

The  
University  
Of  
Sheffield.

**High  
Performance  
Computing with  
Iceberg.**

**CiCS**  
Mike Griffiths  
Bob Booth  
November 2005  
AP-Unix4

# Contents

<b>1.</b>	<b>REVIEW OF AVAILABLE COMPUTE RESOURCES</b>	<b>3</b>
1.1	GENERAL INFORMATION ABOUT ICEBERG	3
1.2	FACTS AND FIGURES ABOUT ICEBERG	3
1.3	OPERATING SYSTEM	3
1.4	RESOURCES WITH WHITE ROSE GRID AND NATIONAL GRID SERVICE	4
<b>2.</b>	<b>ACCESSING HPC RESOURCES</b>	<b>5</b>
2.1	CONNECTING TO ICEBERG	5
2.2	LOGGING OUT	6
2.3	PASSWORDS	6
<b>3.</b>	<b>SOFTWARE DEVELOPMENT ENVIRONMENT</b>	<b>7</b>
3.1	COMPILERS	7
<b>4.</b>	<b>SCHEDULING JOBS</b>	<b>10</b>
4.1	SUN GRID ENGINE	10
4.2	EXECUTION NODES AND QUEUES	10
4.3	INTERACTIVE COMPUTING	11
4.4	SGE COMMANDS	11
4.5	SUBMITTING JOBS	12
4.6	MULTIPROCESSOR JOBS USING JOB ARRAYS	14
<b>5.</b>	<b>RUNNING &amp; BUILDING MULTIPROCESSOR APPLICATIONS</b>	<b>15</b>
5.1	SHARED MEMORY PROGRAMMING WITH OPENMP	15
5.2	PARALLEL PROGRAMMING WITH MPI	17
<b>6.</b>	<b>RUNNING APPLICATIONS</b>	<b>21</b>
6.1	FLUENT EXAMPLE	21
<b>7.</b>	<b>RESOURCES</b>	<b>23</b>
7.1	USING FUSE TO MOUNT FILE SYSTEMS ON ICEBERG	23

# 1. Review of Available Compute Resources

## 1.1 General Information About Iceberg

The system was supplied by Sun Microsystems and has 2.4GHz AMD Opteron (PC technology) processors, currently the fastest available for computation, at about three times the performance of processors on Titania. The operating system is 64-bit Scientific Linux, which is Redhat based. Iceberg has 160 processors for general use and connection to the White Rose Grid. Eighty of these processors are in 4-way nodes with 16GB main memory coupled by a double speed (4Gb/s) low latency Myrinet network, for parallel and high memory use. A further 80 processors are in 2-way nodes with 4GB main memory, for general high throughput computation use. In addition 160 processors in 2-way nodes are reserved for GridPP (the particle physics Grid). Iceberg is connected to the White Rose Grid.

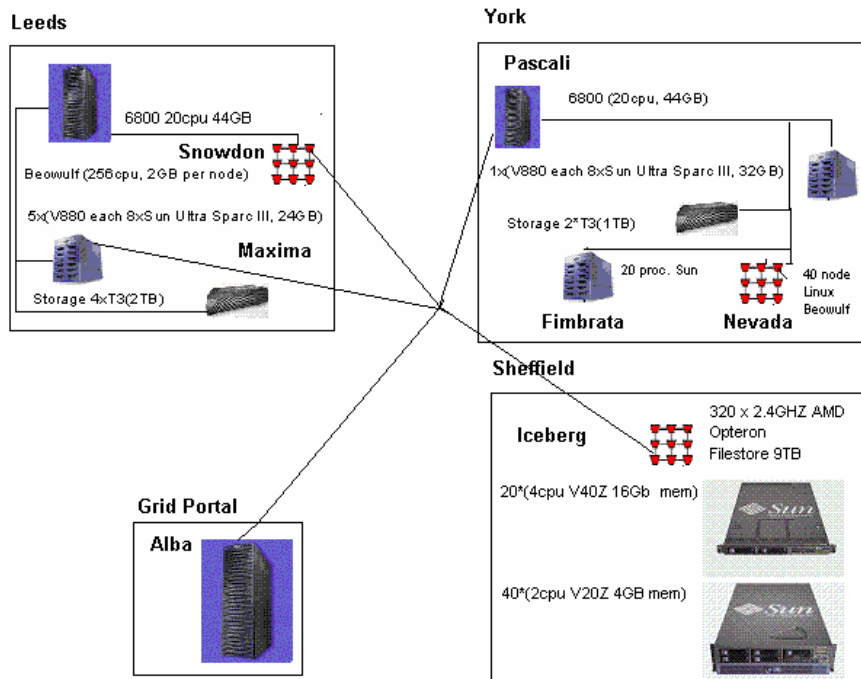
## 1.2 Facts and Figures about Iceberg

- Processors: 320
- Performance: 300GFLOPs
- Main Memory: 800GB
- Filestore: 9TB
- Temporary disk space: 10TB
- Physical size: 8 racks
- Power usage: 50KW

