

# User's Guide to the HPCx Service

## Version 2.02

January 14, 2010

### Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	The IBM eServer 575 system Code of Conduct . . . . .	6
<b>2</b>	<b>Architecture Overview</b>	<b>6</b>
<b>3</b>	<b>Getting started</b>	<b>7</b>
3.1	Web Administration Account . . . . .	7
3.2	Machine accounts . . . . .	7
3.3	Logging in . . . . .	8
3.4	Password changes . . . . .	8
3.5	System maintenance . . . . .	8
3.6	Shells, etc. . . . .	9
3.7	Transferring files . . . . .	9
3.7.1	Transferring large amounts of data to/from HPCx: bbFTP . . . . .	9
3.8	Editors . . . . .	9
<b>4</b>	<b>User Resource Management</b>	<b>10</b>
4.1	What resources do I have - time? . . . . .	10
4.2	What resources do I have - storage? . . . . .	10
4.2.1	Diskspace . . . . .	10
4.2.2	Tapestorage . . . . .	11
4.3	The HPCx Administration Web Site . . . . .	11
4.4	Charging . . . . .	11
<b>5</b>	<b>Compilation</b>	<b>12</b>
5.1	General remarks on compilation . . . . .	12
5.1.1	Use of /tmp by the compilers . . . . .	12
5.2	Compiling sequential code . . . . .	12
5.3	Compiling MPI code . . . . .	13
5.3.1	Fortran . . . . .	13
5.3.2	C . . . . .	13
5.3.3	C++ . . . . .	13
5.4	Compiling OpenMP code . . . . .	13
5.5	Other useful compiler switches . . . . .	13
5.6	Example Makefiles . . . . .	14
5.6.1	MPI fixed format Fortran code . . . . .	14

5.6.2	MPI Fortran 90 code	15
5.6.3	MPI C code	16
5.6.4	OpenMP fixed format Fortran code	17
5.6.5	OpenMP Fortran 90 code	18
5.6.6	OpenMP C code	18
5.6.7	Mixed MPI/OpenMP fixed format Fortran code	19
5.6.8	Mixed MPI/OpenMP Fortran 90 code	20
5.6.9	Mixed MPI/OpenMP C code	21
<b>6</b>	<b>File Management</b>	<b>22</b>
6.1	Use of /tmp	22
6.1.1	Use of /tmp by the compilers	22
6.1.2	Use of /tmp - gotchas for Fortran programmers	22
6.2	Other file management issues	23
<b>7</b>	<b>Porting Codes</b>	<b>23</b>
7.1	Tips for FORTRAN programmers	23
7.1.1	Useful compiler options	23
7.2	Timers	23
7.2.1	Fortran codes	23
7.2.2	C/C++ codes	24
7.2.3	Java codes	25
7.3	Default sizes	26
7.3.1	FORTRAN codes	26
7.3.2	C/C++ codes	26
7.4	MPI Datatypes	27
7.4.1	Fortran	27
7.4.2	C	27
<b>8</b>	<b>Batch Processing</b>	<b>27</b>
8.1	Batch System Overview	28
8.2	Creating a job command file	28
8.2.1	MPI job command files	28
8.2.2	MPI job with less than 16 tasks per LPAR	31
8.2.3	OpenMP job command files	31
8.2.4	Mixed MPI/OpenMP job command files	32
8.2.5	Enabling core files	32
8.2.6	Enabling Simultaneous Multi-Threading (SMT)	32
8.2.7	Serial job command files	33
8.2.8	Chaining serial and parallel jobs	33
8.3	Batch job allocation and queueing policy	34
8.4	Job submission - the llsubmit command	34
8.5	Job status information - the llq command	35
8.6	Cancelling a Job - the llcancel command	35
8.7	Machine Status information - the llstat command	35
8.8	Notification of job completion	35
8.9	Using the xloadl GUI	36
<b>9</b>	<b>Interactive Execution</b>	<b>36</b>
9.1	Purpose of the interactive queue	36
9.2	Using the interactive queue	36
9.2.1	Creating a LoadLeveler job script for interactive use	36
9.2.2	Setting up environment variables	37
9.2.3	Running a program interactively	37

9.2.4	LoadLeveler attributes for interactive processes . . . . .	37
9.3	Gotchas! . . . . .	38
<b>10</b>	<b>Tools</b>	<b>38</b>
10.1	Running The Parallel Debugger pdbx . . . . .	38
10.1.1	Loading the parallel program . . . . .	38
10.1.2	Tracing the program instances in the debugger . . . . .	39
10.2	Totalview . . . . .	39
10.3	VAMPIR . . . . .	40
10.3.1	Tracing your Parallel Application with VampirTrace . . . . .	40
10.3.2	Viewing Your Tracefiles with Vampir . . . . .	40
10.3.3	Documentation . . . . .	41
10.4	Paraver . . . . .	41
10.5	gprof and xprofiler . . . . .	41
10.6	Hardware Performance Monitor (HPM) Toolkit . . . . .	42
10.6.1	The hpmcount utility . . . . .	42
10.6.2	The libhpm library and the hpmviz utility . . . . .	43
10.7	MPI Trace Tools . . . . .	44
10.8	KOJAK . . . . .	44
10.9	ParaView . . . . .	45
10.9.1	Launching the Client . . . . .	45
10.9.2	Launching the Server . . . . .	45
10.9.3	Python scripting . . . . .	46
10.10	TAU . . . . .	46
10.10.1	MPI Profiling . . . . .	46
10.10.2	Instrumentation . . . . .	47
10.11	Subversion (SVN) . . . . .	47
10.12	NAMD & VMD - Interactive Molecular Dynamics . . . . .	47
<b>11</b>	<b>Libraries</b>	<b>48</b>
11.1	IBM Provided Libraries . . . . .	48
11.1.1	ESSL . . . . .	48
11.1.2	Parallel ESSL . . . . .	49
11.1.3	BLACS . . . . .	49
11.1.4	MASS . . . . .	50
11.2	Other Libraries . . . . .	50
11.2.1	LAPACK . . . . .	50
11.2.2	ScaLAPACK . . . . .	50
11.2.3	PLAPACK . . . . .	51
11.2.4	FFTW . . . . .	51
11.2.5	HSL . . . . .	52
11.2.6	Parallel HDF5 . . . . .	52
11.2.7	MPI splitting library . . . . .	53
11.3	Libraries and porting to the IBM . . . . .	55
<b>12</b>	<b>Optimisation</b>	<b>55</b>
<b>13</b>	<b>Applications</b>	<b>57</b>
<b>14</b>	<b>Support, Training and Documentation</b>	<b>57</b>
14.1	The HPCx Administration Web Site . . . . .	57
14.2	The Helpdesk . . . . .	58
14.3	Training . . . . .	58





# 1 Introduction

Welcome to the HPCx service.

The HPCx service is the UK science community's newest capability computing service, and is operated by EPCC and CCLRC on behalf of the Engineering and Physical Sciences Research Council (EPSRC).

This document is an introduction to the HPCx service, and is intended both to target new users of the IBM eServer 575 system, and to provide a handy reference guide for experienced users.

Electronic and printable copies of the most up to date version of this document are accessible from:

<http://www.hpcx.ac.uk/support/introduction/index.html>

## 1.1 The IBM eServer 575 system Code of Conduct

The IBM eServer 575 system is a multi-user system. Although we try to set it up in such a way that everyone has fair access to the resources available, we depend to a very large extent on the goodwill and cooperation of all the users.

To help with this a Code of Conduct has been drawn up and if you follow it, you can be fairly sure that you are not impeding other people's work. The Code of Conduct is linked from the Policy Documents page:

<http://www.hpcx.ac.uk/services/policies/>

which also contains links to various other important HPCx policy documents, including Terms and Conditions of Access, Privacy Policy, Charging Model, Capability Incentive Scheme, etc.

## 2 Architecture Overview

The HPCx system consists out of 160 IBM eServer 575 LPARs for the compute and 8 IBM eServer 575 LPARs for login and disk I/O. Each eServer LPAR contains 16 processors, the maximum allowable by the hardware. The service offers a total of 2560 processors for computations.

Throughout this document we also use the names *logical partition (LPAR)* and *frame* as a synonym for the compute node.

The eServer 575 compute nodes utilise IBM Power5 processors. The Power5 is a 64-bit RISC processor implementing the PowerPC instruction set architecture. It has a 1.5 GHz clock rate, and has a 5-way super-scalar architecture with a 20 cycle pipeline. There are two floating point multiply-add units each of which can deliver one result per clock cycle, giving a theoretical peak performance of 6.0 Gflop/s. There is one divide and one square root unit, which are not pipelined.

The processor has 120 integer and 120 floating-point registers. There is extensive hardware support for branch prediction, and both out-of-order and speculative execution of instructions. There is a hardware prefetch facility: loads to successive cache lines trigger prefetching into the level 1 cache. Up to 8 prefetch streams can be active concurrently.

The level 1 cache is split into a 32 Kbyte data cache and a 64 Kbyte instruction cache. The level 1 data cache has 128-byte lines, is 2-way set associative and write-through.

Each chip contains two processors with their own level 1 caches and a shared level 2 cache. The level 2 cache is a 1.9 Mbyte combined data and instruction cache, with 128 byte lines and is 10-way set associative and write-back.

New to the Power5 is *simultaneous multi-threading*, or SMT. With SMT, each processor can support two

instruction streams, allowing the simultaneous execution of two threads. These streams appear as logical processors, two per physical processor, four per chip. However, these logical processors share the physical processor's level 1 instruction and data cache, floating-point, and other functional units. Enabling SMT will also cause the level 2 and 3 caches to be shared among the four logical processors. SMT is now user configurable, see Section 8 for more details.

Each chip is packaged, together with a level 3 cache, into a Dual-Core Module (DCM). The level 3 cache is 36 MBytes and is shared between the 2 processors, equivalent to 18 Mbytes per processor. It has 256 byte lines, and is 12-way set associative and write-back.

Each eServer node contains 8 DCMs (16 processors) and has 32 Gbytes of main memory.

Inter node communication is provided by an IBM's High Performance Switch (HPS), also known as "*Federation*." Each eServer node (frame) has two network adapters and there are two links per adapter, making a total of four links between each of the frames and the switch network.

## 3 Getting started

### 3.1 Web Administration Account

All users of the service must first register with HPCx via the HPCx web site. To do this, users will need the project code and password provided by their project Principal Investigator (PI) or supervisor. If the PI does not have these, the PI needs to contact the HPCx Helpdesk. The email address is:

`support@hpcx.ac.uk`

Once you have the project code and password, go to the HPCx web site at:

`http://www.hpcx.ac.uk/projects/`

and follow the instructions for new user registration. Users will be asked to provide a preferred e-mail address which will act as a unique identifier for HPCx administration. All communication with the user will be via this e-mail address, so it is essential that it is kept up-to-date. This is the user's responsibility.

Once the registration form is submitted, both the user and the project PI will receive an e-mail to that effect. Once confirmation of the registration from the PI has been received, an administration account will be created. The user will then receive a further e-mail with details of the account and initial password.

If encountering a delay between registration and receipt of account details, please check first with the project PI: it is their responsibility to confirm the registration. The process should usually take 1–2 working days.

Please report any problems with the web-site to:

`support@hpcx.ac.uk`

See section 4.3 and section 14.1 for more details of the web-site.

### 3.2 Machine accounts

It is important to realise that users will need to register for a web administration account before receiving a login account and password for the service machine. The former is used to access administration services via the web site, while the latter is a normal machine login account. The two should not be confused.

Machine accounts are requested via the web administration pages; again, go to:

<http://www.hpcx.ac.uk/projects/>

and follow the instructions.

### 3.3 Logging in

Users can log into the HPCx front end via `ssh` (and `ssh` only). For example,

```
$ ssh userid@login.hpcx.ac.uk
userid@login.hpcx.ac.uk's password:
```

The RSA key finger print of `login.hpcx.ac.uk` is

```
b8:01:60:9e:fe:46:67:d3:cf:56:ba:e0:cf:92:65:32.
```

Note that users will not be able to log into `hpcx.ac.uk` — this is not a service machine.

Users wishing to run X11 clients on the HPCx machine should use the `-X` option of `ssh` in addition when connecting to the HPCx front end

```
$ ssh -X userid@login.hpcx.ac.uk
```

Setting the display manually and using the `xhost` command to enable X11-traffic from `login.hpcx.ac.uk` is strongly discouraged for security reasons. When using PuTTY on a Windows machine X11-traffic can be enabled via

```
Connections > SSH > Tunnels > Enable X11 Forwarding
```

For security reasons, first-time users will be prompted to change their initial password via the `passwd` command. Passwords should be at least 6 characters long with at least 1 non-alphanumeric character. Users will not be allowed to re-use an existing password. Passwords will have a maximum lifetime of 6 months, after which a change will be requested.

Please note that you will be logged out of the machine after 5 hours of idle time.

### 3.4 Password changes

There are two sets of circumstances where a password change might be required:

*The password is known*, in which case the `passwd` command should be used. This will involve the user in a dialogue which is a little more involved than the usual `passwd` dialogue and which in fact silently logs the user onto the HPCx Administration Web Site machine. However, this dialogue is self-documenting, and should only take a moment to complete. Once completed, please note that it will take the new password several minutes to propagate through the whole HPCx machine. Note that this is the same dialogue as is used for changing shells, but changing the password is one of the two options presented.

*The password has been forgotten*, or is unknown for some other reason. In this case, requests for a new password must be made via the 'Your User Accounts' section of the HPCx Administration Web Site. See sections section 3.1 and section 14.1 for more details of the web-site.

