one or more projects accounts, also in a concurrent mode: for instance, an individual user could be the Principal Investigator (PI) for one project, and could also wish to join another project as a collaborator.

4.1.3. On how to select among different Accounts

Users must be associated to (at least) one Account in order to submit jobs to the resource manager in batch mode. Otherwise, only the interactive environment will be available with all the relevant limits in terms of elapsed time, memory and scalability.

In order to access the batch mode, by submitting jobs to the resource manager, it is mandatory to specify on which account (which project budget) the billing for the run is to be referred.
This is done by specifying the so called “account_no”, in a form that depends on the resource manager itself.

Add the following entry in the LoadLeveler submission script:

```
# @ account_no = <account_no>
```

with `<account_no>` being one of the accounts the submitting user is authorized for. If this entry is not included in the batch submission script, the resource manager will reject the request. In order to retrieve the current status of an account, the command `saldo` should be used:

```
$ saldo -b
```

this command gives the list of the associated projects, along with their status (e.g. remaining budgets, expiring dates) and with the user's daily consumption for each of the authorized accounts.

**NOTE:** Each user (with login credentials), can exploit the execution class named “Interactive” on SP6 for debugging purposes (please refer to the SP6 user guide): this class has a wall-clock limit of 600 seconds and NO BILLING is active for this class.

### 4.1.4. Account life: running out of budget

Whenever an account is running out of budget (in CPU hours), or when its expiring date is met, the account itself will be shut down. This meaning that all the usernames referring to that account, won't be able to submit batch jobs on that project budget.

Nevertheless, they will still be able to access the HPC platform in order to perform some lightweight post-processing (e.g. using the interactive class) and/or to retrieve their data. Usernames will be kept alive for a whole year after their last (most recent) account has been shut down.

### 4.1.5. Account set up

The mapping between users and projects (alt. between Username and Account), is defined by the Principal Investigator of each active project. By accessing the “UserDB” portal (userdb.hpc.cineca.it), each PI can dynamically modify the list of his/her collaborators, in addition to visualising all relevant information on the project history.

**NOTE:** The UserDB portal is under development. Not all features could be available at a given time.

### 4.1.6. New budgeting rules

For the new project grants also the budgeting rules have been completely modified: instead of actual cpu time, the billing is now based on elapsed time and the effective number of cores (reserved, not used!) by the batch job.

These new rules, which are already applied in most if not all computer centres, are fairer for users because they discourage the submission of jobs reserving large numbers of resources (e.g. cores or memory) but which in the end aren't actually used.

In particular on SP6, if you require a memory per task larger than the “average” memory (3.4G per core or 1.7G per cpu), this will result in a bigger request in terms of number of cores and in a possible dramatic increase of the cost of your job.

In order to help you to better understand the real requirements of a job, a rough analysis is given at submit time:
This job will cost you: (elapsed * <n_proc>) hours
--------------------------------------------------

where \(<n\_proc>\) is the actual number of processors that will be accounted, taking into consideration the memory request.

### 4.1.7. The CINECA's database for HPC users and projects

A new portal is under development for users of the HPC systems in CINECA. Its name is UserDB and it can be accessed at the address:

https://userdb.hpc.cineca.it

Here the user interested in using HPC services in CINECA can register and insert all required information for getting access to the HPC platforms.

The same portal allows the users to inspect the status of their account, to manage the list of collaborators of their projects, to access other services like “courses” and project calls at national level.

### 4.2. Accounting on Huygens

On Huygens, the system usage is measured in PNU (Processor Node Uren in Dutch). Unfortunately, the name of this unit is extremely misleading. On Huygens, 1 PNU is equal to using 1 core for 1 hour.

For non-shared jobs the accounting is being done based on the number of nodes and the wall clock time used. The cost of a non-shared job is equal to the number of wall clock hours the job takes times the number of nodes times 32 (being the number of available cores per node).

*Note:* this is independent of the number of (MPI) tasks per node. For example you will be charged for 256 PNU when using 2 nodes (32 cores) for 4 hours: \(4 \times 2 \times 32 = 256\) PNU.

For shared jobs, accounting is being done according to the actual CPU time used, i.e. using 1 CPU hour costs 1 PNU.

The annual amount of PNUs available on the Huygens Power6 system is approximately 23 M.

### 4.2.1. Getting budget and accounting information

The `accuse` command is available to view the daily or monthly usage per user or per account. The default usage is the aggregate usage for a user per month starting from the running year. The man page of `accuse` contains a more detailed description.

The `accinfo` command is available to view the budget and the usage for yourself or the account you belong to. There is also a man page for `accinfo`.

*Note:* the information given by the above mentioned commands is up-to-date for the situation the day before the commands are issued. This is because the accounting information is updated once per day at approximately 24:00 hours.

### 5. Programming Environment / Basic Porting

The programming environment of the IBM Power6 machine contains a choice of compilers for the main scientific languages (Fortran, C and C++), debuggers to help users in finding bugs and errors in the codes, profilers to help in code optimization and numerical libraries for getting performance in a simple way.
Since this is a massively parallel system, it is mandatory to interact with the parallel environment by adopting a parallel paradigm (MPI and/or OpenMP), using parallel compilers, linking parallel libraries, and so on.

The user codes, once written, are usually executed within the batch environment, using the job language of the native scheduler.

## 5.1. Available compilers

On the IBM Power6 the native IBM compilers (XL) are available for Fortran, C and C++. Other compilers (gnu, Portland, ...) can be present: check the available modules, the compiler section in particular, with the module avail command.

### Compiler invocation

C, C++ : xlc, xlC  
   xlc_r, xlC_r (Thread safe)

Fortran: xlf, xlf90  
   xlf_r, xlf90_r (Thread safe)

The “_r” invocations instruct the compiler to link and bind object files to thread-safe components and libraries, and produce thread-safe object code for compiler-created data and procedures. We recommend using the “_r” versions.

You can check the compiler version using the -qversion flag:

```
  xlc -qversion  
  xlf -qversion
```

### Compiling parallel MPI programs (distributed memory parallelism)

The way MPI programs are compiled differs quite a bit between AIX (SP6) and Linux (Huygens). Therefore, the incantations for compiling MPI programs on both systems are presented separately. All wrappers around the compilers start with “mp”. On AIX the suffix “_r” can be added behind the wrapper to indicate thread-safe compilation, while on Linux one can specify the compiler to be used on the command line behind the -compiler flag.

#### AIX (SP6 at CINECA)

C, C++ : mpcc, mpCC  
   mpcc_r, mpCC_r (Thread safe)

Fortran: mpxlf, mpxlf90  
   mpxlf_r, mpxlf90_r (Thread safe)

#### Linux (Huygens at SARA)
C, C++ : mpcc
    mpCC
    mpCC -cpp

Note 1: To use the C++ bindings instead of the ones for C, please add the -cpp flag.

Note 2: On Huygens the default is "_r". If another compiler, like xlC, is desired it can be specified with the option -compiler xlC.

Fortran: mpfort
    mpfort -compiler <fortran_version>

Without the -compiler option the default xlf_r is taken. Another compiler can be chosen by specifying -compiler <fortran_version>, in which <fortran_version> can be xlf, xlf90, xlf90_r, xlf95, xlf95_r, xl2003 or xl2003_r.

Compiling parallel OpenMP programs (shared memory parallelism)

XL compilers support OpenMP, a shared-memory parallelism construct. OpenMP provides an easy method for SMP-style parallelization of discrete, small sections of code, such as a do loop. OpenMP can only be used among the processors of a single node. For use with production scale, multi-node codes, OpenMP threads must be combined with MPI processes.

To compile a C, C++ or Fortran code with OpenMP directives use the thread-safe (_r) version of the compiler and the "-qsmp=omp" directive. It should be noted that the -qsmp=omp option is required for both the compile step and the link step. XL C/C++ and Fortran support OpenMP V3.0.

For compiling OpenMP programs you must use the "_r" version of the compiler and specify the "-qsmp" option:

% xlf_r -qsmp=omp -qnosave -o execname filename.f
% xlf90_r -qsmp=omp -o execname filename.f
% xlc_r -qsmp=omp -o execname filename.c
% xlC_r -qsmp=omp -o execname filename.C

Please note that for the invocations xlf, f77, fort77 and most importantly xlf_r, all local variables are STATIC (also known as SAVE'd variables), and are therefore not thread-safe! Please add the flag -qnosave to generate thread-safe code with these invocations.