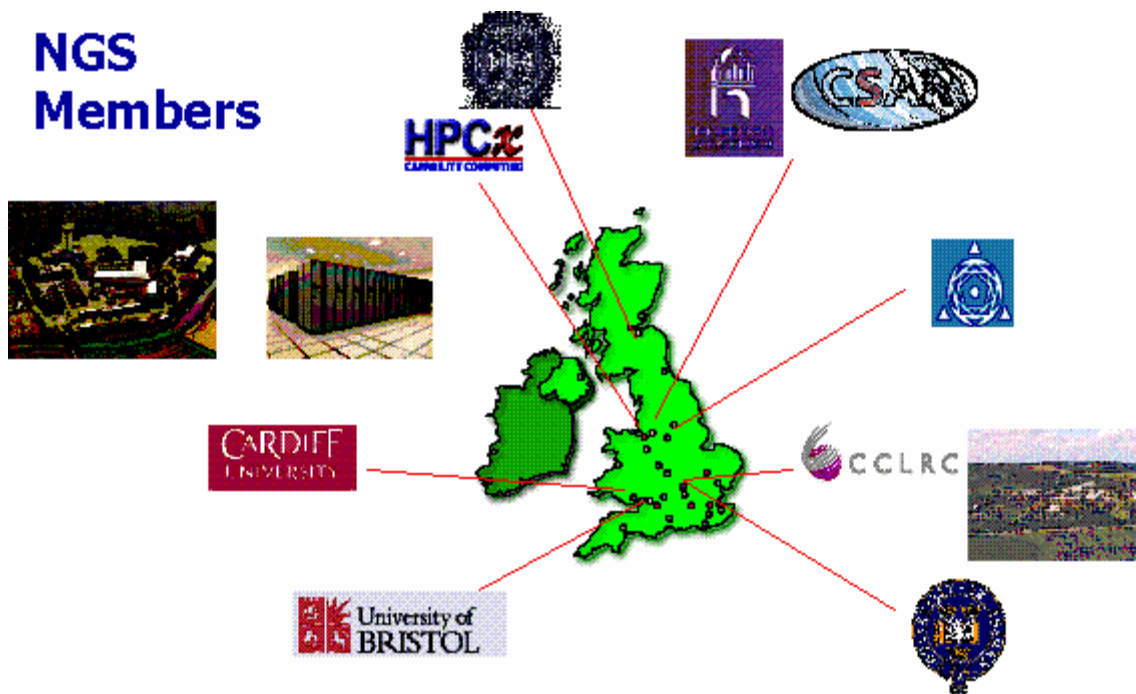
## 1.3 Operating System

- The operating system is 64-bit Scientific Linux, which is Redhat based, installed on all nodes and the head node.
- Has GNU compilers: gcc, g++, g77.
- Has Portland compilers pgcc, pgf90, pgCC.
- Sun Grid Engine for batch scheduling.
- OpenMP for shared memory programming.
- MPICH-GM implementation of the MPI standard.

## 1.4 Resources with White Rose Grid and National Grid Service



White Rose Grid Machines



NGS Resources

## 2. Accessing HPC Resources

Staff requiring a HPCS account should apply to User Registration

- For an account apply to User Registration e-mail: `ucards-reg@sheffield.ac.uk`
- Research postgraduates should ask their supervisor to make the request on their behalf.
- From a PC you can access Iceberg using `ssh`, Exceed software for graphical connection, the Putty client for command line input.
- General instructions for Exceed are available from the CiCS web pages.