### 3.5 System maintenance

By default, the system is booked for preventative maintenance from 1200 until 2400 on the 2nd and 4th Wednesdays of each month. These slots may not be taken, and if they are, then they may not be for the full duration – a stop/go decision will normally be made by 1200 hrs on the preceding Monday.

Watch the login message for details of these sessions.

### 3.6 Shells, etc.

The HPCx machines run AIX, the IBM version of Unix. The default login shell is the Korn shell (ksh). Users who prefer a different shell may change their login shell by running the chsh command.

This will involve the user in a dialogue which is a little more involved than the usual chsh dialogue and which in fact silently logs the user onto the HPCx Administration Web Site machine. However, this dialogue is self-documenting, and should only take a moment to complete (in fact it is the same dialogue as is used for changing passwords, but changing shell is one of the two options presented).

Currently available supported shells are:

```
bsh
bash
csh
ksh
```

Users are requested to restrict their choice of shell to this list.

### 3.7 Transferring files

File transfer to and from the HPCx machine must be achieved via scp or sftp. For example:

```
userid1@host$ scp template.tar userid2@login.hpcx.ac.uk:.
userid2@login.hpcx.ac.uk's password:
template.tar 100% |*****| 5170 00:00
userid1@host$
```

#### 3.7.1 Transferring large amounts of data to/from HPCx: bbFTP

For a description of using bbFTP to transfer large amounts of data between HPCx and HECToR please see the bbFTP guide on the HECToR website:

<http://www.hector.ac.uk/support/documentation/guides/bbftp/>

bbFTP is an open-source secure multistreamed file transfer utility, released under the GNU General Public License. It was written by Gilles Farrache at IN2P3 Computing Center in Lyon, France. bbFTP has been optimised for the transfer of large files: it securely authenticates, but then transfers file data in multiple unencrypted streams, thereby avoiding the computational overhead of encryption and decryption. bbFTP also allows large windows as defined in RFC1323, and therefore is well suited to the problem. Previous studies of grid-ftp have shown that there is good scope for improvement over SCP performance. grid-ftp however, requires a non-trivial certificate based authentication mechanism that is often too large a barrier for user uptake. bbFTP allows password or public key based authentication in the same manner as scp, so is immediately accessible to users with minimal setup costs.

bbFTP is available from <http://doc.in2p3.fr/bbftp/index.html>

### 3.8 Editors

The following text editors are available on the HPCx service machine:

## 4 User Resource Management

### 4.1 What resources do I have - time?

Users can always get a brief summary of their projects and the amount of resource remaining in those projects with the `budgets` command:

```
bash-2.05a$ budgets
z001:      999999 AU      416666:15:0
```

### 4.2 What resources do I have - storage?

#### 4.2.1 Diskspace

Disk space on the HPCx service is available in several varieties: **homespace**, **workspace** and **temporary scratch space**.

- Homespace is intended for permanent files: source and binary codes, permanent small datasets, etc. This is the area that is backed up.
- Workspace is not backed up. We expect this to be used for input and output files for current work. Workspace is considerably larger than homespace.

Workspace for each user is located at:

```
/hpcx/work/project_code/group_code/username
```

where *project\_code* is the code for your project (eg, x01); *group\_code* is the code for your project group, if your project has groups, or the same as the project code, if not (eg, x01-a) *username* is your login name.

For example:

```
/hpcx/work/z001/z001/johnf
```

- Temporary scratch space is available for the duration of your job. To access this, you have to set the variable

```
JTMPDIR=`lljtmp`
```

in your LoadLeveler script. The variable `JTMPDIR` contains the name of the temporary directory and can be used with the standard UNIX commands, e.g.

```
cd $JTMPDIR
```

The temporary scratch space (4.5TB at the time of writing) is shared between all jobs running on the machine on a first come first served basis. Unfortunately no guarantees — another job(s) might have taken it all.

**Important note:** The scratch space is erased once your job finishes. You have to copy the data from inside your LoadLeveler script into your home or work space.

User and group quotas can be interrogated with the `mmlsquota` command:

```
bash-2.05a$ mmlsquota -g z001
Block Limits | File Limits
Filesystem type KB quota limit in_doubt grace | files quota limit in_doubt grace

Disk quotas for group z001 (gid 900):
hsm GRP 3900800 0 10485760 146624 none | 14910 0 0 115 none
work GRP no limits
tmpchkpt GRP no limits
home GRP no limits
```

Note, this command is in `/usr/lpp/mmfs/bin/` which should be added to the path before use.

## 4.2.2 Tapestorage

The tape storage comprises a 4 frame 3584 tape library (1363 tape capacity) with 48 LTO tape drives. There is a separate user guide for the tape archiver [http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0405.pdf](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0405.pdf)

## 4.3 The HPCx Administration Web Site

All users have a login and password on the HPCx Administration Web Site (aka the SAF page):

<https://www.hpcx.ac.uk/login.jsp>

Once logged into this web page, users can find out much about their usage of the HPCx system, including:

- machine status
- project details
- detailed usage information including report generation

These features are largely self-documenting and are not described further here.

See section 3.1 and section 14.1 for more details of the web-site.

## 4.4 Charging

The standard charging policy on HPCx is to multiply the elapsed wall-clock time for each job by the number of nodes used. Since parallel production jobs are granted exclusive access to their nodes, each node is charged with the cost of all its 16 CPUs, even if your job uses only a fraction of it. For example, a job lasting 1 hour on 50 CPUs from four nodes will be charged as 64 CPU hours. For most jobs it will be most economic to ask for multiples of 16 CPUs, that is 16, 32, 48, 64, ... CPUs.

However, there are two exceptions to this rule: serial and interactive jobs. In both these cases users are only charged for the CPU time actually used.

For serial jobs we would expect this to be almost identical to the wall-clock time as we don't over subscribe CPUs (ie: there are never more serial processes than the number of CPUs dedicated to serial jobs). For interactive jobs we allow only one process to run on each CPU, so users will be charged for the number of processes multiplied by the elapsed wall-clock time.

## 5 Compilation

### 5.1 General remarks on compilation

#### 5.1.1 Use of `/tmp` by the compilers

By default the IBM compilers use `/tmp` to store temporary information. If `/tmp` fills up or information placed there is erased the compilation may fail, quite often with no clear indication on the cause of the problem. For reasons outlined in section 6.1, users should not normally use `/tmp` and files placed on `/tmp` can get deleted by the system. To avoid the compilation failing at random places in case of the login node being very busy, users can set the environment variable `TMPDIR` to a directory in their own home or workspace. For example in `ksh` typing

```
mkdir ~/MyTemp
export TMPDIR=~/MyTemp
```

makes the compilers use the directory `MyTemp` in your home space.

### 5.2 Compiling sequential code

Serial codes are compiled with:

- `xlf_r` (FORTRAN 77)
- `xlf90_r` (Fortran 90)
- `xlc_r` (C)
- `x1C_r` (C++)
- `javac` (Java)

The `_r` suffix stands for *re-entrant* and denotes that the compiler should generate thread safe code. This is essential if you are using OpenMP or threads, but it also helps with MPI codes, so it is always recommended. There are several reasons for this:

- They allow MPI-2 features to be utilised.
- They give access to numerical libraries which are more stable and allow both 32- and 64-bit addressing.
- They are not detrimental to a code's performance.

Please note that for Fortran code the `-qsuffix` flag will be needed if your Fortran files do not have a `.f` or `.F` suffix, e.g.

```
xlf90_r -qsuffix=f=f90 -o mycode -q64 mycode.f90
```

For Java codes, the `-O` flag may be used for optimisation. e.g.

```
javac -O mycode.java
```

## 5.3 Compiling MPI code

### 5.3.1 Fortran

To compile Fortran MPI code the basic command is `mpxlf90_r`, e.g.:

```
mpxlf90_r -qsuffix=f=f90 -q64 -o mycode mycode.f90
```

The `mp` prefix denotes that the compiler is for multi-processor (MPI) programs.

Please note the comment above (Compiling sequential code) for the `-qsuffix` flag.

### 5.3.2 C

`mpcc_r` is used to compile MPI C code, e.g.:

```
mpcc_r -q64 -o mycode mycode.c
```

Please have a look at the below remark on the `-q64` option.

### 5.3.3 C++

`mpCC_r` is used to compile MPI C++ code:

```
mpCC_r -q64 -o mycode mycode.C
```

If your code is using the C++ bindings of MPI instead of the standard C bindings, that is utilising functions which start with “MPI: :” instead of “MPI\_”, you need to define the pre-processor variable `_MPI_CPP_BINDINGS`. This can be achieved by using the `-D` compiler flag:

```
mpCC_r -q64 -DMPI_CPP_BINDINGS -o mycode mycode.C
```

Please have a look at the below remark on the `-q64` option.

## 5.4 Compiling OpenMP code

Use the `-qsmp` option to turn on shared-memory parallelisation. You must use the `_r` versions of the compilers which provide thread-safe versions of the Fortran/C/C++ compilers, e.g.

```
$ export OMP_NUM_THREADS=2
$ xlf90_r -qsmp=omp -qsuffix=f=f90 -o hello hello.f90
* hello === End of Compilation 1 ===
1501-510 Compilation successful for file hello.f90.
$ ./hello
hello from 0 of 2
hello from 1 of 2
```

## 5.5 Other useful compiler switches

### **-q64**

By default the `xlf90_r` compiler uses 32-bit addressing. To change this to use 64-bit addressing the `-q64` options can be used. All object files making up the same executable must be compiled **and** linked with the `-q64` flag.

This flag increases the amount of memory that a program can use and removes some of the restrictions on shared memory segments so the 64-bit MPI library has some additional optimisations. This flag is therefore a good choice for most codes. One possible disadvantage is that all pointers will double in size so there may be a performance impact on codes that use a lot of pointers, this mainly concerns codes written in C or C++.

If you want to make your own 64-bit object code libraries using the `ar` command you will also need to set the environment variable `OBJECT_MODE=64` or use the `-X 64` flag. Setting `OBJECT_MODE=64` is an alternative to specifying `-q64` as a linker option.

#### **-qrealsize=8**

It is important to note that the default size of variables declare `REAL` is 4 bytes (on a Cray, for example, it is 8 bytes). The `-qrealsize` option allows one to specify either 4 or 8 bytes for the default size. Please also note that changing this value from 4 to 8 bytes also changes the size of `COMPLEX` variables to  $2 \times 8$  bytes, `DOUBLE COMPLEX` variables to  $2 \times 16$  bytes and `DOUBLE PRECISION` variables from 8 bytes to 16 bytes. Integers (and logicals) are also promoted to 8 bytes. It is up to the user to ensure that the Fortran/C types are consistent with the MPI types, which will *not* be altered automatically.

Please see the section on 'Porting code' for details of Fortran, C and MPI datatypes.

## **5.6 Example Makefiles**

Example Makefiles, batch scripts and short test programs are available at

<http://www.hpcx.ac.uk/support/FAQ/template.tar>

The Makefiles are also shown below.

To compile the examples, just use:

```
make -f Makefile.MPI.[f|f90|c] (for .[f|f90|c] MPI files)
make -f Makefile.OMP.[f|f90|c] (for .[f|f90|c] OMP files)
make -f Makefile.MIX.[f|f90|c] (for .[f|f90|c] mixed MPI/OMP files)
```

It is probably best to do a `make -f Makefile.X.Y clean` beforehand too.

These examples have deliberately been written using 2 separate files (e.g. `mpihello.f90` and `hello.f90`) to ensure the Makefiles are not completely trivial.

### **5.6.1 MPI fixed format Fortran code**

Example Makefile (`Makefile.MPI.f`) to compile code with two MPI fixed format Fortran source files: `mpihello.f` and `hello.f`.

```
MF=      Makefile.MPI.f

FC=      mpixlf_r
FFLAGS=  -q64 -O3 -qarch=pwr4 -qtune=pwr4
LFLAGS=  $(FFLAGS)

EXE=     mpihello

SRC= \
    mpihello.f \
    hello.f
```

```

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f .o

OBJ=    $(SRC:.f=.o)

.f.o:
    $(FC) $(FFLAGS) -c $<

all:    $(EXE)

$(EXE): $(OBJ)
    $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.2 MPI Fortran 90 code

Example Makefile (Makefile.MPI.f90) to compile code with two MPI Fortran 90 source files: mpihello.f90 and hello.f90.

```

MF=    Makefile.MPI.f90

FC=    mpixlf90_r
FFLAGS= -qsuffix=f=f90 -q64 -O3 -qarch=pwr4 -qtune=pwr4

LFLAGS= $(FFLAGS)

EXE=    mpihello

SRC= \
    mpihello.f90 \
    hello.f90

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f90 .o

OBJ=    $(SRC:.f90=.o)

```

```

.f90.o:
    $(FC) $(FFLAGS) -c $<

all:    $(EXE)

$(EXE): $(OBJ)
    $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.3 MPI C code

Example Makefile (Makefile.MPI.c) to compile code with two C source files: mpihello.c and hello.c

```

MF=      Makefile.MPI.c

CC=      mpcc_r
CFLAGS=  -q64 -O3 -qarch=pwr4 -qtune=pwr4

LFLAGS=  $(CFLAGS)

EXE=     mpihello

SRC=     \
    mpihello.c      \
    hello.c

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .c .o

OBJ=     $(SRC:.c=.o)

.c.o:
    $(CC) $(CFLAGS) -c $<

all:     $(EXE)

$(EXE):  $(OBJ)
    $(CC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ):  $(MF)

```

```

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

#### 5.6.4 OpenMP fixed format Fortran code

Example Makefile (Makefile.OMP.f) to compile code with two OMP fixed format Fortran source files: omphello.f and hello.f.