Compiling parallel hybrid programs (MPI + OpenMP)

OpenMP and MPI can be freely mixed in your code. You must use a "multiprocessor" and "thread-safe" compiler invocation with the -qsmp=omp option, i.e.:

AIX (SP6 at CINECA)

mpxlf90_r -qsmp=omp
mpcc_r -qsmp=omp
mpCC_r -qsmp=omp

Linux (Huygens at SARA)
mpfort -qsmp=omp -qnosave  
mpfort -compiler xlf90_r -qsmp=omp  
mpcc -qsmp=omp  
mpCC -qsmp=omp (C MPI bindings)  
mpCC -qsmp=omp -cpp (C++ MPI bindings)

**Memory Usage Modes**

The IBM XL compilers can support both 32 and 64-bit addressing through the `-q32/-q64` options. Check the default mode of your specific site, CINECA or SARA. In any case you can change the default mode of the compilers by setting an environment variable:

```bash
export OBJECT_MODE=64
export OBJECT_MODE=32
```

Please note that `OBJECT_MODE=64` is default on Huygens at SARA.

The addressing mode can also be specified directly to the XL compilers with the `-q` option:

```
xlf -q64
xlf -q32
```

Using the 64-bit option causes all pointers to be 64 bits in length and increases the length of “long” datatypes from 32 to 64 bits. It does not change the default size of any other datatype.

The following points should be kept in mind when choosing the memory usage mode:

- You cannot bind object files that were compiled in 32-bit mode with others compiled in 64-bit mode. You must recompile to ensure that all objects are in the same mode.
- Your link options must reflect the type of objects you are linking. If you compiled 64-bit objects, you must also link these objects with the `-q64` option.

It is recommended that users compile in 64-bit mode unless they have compatibility issues within their code.

**Recommended Optimization Options**

The recommended optimization options for the XL compilers are

```
-O3 -qstrict -qarch=auto -qtune=auto
```

A more aggressive optimization is possible, for example using the following options:

```
-O3, -qhot, -qipa=level=2 ...
```

Instead of `-O3` higher levels `-O4` or `-O5` can be tried. But, in some cases, these may cause problems so it is advisable to start with a weaker optimization level. For the full range of compiler and linker options see the appropriate man page. For more information see Section 7.
Recommended Debugging/Profiling Option

Debugging requires the -g option and profiling additionally requires the -p or -pg (recommended) option (also at link time). In both cases it is recommended to use the -qfullpath option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-g</td>
<td>Generates debug information for debugging tools.</td>
</tr>
<tr>
<td>-p</td>
<td>Sets up the object files produced by the compiler for profiling.</td>
</tr>
<tr>
<td>-pg</td>
<td>like -p, but it produces more extensive statistics.</td>
</tr>
<tr>
<td>-qfullpath</td>
<td>Records the full or absolute path names of source and include files in object files compiled with debugging information (when you use the -g option).</td>
</tr>
</tbody>
</table>

Here are some examples:

```
xlc_r -g -qfullpath -O0 hello.c
xlf_r -g -pg -qfullpath -O3 -qstrict -qarch=auto -qtune=auto hello.f90
```

For more details see Section 8.2.

Further Documentation

A collection of IBM documentation files, including programming and reference guides:


5.2. Available (vendor optimized) numerical libraries

The vendor optimized numerical library for this platform are ESSL, PESSL and MASS: optimized numerical libraries from IBM.

These libraries ensure the best performance for your programs even if, being available only on IBM systems, they are not good for portability. If you want to have a better portability among different vendor environments, you should use other libraries, like LAPACK. Since LAPACK is available on multiple platforms, it can guarantee portability. At the same time, since it is compiled on Power6 using the ESSL library, it guarantees also good performance.

Other numerical libraries can be available on the system; you can verify it by using the command:

```
module avail
```

and looking at the library section.

ESSL: Engineering and Scientific Subroutine Library from IBM

Taken from the ESSL website [http://www.ibm.com/systems/p/software/essl]

ESSSL is a collection of subroutines providing a wide range of performance-tuned mathematical functions for many common scientific and engineering applications. The mathematical subroutines are divided into nine computational areas:
• Linear Algebra Subprograms
• Matrix Operations
• Linear Algebraic Equations
• Eigensystem Analysis
• Fourier Transforms, Convolutions, Correlations and Related Computations
• Sorting and Searching
• Interpolation
• Numerical Quadrature
• Random Number Generation

ESSL provides two run-time libraries; both libraries support both 32-bit and 64-bit environment applications:

• The ESSL Serial Library provides thread-safe versions of the ESSL subroutines for use on all processors. You may use this library to develop your own multithreaded applications.

• The ESSL Symmetric Multi-Processing (SMP) Library provides thread-safe versions of the ESSL subroutines for use on all SMP processors. In addition, some of these subroutines are also multi-threaded, meaning, they support the shared memory parallel processing programming model. You do not have to change your existing application programs that call ESSL to take advantage of the increased performance of using the SMP processors; you can simply re-link your existing programs.

Both libraries are designed to provide high levels of performance for numerically intensive computing jobs and both provide mathematically equivalent results. The ESSL subroutines can be called from application programs written in Fortran, C, and C++.

Compiling and linking your code with the ESSL library

To compile your code (C or Fortran) with the ESSL library you only need to add the appropriate “-l” option:

```
xlf_r prog.f -lessl
```

If you are accessing ESSL from a Fortran program, you can compile and link using the commands shown in the table below.

<table>
<thead>
<tr>
<th>ESSL Library</th>
<th>Environment</th>
<th>Fortran Compile Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>32-bit integer, 32-bit pointer</td>
<td>xlf_r -O -q32 -qnosave xyz.f -lessl</td>
</tr>
<tr>
<td></td>
<td>32-bit integer, 64-bit pointer</td>
<td>xlf_r -O -q64 -qnosave xyz.f -lessl</td>
</tr>
<tr>
<td></td>
<td>64-bit integer, 64-bit pointer</td>
<td>xlf_r -O -q64 -qnosave -qint-size=8 xyz.f -lessl6464</td>
</tr>
<tr>
<td>SMP</td>
<td>32-bit integer, 32-bit pointer</td>
<td>xlf_r -O -q32 -qnosave xyz.f -lesslmp -lxslmp</td>
</tr>
<tr>
<td></td>
<td>32-bit integer, 64-bit pointer</td>
<td>xlf_r -O -q64 -qnosave xyz.f -lesslmp -lxslmp</td>
</tr>
</tbody>
</table>
Further Documentation

- Parallel ESSL [https://subtrac.sara.nl/userdoc/wiki/huygens/usage/progavail/pessl]

LAPACK

LAPACK is a library written in Fortran77 that provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision. LAPACK is freely available and is highly portable.

Compiling and linking example

```bash
module load lapack
xlf_r prog.f -L$LAPACK_LIB -llapack -lessl (on SP6 at CINECA)
xlf_r prog.f -L$SARA_LAPACK_LIB -llapack -lessl (on Huygens at SARA)
```

Further Documentation

- The homepage of LAPACK [http://www.netlib.org/lapack/]

5.3. MPI

The distributed memory parallelism can be easily exploited by your C/C++ or Fortran codes by means of the MPI (Message Passing Interface) library. On the IBM Power6 system, a vendor specific version of MPI is available.

Running Parallel MPI programs interactively

MPI programs on the Power6 system run normally under the “Parallel Operating Environment” (POE). Programs compiled with the “mp” compiler wrappers automatically invoke POE.

There are two different ways to run a parallel program:

1. explicit poe

```bash
po e <prog> <program options> <poe options>
```

Example 1:

```bash
po e ./myexe -procs 4
```

Example 2:
2. implicit poe

export <poe variables>
<prog> <program options>

The most useful <poe variables> are:

- **MP_PROCS** - the number of processes to start
- **MP_HOSTFILE** - a file containing the names of the nodes

**Example 1:**

```bash
export MP_PROCS=4
done
./myprog <parameters>
```

**Example 2:**

```bash
export MP_HOSTFILE=$HOME/myhostfile
export MP_PROCS=4
> $MP_HOSTFILE
for i in `seq $MP_PROCS` ; do
    hostname >> $MP_HOSTFILE
done
done
./myprog <parameters>
```

For more information about poe options or poe environmental variables see the man-page of poe.