### 2.1 Connecting to Iceberg

#### Access Iceberg From a Unix Workstation

- log in to the workstation as usual
- `ssh -X username@iceberg.shef.ac.uk`
- log in with your username and directory password

#### From the Desktop Managed service

- Locate Exceed on the **Start** menu
- under **Applications** and **Internet**, then select **Xsession**
- In the window that appears select **Xterm.sx Xterm Login**
- In the new window that appears, specify Iceberg as the **host** and enter your username and password in the appropriate boxes
- Click **Run** on the menu bar
- You will be logged onto Iceberg and an Xterm window will appear

#### From an Unmanaged Networked PC

If you have used Exceed before you will know what to do. If not,

- select **Xstart** from the **Exceed** menu, under **Start>Programs**.
  - In the dialog box that appears, type Iceberg into the **host** field, and type your username in the **User ID** field.
  - Then, in the Command field type the following command.  
`xterm -ls -display @d em`
  - Save this information by selecting **Save** from the **File** menu.
  - When you have provided a name click **OK**.
  - Once the dialog box is filled in start your X terminal session by clicking the **Run** command in the menu bar.
  - In the next dialog box type in your password and click **OK**.
  - An X terminal window will open and you will be logged into your account.
-

## 2.2 Logging Out

When you have finished using Iceberg you will need to exit cleanly. First close down all running applications. In your Telnet or Xterm window type the command `logout`. If you are running Exceed you will also need to locate the **Exceed** item on the taskbar. Right-click this and select **Close** from the menu.

## 2.3 Passwords

When your account is activated, you will have your usual username and your authentication password. Although your Novell and authentication passwords are separate entities, in practice both passwords are usually identical. If your Novell and directory passwords are different you can synchronise them using the web form at [www.shef.ac.uk/cics/password](http://www.shef.ac.uk/cics/password) If you do not have a Novell password you can change your directory password by e-mail [ucards-reg@sheffield.ac.uk](mailto:ucards-reg@sheffield.ac.uk)

### 3. Software Development Environment

The operating system on iceberg provides full facilities for scientific code development, compilation and execution of programs. The development environment includes debugging tools provided by the Portland test suite, the distributed debugging tool DDT and the eclipse IDE.

#### 3.1 Compilers

C and Fortran programs may be compiled using the GNU or Portland Group. The invoking of these compilers is summarized in the following table:

Language	GNU Compiler	Portland Compiler
C	gcc	pgcc
C++	g++	pgCC
Fortran 77	g77	pgf77
Fortran 90/95		pgf90

All of these commands take the filename containing the code to be compiled as one argument followed by numerous options. Details of these options may be found through the UNIX man facility, e.g. to find details about the Portland f90 compiler use:

```
man pgf90
```

Common options for the portland compilers are shown in the table below

Option	Effect
-c	Compile, do not link.
-o exefile	Specifies a name for the resulting executable.
-g	Produce debugging information (no optimization).
-Mbounds	Check arrays for out of bounds access.
-fast	Full optimisation with function unrolling and code reordering.
-Mvect=sse2	Turn on streaming SIMD extensions (SSE) and SSE2 instructions. SSE2 instructions operate on 64 bit floating point data.
-Mvect=prefetch	Generate prefetch instructions.
-tp k8-64	Specify target processor type to be opteron processor running 64 bit system.
-g77 libs	Link time option allowing object files generated by g77 to be linked into programs (n.b. may cause problems with parallel libraries).

## Compiling a Serial Fortran Program

Assuming that the Fortran 77 program source code is contained in the file `mycode.f`, to compile using the Portland group compiler type:

```
% pgf77 mycode.f
```

In this case the code will be output into the file `a.out`. To run this code issue:

```
% ./a.out
```

at the UNIX prompt. To add some optimization, when using the Portland group compiler, the `-fast` flag may be used. Also `-o` may be used to specify the name of the compiled executable, i.e.:

```
% pgf77 -o mycode -fast mycode.f
```

The resultant executable will have the name `mycode` and will have been optimized by the compiler.

## Compilation of a Serial C Program

Assuming that the program source code is contained in the file `mycode.c`, to compile using the Portland C compiler, type:

```
% pgcc -o mycode mycode.c
```

In this case, the executable will be output into the file `mycode` which can be run by typing its name at the command prompt:

```
% ./mycode
```

## Debugging

The Portland group debugger is a symbolic debugger for Fortran, C, C++ programs. It allows the control of program execution using breakpoints, single stepping and enables the state of a program to be checked by examination of variables and memory locations. The PGDBG debugger is invoked using the `pgdbg` command as follows:

```
pgdbg arguments program arg1 arg2.. argn
```

Where, arguments may be any of the `pgdbg` command line arguments. `program` is the name of the target program being debugged, `arg1`, `arg2`,... `argn` are the arguments to the program.

To get help for `pgdbg` use:

```
pgdbg -help
```

The PGDBG GUI is invoked by default using the command `pgdbg`.

Note that in order to use the debugging tools applications must be compiled with the `-g` switch thus enabling the generation of symbolic debugger information.