```

MF=      Makefile.OMP.f

FC=      xlf_r
FFLAGS=  -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp,noauto
LFLAGS=  $(FFLAGS)

EXE=     omphello

SRC= \
    omphello.f      \
    hello.f

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f .o

OBJ=     $(SRC:.f=.o)

.f.o:
    $(FC) $(FFLAGS) -c $<

all:     $(EXE)

$(EXE): $(OBJ)
    $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.5 OpenMP Fortran 90 code

Example Makefile (Makefile.OMP.f90) to compile code with two OpenMP Fortran 90 source files: omphello.f90 and hello.f90

```
MF=      Makefile.OMP.f90

FC=      xlf90_r
FFLAGS=  -qsuffix=f=f90 -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp,noauto
LFLAGS=  $(FFLAGS)

EXE=     omphello

SRC=     \
         omphello.f90   \
         hello.f90

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f90 .o

OBJ=     $(SRC:.f90=.o)

.f90.o:
        $(FC) $(FFLAGS) -c $<

all:     $(EXE)

$(EXE):  $(OBJ)
        $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ):  $(MF)

tar:
        tar cvf $(EXE).tar $(MF) $(SRC)

clean:
        rm -f $(OBJ) $(EXE) core
```

### 5.6.6 OpenMP C code

Example Makefile (Makefile.OMP.c) to compile code with two OMP C source files: omphello.c and hello.c.

```
MF=      Makefile.OMP.c

CC=      xlc_r
```

```

CFLAGS= -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp:noauto

LFLAGS= $(CFLAGS)

EXE=    omphello

SRC= \
    omphello.c    \
    hello.c

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .c .o

OBJ=    $(SRC:.c=.o)

.c.o:
    $(CC) $(CFLAGS) -c $<

all:    $(EXE)

$(EXE): $(OBJ)
    $(CC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.7 Mixed MPI/OpenMP fixed format Fortran code

Example Makefile (Makefile.MIC.f) to compile code with two MPI/OpenMP fixed format Fortran source files: mixhello.f and hello.f.

```

MF=    Makefile.MIX.f

FC=    mpxlf_r
FFLAGS= -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp,noauto

LFLAGS= $(FFLAGS)

EXE=    mixhello

SRC= \
    mixhello.f    \
    hello.f

```

```

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f .o

OBJ=    $(SRC:.f=.o)

.f.o:
    $(FC) $(FFLAGS) -c $<

all:    $(EXE)

$(EXE): $(OBJ)
    $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.8 Mixed MPI/OpenMP Fortran 90 code

Example Makefile (Makefile.MIX.f90) to compile code with two mixed MPI/OpenMP Fortran 90 source files: mixhello.f90 and hello.f90

```

MF=    Makefile.MIX.f90

FC=    mpxlf90_r
FFLAGS= -qsuffix=f=f90 -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp,noauto

LFLAGS= $(FFLAGS)

EXE=    mixhello

SRC= \
    mixhello.f90 \
    hello.f90

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f90 .o

OBJ=    $(SRC:.f90=.o)

```

```

.f90.o:
    $(FC) $(FFLAGS) -c $<

all:    $(EXE)

$(EXE): $(OBJ)
    $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core

```

### 5.6.9 Mixed MPI/OpenMP C code

Example Makefile (Makefile.MIX.c) to compile code with two mixed MPI/OpenMP C source files: mixhello.c and hello.c.

```

MF=      Makefile.MIX.c

CC=      mpcc_r
CFLAGS=  -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qsmp=omp:noauto

LFLAGS=  $(CFLAGS)

EXE=     mixhello

SRC= \
    mixhello.c \
    hello.c

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .c .o

OBJ=     $(SRC:.c=.o)

.c.o:
    $(CC) $(CFLAGS) -c $<

all:     $(EXE)

$(EXE): $(OBJ)
    $(CC) $(LFLAGS) -o $@ $(OBJ)

```

```
$(OBJ): $(MF)

tar:
    tar cvf $(EXE).tar $(MF) $(SRC)

clean:
    rm -f $(OBJ) $(EXE) core
```

## 6 File Management

### 6.1 Use of /tmp

Users should use their areas in /home for their development files and /work for temporary and run-time files.

The /tmp area is principally intended for the temporary storage of data used by the system and system programs. Because of the size and number of jobs normally run on the HPCx service, it is impractical to allow users create files in /tmp as it soon overflows. Normally, the system ensures that there is free space on /tmp by periodically removing files as necessary, when they are no longer in use. If /tmp is filling up at a dangerous rate, however, any files there may be deleted automatically at any time, in order to ensure that the system operates safely.

For this reason, users should not normally place files on /tmp.

#### 6.1.1 Use of /tmp by the compilers

The IBM Fortran, C and C++ compilers use /tmp to store temporary information. This /tmp area is shared with all other interactive users and occasionally /tmp may fill up or temporary files may be erased. In this case the compilation may fail, quite often with no clear indication of what the problem is. Setting the environment variable TMPDIR makes the compilers use a different directory. Please see section 5.1.1 for an example.

#### 6.1.2 Use of /tmp - gotchas for Fortran programmers

If a Fortran program creates a file of type SCRATCH, the file is given a name starting with xlf . . . and by default it is placed in the /tmp directory. If the program then doesn't exit properly for whatever reason, the file is not removed, but continues to exist.

There is a /tmp area for every LPAR. If that area gets full, AIX on that lpar crashes.

On some recent occasions, failed program runs have left a lot of files in the /tmp areas and we need to make sure that this doesn't have any catastrophic effects on the HPCx system.

The easiest way to do this is for users to arrange for SCRATCH files to be created somewhere else. You can do this by defining the TMPDIR environment variable to point to another directory, e.g. in the users workspace.

We would ask *all* users to follow this simple procedure. If the /tmp directories continue to fill up we will have to put restrictions on this, which would inconvenience everybody.

## 6.2 Other file management issues

For information relating to data storage and contingency plans in the case of data loss, please refer to:

<http://www.hpcx.ac.uk/services/storage/index.html>

<http://www.hpcx.ac.uk/services/policies/contingency.html>

## 7 Porting Codes

This section provides some helpful information on porting codes to the IBM p690 Regatta system.

### 7.1 Tips for FORTRAN programmers

#### 7.1.1 Useful compiler options

The following compiler options may be useful when porting your code to the IBM p690 Regatta system:

compiler option	effect
-initauto	initialises variables all variables to zero at compile time
-eI	equivalent to having IMPLICIT NONE throughout the code

#### 7.2 Timers

It is often useful to include time keeping routines within your code. This section describes the timing routines available on the IBM p690 Regatta system.

##### 7.2.1 Fortran codes

The Fortran 90 specific `SYSTEM_CLOCK` offers good portability, however due to its implementation as 32-byte integer we recommend it for simple tasks only.

```
integer :: clock0, clock1, clockmax, clockrate, ticks
real    :: secs
call system_clock(count_max=clockmax, count_rate=clockrate)
call system_clock(clock0)

! code to be timed

call system_clock(clock1)

ticks = clock1-clock0
ticks = mod(ticks+clockmax, clockmax) ! reset negative numbers
secs = float(ticks)/float(clockrate)
write(*,*) 'Code took ', secs, ' seconds'
```

MPI also defines a timer, namely `MPI_Wtime()` which returns a double precision number of seconds, representing elapsed wall-clock time since some time in the past. `MPI_Wtime()` provides a good and portable solution for Fortran codes using MPI.

Note this does not involve clock ticks, so use of this timer does not need to determine the clock rate. If one wishes to determine the resolution of `MPI_Wtime`, then the routine `MPI_Wtick()` should be employed, which returns the number of seconds between successive clock ticks as a double precision value.

```
DOUBLE PRECISION :: start, end
start = MPI_Wtime()

! code to be timed

end = MPI_Wtime()
print*, 'That took ', end-start, ' seconds'
```

Note that the times returned are local to the node that called them. There is no requirement that different nodes return “the same time”. MPI provides a boolean variable, namely `MPI_Wtime_is_global`, which indicates whether clocks are synchronised.

Lastly, the following table gives a list of IBM SP vendor-specific calls which may be of use when timing FORTRAN codes. NB. these routines are non-portable.

Name	Function
<code>clock_</code>	returns current time, in ASCII hh:mm:ss format
<code>date</code>	returns the current date, in ASCII mm/dd/yy format
<code>idate_</code>	returns date in numerical form
<code>irtc</code>	returns real-time clock
<code>itime_</code>	returns time in numerical form
<code>jdate</code>	returns the current Julian day-number
<code>rtc</code>	returns real-time clock
<code>timef</code>	returns elapsed wall-clock time
<code>time_</code>	returns the value of time in seconds

Here is an example of a timing module which uses `irtc`:

```
module time_tool

contains
  function now_time()
    real(kind=8) now_time
    integer(kind=8) nano_time ,irtc
    external irtc

    nano_time=irtc()
    now_time = real(nano_time,kind=8) * 1.0e-9_8
  end function now_time
end module time_tool
```

## 7.2.2 C/C++ codes

### ANSI C

The `clock()` function returns clock ticks.

```

#include <time.h>

clock_t start, end;
double elapsed;

start = clock();
  <code to be timed>
end = clock();
elapsed = ((double) (end - start)) / CLOCKS_PER_SEC;

```

The `gettimeofday()` function has a resolution of microseconds.

### C++

```

#include <sys/time.h>
struct timeval *Tps, *Tpf;
void *Tzp;
Tps = (struct timeval*) malloc(sizeof(struct timeval));
Tpf = (struct timeval*) malloc(sizeof(struct timeval));
Tzp = 0;
gettimeofday (Tps, Tzp);
  <code to be timed>
gettimeofday (Tpf, Tzp);
printf("Total Time (usec): %ld\n",
      (Tpf->tv_sec-Tps->tv_sec)*1000000
      + Tpf->tv_usec-Tps->rvr_usec);

```

As with FORTRAN codes, `MPI_Wtime()` may also be used with C/C++ codes to determine the time since a particular date.

### C++ example

```

#include <mpi.h>
double start, finish;

start = MPI_Wtime();
  <code to be timed>
finish = MPI_Wtime();

printf("Final Time: %f", finish-start);
/* Time is in milliseconds since a particular date */

```

### 7.2.3 Java codes

For Java codes, the `System.currentTimeMillis()` may be used to determine the time since a particular date.

```

long start = System.currentTimeMillis();
  <code to be timed>
long finish = System.currentTimeMillis();
printf("Final Time: %f", finish-start);
/* Time is in milliseconds since a particular date */

```

## 7.3 Default sizes

### 7.3.1 FORTRAN codes

The default sizes and available datatypes of the SP are as follows

Type	Length (bytes)
character	1
complex	2 × 4
double complex	2 × 8
double precision	8
integer/logical	4
real	4

NB. double complex is not standard FORTRAN.

The SP XL Fortran compiler flag `-qrealsize=8` can be used to promote all default reals, and real constants, to 8 bytes. Integers and logicals can be similarly promoted using `-qintsize=8`.

A portable alternative is to use the Fortran 90 KIND syntax.

Type	KIND	*	Length (bytes)
complex	4	8	2 × 4
	8	16	2 × 8
	16	32	2 × 16
integer/logical	1	1	1
	2	2	2
	4	4	4
	8	8	8
real	4	4	4
	8	8	8
	16	16	16

### 7.3.2 C/C++ codes

These are the default sizes of the standard C/C++ datatypes on the IBM SP.

Type	Length (bytes)
bool	1
char	1
wchar_t	4
short	2
int	4
long	4/8
float	4
double	8
long double	8/16

NB. bool is C++ only.

The second number given for both long and long double applies when the code is compiled in 64 bit mode.

## 7.4 MPI Datatypes

The default sizes of MPI datatypes are different on the different machines:

### 7.4.1 Fortran

Type	Length (bytes)
MPI_CHARACTER	1
MPI_COMPLEX	$2 \times 4$
MPI_DOUBLE_COMPLEX	$2 \times 8$
MPI_DOUBLE_PRECISION	8
MPI_INTEGER	4
MPI_LOGICAL	4
MPI_REAL	4

**NB:** If the associated compiler flags are employed to increase the precision of default reals, then the interpretation of MPI\_REAL is NOT changed.

MPI defines additional datatypes with explicit sizes. These can be used to provide some portability:

Type	Length (bytes)	Type	Length (bytes)
MPI_COMPLEX8	$2 \times 4$	MPI_LOGICAL1	1
MPI_COMPLEX16	$2 \times 8$	MPI_LOGICAL2	2
MPI_COMPLEX32	$2 \times 16$	MPI_LOGICAL4	4
MPI_INTEGER1	1	MPI_LOGICAL8	8
MPI_INTEGER2	2	MPI_REAL4	4
MPI_INTEGER4	4	MPI_REAL8	8
MPI_INTEGER8	8	MPI_REAL16	16

### 7.4.2 C

Type	Length (bytes)	Type	Length (bytes)
MPI_CHAR	1	MPI_SHORT	2
MPI_DOUBLE	8	MPI_SIGNED_CHAR	1
MPI_FLOAT	4	MPI_UNSIGNED	4
MPI_INT	4	MPI_UNSIGNED_CHAR	1
MPI_LONG	4	MPI_UNSIGNED_LONG	4
MPI_LONG_DOUBLE	8	MPI_UNSIGNED_LONG_LONG	8
MPI_LONG_LONG	8	MPI_UNSIGNED_SHORT	2
MPI_LONG_LONG_INT	8	MPI_WCHAR	2

## 8 Batch Processing

Batch processing on the HPCx system is controlled using LoadLeveler, a batch job scheduling application produced by IBM:

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sg246038.pdf>

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2278810.pdf>

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2278820.pdf>

## 8.1 Batch System Overview

LoadLeveler provides the facility to build, submit and process batch jobs on the HPCx system. Users can interact with LoadLeveler either using the command line, or using the LoadLeveler GUI `xloadl` (see Section 8.9).