---

**Running Parallel MPI programs in a batch environment**

When the phrase

```bash
# @ job_type = parallel
```

is present in a job script, the native scheduler of the system (LoadLeveler) generates an environment suitable for parallel jobs. The number of processes is determined from the value of `@node`, `@tasks_per_node` or `@total_tasks` clauses included in the job script.

*Note:* Under control of LoadLeveler poe ignores the **MP_PROCS** environment variable and the **-procs** flag as well as some other explicit settings.

This example is for a job running 128 processes on 4 nodes:

```bash
# @ node = 4
# @ tasks_per_node = 32
```

This example is for a job, running 8 processes.
# @ total_tasks = 8

**Running Parallel MPI programs interactively**

For testing parallel programs, it can be convenient to run them interactively. However, usually a limited wall-clock time and a limited number of tasks are allowed for interactive jobs. It is not allowed to run long lasting (parallel) programs on the interactive nodes; these programs should be run in a job.

### 5.4. OpenMP

For programs compiled with `-qsmp=omp`, the actual number of threads that will be used when executed is defined at the execution time by the LoadLeveler environment (`#@parallel_threads`) or by environmental variables (`OMP_NUM_THREADS`). However, the number of threads should be less or equal to the number of processors on the node. The default number is the number of online processors on the machine.

**Running parallel OpenMP programs**

After the number of threads is set in the environment variable `OMP_NUM_THREADS` the program will run with as many threads as indicated. For example 16 threads:

```bash
export OMP_NUM_THREADS=16
./exename
```

A job script for running a OpenMP programs is as follows:

```bash
# @ job_type = serial
# @ parallel_threads = 8
# @ queue
./exename
```

Note that you do not have to use poe for pure OpenMP codes that are intended to run on a single node.

### 5.5. MPI-OpenMP hybrid parallelism

As indicated before, a hybrid parallel program needs to be built with any of the “mp” compiler wrappers in combination with the “-qsmp=omp” compiler flag.

**Running hybrid parallel programs**

To run the program `hello` on two nodes with 1 MPI process per node and 16 OpenMP threads per node, the following commands need to be executed:

```bash
export OMP_NUM_THREADS=16
poe ./hello -nodes 2 -tasks_per_node 1
```

Here’s a LoadLeveler script to run the code on two nodes with 2 total MPI tasks and 16 OMP threads per node.
5.6. Batch system / job command language

A batch job is the typical way users run production applications on HPC machines. The user submits a batch script to the batch system (LoadLeveler). This script specifies, at the very least, how many nodes and cores the job will use, how long the job will run, and the name of the application to run. The job will advance in the queue until it has reached the top, when it will then be launched on the compute nodes. The output of the job will be available when the job has completed. On request, the user can be notified with an email for every step of his job within the batch environment.

LoadLeveler is the native batch scheduler for the IBM Power machine.

In order to submit a batch job, a LoadLeveler script file must be written with directives for the scheduler followed by the commands to be executed. The script file has then to be submitted using the llsubmit command.

The basic LoadLeveler commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>llsubmit job.cmd</td>
<td>Submit the job described in the &quot;job.cmd&quot; file (see below for the scripts syntax)</td>
</tr>
<tr>
<td>llq</td>
<td>Return information about all the jobs waiting and running in the LoadLeveler queues</td>
</tr>
<tr>
<td>llq -u $USER</td>
<td>Return information about your jobs only in the queues</td>
</tr>
<tr>
<td>llq -l &lt;jobID&gt;</td>
<td>Return detailed information about the specific job</td>
</tr>
<tr>
<td>llq -s &lt;jobID&gt;</td>
<td>Return information about why the job remains queued</td>
</tr>
<tr>
<td>llcancel &lt;jobID&gt;</td>
<td>Cancel a job from the queues, either it is waiting or running</td>
</tr>
<tr>
<td>llstatus</td>
<td>Return information about the status of the machine</td>
</tr>
</tbody>
</table>

In general, users do not need to specify a class (queue) name: LoadLeveler chooses the appropriate class for the user depending on the resources requested. For a list of available classes and resources limits on specific Power6 system see the local documentation.

LoadLeveler script file syntax

In order to submit a batch job, you need to create a file, e.g. job.cmd, which contains

- LoadLeveler directives specifying how much wall-time, processors, resources you wish to allocate to your job.
- shell commands and programs which you wish to execute. The file paths can be absolute or relative to the directory from which you submit the job.

Once created, this file must be submitted with the llsubmit command:
> lnsubmit job.cmd

It will then be queued into a “class”, depending on the directives specified in the script file (usually based on the resources requested).

Note that if your job tries to use more resources (memory or time) than requested, it will be killed by the scheduler. On the other hand, if you require more resources than needed, you risk to pay more or to wait longer before your job is taken into execution. For these reasons, it is important to design your jobs in such a way that they request the right amount of resources.

The first part of the script file contains directives for LoadLeveler specifying the resources needed by the job, in particular:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#@ wall_clock_limit = <a href="">hh:mm:ss</a></td>
<td>Sets the maximum elapsed time for the job. (ex: 24:00:00)</td>
</tr>
<tr>
<td>#@ resources = ConsumableMemory(&lt;N&gt;)</td>
<td>Sets the maximum memory per task. (ex: 350MB, 20GB, ...)</td>
</tr>
<tr>
<td>#@ job_type = parallel</td>
<td>“parallel” informs LoadLeveler to setup a parallel (MPI) environment for running the job;</td>
</tr>
<tr>
<td>#@ job_type = serial</td>
<td>“serial” is used for serial job or parallel (pure OpenMP only).</td>
</tr>
<tr>
<td>#@ total_tasks = &lt;Ntask&gt;</td>
<td>Sets the number of MPI tasks. Do not use for pure OpenMP jobs</td>
</tr>
<tr>
<td>#@ parallel_threads = &lt;Nthreads&gt;</td>
<td>Sets the number of threads (OpenMP) per task. Do not use for pure MPI jobs</td>
</tr>
<tr>
<td>#@ task_affinity = core(&lt;Nthreads&gt;)</td>
<td>Controls the placement of single MPI tasks on one or more physical processor core(s)</td>
</tr>
<tr>
<td>#@ task_affinity = cpu(&lt;Nthreads&gt;)</td>
<td>Controls the placement of single MPI tasks on one or more logical CPUs (two for each core)</td>
</tr>
<tr>
<td>#@ class = &lt;class name&gt;</td>
<td>Asks for a specific class of the scheduler (only if enabled at the local site)</td>
</tr>
<tr>
<td>#@ account_no =</td>
<td>Informs LoadLeveler about the accounting number to be charged for this job (only if enabled at the local site)</td>
</tr>
<tr>
<td>#@ queue</td>
<td>concludes the LoadLeveler directives</td>
</tr>
</tbody>
</table>

Example job scripts

Serial

This is a serial (one task) job, requesting 1 hour of wall-time and 2GB of memory (max 109 GB). The job file, the exec file (myjob) and the input file (input) are in the same directory where the job is submitted from. This directory needs to be a file system mounted on execution nodes:

#!/bin/bash
#
# @ shell = /bin/bash
# @ job_name = myjob
# @ output = myjob.$(jobid).out
# @ error = myjob.$(jobid).err
# @ wall_clock_limit = 1:00:00
# @ job_type = serial
# @ resources = ConsumableMemory(2Gb)
# @ account_no = <your_account_number>
# @ queue

./myjob < input > output

**Pure MPI**

This is a template for a parallel (pure MPI) job on 32 cores, requesting 20 minutes of wall-time and 320 Mb memory/core:

```bash
#!/bin/bash
#
# @ shell = /bin/bash
# @ job_name = myjob
# @ output = myjob.$(jobid).out
# @ error = myjob.$(jobid).err
# @ wall_clock_limit = 0:20:00
# @ resources = ConsumableMemory(320Mb)
# @ job_type = parallel
# @ total_tasks = 32
# @ task_affinity = core(1)
# @ account_no = <your_account_number>
# @ queue

./myMPI < input > output
```

**Pure OpenMP**

Parallel (shared memory) job: 1 task and 32 threads on 32 cpus on 1 node (smp), requesting 20 mins of wall-time and 50 GB memory.