The following table gives the runtime limits the present LoadLeveler setup allows, depending on the number of tasks/threads. The table assumes 16 tasks/threads are allocated per node.

max. tasks/threads	max. runtime	Comment
16	48h	Largest OpenMP job
128	48h	Largest Capacity job
1024	48h	Capability jobs
1536	1h	largest partition available
serial	12h	special serial queue

There are special queues offering a higher priority for testing and debugging jobs using up to 128 tasks. These queues are available from 9:00 to 18:00 Monday to Friday. To access these queues you have to ask for a `wall_clock_limit` of 20 minutes or less.

Users should note that capacity and capability jobs are run in different regions of the machine. Capability jobs often achieve a faster turn-around time. Users are therefore encouraged to run on 256 processors or more. If you require help to achieve scalability to this level then please contact the helpdesk ([helpdesk@hpcx.ac.uk](mailto:helpdesk@hpcx.ac.uk)). Users of OpenMP should note that it is possible to run many separate OpenMP jobs simultaneously within a single script, enabling them to submit jobs using more than 16 processors (the limit for a single OpenMP program). For details on how to do this see the FAQ entry <http://www.hpcx.ac.uk/support/FAQ/#omptaskfarm>.

Simultaneous Multithreading (SMT) is a new feature of the Power5 processor that allows two task/threads to be executed simultaneously by a single CPU. If SMT is enabled, the numbers of tasks/threads in the table above should be multiplied by two. See Section 8.2.6 for further details.

## 8.2 Creating a job command file

Every LoadLeveler job must be submitted through a job command file. This describes the job to be submitted and contains a number of LoadLeveler keyword statements which specify the various resources needed by the job.

Job command files can either be created using a text editor or by using the `xloadl` tool. Job command files have no specific naming convention, and may be given whatever name you wish. LoadLeveler keywords in the file are case insensitive.

### 8.2.1 MPI job command files

This is a simple LoadLeveler script for running on 256 processors. It also contains our presently recommended setting of the environment variables for **production** jobs. These setting and values proved to give good performance for a wide variety of applications. Tweaking individual settings might lead to further improvements for some applications. For a **development** job `MP_EAGER_LIMIT` should be changed to 0.

```
#@ shell = /bin/ksh
#
#@ job_name = myrun
```

```

#
#@ job_type = parallel
#@ cpus = 256
#@ node_usage = not_shared
#
#@ network.MPI = csss,shared,US
#@ bulkxfer = yes
#
#@ wall_clock_limit = 00:20:00
#@ account_no = z001
#
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue

# suggested environment settings:
export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export MEMORY_AFFINITY=MCM
export MP_TASK_AFFINITY=MCM

poe ./my_executable

```

Note that all the lines in this command file are required, unless stated below. The meaning of each line in this command file is as follows:

**shell = /bin/ksh**

Specifies the shell to be used for the job.

**job\_name = my\_run**

This allows you to give a name to the job. This is not mandatory, but is useful for identifying output files (see later).

**job\_type = parallel**

This informs LoadLeveler that this is a parallel job which requires scheduling on multiple processors.

**cpus = 256**

Specify the number of CPUs you need. This can be any number, however since the machine is partitioned into LPARs of 16 CPUs, we recommend multiples of 16, e.g. 16, 32, 48, 64, .... If you select a number of CPUs different from the above, you will be charged for the next full multiple of 16. To be specific, if you ask for 59 CPUs, you will be charged for  $4 \times 16 = 64$  CPUs. This is because your jobs is granted exclusive access to the frames it occupies.

**node\_usage = not\_shared**

This ensures that your job will have exclusive use of the LPAR.

**network.MPI = csss,shared,US**

By default, MPI jobs on HPCx use IP communications not the dedicated US user-space communications. As US has better latency, we recommend that this should be used in MPI scripts by default. However, if you use a single LPAR you may see something similar to the following error:

```

US is not valid for single nodes. Delete "<whatever #@network is given>"
or use IP

```

If this is the case then do not include this line as US communications do not function on a single LPAR. By leaving out the line, IP communications will be used and this error should not occur.

**bulkxfer = yes**

Switches on Remote Direct Memory Access (RDMA) to boost the maximum bandwidth available via the switch network. We expect this to be beneficial to most codes, but a few applications might see a decrease in performance. We recommend comparing the performance of your application with and without this line in the script.

**wall\_clock\_limit = 00:20:00**

Specifies a wall clock limit of 20 minutes for the job. The wall clock limit has the format hh:mm:ss or mm:ss. The time chosen has to be shorter than the time limit for the number of tasks you are using. See Section 8.1 for details.

**account\_no = z001**

You must set `account_no` to a valid budget for your project (e.g. y001, y002, ...). You can determine the budgets you have access to with the `budgets` command:

```
$ budgets
z001:      1000000 AU          416666:40:0
```

**output = \$(job\_name).\$(schedd\_host).\$(jobid).out**

**error = \$(job\_name).\$(schedd\_host).\$(jobid).err**

These lines specify the files to which stdout and stderr from your job will be redirected. There is no default, so you must set something here. The use of `$(schedd_host).$(jobid)` is recommended as this matches the `hostid/jobid` reported by LoadLeveler (see below).

**notification = never**

Suppresses email notification of job completion. To receive email notification, please see below.

**queue**

This line tells LoadLeveler to enqueue the job: this is essential!

**export MP\_EAGER\_LIMIT=65536**

Tweak the MPI library. Most applications perform best for a value of 65536, however you might want to check your application. When running on more than 256 CPUs, `poe` will reduce the `MP_EAGER_LIMIT` to a lower value. Ignore the corresponding warning in your error file. For **development work** set `export MP_EAGER_LIMIT=0`. If your code doesn't work for `MP_EAGER_LIMIT=0`, there is a problem with the way it uses MPI, which needs mending.

**export MP\_SHARED\_MEMORY=yes**

Use shared memory inside your logical partition (frame). Don't change.

**export MEMORY\_AFFINITY=MCM**

Use the memory closest to the cpu.

**export TASK\_AFFINITY=MCM**

Ensure that processes do not migrate between MCMs.

**poe ./my\_executable**

This line executes an MPI executable called `my_executable` in the current directory. `poe` is the MPI job launcher: note that you do not have to specify the number of processes here: it is automatically derived from the number of requested with the LoadLeveler keyword `cpus`.

By default LoadLeveler uses your login shell to interpret the command file. You may need to modify this line to use the syntax of your login shell for setting environment variables. Alternatively, you can specify a different shell to interpret the command file by adding a line such as:

```
#@ shell = /bin/ksh
```

## 8.2.2 MPI job with less than 16 tasks per LPAR

Some applications perform better when leaving a number of processors on your LPAR unused. This leaves these processors free to deal with the needs of the operating system and threads originating from the MPI library. Keep in mind that you will be charged for these processors.

Adding the optional line

```
#@ tasks_per_node = 15
```

above the “#@ queue”, will place 15 MPI-tasks on a single LPAR. In this case you should specify multiples of 15 for the required number of CPUs (LoadLeveler keyword cpus). Remember you still have to pay for 16 CPUs per LPAR.

## 8.2.3 OpenMP job command files

The HPCx Phase 2a system is operated as a cluster of 16-way SMPs. This makes it a more attractive resource to run applications in shared memory using OpenMP. Below is a simple sample script to run an OpenMP code with 16 threads.

```
#@ shell = /bin/ksh
#
#@ job_name = my_openmp_run
#
#@ job_type = parallel
#@ cpus = 1
#@ node_usage = not_shared
#
#@ wall_clock_limit = 00:20:00
#@ account_no = z001
#
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue
#

export XLSMPOPTS=spins=0:yields=0
export OMP_NUM_THREADS=16

./my_omp_executable
```

### Comment:

The lines

```
#@ job_type = parallel
#@ cpus = 1
#@ node_usage = not_shared
```

ensure LoadLeveler dedicates a full frame with 16 processors to your job but doesn't place more than 1 task. Obviously your account gets charged for all 16 processors of the frame dedicated to your job. The command `export XLSMPOPTS=spins=0:yields=0` forces the threads to adopt busy-waiting where they keep executing in a tight loop looking for new work instead of going to sleep.

## 8.2.4 Mixed MPI/OpenMP job command files

Please email [support@hpcx.ac.uk](mailto:support@hpcx.ac.uk) if you are interested in running mixed MPI/OpenMP applications.

## 8.2.5 Enabling core files

By default a failing job on HPCx will not produce a core file. To enable the production of core files, you need to add a line similar to

```
#@ core_limit = 10mb
```

to the header of your LoadLeveler script. The specified size applies to each MPI task individually. The files get truncated to this size. For large core files this might need adjusting.

## 8.2.6 Enabling Simultaneous Multi-Threading (SMT)

Simultaneous Multi-Threading is now a user configurable option within LoadLeveler, and is available for all jobs using “#@ node\_usage = not\_shared”. This excludes serial and interactive jobs. SMT allows two separate threads or tasks to be executed simultaneously on each physical processor, splitting it into two logical processors.

To enable SMT for a job, you must include the SMT feature keyword in your job submission script:

```
#@ requirements = ( Feature == "SMT" )
```

SMT will be enabled prior to the beginning of your job, and disabled at completion, making 32 processors available per node. For MPI jobs, you should therefore increase your tasks per node by setting:

```
#@ tasks_per_node = 32
```

Similarly, for Mixed or OpenMP jobs,

```
export OMP_NUM_THREADS=32
```

As usual, nodes can be underpopulated if you require this (MPI tasks or OpenMP threads less than 32).

Many applications stand to benefit significantly from SMT, but please note that this is an advanced feature, and the performance of some applications may not improve. For more details of SMT and its potential performance benefits, see:

[http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0604.pdf](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0604.pdf)

**Important Note:** When using SMT, you should be careful how you interpret timing calls to your code. The “User” and “System” times commonly reported accumulate only when the thread concerned is actually running, hence a processor running two threads will report only about half of its true execution time with a CPU timer. It is important to make sure you use a “wallclock” or “elapsed time” timer, which measures the amount of real time spent running your application. For this purpose, we recommend the `MPI_Wtime()` and `irtc()` routines, both of which are good wallclock timers.

Please email [support@hpcx.ac.uk](mailto:support@hpcx.ac.uk) if you experience problems when using SMT.

## 8.2.7 Serial job command files

```
#@ shell = /bin/ksh
#
#@ job_name = my_serial_run
#
#@ job_type = serial
#@ node_usage = shared
#
#@ wall_clock_limit = 00:20:00
#@ account_no = z001
#
#@ output = $(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue
#

./my_serial_executable
```

This job will run in a serial class: see Section 8.3.

For serial jobs “#@ node\_usage = shared” has to be specified.

## 8.2.8 Chaining serial and parallel jobs

LoadLeveler allows a batch job to be broken into a series of dependent job-steps. Each job step progresses through the queues independantly but will not start until all of the job-steps it depends on have completed. This can be very useful for adding serial pre/post-processing steps to a parallel job. This is particularly important on HPCx if you wish to copy data in and out from a remote system because only the login node and the node that runs the serial classes can make network connections to remote systems.

```
#@ wall_clock_limit = 00:10:00
#@ account_no = z001
#@ job_name = hello
#@ output = $(job_name).$(jobid).$(stepid).out
#@ error = $(job_name).$(jobid).$(stepid).err
#@ job_type = serial
#@ node_usage = shared
#@ executable = serial_script1.sh
#@ step_name = serial_1
#@ queue
#@ executable = parallel_script.sh
#@ node_usage = not_shared
#@ dependency = serial_1 == 0
#@ job_type = parallel
#@ step_name = parallel_1
#@ cpus=16
#@ queue
#@ node_usage = shared
#@ executable = serial_script2.sh
```

```
#@ dependency = parallel_1 == 0
#@ job_type = serial
#@ step_name = serial_2
#@ queue
```

The LoadLeveler script consists of a series of job-steps separated by the *queue* keyword. Any keyword that is not explicitly overridden is inherited by the following steps. The commands to be run at each step needs to be placed in a separate script. In the above example each step only continues if the previous step exited with a zero return value.

### 8.3 Batch job allocation and queueing policy

Job allocation and queueing policy are controlled to some extent by machine configuration which can change to accommodate changing user demand, so only a brief overview of a 'typical' configuration is given here

LoadLeveler execution queues are called classes.

Jobs submitted via `llsubmit` (see below) are automatically assigned to the appropriate class based on the job type, number of CPUs and wall clock limit requested in the LoadLeveler command file. This also applies to serial jobs.

Any given job will be allocated to the class whose number of CPUs and wall clock time is the next above that which has been requested in the LoadLeveler script file.

Allocation of processors for parallel jobs is by node (i.e. in multiples of 16), and is exclusive, even if not all the processors on the node are used by a job. The relevant budget will be charged with the wall clock time multiplied by the number of nodes. Regardless of how many processors in each node are actually used each node will be charged as 16 CPUs. The same conditions apply to all parallel jobs.

LoadLeveler is operated with a backfilling policy in operation. This means that for jobs requesting large number of nodes, LoadLeveler will identify a time slot in the future for the job to run in. It will then prevent smaller jobs from running which would impinge on this time slot. Therefore, small jobs may not run when there appear to be sufficient free processors available. This policy means that large jobs will not get "*frozen out*" because sufficient processor are never available at the same time.

Of course, this sometimes has an impact on the throughput of smaller jobs, but users should never ask for a longer wall-clock time than a job needs in the hope that this will speed up the throughput of the job. It may well do so occasionally, but overall, the "*optimal*" strategy is always to ask for just as much wall clock time as it is believed the job will need.

The available classes can be viewed using the `llclass` command. The column "*Description*" informs you about the time and CPU limit of each class.

### 8.4 Job submission - the `llsubmit` command

Jobs are submitted to the system via the `llsubmit` command, or via the `xloadl` GUI (see Section 8.9).

For example:

```
$ llsubmit hello.ll
```

where `hello.ll` is the LoadLeveler command file. The filter does a number of checks on your command file. If it spots a problem, you will receive an error message and the job will not be submitted.

## 8.5 Job status information - the `llq` command

The `llq` command displays information on the current status of LoadLeveler jobs, as does the `xloadl` GUI.

The status column (ST) provides details of the current status of all the jobs on the system. The most common states are: R (Running), I (Idle), ST (Starting) and C (completed). Queueing jobs will usually be idle, signifying that they are being considered to run, but a machine has not yet been selected to run the job on.

You may find out further details on one of your own specific jobs using the `-s` flag.

## 8.6 Cancelling a Job - the `llcancel` command

To cancel a job you should use the `llcancel` command:

```
$ llcancel 13f42.124.0
llcancel: Cancel command has been sent to the central manager.
```

## 8.7 Machine Status information - the `llstat` command

Use the `llstat` command to display information about the overall status of the system, including which jobs are running on which frames.

The `llstatus` command can be used to display a much more comprehensive (and arguably, less comprehensible) summary of the machines state.

## 8.8 Notification of job completion

All the sample LoadLeveler script files in this section contain the line:

```
#@ notification = never
```

This means that users are not informed (by email) when a batch job completes.

To receive an email message on a remote machine when a batch job completes, edit the LoadLeveler script and replace:

```
#@ notification = never
```

with:

```
#@ notify_user = email-address@another.machine.somewhere.else
#@ notification = complete
```

where *email-address@another.machine.somewhere.else* is the address to which notification is to be sent. One can also use `always` instead of `complete`.

## 8.9 Using the xloadl GUI

The main xloadl window has three panels. The top panel show the current queue status (equivalent to the output from llq), the centre panel shows the machine status (equivalent to the output from llstatus), and the bottom panel shows messages from LoadLeveler, including the output from LoadLeveler commands executed via xloadl.

To submit a job, click on `File -> Submit a job...` in any of the panels. This pops up a new window, which allows you to select, edit and submit a command file.

To view job details, click on the job in the top panel and then click on `Actions -> Details`.

To cancel a job, select `Actions -> Cancel` in the top panel.

## 9 Interactive Execution

At the time of writing, two LPARs (a total of 32 CPUs) have been set up to support interactive use on the HPCx system.

### 9.1 Purpose of the interactive queue

Please note, the interactive queue is intended for tasks such as debugging, visualisations etc. It is not meant for production jobs. **Production jobs must use the batch queues.**

### 9.2 Using the interactive queue

The main steps to running an interactive parallel job are:

- Creating a LoadLeveler job script for interactive
- Setting up environment variables
- Running the program from the command line

Each of these steps is described in more detail in the following sections.

#### 9.2.1 Creating a LoadLeveler job script for interactive use

All interactive parallel jobs must use a LoadLeveler job command file. This file contains a number of LoadLeveler keyword statements which specify the various requirements of the interactive job. In practice, the script file is very little different from the equivalent background (batch) script file.

Here is an example of a simple LoadLeveler script file for an interactive MPI job:

Here is a sample script for an MPI application using 5 CPUs (the remaining CPUs will be available to other users, be nice):

```
#@ job_type = parallel
#@ job_name = hello
#
#@ cpus = 5
#
```

```

#@ node_usage = shared
#
#@ wall_clock_limit = 00:10:00
#@ account_no = z001
#
#@ notification = never
#
#@ class = inter32_1
#
#@ queue
#

# Submit this script from the command line as follows:
#
# poe ./my_executable -llfile ./my_script.ll

```

In contrast to batch jobs, interactive jobs have to specify a job class. The “#@ class = inter32\_1” statement serves this purpose.

For interactive jobs “#@ node\_usage = shared” has to be specified.

Note that output and error files should **not** be specified in order that the output is directed to the screen. Also note that, unlike for batch jobs, neither poe nor the executable are run from within the scriptfile – these are specified at job submission time (see ‘Running a program interactively’, below).

### 9.2.2 Setting up environment variables

Any environment variables, whether specific to the Parallel Operating Environment or the parallel application should be defined **and exported** from within the interactive session before running poe. Note that, for interactive batch jobs, it is not possible to set environment variables from within the LoadLeveler script (unlike for standard batch jobs).

For example (using /bin/ksh format), to specify the use of shared memory for message passing between tasks on the same node type

```
export MP_SHARED_MEMORY=yes
```

You have to set all environment variables required by your job before moving onto the next step

### 9.2.3 Running a program interactively

Run the program interactively using poe:

```
poe ./my_executable -llfile ./my_script.ll
```

where:

“my\_script.ll” is the name of the file containing the LoadLeveler keyword statements.

### 9.2.4 LoadLeveler attributes for interactive processes

The following LoadLeveler attributes are set for interactive processes (default values in brackets):

**Wall\_clock\_limit**

The maximum wall clock (elapsed time) limit (1 hour)

**Stack\_limit**

The maximum stack size per task instance (20mb)

**Rss\_limit**

The maximum amount of physical memory per task instance (884mb = 26.96gb/num\_tasks)

**Data\_limit**

The maximum data segment size per task instance (884mb-20mb=864mb)

**Core\_limit**

The maximum core size file per task instance (960mb).

### 9.3 Gotchas!

If an interactive job not run and reports the following message

```
LoadLeveler: 2544-870 Step fNN.N.0 was not considered to be run in
this scheduling cycle due to its relatively low priority or because there
are not enough free resources
```

There are fewer free CPUs on in the interactive region than you have asked for.

## 10 Tools

### 10.1 Running The Parallel Debugger `pdbx`

To run the parallel debugger (`pdbx`) the following steps should be carried out:

- Compile a parallel program using the appropriate `mpxx_r` compiler shell script (e.g. `mpcc_r`, `mpxlf90_r`) and specify the `-g` option.
- Create a LoadLeveler job script for interactive use
- Set-up any environment variables
- Load the correct number of instances of the program into the parallel debugger on all LPARs allocated
- Trace the program from within the debugger

The last two of these steps are described in more detail in the next section.

#### 10.1.1 Loading the parallel program

From the command line type:

```
pdbx ./prog.exe -llfile ./llscriptfile -procs N
```

where: `prog.exe`

is the name of the parallel program

`llscriptfile`  
is the name of the file containing the LoadLeveler keyword statements

`-procs N`  
specifies the total number of instances of MPI tasks. `N` should match the `cpus`-keyword in your LoadLeveler script file.

After initialisation the:

```
pdbx(all)
```

prompt should be displayed.

### 10.1.2 Tracing the program instances in the debugger

- Type

```
tasks long
```

at the `pdbx(all)`-prompt, to confirm all the instances are ready to be traced.

For example if 2 instances of the program were loaded the output should be something like:

```
0:Debug ready 11f35 172.31.6.137 0  
1:Debug ready 11f35 172.31.6.137 0
```

- To set a break point in the code for all instances type:

```
stop at linenum
```

where `linenum` denotes the line number in the source code.

- To continue execution type

```
cont
```

- To exit the debugger type `quit`

For more details on the use of the `pdbx` debugger refer to the IBM Parallel Environment for AIX - Operation and Use, Volume 2.

## 10.2 Totalview

Totalview is a powerful sophisticated debugger which enables the debugging, analysis and tuning of serial and parallel programs. Currently, one may run Totalview interactively using up to 32 processors on HPCx.

Before running Totalview you must go through some setup stages which are detailed at:

<http://www.hpcx.ac.uk/support/FAQ/totalview/>

To start Totalview, type:

```
/usr/local/packages/totalview/tv6
```

Totalview has built-in documentation, and is more fully documented at:

<http://www.etnus.com/Support/docs/index.html>

## 10.3 VAMPIR

VAMPIR (Visualisation and Analysis of MPI Resources) is a commercial post-mortem trace visualisation tool from Intel GmbH, Software & Solutions Group, the former Pallas HPC group. It uses the profiling extensions to MPI and permits analysis of the message events where data is transmitted between processors during execution of a parallel program. Event ordering, message lengths and times can all be analysed. The tool comes in two components - VampirTrace and Vampir. VampirTrace is a library which when linked and called from a parallel program, produces an event tracefile. The Vampir tool interprets the event tracefiles and represents the data in a graphical form for the user. he present

In order to run the Vampir/Vampirtrace tools, you will need to set your PAL\_ROOT and PAL\_LICENSEFILE and VT\_ROOT environment variables. For example for a bash shell:

```
export PAL_ROOT=/usr/local/packages/vampir
export VT_ROOT=/usr/local/packages/vampir
export PAL_LICENSEFILE=/usr/local/packages/vampir/etc/license.dat
```

**N.B.** These environment variables will need to be set in any **LoadLeveler batch scripts** you use when you want to create a vampir tracefile. The present license works on up to 512 processors.

### 10.3.1 Tracing your Parallel Application with VampirTrace

**Basic Usage:** Using the basic functionality of VampirTrace for MPI is straightforward: relink your MPI application with the appropriate VampirTrace `-lVT` library, add the environment variables outlined above to your batch script and execute the application as usual.

There are different versions of this library for 32-bit addressing and 64-bit addressing. You have to specify the appropriate library path to pick up the correct version. You also need to add `-lld` library.

For 32-bit addressing:

```
mpxlf90_r -o hello hello.f -L${PAL_ROOT}/lib -lVT -lld
mpxlf90_r -q32 -o hello hello.f -L${PAL_ROOT}/lib -lVT -lld
```

For 64-bit addressing:

```
mpxlf90_r -q64 -o hello hello.f -L${PAL_ROOT}/lib64 -lVT -lld
```

**Instrumented Tracing:** By using keywords, section-specific information can be built into the trace using subroutine calls. Trace calls can be limited to time-critical program sections (in particular this can be useful to limit the size of the tracefile produced). In Fortran code, this involves adding calls to `VTBEGIN` and `VTEND` around the section of interest in the source. The equivalent in C is `VT_begin` and `VT_end`. You will also need to include a header file (`VT.inc` for Fortran and `VT.h` for C). The include path is

```
-I${PAL_ROOT}/include
```

### 10.3.2 Viewing Your Tracefiles with Vampir

The output from VampirTrace files is viewed using Vampir. Simply run the Vampir executable to load the graphical viewer, and load your tracefiles through the menu system.

The vampir executable lives at:

```
/usr/local/packages/vampir/bin
```

or

```
${PAL_ROOT}/bin
```

It can be useful to add this path to your PATH variable.

### 10.3.3 Documentation

Users of the service can access the documentation on the service machine

```
/usr/local/packages/vampir/doc/Vampir-userguide.pdf  
/usr/local/packages/vampir/doc/VT.pdf
```

Further information about Vampir/VampirTrace can be found at:  
<http://www.pallas.com/e/products/vampir/index.htm>

## 10.4 Paraver

Paraver is a performance visualisation and analysis tool that can be used to analyze MPI, OpenMP and Mixed-mode programs on HPCx. Hardware counter profiling is also included. A guide to setting up your jobs for Paraver profiling on HPCx can be found at:

<http://www.hpcx.ac.uk/support/FAQ/paraver.html>

## 10.5 gprof and xprofiler

gprof is a simple text-based utility that provides procedural-level profiling of serial and parallel codes. This helps users to identify how much time is being spent in subroutines and functions. xprofiler generates a graphical display of the performance, and provides application profiling at the source statement level.

Both gprof and xprofiler are very simple to use:

- Compile the code with the `-pg` option, in addition to optimisation flags. If you use xprofiler, using `-g` in addition to the `-pg` option will offer profiling at source line level, however the `-g` will degrade the performance and is incompatible with some optimisation flags (e.g. inlining).
- Run the parallel code as usual.
- Each process will write an additional file to disk named `gmon.out.pid`
- Process the output with gprof or xprofiler where `exec_file` is the name of the compiled executable:

```
gprof exec_file gmon.out.pid
```

or

```
xprofiler
```

(after starting `xprofiler` use the File > Load Files dialogue box to direct it towards the executable file and the `gmon.out.pid` file.)

`gprof` and `xprofiler` facilitate analysis of CPU usage only. They cannot provide other types of profiling information, such as CPU idle, I/O or communication.

The latest version of `xprofiler` (from IBM research) is installed in

```
/usr/local/packages/actc/hpct/bin/xprofiler
```

and offers additional features such as histograms when using profiling at the source line level.

NB. To include LAPACK in your profile, link using `-L/usr/local/lib -llapack_profile`.

Information about `gprof` and `xprofiler` can be found in the of the IBM Parallel Environment for AIX:

Operation and Use, Volume 2 at:

<http://www-1.ibm.com/servers/eserver/pseries/library/spbooks/pe.html>

## 10.6 Hardware Performance Monitor (HPM) Toolkit

The HPM toolkit consists of three components:

- The `hpmcount` utility which may be used in conjunction with existing serial and parallel programs to provide information on execution time (wallclock time), hardware performance counters, hardware metrics and resource utilisation statistics.
- The `libhpm` library and the `hpmviz` utility which must be used in conjunction with each other. The `libhpm` library is used to insert instrumenting code into different parts of the user program. At execution time, similar output to that of `hpmcount` is produced for each instrumented section. The output may be examined using the `hpmviz` utility.  
The `libhpm` library may be used with serial and parallel (MPI, OMP) code written in C, C++ and Fortran.