**NOTE:** the job_type must be set to “serial”, since it refers to a single task:

```bash
#!/bin/bash
#
# @ shell = /bin/bash
# @ job_name = myjob
# @ output = myjob.$(jobid).out
# @ error = myjob.$(jobid).err
# @ wall_clock_limit = 0:20:00
# @ resources = ConsumableMemory(50Gb)
# @ job_type = serial
# @ parallel_threads=32
# @ task_affinity = core(32)
# @ account_no = <your_account_number>
# @ queue

echo "Using $OMP_NUM_THREADS threads"
./myOMP < input > output
```

**Hybrid/mixed mode MPI/OpenMP**

Hybrid jobs are MPI job where each MPI task is splitted into several threads.
In this example we have a 16 tasks job (MPI) on 2 nodes and 8 threads per task (OpenMP), requesting 20 minutes of wall-time and 50 GB memory. The 50 Gbytes memory request refers to each of the 16 MPI tasks.

```bash
#!/bin/bash
# @ shell = /bin/bash
# @ job_name = myjob
# @ output = myjob.$(jobid).out
# @ error = myjob.$(jobid).err
# @ wall_clock_limit = 0:20:00
# @ resources = ConsumableMemory(50Gb)
# @ job_type = parallel
# @ node = 2
# @ tasks_per_node = 8
# @ parallel_threads = 4
# @ task_affinity = core(4)
# @ account_no = <your_account_number>
# @ queue

echo "Using 16 MPI tasks, each splitted into $OMP_NUM_THREADS threads"
./myHybrid < input > output
```

**ST and SMT mode**

Power6 cores can schedule two processes (threads) in the same clock cycle, and it is possible to use a single core as two virtual CPUs. This mode of Power6 is called Simultaneous Multi Threading (SMT), and some applications may show better performance if they are run in SMT mode. SMT can be exploited with specific LoadLeveler keywords:

```bash
# @total_tasks=64
# @task_affinity=cpu(1)
```

in this way you can execute up to 64 MPI tasks per node. If you want to run in Single Thread (ST) mode instead, you have to specify:

```bash
# @total_tasks=32
# @task_affinity=core(1)
```

In case you have a hybrid application (MPI+OpenMP) the keywords for the SMT mode become:

```bash
# @total_tasks=8
# @parallel_threads=8
# @task_affinity=cpu(8)
```

where we have assumed that you want to use 8 MPI tasks and 8 OpenMP threads per task (for a total of 64 threads on 1 node).

Whereas the keywords for ST mode become:

```bash
# @total_tasks=8
# @parallel_threads=4
```
where we have assumed that you want to use 8 MPI tasks and 4 OpenMP threads per task (for a total of 32 threads on 1 node).

Note: On Huygens the SMT mode is switched on by default. Switching it off SMT is laborious and is not needed. Whenever a user has the feeling that 64 threads is harming the performance of the application, he or she is advised to lower the total number of tasks to 32, or even lower.

Multi-step jobs

Every scheduler has the possibility to chain multiple jobs. In the following, we describe the details to execute a multi-step job with LoadLeveler.

In a typical multi-step job, the various steps are terminated by the `#@queue` statement. However, you need to tell LoadLeveler not to execute all the steps at the same time (this is the standard behavior), but to wait for the completion of the previous one (except for the first). This is done with the `#@dependency` keyword.

In the following example, two steps are defined, each of them with its own input, output files and executable. The second step will be executed only if the first one completes correctly.

```
# @ input = step00.inp
# @ output = step00.out
# @ executable = program00.exe
# @ step_name = step00
# @ queue
#
# @ input = step01.inp
# @ output = step01.out
# @ executable = program01.exe
# @ step_name = step01
# @ dependency = step00 == 0
#
# queue
```

This is another example where the script to be executed in each step is similar and is reported in the second part of the script after the last `#@queue` statement. In order to differentiate the different steps you can use the $LOADL_STEP_NAME environmental variable. Be careful: you need to specify `#@total_tasks` for each step.

```
#!/bin/bash
# @ job_name = NAMDjob
# @ error = $(job_name).$(jobid).$(stepid).err
# @ output = $(job_name).$(jobid).$(stepid).out
# @ wall_clock_limit = 24:00:00
# @ job_type = parallel
# @ task_affinity=cpu(1)
# @ account_no = (in case you need to specify it)
# @ resources = ConsumableMemory(500Mb)
# @
# @ total_tasks=128
# @ step_name=step_0
# @ queue
# @
# @ total_tasks=128
```
Further documentation


6. Performance Analysis

Performance analysis means looking at the program execution in order to check where bottlenecks or other performance problems may happen. In this context, performance analysis tools are essential to optimize a program's performance. These tools may assist you in understanding what a program is really doing and suggest how the performance could be improved.

Not on all systems the same tools are installed. Please check with the following command which tools are available on your system:

module avail

The tools which will be described along this Section are:

- HPM
- PAPI
- Scalasca
- Extrae
- Vampir

6.1. Available Performance Analysis Tools

In this section we will introduce some of the tools available for IBM Power architectures:

HPM: Hardware Performance Monitor

Description: It is a library developed for performance measurement of applications running on different IBM systems, such as PPC970, Power, BG/L and BG/P.

The library “libhpm” provides a programming interface to start and stop performance counting for an application. The part of the application between the start and stop of performance counting is called an instrumentation section.
This library supports serial and parallel (Message Passing Interface (MPI), threaded, OpenMP and mixed mode) applications, written in Fortran, C, and C++.

Notice that “libhpm” collects information and performs summarization during run time. Thus, there can be considerable overhead if instrumentation sections are inserted inside inner loops.

Depending on the processor architecture, there are several hundreds of registers which can be read by HPM - 6 registers for Power6.

From the user's point of view, it is a useful tool to monitor hardware events of your application. As a consequence, the user will get deep understanding of what the code is doing and how it is performing in the selected hardware.

Functionalities: For IBM systems, HPM can be found under the IBM High Performance Computing Toolkit (IHPCT). IHPCT includes:

- Trace Library: These libraries collect profiling and tracing data for MPI and TurboSHMEM programs.
- SiGMA: The Simulation Guided Memory Analyzer (SiGMA) is a toolkit designed to help programmers to understand the precise memory references in scientific programs that are causing poor utilization of the memory subsystem.
- Dynamic Performance Monitoring Interface for OpenMP: DPOMP is a standard API for performance monitoring of OpenMP applications.
- Peekperf: It is a tool that allows you to map your collected performance data back to the source code. It has a GUI from which you can control instrumentation, execute the application, and visualize and analyze the collected performance data within the same user interface.

PAPI

Description: It specifies an application programming interface (API) for accessing hardware performance counters available on most modern microprocessors. These counters exist as a small set of registers that count events, this is, occurrences of specific signals related to the processor's function.

From the user's point of view, PAPI can be used as an API to read performance counters, or it can be used as part of other Performance Analysis Tools such as Scalasca or Vampir (see below). PAPI reads out the same counters as HPM. HPM is only available on IBM architectures, while PAPI is a standard cross-platform API.

Functionalities

- Basic events: clock cycles, retired instructions
- Instruction execution: instruction decode, issue and execution, data and control speculation, and memory operations.
- Cycle accounting events: stall and execution cycle breakdowns
- Branch events: branch prediction
- Memory Hierarchy: instruction and data caches
- System events: operating system monitors, instruction and data TLBs

Scalasca

Description: It is a performance analysis toolset developed at the Jülich Supercomputing Centre. It has been specifically designed for use on large-scale systems including IBM Blue Gene and Cray XT. It is also suitable for smaller HPC platforms using MPI and/or OpenMP.
It supports an incremental performance analysis process that integrates runtime summaries with in-depth studies of concurrent behavior via event tracing, adopting a strategy of successively refined measurement configurations.

From the user's point of view, Scalasca measures and analyzes the runtime behaviour of your application. Once the application has been executed, you can detect possible bottlenecks, and therefore optimize your application.

**Functionalities:** In order to evaluate the behaviour of parallel programs, Scalasca takes performance measurements at runtime to be analyzed postmortem. The user of Scalasca can choose between two different analysis modes:

- Performance overview on the call-path level via runtime summarization (profiling)
- In-depth study of application behavior via event tracing

**Extrae**

*Description:* Extrae is a dynamic instrumentation package to trace programs compiled and run with the shared memory model (like OpenMP and pthreads), the message passing (MPI) programming model or both programming models (different MPI processes using OpenMP or pthreads within each MPI process).

Extrae generates trace files that can be later visualized with Paraver, which is an extremely configurable visualization and analysis tool.

Paraver reads a trace file (.prv) and generates a plot showing both computation and communication per thread. Threads are represented along the Y-axis and time is on the X-axis.

From the user's point of view, Extrae intercepts MPI/OpenMP sections in your code at execution time. Once the program has finalized, a tracefile is generated, and it can be opened with Paraver. In Paraver, you can follow the execution of your processes/threads and detect possible bottlenecks.

*Functionalities:* It can be used to analyze:

- MPI
- OpenMP
- MPI+OpenMP
- Java
- Hardware counters profile
- OS activity

**Vampir**

*Description:* It is a toolset from Technische Universität Dresden to collect and visualize event traces for MPI applications.

Vampir implements optimized event analysis algorithms and customizable displays which enables a fast and interactive rendering of very complex performance monitoring data. Ultra large data volumes can be analyzed with a parallel version of Vampir which is available on request. Vampir is based on standard X-Windows and works on many Unix systems.

From the user's point of view, Vampir provides a Graphic User Interface which allows users to follow program behaviour, and check for inefficient sections of the code.
**Functionalties:** It provides an easy to use analysis framework which enables developers to quickly display program behavior at any level of detail:

- It converts performance data obtained from a program into different performance views.
- It supports navigation and zooming within these displays.
- It helps to identify inefficient parts of code.
- It leads to more efficient programs.

### 6.2. Hints for interpreting results

In this section we will give an overview on how to interpret the results obtained during performance analysis with the previous tools.

#### HPM

On IBM Power architectures, different event sets have to be selected. When using HPMCOUNT use the `-g` option and when running code instrumented with LIBHPM use the environment variable `HPM_EVENT_SET` to specify the event set.

There are five different sets which are particularly useful. Each set consists of a number of raw counters and a set of derived metrics, which are often easier to judge than the raw counters. The `-l` option of HPMCOUNT provides a complete listing of the raw counters for all sets.

When instrumenting the application code using LIBHPM, it is easy to exclude the overheads from the measured code segments and derived metrics using the wall-clock time will produce meaningful results even when used in short runs.

The user time is only available in those sets which feature the counter `PM_CYC`. `PM_CYC` counts the processor cycles consumed by the application. The user time is calculated by dividing the number of cycles by the processor frequency.

If no event set is specified, the default is Event set 60. It is used for counting cycles, total instruction and various floating point operations. The floating point operations include: divisions, multiply-adds, load, stores and completed floating point instructions.

In the following tables, we can see the available Raw counters and some Derived Metrics. Further information can be found at the IBM website. [http://domino.research.ibm.com/comm/research_projects.nsf/pages/hpc/index.html](http://domino.research.ibm.com/comm/research_projects.nsf/pages/hpc/index.html)

<table>
<thead>
<tr>
<th>Raw counters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM_FPU_FDIV</td>
<td>Number of floating point divisions (hardware)</td>
</tr>
<tr>
<td>PM_FPU_FMA</td>
<td>Number of floating point multiply-additions</td>
</tr>
<tr>
<td>PM_FPU0_FIN</td>
<td>Operations on floating point unit 0 producing a result</td>
</tr>
<tr>
<td>PM_FPU1_FIN</td>
<td>Operations on floating point unit 1 producing a result</td>
</tr>
<tr>
<td>PM_CYC</td>
<td>Number of processor cycles</td>
</tr>
<tr>
<td>PM_FPU_STF</td>
<td>Number of floating point stores (by floating point unit)</td>
</tr>
<tr>
<td>PM_INST_CMPL</td>
<td>Number of completed instructions</td>
</tr>
<tr>
<td>PM_LSU_LDF</td>
<td>Number of floating point loads (by load store unit)</td>
</tr>
</tbody>
</table>

| Derived Metrics | |
|-----------------||

37
Utilization rate | User time divided by wall-clock time in percent
---|---
Load and store operations | Total number of floats loaded and stored in 1000000 operations
Instructions per load/store | Completed instructions divided by result of the previous line
MIPS | Completed instructions divided by wall-clock time in 1000000/s
Instructions per cycle | Completed instructions divided by number of cycles
HW Float point instructions per Cycle | Sum of result-producing operations on both FPUs divided by the number of cycles
Floating point instructions + FMAs (flips) | Sum of result-producing operations on both FPUs plus the number of executed floating point multiply-additions minus the stores by the FPUs. Each FMA contains 2 calculations in a single instruction. Store instructions contain no calculation.
Flip rate (flips/WCT) | Result of the previous line divided by the wall-clock time in 1000000/sec. Includes many overheads for HPMCOUNT, more useful for LIBHPM
Flips/user time | As above but with user time instead of wall-clock time
FMA percentage | Twice the number of floating point multiply adds divided by the flips in percent
Computation intensity | Number of flips divided by the total number of floats loaded and stored

Figure 7 below shows Peekperf's main window, where much of this information can be graphically analyzed.

**Figure 7. Peekperf's GUI**
PAPI

It provides simple high level functions fully supported on both C and Fortran. Some of the most common functions are:

- PAPI_num_counters - get the number of hardware counters available on the system
- PAPI_flips - simplified call to get Mflips/s (floating point instruction rate), real and processor time
- PAPI_flops - simplified call to get Mflops/s (floating point operation rate), real and processor time
- PAPI_ipc - gets instructions per cycle, real and processor time
- PAPI_accum_counters - add current counts to array and reset counters
- PAPI_read_counters - copy current counts to array and reset counters
- PAPI_start_counters - start counting hardware events
- PAPI_stop_counters - stop counters and return current counts

If any of the above functions fails, it will return an error code. The values that are greater than or equal to zero indicate success and those that are less than zero indicate failure. The complete list can be checked in PAPI's User Guide. [http://icl.cs.utk.edu/papi/]

Scalasca

The results of the automatic analysis are stored in one or more reports in the experiment archive. These reports can be processed and examined using the “scalasca -examine” command on the experiment archive:

scalasca -examine epik_<title>

Post-processing is done the first time that an archive is examined, before launching the CUBE3 report viewer. If the scalasca -examine command is executed on an already processed experiment archive, or with a CUBE file specified as argument, the viewer is launched immediately.

A short textual score report can be obtained without launching the viewer:

scalasca -examine -s epik_<title>

This score report comes from the cube3_score utility and provides a breakdown of the different types of region included in the measurement and their estimated associated trace buffer costs, aggregate trace size (total_tbc) and largest process trace size (max_tbc), which can be used to specify an appropriate ELG_BUFFER_SIZE for a subsequent trace measurement.

The CUBE3 viewer can also be used on an experiment archive or CUBE file:

cube3 epik_<title>
cube3 <file>.cube

However, we must keep in mind that no post-processing is performed in this case, so that only a subset of Scalasca analyses and metrics may be shown.

Figure 8 shows an explanatory image of Scalasca's GUI, and further information can be found at the website of Scalasca. [http://www.scalasca.org]
Extrae

Figure 9 shows an example of a Paraver trace. Specifically, it is a zoom in on a WRF iteration, executed with 32 processes (MPITrace). The blue area represents computation, orange and red areas represent communication and the yellow lines represent the source and destination of MPI communications. A user guide can be found on the BSC website: http://www.bsc.es/sites/default/files/public/computer_science/performance_tools/extrae-user-guide.pdf

Vampir

It provides a large set of different timeline and summarizing chart representations of performance events, which allow to pinpoint the real cause of performance problems. Most displays have context-sensitive menus which provide additional information and customization options.
There are two types of displays:

- **Timeline displays**: for process groups or single processes show application activities and communication along a time axis which can be zoomed and scrolled.

- **Summary displays**: provide quantitative results for arbitrary portions of the timelines. References to source code lines are available with compiler support.