### 10.6.1 The `hpmcount` utility

Serial program usage:

```
hpmcount -o filename ./program
```

where *program* is the name of the executable and *filename* is the name of the file in which the output will be stored.

Parallel program usage:

Simply specify `hpmcount` in the LoadLeveler script file as follows:

```
poe hpmcount -o filename ./program
```

**Pitfall:** The following versions are **wrong** and will lead to `hpmcount` investigating the performance of `poe` instead of your program:

```
hpmcount -o filename poe ./program  
hpmcount -o filename ./program
```

For MPI and OpenMP programs respectively.

In the parallel case, *filename* is the prefix for a sequence of files (one for each processor or task) named *filename\_taskid.processid*. Each of these files contains summary performance information for each processor.

### 10.6.2 The `libhpm` library and the `hpmviz` utility

In order to be able to use the `hpmviz` utility, the program must first be instrumented with calls to the appropriate `libhpm` routines. The full documentation should be studied, but briefly, for a free format Fortran 90 MPI source file *mpiprogram.f90*:

- insert the line:  

```
#include "f_hpm.h"
```

immediately after your variables declaration.
- Insert the lines:  

```
call f_hpminit(rank, 'mpiprogram')  
call f_hpm_terminate(rank)
```

in the code *once only* (immediately after MPI initialisation, and immediately before exiting MPI, as shown). *mpiprogram* is the name of the executable.
- Instrument the relevant sections of code by bracketing them with the lines:  

```
call f_hpmstart(no, 'name')  
call f_hpmstop(no)
```

where *no* is a unique integer  $0 < no < 100$  and *name* is a name identifying the block being instrumented.
- compile with:  

```
$ mpxlf90_r -qsuffix=f=f90 -qsuffix=cpp=f90 -o mpiprogram mpiprogram.f90  
-I/usr/pmpi/include -L/usr/pmpi/lib -lhpm -lpmapi
```
- Note the `-qsuffix=cpp=f90` flag which processes the header file included earlier.  
Run the program in the usual way, after which a sequence of files with the suffix `.viz` will have been generated which can be examined with `hpmviz`. Each file will contain summary information about each instrumented section of the code.  
In addition, files prefixed `perfhpm` will have been generated for each processor which contain information similar to that generated by the `hpmcount` utility.
- To run `hpmviz`  

```
$ usr/local/packages/actc/hpmtk/bin/hpmviz &
```

and load in the required `.viz` files. Left hand portion of the window will display the statistics for each block. Clicking the block name in the left hand side will highlight the block code in the right hand side of the window.

For OpenMP programs, use the thread-safe version of the library `-lhpm_r` and routines `f_hpmtstart` and `f_hpmtstop`.

For C and C++ programs, drop the `f_` prefix.

A full guide on how to use the HPM toolkit on HPCx is available as technical report HPCxTR0307:  
[http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0307/index.html](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0307/index.html)

The original IBM documentation will be found at:

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/HPM.html>

## 10.7 MPI Trace Tools

In order to gain detailed information about MPI communication times, three trace-wrapper libraries are provided:

- `mpitrace`  
wrappers for low-overhead MPI elapsed-time measurements
- `mpihpm`  
the trace wrappers plus power-4 hpm counter data
- `mpiprof`  
provides elapsed-time call-graph data for MPI routines

To use `mpitrace`:

- Link application code with `-L/usr/local/lib -lmpitrace`, ensuring that this appears on the link line *before* `-lmpi`
- Run the application. By default, each MPI task will create files in the working directory with names: `mpi_profile.pid`
- The `mpi_profile.pid` files contain useful summary information including no. of calls per routine, comms and elapsed time and message size distributions.

The number of output files can be reduced by setting the environment variable `TRACE_SOME` to "yes" or "1".

`mpihpm` provides the same information as `mpitrace` plus Power-4 hpm counter data.

`mpiprof` provides an elapsed-time profile of MPI routines including some call-graph information in order to identify communication time on a per-subroutine basis. To use `mpiprof`, the code must be compiled with `-qtable=full` or `-g` as an additional compiler option.

More information about the `mpitrace`, `mpihpm` and `mpiprof` libraries can be found at:

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/mpitrace>

## 10.8 KOJAK

KOJAK is a set of tools designed to analyse the performance of and find bottlenecks in parallel applications. Information can be gained about the communications and hardware events which occur when the application is run. For full information, please see the documentation and examples in the `/usr/local/packages/kojak/kojak` directory and the web page <http://icl.cs.utk.edu/kojak/>.

To use on HPCx, first a few directives must be added to the source code. At least

```
!POMP$ INST INIT
!POMP$ INST BEGIN(name)
```

for Fortran, or

```
#pragma pomp inst init
#pragma pomp inst begin(name)
```

for C are needed as the first executable statements of the main routine and

```
!POMP$ INST END(name) (Fortran)
#pragma pomp inst end(name) (C)
```

are needed as the last executable statements. `name` can be chosen as the name of the main routine.

Additionally, the user can choose to instrument any additional subroutines or segments of code using the 'begin' and 'end' statements above to gain more clarity (about how events relate to the program structure) at the analysis stage.

Then, if working in 32-bit, add the location of the relevant executables to your path with

```
export PATH=$PATH:/usr/local/packages/kojak/kojak/bin/32
```

If working in 64-bit replace the 32 above with 64.

The application should be built with the name of the compiler preceded by `kinst-pomp`

```
kinst-pomp mpif90 -qsuffix=f=f90 source_code.f90 -o executable.x
```

The executable can then be run in the normal way resulting in the trace file `a.elg` being created in the working directory, which can then be automatically analysed with the command

```
kanal a.elg
```

This will analyse the trace file and display the results in the `cube` browser, which is designed to make it easy to determine performance properties of the application via hierarchical (performance, call and system) trees. A colour coding system helps to identify hot-spots. The manual can be downloaded from <http://icl.cs.utk.edu/kojak/cube>.

## 10.9 ParaView

ParaView is a parallel visualisation application capable of performing many different types of visualisations and supporting many different data types as input. ParaView runs in a client/server environment and has a python scripting interface for batch jobs.

### 10.9.1 Launching the Client

Users can download a client binary to run on their local workstation from <http://www.paraview.org/New/download.html>. If a binary is not available for your particular platform, an X-windows client is available on HPCx. Run it using:

```
/usr/local/packages/paraview/launch_paraview_client
```

Prepare the client to accept a server connection by selecting "File : Connect" and entering `login.hpcx.ac.uk` as the host and "Client / Server (reverse connection)" as the server type. The default port is 11111 and can be left unchanged.

### 10.9.2 Launching the Server

The ParaView server can be run in parallel with up to 32 processors on the interactive queue. Run it using:

```
/usr/local/packages/paraview/launch_paraview_server  
-l <loadleveller_script>
```

The full range of options can be accessed by passing `-h` to the script. Users wishing more control over the launching of the server should read the script and run the relevant executables directly.

An appropriate wall clock limit should be specified in the LoadLeveller script, as if the limit is reached, the server processes will be terminated immediately and any visualisation data that has not been saved to disk will be lost.

Users who are running the client behind a firewall are responsible for ensuring that the connection from `login.hpcx.ac.uk` is accepted. Alternatively, solutions such as VPN or port forwarding over SSH could be used.

For further information on server configuration visit [http://paraview.org/Wiki/Setting\\_up\\_a\\_ParaView\\_Server](http://paraview.org/Wiki/Setting_up_a_ParaView_Server)

### 10.9.3 Python scripting

ParaView provides a Python scripting interface as an alternative to using the GUI. All ParaView functionality is accessed through the `'servermanager'` module, and help can be found by typing `help(servermanager)` in a python shell. Python scripts can be edited, run and tested in the interactive shell available in the GUI under "Tools : Python Shell" or by using the `pvpython` interpreter. Scripts can be executed without user interaction using `pvbatch`, either in serial:

```
/usr/local/packages/paraview/3.3.0/serial/bin/pvbatch
--use-offscreen-rendering <path-to-python-script>
```

or parallel (via a loadleveller script):

```
poe /usr/local/packages/paraview/3.3.0/parallel/bin/pvbatch
--use-offscreen-rendering <path-to-python-script>
```

Note that when using `pvbatch`, ParaView 3.3.0 or later must be used. Also, when running in parallel, the following error message will be displayed: `"vtkXOpenGLRenderWindow (1131a7750): bad X server connection. DISPLAY="`. This is normal and will be removed in a future release.

For further instructions on Python scripting for ParaView visit <http://paraview.org/Wiki/images/f/f9/Servermanager2.pdf>

## 10.10 TAU

TAU (Tuning and Analysis Utilities) is a portable profiling and tracing toolkit. It can profile parallel programs written in FORTRAN, C++ and C which use MPI.

It is installed in `/usr/local/packages/tau` and is available to all users. There are 32 and 64 bit versions installed under `tau-2.17/ibm` and `tau-2.17/ibm64` respectively.

To use TAU simply add `/usr/local/packages/tau/tau-2.17/ibm[64]/bin` to your `$PATH`.

Below is a brief introduction to TAU on HPCx. For full instructions see <http://www.cs.uoregon.edu/research/tau/tau-usersguide.pdf>

### 10.10.1 MPI Profiling

The simplest way to get an application profile using TAU is to use the `tau.poe` wrapper when you run your MPI executable. Simply replace `poe` with `tau.poe` in your LoadLeveller script. When your application exits, one profile file for each process will be written in the current directory.

To view the profile run `paraprof <path_to_profile_files>`. ParaProf is TAU's profile viewing tool. It shows how much time each process has spend in different parts of the code. In this case it will record only the time spent in MPI calls and time spent in user code is bunched together as 'TAU Application'.

### 10.10.2 Instrumentation

To get more information on your code, you can add instrumentation calls to TAU to profile particular functions. TAU also provides a set of wrapper compiler scripts that will automatically instrument all your code using PDT (Program Database Toolkit). The wrapper compilers are called `tau_f90.sh`, `tau_cc.sh`, and `tau_cxx.sh`. They should be used in place of your FORTRAN, C, or C++ compilers in your own makefile. You must then export the `TAU_MAKEFILE` variable and recompile your code to instrument and link with TAU.

e.g.

```
export TAU_MAKEFILE=/usr/local/packages/tau/tau-2.17/ibm[64]/lib/Makefile.tau-mpi-pdt
make clean; make; make install
```

Now run your application and you will again get one profile file for each process, however they will now contain data on all functions, not just MPI calls.

It is possible to generate even more detail by generating trace files instead of profiles. To do this set `TAU_MAKEFILE` to `.../Makefile.tau-mpi-pdt-mpitrace` or `.../Makefile.tau-mpi-pdt-trace` to generate traces of MPI function calls or all function calls respectively. WARNING: full trace files can be very large (1GB/min/process)!

TAU traces can be merged and converted for viewing in other tools e.g. VAMPIR, using the `tau_treemerge.pl` script and `tau_convert` tools. See the TAU user guide for more details.

## 10.11 Subversion (SVN)

The subversion executables can be found in `/usr/local/packages/svn/subversion-1.5.6/bin`.

In order to use Subversion on HPCx you need to have the following settings in your `.subversion` file:

- `http-proxy-host = 148.79.162.144`
- `http-proxy-port = 8080`

## 10.12 NAMD & VMD - Interactive Molecular Dynamics

NAMD is a parallel MD (Molecular Dynamics) code specialised for high performance simulation of large biomolecular systems. VMD (Visual Molecular Dynamics) is a molecular visualisation program for displaying, animating and analysing large biomolecular systems. IMD (Interactive MD) refers to using VMD and NAMD together by connecting VMD to NAMD, providing a method to run the MD simulation interactively.

NAMD can be run on HPCx and viewed and controlled from VMD running on your local machine, using a program called Proxycontrol for the connection, as shown below.

There is a video demonstration here [IMDdemo](#)

- **Enable IMD in NAMD**

Enabling IMD in NAMD is straightforward, and a quick tutorial for IMD can be found online at <http://www.ks.uiuc.edu/Research/vmd/imd/tutorial/>

Add these lines to the NAMD configuration file to enable IMD:

```
IMDon      yes      ;# enable IMD
IMDport    2030     ;# NAMD socket port number, normally a number of (1024-65535)
IMDfreq    1        ;# NAMD data send out frequency
IMDwait    on       ;# blocking wait for VMD connection
```

The underlined 2030 is an arbitrary port normally between 1024 and 65535, which will be used by the NAMD simulation for communication.

- **Start NAMD using the program Proxycontrol**

In the batch file, replace the command

```
/usr/bin/poe ./namd2 input.conf
```

with the command

```
proxycontrol -p 50000 -h 193.62.122.2 -f 65100:peer:2030 /usr/bin/poe
./namd2 input.conf
```

The underlined 2030 is the socket port NAMD will use, and 65100 is the port proxycontrol will open for VMD connection. Now IMD can be used by connecting VMD to HPCx on address [login.hpcx.ac.uk](http://login.hpcx.ac.uk) and port 65100.

- **Load your system in VMD on your local machine**

Start VMD and load a PDB file (or, if you prefer, both the PDB and PSF files) for the same system that is currently running in NAMD.