Figure 10 below shows an explanatory snapshot of Vampir. Further information can be found at the Vampir website. [http://www.vampir.eu/]

**Figure 10. Vampir's GUI**

![Vampir's GUI]

7. Tuning Applications

In this Section several ways to tune applications will be presented, including compiler options and advanced options for parallel usage and programming. Besides tuning these options, affinity is impor-
tant, because the key to performance often lies herein. In order to better understand the affinity topics in this Section, we briefly summarize the relevant architectural features of a p575 (Power6 IH) node.

Each DCM (Dual Chip Module) has a (dual-core) Power6 processor chip and an L3 cache chip. A “quad” constitutes 4 DCMs. For historical reasons, a “quad” is most commonly referred to as an MCM (Multi Chip Module). All DCMs in an MCM are fully connected and share their memory. A full node, in turn, constitutes 4 MCMs. The 4 MCMs in a node again are fully connected.

Each core is capable of two-way simultaneous multithreading (SMT). A node has 32 cores and can run 64 hardware threads. Both in LoadLeveler and in the Linux kernel, the hardware threads are called cpus: core0 = cpu0 + cpu1, core1 = cpu2 + cpu3, ..., core31 = cpu62 + cpu63. These of course are virtual or logical cpus, however, the word “virtual” or “logical” is commonly dropped.

Processor affinity takes advantage of the fact that some remnants of a process may remain in one processor's state (in particular, in its cache) from the last time the process ran. Scheduling it to run on the same processor the next time could result in the process running more efficiently by reducing performance-degrading situations such as cache misses. This minimizes thread migration and context-switching cost among cores. It also improves the data locality and reduces the cache-coherency traffic among the cores (or processors).

Although the MCMs are fully connected, MCMs are NUMA nodes (NUMA domains): intra node memory access to/from another MCM is slower. MCM0 = cpu0, ..., cpu15; MCM1 = cpu16, ..., cpu31; ..., MCM3 = cpu48, ..., cpu63.

Finally, each Power 575 compute node supports up to two 4 port IBM Infiniband Host Channel Adapters (HCA). Each HCA has dual 4x DDR ports to connect into the QLogic fabric.

Note: the current version of Section 7 assumes the Linux operating system, IBM Parallel Environment (a.o. for MPI), IBM XL compilers (most relevant for OpenMP) and LoadLeveler (because of the tight integration with IBM PE and the OpenMP implementation of the IBM XL compilers).

### 7.1. Advanced / aggressive compiler flags

This section will cover some compiler flags needed for tuning an application.

Before going into IBM (hardware) specific compiler options, let's first have a look at the general optimizations. IBM specific options will be explained in the next section.

The general optimizations include:

- Function inlining
- Dead code elimination
- Array dimension padding
- Structure splitting and field reordering

These optimizations can be performed with the flag -On, where n is the level of optimization. In the end, these optimizations lead to a smaller and more efficient code.

In the following tables you can see the different levels of optimization for IBM XL and GCC:

<table>
<thead>
<tr>
<th>IBM XL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O0</td>
<td>Performs only quick local optimizations such as constant folding and elimination of local common subexpressions.</td>
</tr>
<tr>
<td>-O2</td>
<td>Performs optimizations that the compiler developers considered the best combination for compilation speed and runtime performance. The opti-</td>
</tr>
</tbody>
</table>
Perform some memory and compile-time intensive optimizations in addition to those executed with -O2. The -O3 specific optimizations have the potential to alter the semantics of a program. The compiler guards against these optimizations at -O2 and the option -qstrict is provided at -O3 to turn off these aggressive optimizations. Specifying -O3 implies -qhot=level=0.

- **-O3**
  - This option is the same as -O3, but also:
    - sets the -arch and -tune options to the architecture of the compiling machine.
    - sets the -qcache option most appropriate to the characteristics of the compiling machine.
    - sets the -qipa option.
    - sets the -qhot option to level=1.

- **-O4**
  - This option is the same as -O3, but also:
    - sets the -arch and -tune options to the architecture of the compiling machine.
    - sets the -qcache option most appropriate to the characteristics of the compiling machine.
    - sets the -qipa option.
    - sets the -qhot option to level=1.

- **-O5**
  - Equivalent to -O4 -qipa=level=2

---

**IBM specific processor features include Altivec, a floating point and integer SIMD instruction set, which allows to exploit the SIMD and parallel processing capabilities of the PowerPC processor. It can be switched on in the XL compiler suite and in GCC.**

### IBM XL
- `-qenablevmx` -qaltivec
- `-maltivec` -mabi=altivec

### GCC
- `-march -mtune` -march -mtune
- `-mabi -mabi` -mabi

There are also some aggressive compiler flags, such as:

<table>
<thead>
<tr>
<th>IBM XL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-qhot</code></td>
<td>Instructs the compiler to perform high-order loop analysis and transformations during optimization.</td>
</tr>
<tr>
<td><code>-qipa</code></td>
<td>Interprocedural analysis (IPA) enables the compiler to optimize across different files (whole-program analysis), and can result in significant performance improvements.</td>
</tr>
</tbody>
</table>
Inline functions are expanded in any context in which it is called. This avoids the normal performance overhead associated with the branching for a function call, and it allows functions to be included in basic blocks.

The option `-qessl` includes the hardware optimized library ESSL (see Section 5). Test examples run by SARA personnel have indicated that `-qessl` in combination with `-O3 -qhot -qstrict` increased the speed of test program with a factor of 4.

In addition, there are some specific flags available for IBM Power architectures, such as:

<table>
<thead>
<tr>
<th>IBM XL</th>
<th>GCC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-qarch=&lt;suboption&gt;</code></td>
<td><code>-mcpu=&lt;suboption&gt;</code></td>
<td>Specifies the architecture system for which the executable is optimized.</td>
</tr>
<tr>
<td><code>-qtune=&lt;suboption&gt;</code></td>
<td><code>-mtune=&lt;suboption&gt;</code></td>
<td>Specifies additional performance optimizations.</td>
</tr>
</tbody>
</table>

### 7.2. Advanced MPI usage

#### 7.2.1. Tuning / environment variables

The text for this Section has mostly been taken from “IBM Tuning Guide For High Performance Computing Applications On IBM POWER6” by Luigi Brochard et al. from 2009.

For better performance, the following environment variables can be used on top of the LoadLeveler or PE affinity settings (see Section 7.2.3 and Section 7.3.1). It should be noted that these settings have been thoroughly tested on IBM Power6 with AIX. Therefore some of the environment variables may not be beneficial for the performance on Linux (Huygens at SARA).

```bash
export MP_SYNC_QP=YES
export MP_RFIFO_SIZE=16777216
export MP_SHM_ATTACH_THRESH=500000
export MP_EUIDEVELOP=min
export MP_USE_BULK_XFER=yes
```

Large computer systems like Huygens (SARA) and SP6 (CINECA) make use of remote direct memory access (RDMA). RDMA supports zero-copy networking by enabling the network adapter to transfer data directly to or from application memory, eliminating the need to copy data between application memory and the data buffers in the operating system. Such transfers require no work to be done by CPUs, caches, or context switches, and transfers continue in parallel with other system operations. When an application performs an RDMA Read or Write request, the application data is delivered directly to the network, reducing latency and enabling fast message transfer. For Power6 system the following RDMA specific variables can be set:

```bash
export MP_RDMA_MTU=4K
export MP_BULK_MIN_MSG_SIZE=64k
export MP_RC_MAX_QP=8192
```

```bash
MP_SYNC_QP=YES
```
On IB systems, QP information must be exchanged between two tasks before messages can be sent. For FIFO/UD traffic, it's recommended that the exchange of this information be done in MPI_Init() by setting MP_SYNC_QP=YES. By forcing this QP exchange to occur up-front, some variation in communication performance can be eliminated.

\textbf{MP_RFIFO\_SIZE=16777216}

The default size of the receive FIFO used by each MPI task is 4MB. Larger jobs are recommended to use the maximum size receive FIFO by setting MP_RFIFO_SIZE=16777216.

\textbf{MP_SHM\_ATTACH\_THRESH=500000}

LAPI (Low-level Application Programming Interface is a reliable and efficient communication library implemented on IBM systems) has two modes of sending shared memory messages. For smaller messages slot mode is used to copy messages from one MPI task to another. For larger messages, it's possible to map the shared memory segment of one task to another, thereby saving a copy at the cost of some overhead of attaching. The MP_SHM\_ATTACH\_THRESH variable defines the minimum size message for which attach mode is used. Depending on the type of job different cross-over points may provide optimal performance, but 500000 is often a reasonable starting point when tuning this value. The default depends on how many tasks are running on the node.

\textbf{MP_EUIDEVELOP=min}

The MPI layer will perform checking on the correctness of parameters according to the value of MP_EUIDEVELOP. As these checks can have a significant impact on latency, when not developing applications it is recommended that MP_EUIDEVELOP=min be set to minimize the checking done at the message passing interface layer.

\textbf{MP_USE\_BULK\_XFER=yes}

Setting MP_USE\_BULK\_XFER=yes will enable RDMA. On IB systems using RDMA will generally give better performance at lower task counts when forcing RDMA QP information to be exchanged in MPI_Init() (via setting LAPI_DEBUG\_RC\_INIT\_SETUP=yes). When the RDMA QP information is not exchanged in MPI_Init(), there can be delays due to QP information exchange until all tasks have synced-up.

The benefit of RDMA depends on the application and its use of buffers. For example, applications that tend to re-use the same address space for sending and receiving data will do best, as they avoid the overhead of repeatedly registering new areas of memory for RDMA.

RDMA mode will use more memory than pure FIFO mode. Note that this can be curtailed by setting MP_RC\_MAX\_QP to limit the number of RDMA QPs that are created.

\textbf{MP_BULK\_MIN\_MSG\_SIZE=64k}

The minimum size message used for RDMA is defined to be the maximum of MP_BULK\_MIN\_MSG\_SIZE and MP_EAGER\_LIMIT. So if MP_EAGER\_LIMIT is defined to be higher than MP_BULK\_MIN\_MSG\_SIZE, the smallest RDMA message will be limited by the eager limit.

\textbf{MP_RC\_MAX\_QP=8192}

This variable defines the maximum number of RDMA QPs that will be opened by a given MPI task. Depending on the size of the job and the number of tasks per node, it may be desirable to limit the number of QPs used for RDMA. By default, when the limit of RDMA QPs is reached, future connections will all use FIFO/UD mode for message passing.

\textbf{MP_EAGER\_LIMIT=65536}

The larger this number, the more (short) MPI messages are sent without immediate response from the receiver (they are outstanding). In this way message are bundled in larger chunks, which can be
advantageous. Therefore, a test with the maximum of 65536 is not a bad idea. While debugging your code for validity of MPI calls, you can set this limit to zero.

7.2.2. Mapping tasks on node topology

Since all nodes are fully connected, all nodes are created equal and performance-wise there is no need to map tasks to specific nodes.

7.2.3. Task affinity

The key to MPI performance often lies in processor affinity of MPI tasks. For large core counts (1000 or more) we frequently observe a performance gain of a factor 2 - 4.

The IBM Parallel Environment 5.1 (PE) has extended its affinity support with additional rsets for cases that correspond to the latest LoadLeveler affinity support. PE's support of affinity domains is controlled by the MP_TASK_AFFINITY environment variable. It provides roughly equivalent functionality to LoadLeveler's JCF keywords, except PE can provide only one core or one logical CPU to each MPI task.

The MP_TASK_AFFINITY settings are ignored for batch jobs. To give a batch job affinity, the appropriate LoadLeveler JCF keywords should be used.

For interactive jobs, POE will query the LoadLeveler API to determine if the resource manager provides affinity support at the requested level. When running a version of LoadLeveler with full affinity support, PE will simply convert the requested MP_TASK_AFFINITY environment variable to the appropriate JCF settings as follows:

Table 2. POE Environment Variable Settings and corresponding LL JCF lines generated.

<table>
<thead>
<tr>
<th>POE Environment Variable Setting:</th>
<th>LoadLeveler JCF lines that will Be Generated:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_TASK_AFFINITY = MCM</td>
<td>#@rset = rset_mcm_affinity</td>
</tr>
<tr>
<td>MP_TASK_AFFINITY = CORE</td>
<td>#@rset = rset_mcm_affinity</td>
</tr>
<tr>
<td></td>
<td>#@task_affinity = core(1)</td>
</tr>
<tr>
<td>MP_TASK_AFFINITY = CPU</td>
<td>#@rset = rset_mcm_affinity</td>
</tr>
<tr>
<td></td>
<td>#@task_affinity = cpu(1)</td>
</tr>
<tr>
<td>MP_TASK_AFFINITY = SNI</td>
<td>#@rset = rset_mcm_affinity</td>
</tr>
<tr>
<td></td>
<td>#@mcm_affinity_options = MCM_SNI_PREF MCM_DISTRIBUTE</td>
</tr>
<tr>
<td>MP_TASK_AFFINITY = list</td>
<td>Ignored</td>
</tr>
</tbody>
</table>

The MP_TASK_AFFINITY=list approach defines a list of MCM rset numbers that have already been created.

The MP_TASK_AFFINITY=SNI setting is probably not needed for the 575.

The simplest way to achieve processor affinity for MPI tasks is using the task_affinity LoadLeveler keyword:

```bash
# @ tasks_per_node = 4
# @ parallel_threads = 8
#
# @ rset = rset_mcm_affinity
# @ mcm_affinity_options = mcm_distribute mcm_mem_pref mcm_sni_none
```
Best Practice Guide - IBM Power

```python
# @task_affinity = core(x)

or

# @ tasks_per_node = 4
# @ parallel_threads = 8
#
# @ rset = rset_mcm_affinity
# @ mcm_affinity_options = mcm_distribute mcm_mem_pref mcm_sni_none
# @task_affinity = cpu(x)
```

The `core` keyword specifies that each task in the job is bound to run on as many processor cores as specified by x. The `cpu` keyword indicates that each task of the job is constrained to run on as many cpus as defined by x. For Huygens at SARA we created a custom submit filter that amongst others add a `task_affinity = core or task_affinity = cpu` keyword for the most common cases: `tasks_per_node = 32 and tasks_per_node = 64`, respectively. If you want more control over the task placement, you can use either wrapper scripts using the `taskset(1)` or `numactl(1)` Linux commands, or the launch tool. The launch tool is the simplest to use.

The target cpus for the MPI tasks are selected through the environment variables `TARGET_CPU_LIST`. The value “-1” denotes all cpus otherwise use a list like “0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62”.

To obtain the affinitization provided by `launch` simply replace program invocations such as

```bash
...  # @ queue
  ./MpiProgram ...
poe ./MpiProgram ...
```

by

```bash
...  # @ skip_rset
  # @ queue
  module load launch
  poe launch ./MpiProgram ...
```

Please note that the `skip_rset` LoadLeveler keyword is a SARA extension that prevents automatic affinitization by SARAs `llsubmitfilter`.

### 7.2.4. Adapter affinity

On Power6 InfiniBand systems, the recommended value for `adapter_affinity` in the `mcm_affinity_options` is `mcm_sni_none`.

### 7.3. Advanced OpenMP usage

#### 7.3.1. Tuning, Environment variables and Thread affinity

There appears to be no LoadLeveler support for pure OpenMP affinity. The simplest way to achieve processor affinity for OpenMP threads is using environment variables. Furthermore, the OpenMP
standard does not prescribe environment variables for this. Consequently, every compiler vendor uses its own environment variables. For the IBM XL compilers you can use XLSMOPTS startproc and stride sub options:

```
export OMP_NUM_THREADS=32
export XLSMOPTS=startproc=0:stride=2
```

Processor numbers refer to virtual or logical CPUs 0..63. Therefore, the specification startproc=0:stride=2 binds the OpenMP threads to the 32 physical cores.


### 7.4. Hybrid programming

#### 7.4.1. Optimal tasks / threads strategy

As already indicated, the memory and CPUs inside the nodes are located on Multi Chip Modules (MCMs). Each node contains 4 of these MCMs. For a hybrid run, the user is strongly advised to use a multiple of 4 MPI tasks per node: each task uses its own MCM, or in case of a multiple of 4, an equal number of tasks runs on each MCM. This strategy is illustrated with an example in the next section.