- **Connect to NAMD**

Open the "IMD" or "Simulations-¿IMD Connect" plugin from the VMD "Extensions" menu (depends on which version you're running), and enter the hostname of the computer running NAMD ([login.hpcx.ac.uk](http://login.hpcx.ac.uk)) and the port that proxycontrol has opened (65100). Click "Connect". After a few seconds, you should see your molecule start to move: you are watching your simulation in real-time!

## 11 Libraries

A number of numerical libraries are available for use by programmers, some are provided by IBM, some are in the public domain. This section gives a very brief summary of some of what is available, and also indicates where more information can be obtained.

### 11.1 IBM Provided Libraries

#### 11.1.1 ESSL

ESSL (Engineering and Scientific Subroutine Library) contains a large number of highly tuned serial numerical routines. It includes the BLAS for basic linear algebra, and also routines that cover such areas as linear equation solving, both dense and sparse, eigensolvers, courier analysis, quadrature, interpolation, random number generation and sorting. As such in many ways it plays a similar role to `libsci` on Cray machines. A PDF document detailing the full capabilities of ESSL and the interfaces to the routines contained in it is available here:

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/am501401.pdf>

Note that to use ESSL `-lessl` must be included on the link line. Further, unlike `libsci`, ESSL typically uses its own proprietary interfaces to these routines, and in particular contains only a very small subset of the LAPACK library.

NB: `-lessl` is a 32- and 64-bit thread-safe library.

### 11.1.2 Parallel ESSL

Parallel versions of ESSL comes in two forms.

The first is the ESSL SMP library. This contains a threaded subset of the ESSL routines, and so can be used to parallelise operations within a shared memory partition. The number of threads employed by the ESSL SMP library is set by the environment variable `OMP_NUM_THREADS` which has a current default value of 32 on HPCx.

To link in the ESSL SMP library use `-lesslsmp`.

The second is the PESSL distributed data version. This contains a subset of the standard PBLAS and ScaLAPACK routines for performing linear algebra (all the ‘work’ routines are included, but some of the utilities are missing), and also routines for FFTs, Fourier analysis and random number generation. (If one requires PESSL, then one must also link in a BLACS library. See the following section for details on BLACS.)

To link use `-lpesslsmp -lblacssmp`. Both these libraries are 32- and 64-bit thread-safe libraries. NB: If your code is purely MPI, then `OMP_NUM_THREADS=1` should be set in your LoadLeveler script, if you link with these libraries, as they also include thread-based parallelism.

There exists another version of PESSL, namely `-lpessl`, which is not thread-safe and only caters for 32-bit addressing. This library is not affected by the `OMP_NUM_THREADS` environment variable.

If the code is a mixed mode code, i.e. MPI between LPARs and OpenMP inside LPARs, one would link with `-lpesslsmp -lblacssmp -lesslsmp`.

Full information for PESSL can be found in PDF format at:

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/am601301.pdf>

### 11.1.3 BLACS

BLACS, or Basic Linear Algebra Communications Subroutines, are similar to MPI and are built on the same layer as MPI, therefore, their performance should be just as good. However, MPI is a more powerful and more versatile communications library.

Two implementations of BLACS are available on HPCx.

Firstly, the IBM implementation of BLACS. There are two IBM BLACS libraries, namely BLACS and BLACSSMP. BLACS, `-lblacs`, is currently only 32-bit addressing and is incomplete. BLACSSMP, `-lblacssmp`, caters for both 32- and 64-bit addressing and is thread-safe.

Secondly, the public implementation of BLACS, built on top of MPI and is located in `/usr/local/lib`. The user also needs to link `-lblacsCinit` or `-lblacsF77init` as well as `-lblacs`.

The public domain BLACS supports both 32- and 64-bit addressing and is thread-safe.

### 11.1.4 MASS

The highly-optimised 'mathematical intrinsics' are available via the MASS package. Please visit the following web page for more information.  
<http://techsupport.services.ibm.com/server/mass?fetch=home.html>

NB: The current MASS libraries are now included, by default, as part of the compiler process.

However, you may wish to use an alternative MASS library. To do this, ensure `-lmass` appears before `-lm` when linking.

The MASS.readme file is located at `/usr/lpp/mass/MASS.readme` on HPCx. The current default version is 3.3, however, versions 3.0 and 4.2 are also available.

## 11.2 Other Libraries

### 11.2.1 LAPACK

LAPACK contains a very large number of routines that perform serial dense linear algebra, and has been ported to a very large number of machines. It achieves high performance by using the BLAS library, which in the case of IBM (and many other manufacturers) has been highly optimised. It also is the interface that libsci uses, so use of LAPACK may aid in porting codes to the new machine, but please see the note below.

More information on LAPACK may be found at

<http://www.netlib.org/lapack/>

and to use it you need `-lessl -llapack` in your link line. This ordering will ensure that the faster ESSL routines will replace the slower LAPACK routines, although, the argument list of the ESSL routines may not match the argument list of the LAPACK routines. Linking with `-llapack -lessl` will also work.

NB. `-llapack` is a 32- and 64-bit thread-safe library.

You also need to add `/usr/local/lib` to your link path.

### 11.2.2 ScaLAPACK

ScaLAPACK is the distributed memory version of LAPACK. As mentioned above, some of the ScaLAPACK routines are included in PESSL (see above), but to aid in porting, the public domain version is provided. More information may be found at

<http://www.netlib.org/scalapack/>

To use the public domain version of ScaLAPACK, use

```
-lessl -lblacs -lblacsF77init -lscalapack
```

You also need to add `/usr/local/lib` to your link path. Note that default version of ScaLAPACK on HPCx is a 32- and 64-bit thread-safe library.

If the ScaLAPACK routine is included in IBM's own PESSL, then you can use this library instead. See the section on PESSL above for details.

### 11.2.3 PLAPACK

Release 3.2 of the Parallel Linear Algebra Package (PLAPACK) is installed on HPCx.

The header files are located at `/usr/local/packages/plapack/INCLUDE` and are included by adding `-I/usr/local/packages/plapack/INCLUDE` to your compile line.

The libraries themselves are located in `/usr/local/packages/plapack` and are included by adding `-L/usr/local/packages/plapack -lPLAPACK` to your compile line.

More information and documentation about PLAPACK can be found at

<http://www.cs.utexas.edu/users/plapack/>.

#### Eigensolvers

A report about the QR and the MR<sup>3</sup>-algorithm based eigensolvers coming with PLAPACK is at

[http://www.hpcx.ac.uk/research/hpc/technical\\_reports/HPCxTR0406.pdf](http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0406.pdf).

If you want to use the beta-version of the MR<sup>3</sup>-algorithm based Eigensolver we recommend to copy `/usr/local/packages/plapack/ParEig-1.2.tgz` into your local directory where you can change the code to your needs. The `README/` files in the directories `Tridiag/` and `Dense` explain in detail of how to use it. You might especially want to change the variables `PRINT`, `CHECK`, `TIME` in `Tridiag/global.h` and `mpiexec`, `nprows`, `npcols`, `nb_distr`, `nb_alg`, `nb_alg2`, `n` and `right` in `Dense/test_sym_eig.c` and also adapt the code to read in your specific matrix. Set `HOME = /usr/local/packages/plapack` and `PLAPACK_ROOT = $(HOME)` in `Dense/Makefile` in addition. Please cite

A Parallel Eigensolver for Dense Symmetric Matrices Based on Multiple Relatively Robust Representations.

Paolo Bientinesi, Inderjit S. Dhillon, Robert A. van de Geijn.

Accepted for publication on SIAM Journal on Scientific Computing, 2003

when using the MR<sup>3</sup>-algorithm based Eigensolver.

An example makefile and C-routines of how to use the QR-Eigensolver can be copied from `/usr/local/packages/plapack/QR.example.tar.gz`, there is no example input file included.

### 11.2.4 FFTW

FFTW (Fastest Fourier Transform in the West) is a set of self-optimising Fourier transform routines which can be faster than those provided in ESSL/PESSL. Serial, threaded and distributed data versions are available. For more information see

<http://www.fftw.org/>

Since the interface is incompatible between FFTW version 2.x and version 3.x, we presently have the versions 2.1.5 and 3.0.1 installed on the service.

The version 2.1.5 of the FFTW library has been installed for both single- and double-precision for the serial, threaded and MPI versions. Further, the libraries are both installed for both 32- and 64-bit compilations.

The header files are located at `/usr/local/packages/fftw/include` and are included by adding `-I/usr/local/packages/fftw/include` to your compile line.

The libraries themselves are located in `/usr/local/packages/fftw/lib` and are included by adding `-L/usr/local/packages/fftw/lib` to your compile line.

The information files are located in `/usr/local/packages/fftw/info`.

For version 3.0.1 the the 32- and 64-bit libraries are installed in two different directories, namely `/usr/local/packages/fftw/fftw3_32` and `/usr/local/packages/fftw/fftw3_64` respectively.

So, to employ, say, the double-precision serial 32-bit FFTW library, one would compile with

```
xlf90_r code.f -I/usr/local/packages/fftw/include \  
-L/usr/local/packages/fftw/fftw3_32/lib -ldfftw
```

Similarly, to employ the 64-bit single-precision serial FFTW library in an MPI code, one would compile using

```
mpxlf90_r -q64 code.f -I/usr/local/packages/fftw/include \  
-L/usr/local/packages/fftw/fftw3_64/lib -lsfftw
```

The required include files are located in `/usr/local/packages/fftw/fftw3_32/include` and `/usr/local/packages/fftw/fftw3_64/include`.

### 11.2.5 HSL

HSL, formerly the Harwell Subroutine library, is a collection of ISO Fortran codes for large scale scientific computation written and maintained by the Numerical Analysis Group at Rutherford-Appleton Laboratory. A large range of problems are addressed, but unlike many of the libraries mentioned elsewhere, sparse equation solving is a particular forte of HSL. More information is available from

<http://www.cse.clrc.ac.uk/nag/hsl/>

To link you need `-lhs12004`, given that `/usr/local/lib` is added to your link path.

### 11.2.6 Parallel HDF5

Parallel HDF5 (Hierarchical Data Format) is software for scientific data management. It includes I/O libraries and tools for analyzing, visualising, and converting scientific data. Parallel HDF5 uses MPI-IO calls for parallel file access. For more information see

<http://hdf.ncsa.uiuc.edu/>

The libraries are in `/usr/local/packages/hdf5/lib`. In addition, an improved version of the gzip library has been installed at `/usr/local/packages/hdf5/zlib/lib`. If you are using gzip compression within HDF5 you are advised to link to this version of the gzip library rather than the default system one. The current version is 1.6.4 which is compiled in 64-bit mode only. The complete information about the installation options can be found in `/usr/local/packages/hdf5/libhdf5.settings` and `/usr/local/packages/hdf5/libhdf5_fortran.settings`

Users may call this library from within a serial code, however, the associated compiler must be the parallel version, i.e. if the code uses `xlf90_r` or `xlcr`, then to use this parallel library, one will need to employ `mpxlf90_r` or `mpxlc_r`, respectively.

Example Makefile:

```
MF=      Makefile  
FC=      mpxlf90_r  
FFLAGS=  -qsuffix=f=f90 -q64 -O3 -qarch=pwr4 -qtune=pwr4 \  
         -I/usr/local/packages/hdf5/include \  
         -L/usr/local/packages/hdf5/lib
```

```

        -L/usr/local/packages/hdf5/lib \
        -I/usr/local/packages/hdf5/lib \
        -L/usr/local/packages/hdf5/zlib/lib \
        -lhdf5_fortran -lhdf5 -lgpfs -lz
LFLAGS= $(FFLAGS)

EXE=    prog.exe
SRC=    prog_withHDF5.f90

#
# No need to edit below this line
#

.SUFFIXES:
.SUFFIXES: .f90 .o

OBJ=    $(SRC:.f90=.o)

.f90.o:
        $(FC) $(FFLAGS) -c -o $(OBJ) $<

all:    $(EXE)

$(EXE): $(OBJ)
        $(FC) $(LFLAGS) -o $@ $(OBJ)

$(OBJ): $(MF)

tar:
        tar cvf $(EXE).tar $(MF) $(SRC)

clean:
        rm -f $(OBJ) $(EXE) core

```

The HDF5 module can then be used by including `use HDF5` in the code.

Some useful HDF5 tools are located in `/usr/local/packages/hdf5/bin`.

### 11.2.7 MPI splitting library

This library allows users to run many small jobs using the same binary from one batch script. The MPI splitting library creates a split communicator for all MPI calls. To use this library, the target code only needs to be linked to the splitting library to override the default MPI library. Several environment variables need to be set in the batch job file which then control the behaviour of the split communicator.

To recompile and link to this library, users should link with the following commands for c:

```
mpicc_r -q64 -o main main.c -L/usr/local/packages/mpisplit/lib -lmpi_split
```

and for Fortran:

```
mpxlf90_r -qsuffix=f=f90 -q64 -o fmain fmain.f90 -L /usr/local/packages/mpisplit/lib -lfo
```

To use the library several environment variables need to be set. `SMPI_GROUP_SIZE` is the number of processors in each sub-group. It should satisfy `#cpus mod group_size = 0`. If this condition is not met, then the group size is set to `#cpus`. As there will be several instances of the application running, each instance will need its own input and output files. `SMPI_DIR_PREFIX` sets the prefix for directory for the IO for each group of processors. The group number (with a ".") is automatically appended to the prefix. So if `SMPI_DIR_PREFIX=Group` is set then the separate directories for each group would be `Group.0`, `Group.1`, etc. The remaining two environment variables deal with the stdout and stderr. `SMPI_STDOUT_REDIRECT` can be set to one of three values. The default is `NO` in which case all the output from each instance of the main program executing will be written to the stdout. The second option is `GROUP` where a separate stdout file for each group is written to the `SMPI_DIR_PREFIX` directory. The third option is `ALL`, in this case each processor writes to a separate file, again in the `SMPI_DIR_PREFIX` but with the rank of the processor appended. Similarly for the stderr.

Below is a sample batch script file, with 32 CPUS split into 4 groups of 8, with a directory `Group.n` for each group for io files, and the stdout and stderr written separately for each group into that directory.

```

#@ shell = /bin/ksh
#
#@ job_name = hello
#
#@ job_type = parallel
#@ cpus = 32
#@ node_usage = not_shared
#
#@ network.MPI = csss,shared,US
#@ bulkxfer = yes
#
#@ wall_clock_limit = 00:10:00
#@ account_no = z001
#
#@ output = 4$(job_name).$(schedd_host).$(jobid).out
#@ error = $(job_name).$(schedd_host).$(jobid).err
#@ notification = never
#
#@ queue

# suggested environment settings:
export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export MEMORY_AFFINITY=MCM
export MP_TASK_AFFINITY=MCM

#
# Above lines are common settings in every batch file
#-----
# Following lines are Splitting Library specific
#

export SMPI_GROUP_SIZE=8

export SMPI_DIR_PREFIX=Group

export SMPI_STDOUT_REDIRECT=GROUP
export SMPI_STDERR_REDIRECT=GROUP

```

poe ./main

### 11.3 Libraries and porting to the IBM

In this section a small number of porting issues are very briefly addressed. Mostly they are about porting from Cray systems to the IBM, but some are more general.

- When using BLAS **do not link with -lblas!**  
`libblas.a` is provided on IBM systems, and while it does contain a BLAS library it is actually only the compiled version of the public domain source, and as such has very low performance. Instead use `-lessl`.
- ESSL interfaces  
As noted above ESSL, BLAS aside, does not have standard interfaces to its routines.
- Cray and IBM default real
  - On Cray systems the default real is 64 bits, while this corresponds to (in FORTRAN 77 terms) double precision on IBM systems. As far as libraries are concerned this affects porting in two ways.

- On Crays when calling BLAS/LAPACK etc. routines the 'S' form is used e.g. SGEMM, SAXPY, SSYEV. On IBM these need to be converted to the 'D' form i.e. DGEMM, DAXPY, DSYEV. Similarly complex routines need to be converted from the 'C' form to 'Z'. This may either be done by hand or by using the `-brename` option to the linker.
- Not only the routine names must be altered, but also any floating point constants in the argument list e.g.

```
Call saxpy( n, 1.0, a, 1, b, 1 )
```

on a Cray becomes something like

```
Call daxpy( n, 1.0d0, a, 1, b, 1 )
```

on IBM. This can be done either as above, but a better method is to use Fortran 90 parameterised types.

- Linking  
As noted above on IBM systems you always need to specify the libraries you are using, which is not always the case on Cray systems.

## 12 Optimisation

The modern hardware of the HPCx system, and its sophisticated compilers mean that many time-honoured code 'hand-crafting' optimisation techniques are irrelevant, and should not be necessary except in the pathological case.

The use of appropriate optimisation switches at compile time can lead to significant (sometimes dramatic) improvements in execution time

Use of the thread-safe re-entrant versions of compilers (`[mp]xlf90_r`, `[mp]xlc_r`, `[mp]xlc_r`, etc) is essential for any code which is required to be thread-safe. There is no performance hit when using these compilers for codes which are not required to be thread-safe, so we recommend their use for *all* programs.

Fortran and C compilers have a similar set of optimisation switches, some of these are described briefly here. Users should refer to the relevant compiler documentation for more detailed information.

- `-q64`

Enables 64-bit addressing which allows better memory management for *all* programs, even if 64-bit addressing is not explicitly exploited in the program.

- `-qarch=pwr4` or `-qarch=pwr5`

The `-qarch` flag specifies the instruction set architecture of the machine, and may take advantage of instructions only available on the specified machine. `pwr4` specifies that this is for the POWER 4 system. `-qarch=auto` may take advantage of instructions only available on the *compiling* machine (or similar machines). Initial test show that compiling with `pwr5` can actually slow codes down. It may be worth experimenting with both `pwr4` and `pwr5` to see which works best for your code.

- `-qhot`

The `-qhot` flag forces the compiler to carry out certain high-order transformations of the source code. For example, loops with similar trip counts and no dependencies may be merged; inner and outer loops may be interchanged so that the innermost loop counter varies most rapidly; in some cases, intrinsic functions will be extracted from loops and computed in batches (vectorisation). `-qhot` is turned on automatically at `-O4` and higher.

- `-qtune=pwr4` or `-qtune=pwr5`

The `-qtune` flag biases the optimisation towards execution on a given machine, `pwr4` specifies that this is for the POWER 4 system. `-qtune=auto` generates code which is automatically tuned for the *compiling* machine (or similar machines). As with the `qtune` flag, `pwr5` may slow down codes. Again, it is worth experimenting with both options to see which works best for your code.

- `-O0`, `-O2`, `-O3`, `-O4`, `-O5`

The `-O` flag is the main compiler optimisation flag and can be specified with a range of values.

- `-O0` (Fortran only)  
Fast compilation and full debugging support.
- `-O2`  
Comprehensive low-level optimisation.  
Global assignment of user variables.  
Elimination of redundant or unused code.  
Machine specific scheduling instructions used..
- `-O3`  
Source manipulation where indicated.  
Inner loop unrolling.  
Software pipelining.  
Whole procedure scoping.
- `-O4`  
High order transformations (see also `-qhot`).  
Inter-procedural analysis (IPA) & inlining.  
Automatic architecture detection & cache tuning.

- -O5  
Aggressive IPA.  
Inlining.

A good mix of compiler switches to use when starting serious optimisation might be:

```
-qhot -qarch=pwr4 -O3 -q64
```

Please note that it is better to compile and debug your code first without optimisation. If your code is in some way non-standard, then optimising may break your code. Optimising your code in the above ways can alter the precise numerical results. If you do not want this to happen then try compiling with the `-qstrict` flag which will overcome this problem, but may result in some performance degradation.

## 13 Applications

HPCx provides a range of pre-compiled scientific application packages on the IBM eServer 575 system. These include:

- AMBER:  
<http://www.hpcx.ac.uk/research/chemistry/amber.html>
- CASTEP:  
<http://www.hpcx.ac.uk/research/materials/castep.html>
- CRYSTAL:  
<http://www.hpcx.ac.uk/research/materials/crystal.html>
- DL-POLY:  
[http://www.hpcx.ac.uk/research/chemistry/dl\\_poly.html](http://www.hpcx.ac.uk/research/chemistry/dl_poly.html)
- GAMESS(UK):  
<http://www.hpcx.ac.uk/research/chemistry/gamess-uk.html>
- GAUSSIAN - not yet available
- H2MOL:  
<http://www.hpcx.ac.uk/research/atomic/h2mol.html>
- NEWT:  
<http://www.hpcx.ac.uk/research/engineering/newt.html>
- POLCOMS:  
<http://www.hpcx.ac.uk/research/environment/polcoms.html>

## 14 Support, Training and Documentation

### 14.1 The HPCx Administration Web Site

The HPCx Administrative Web Site can be accessed from <https://www.hpcx.ac.uk/index.jsp>.

## 14.2 The Helpdesk

Comments, Questions and Queries about the service should be directed to the HPCx Helpdesk [support@hpcx.ac.uk](mailto:support@hpcx.ac.uk).

## 14.3 Training

Please check <http://www.hpcx.ac.uk/support/training/index.html> for details on forthcoming HPCx training courses.

# 15 References and Further Reading

Some items in this list are accessible only via the password-protected User Information Pages. These items are marked with an asterisk (\*).

### **HPCx service documentation:**

The most up to date version of this user guide (and other documentation) is available on the web : [here](#).

### **Other useful local links:**

*HPCx Service Home Page:*

<http://www.hpcx.ac.uk/>

*EPCC Home Page:*

<http://www.epcc.ed.ac.uk/>

*EPCC HPF Page:*

[http://www.epcc.ed.ac.uk/computing/training/document\\_archive/hpf/](http://www.epcc.ed.ac.uk/computing/training/document_archive/hpf/)

*EPCC Java Grande Page:*

[http://www.epcc.ed.ac.uk/computing/research\\_activities/java\\_grande/](http://www.epcc.ed.ac.uk/computing/research_activities/java_grande/)

*EPCC Vampir Page:*

<http://www.epcc.ed.ac.uk/vampir/>

Graham P, *OpenMP: A Parallel Programming Model for Shared Memory Architectures*, Edinburgh Parallel Computing Centre, March 1999.

[http://www.epcc.ed.ac.uk/overview/publications/training\\_material/tech\\_watch/99\\_tw/](http://www.epcc.ed.ac.uk/overview/publications/training_material/tech_watch/99_tw/)

### **IBM Product Documentation:**

The starting point for any search for IBM documentation not listed here is the IBM Product Publications,

(<http://www.ibm.com/support/publications/uk/>) page.

An alternative is to go to the IBM Publications Centre,

(<http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US>)

page and initiate a search from there.

Where possible, links given are to printable user guides (as opposed to reference manuals) in pdf format.

*IBM Documentation Library* (Useful IBM documentation online)

[http://publibn.boulder.ibm.com/cgi-bin/ds\\_rslt?lang=en-US](http://publibn.boulder.ibm.com/cgi-bin/ds_rslt?lang=en-US)

*IBM Product Publications* (starting point for searching for all IBM documentation)

<http://publib.boulder.ibm.com/>  
*RS/6000 SP:Practical MPI Programming*  
<http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245380.pdf>  
*ESSL for AIX Library Guide and Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/am501401.pdf>  
*Parallel ESSL for AIX Library Guide and Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/am601301.pdf>  
*IBM Parallel Environment for AIX: Messages*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794401.pdf>  
*IBM Parallel Environment for AIX: MPI Programming Guide*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794500.pdf>  
*IBM Parallel Environment for AIX: MPI Subroutine Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794600.pdf>  
*IBM Parallel Environment for AIX: Hitchhiker's Guide*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794700.pdf>  
*IBM Parallel Environment for AIX: Operations & Use, Volume 1, Using the Parallel Environment*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794800.pdf>  
*IBM Parallel Environment for AIX: Operations & Use, Volume 2, Using the Parallel Environment*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a22794900.pdf> Volume 2 is the main reference for pdbx, xprofiler, PE Benchmark, PCT, UTE and PVT.  
*LoadLeveler for AIX 5L: Using and Administering*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2278810.pdf>  
*LoadLeveler for AIX 5L: Diagnosis and Messages Guide*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/a2278820.pdf>  
*XL Fortran for AIX: User's Guide*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc0949461.pdf>  
*XL Fortran for AIX: Language Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc0949471.pdf>  
*VisualAge C++: Standard C++ Library Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094949.pdf>  
*VisualAge C++: C/C++ Language Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094957.pdf>  
*IBM C for AIX: C/C++ Language Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094958.pdf>  
*Visual Age C++ for AIX: Compiler Reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094959.pdf>  
*C for AIX: Compiler reference*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094960.pdf>  
*Getting Started with C for AIX: Introduction & Installation Guide*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sc094961.pdf>  
*Workload Management with Loadleveler*  
<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sg246038.pdf>  
*AIX Performance Tools Handbook*

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sg246039.pdf>

*The POWER4 Processor Introduction and Tuning Guide*

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/sg247041.pdf>

*MASS Library*

<http://techsupport.services.ibm.com/server/mass>

*Power4 System Microarchitecture*

<http://www.hpcx.ac.uk/support/documentation/IBMdocuments/power4.pdf>

### **Third-party Documentation**

*Porting Programs from the Cray T3Es to the SP (NERSC, Berkely):*

<http://hpcf.nersc.gov/computers/SP/craytoSP.html>

*LoadLeveler: Maui High Performance Computing Centre*

<http://www.mhpcc.edu/training/workshop/loadleveler/MAIN.html>

*LoadLeveler: National Partnership for Advanced Computational Infrastructure (San Diego)*

<http://www.npaci.edu/BlueHorizon/>

*LoadLeveler: Batch system on the IBM p690 Regatta (Parallab, Bergen)*

<http://www.parallab.uib.no/resources/regatta/batch/>

*LoadLeveler: LoadLeveler on Cheetah (Oak Ridge)*

<http://www.ccs.ornl.gov/Cheetah/LL.html>

*Totalview documentation:*

<http://www.etnus.com/Support/docs/index.html>

*Vampir documentation:*

<http://www.intel.com/cd/software/products/asm-na/eng/cluster/tanalyzer/231039.htm>

*MPI Performance Review (Lawrence Livermore Laboratory):*

[http://www.llnl.gov/computing/tutorials/mpi\\_performance/](http://www.llnl.gov/computing/tutorials/mpi_performance/)

*OpenMP Home Page:*

<http://www.openmp.org/>

*OpenMP Fortran Application Program Interface, OpenMP Architecture Review Board, November 2000.*

<http://www.openmp.org/specs/mp-documents/fspec20.pdf>

*OpenMP C and C++ Application Program Interface, OpenMP Architecture Review Board, March 2002.*

<http://www.openmp.org/specs/mp-documents/cspec20.pdf>

*FFTW Home Page:*

<http://www.fftw.org/>

*LAPACK*

<http://www.netlib.org/lapack/>

*SCALAPACK*

[http://www.netlib.org/scalapack/scalapack\\_home.html](http://www.netlib.org/scalapack/scalapack_home.html)

*HSL*

<http://www.cse.clrc.ac.uk/nag/hsl/>