#### 7.4.2. Task and thread affinity

The simplest way to achieve processor affinity for hybrid MPI tasks / OpenMP threads is using the parallel_threads LoadLeveler keyword:

```
# @ tasks_per_node = 4
# @ parallel_threads = 8
#
# @ rset = rset_mcm_affinity
# @ mcm_affinity_options = mcm_distribute mcm_mem_pref mcm_sni_none
# @ task_affinity = core(8)
```

The parallel_threads keyword assigns separate cpus to individual threads of an OpenMP task. The cpus assigned to threads are selected from the set of cpus or cores assigned to the task. The cpus for individual OpenMP threads of the tasks are selected based on the number of parallel threads in each task and set of cpus or cores assigned to the task.

If you want more control over the thread placement, you can use the hybrid_launch tool.

The target cpus for the MPI tasks / OpenMP threads are selected through the environment variables TARGET_CPU_LIST. The value “-1” denotes all cpus otherwise use a list like “0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62”.

To obtain the affinitization provided by hybrid_launch simply replace program invocations such as

```
...
# @ queue
./HybridProgram ...
```
poe ./HybridProgram ...

by

...
# @ skip_rset
# @ queue
module load launch
poe hybrid_launch ./HybridProgram ...

Please note that the skip_rset LoadLeveler keyword is a SARA extension that prevents automatic affinitization by SARA's llsubmitfilter.

7.5. Memory optimization

7.5.1. Memory affinity (MPI/OpenMP/Hybrid)

In the current LoadLeveler implementation (on Linux), memory affinitization (binding) only appears to work for pure MPI program. Currently, every allowed value of the memory_affinity option in the mcm_affinity_options seem to has the same effect as mcm_mem_req. The following sequence of LoadLeveler keywords results in every MPI tasks being bound to core and the corresponding memory to be bound to the MCM that contains that core.

# @ rset = rset_mcm_affinity
# @ mcm_affinity_options = mcm_distribute mcm_mem_req mcm_sni_none
# @ task_affinity = core

8. Debugging

The aim of this section is to give an overview of the current debugger tools available in common European HPC facilities.

The tools which will be described along this Section are:

- TotalView
- DDT
- Marmot
- Valgrind

8.1. Available debuggers

There are several debuggers used for HPC applications. In this section we will introduce some of them. Please be aware that not all tools may be installed on all Power systems. To check whether a certain debugging tool is installed please type:

module avail

Please load the correct module with the module load command.
TotalView

It is a proprietary debugger for C/C++ and Fortran code that runs on Unix-like Operating Systems such as Linux and Mac OS X systems.

It allows process control down to the single thread, the ability to look at data for a single thread or all threads at the same time, and the ability to synchronize threads through breakpoints. TotalView integrates memory leak detection and other heap memory debugging features. Data analysis features help find anomalies and problems in the target program's data, and the combination of visualization and evaluation points lets the user watch data change as the program executes. TotalView includes the ability to test fixes while debugging. It supports parallel programming including Message Passing Interface (MPI), Unified Parallel C (UPC) and OpenMP. It can be extended to support debugging CUDA. It also has an optional add-on called ReplayEngine that can be used to perform reverse debugging (stepping backwards to look at older values of variables).

From the user's point of view, TotalView allows analyzing the thread's behaviour of your application, as well as detecting memory leaks.

Further information can be found at the Rogue Wave website [http://www.roguewave.com/products/totalview-family/totalview.aspx].

DDT

It is a source-level debugger for scalar, multi-threaded and large-scale parallel C, C++ and Fortran codes. It provides complete control over the execution of a job and allows the user to examine in detail the state of every aspect of the processes and threads within it.

Control of processes is aggregated using groups of processes which can be run, single stepped, or stopped together. The user can also set breakpoints at points in the code which will cause a process to pause when reaching it. When a program is paused, the source code is highlighted showing which lines have threads on them and, by simply hovering the mouse, the actual threads present are identified. By selecting an individual process, its data can be interrogated - for example to find the local variables and their values, or to evaluate specific expressions.

From the user's point of view, DDT is a debugging tool for parallel applications (MPI, OpenMP, CUDA, threads). DDT is able to debug code in workstations, GPUs or clusters.

Further information can be found at Allinea website [http://www.allinea.com/products/ddt].

Marmot

It is a library that uses the so-called PMPI profiling interface to intercept MPI calls and analyse them during runtime. It has to be linked to the application in addition to the underlying MPI implementation, not requiring any modification of the application's source code nor of the MPI library. The tool checks if the MPI API is used correctly and checks for errors frequently made in MPI applications, e.g. deadlocks, the correct construction and destruction of resources, etc. It also issues warnings for non-portable behaviour, e.g. using tags outside the range guaranteed by the MPI-standard. The output of the tool is available in different formats, e.g. as text log file or html/xml, which can be displayed and analysed using a graphical interface. Marmot is intended to be a portable tool that has been tested on many different platforms and with many different MPI implementations.

Marmot supports the complete MPI-1.2 standard for C and Fortran applications and is being extended to also cover MPI-2 functionality.

From the user's point of view, MARMOT surveys the MPI calls and checks the correct usage of these calls and their arguments at execution time.

Further information can be found at the HLRS website [http://www.hlrs.de/organization/av/amt/projects/marmot].
Valgrind

It is a programming tool for memory debugging, memory leak detection, and profiling. Its main application is memcheck. It can detect the following issues:

- Use of uninitialised memory
- Reading/writing memory after it has been free’d
- Reading/writing off the end of malloc’d blocks
- Reading/writing inappropriate areas on the stack
- Memory leaks -- where pointers to malloc’d blocks are lost forever
- Mismatched use of malloc/new/new [] vs free/delete/delete []
- Overlapping src and dst pointers in memcpy() and related functions
- Some misuses of the POSIX pthreads API

From the user's point of view, Valgrind works directly with existing executables, without recompiling or modifying the application. Debugging information is read from the executable and associated libraries, so that error messages can be located in the source code.

Further information can be found at the Valgrind website [http://valgrind.org].

8.2. Compiler flags

In this section we will talk about the compiler flags needed for profiling and/or debugging applications. The table below show common flags for most compilers:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-g</td>
<td>Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF 2). GDB can work with this debugging information.</td>
</tr>
<tr>
<td>-p</td>
<td>Generate extra code to write profile information suitable for the analysis program prof. You must use this option when compiling the source files you want data about, and you must also use it when linking.</td>
</tr>
<tr>
<td>-pg</td>
<td>Generate extra code to write profile information suitable for the analysis program gprof. You must use this option when compiling the source files you want data about, and you must also use it when linking.</td>
</tr>
</tbody>
</table>

There are some specific flags for IBM XL Compilers, such as:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qfullpath</td>
<td>Records the full, or absolute, path names of source and include files in object files compiled with debugging information (-g option).</td>
</tr>
<tr>
<td>-qlinedebug</td>
<td>Generates line number and source file name information for the debugger.</td>
</tr>
<tr>
<td>-qsaveopt</td>
<td>Saves the command-line options used for compiling a source file in the corresponding object file.</td>
</tr>
<tr>
<td>-qcheck</td>
<td>Checks each reference to an array element, array section, or character substring for correctness.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-qflttrap</td>
<td>This option uses trap operations to detect floating-point exceptions and generates SIGTRAP signals when exceptions occur. Compile the program with the -qflttrap option and some combination of suboptions that includes enable.</td>
</tr>
<tr>
<td>-ggdb</td>
<td>Produce debugging information for use by GDB. This means to use the most expressive format available (DWARF 2, stabs, or the native format if neither of those are supported), including GDB extensions if at all possible.</td>
</tr>
<tr>
<td>-Q</td>
<td>Makes the compiler print out each function name as it is compiled, and print some statistics about each pass when it finishes.</td>
</tr>
<tr>
<td>-ftime-report</td>
<td>Makes the compiler print some statistics about the time consumed by each pass when it finishes.</td>
</tr>
<tr>
<td>-fmem-report</td>
<td>Makes the compiler print some statistics about permanent memory allocation when it finishes.</td>
</tr>
<tr>
<td>-save-temps</td>
<td>Store the usual “temporary” intermediate files permanently; place them in the current directory and name them based on the source file.</td>
</tr>
</tbody>
</table>