## Profiling

The PGPROF profiler enables the profiling of single process, multi process MPI or SMP OpenMP, or Programs compiled with the `-Mconcur` option. The generated profiling information enables the identification of portions of the application that will benefit most from performance tuning. Profiling generally involves three stages:

- compilation
- execution
- analysis (using the profiler)

To use profiling it is necessary to compile your program with the following options indicated in the table below:

Option	Effect
<code>-Mprof=func</code>	Insert calls to produce function level <code>pgprof</code> output.
<code>-Mprof=lines</code>	Insert calls to produce line level <code>pgprof</code> output.
<code>-Mprof=mpi</code>	Link in mpi profile library that intercepts MPI calls to record message sizes and count message sends and receives. e.g. <code>-Mprof=mpi,func</code> .
<code>-pg</code>	Enable sample based profiling.

The PG profiler is executed using the command

```
% pgprof [options] [datafile]
```

Where `datafile` is a `pgprof.out` file generated from the program execution.

## 4. Scheduling Jobs

With the HPCS you have to set up a job on the service, using your computer as usual, but then you will need to queue the job to be executed on the execution node farm. Results will be sent to the head node and can be viewed on your computer.

### 4.1 Sun Grid Engine

The queues are handled by the Sun Grid Engine (SGE), which is an enhanced version of the NQS. The HPCS is running SGE 6, which is an upgrade from the version run on the Grid machines. Jobs are prioritised according to computational (cpu) time. Time is allocated fairly with all users being equal. When you submit your first job it will go to the end of the queue, but the scheduler will quickly move it higher up the queue. As you submit more jobs your usage figure will increase, so the scheduler will become less generous. This means that users who submit less jobs will get a more rapid turnover, but the rapid turnaround of the first job may not be repeated for future jobs.

All jobs must be queued using SGE and must not be run on the head node. Any jobs found running on the head node will be killed off. Compute power is limited and we depend upon the cooperation of users. Failure to comply may lead to suspension.

### 4.2 Execution Nodes and Queues

Each Iceberg runs four queues, one for jobs taking up to 8 hours cpu time and three up to 168 hours. However, if a 168 hour queue is empty, it will start processing 8 hour jobs to maximise efficiency. 20 of the iceberg workers have 4 CPUs and 16GB memory (bigmem.q) 40 of the iceberg workers have 2 CPUs and 4GB memory.

Queue Name	Job size limit	System specifications
short.q	8 cpu hours	
long.q	168 cpu hours	2GB memory per cpu
bigmem.q	168 cpu hours	4GB memory per cpu
parallel.q	168 cpu hours	Jobs requiring multiple cpus

## 4.3 Interactive Computing

Software that runs interactively should not be run on the head node. Instead you must run interactive jobs on an execution node (see 'qsh' command below). The time limit for interactive work is 8 hours. Interactive work on the head node will be killed off.

## 4.4 SGE Commands

### qsub

To submit a job to a queue you will use the `qsub` command. First create a text file containing the SGE commands, then submit this file to the SGE using `qsub filename`. This is used to run programs that you have written and compiled, and to solve models that you have created interactively in software such as Fluent. Detailed information about this command, with examples, is available in the next section of this document, as well as in the man pages on the HPCS.

### qsh

To open a new Xterm or Telnet window on an execution node. This will allow you to start interactive software such as Matlab, or write and compile programs in Fortran, C or C++. These will be run on the execution node, and you can switch back to the original window to set up additional multitasking. Under no circumstances should you run background jobs from a `qsh` session on an execution node. This uses a disproportionate amount of resources. Failure to comply may lead to suspension.

### qdel

To delete any job that you have submitted use the `qdel` command.

### qstat

To get information on your submitted jobs use the `qstat` command. Using the `qstat` command with the `-f` or the `-ext` options will offer different levels of details about your queued jobs. Detailed information about each of these commands is available in man pages. Type `man qstat` or `man qsub` for example. In addition, a man page has been written at Sheffield, giving a detailed overview of the SGE and its uses. Type `man sge` to view.

## 4.5 Submitting Jobs

The following commands are used to submit jobs. For a full explanation see the man page on SGE by typing `man sge` whilst connected to Iceberg. To run a program you will use the `qsub` command. First create a text file containing the SGE commands. The first line should be a `#` character then you can put in a line containing the command for the job to be run.

In the following example, the SGE script file `job1` will execute a program called `prog1`, reading data from a file `prog1.dat` and sending results to `prog1.res`.

```
# prog1 < prog1.dat > prog1.res
```

To run this job type:

```
qsub [options] job1
```

where `job1` is the text file containing the SGE commands. By adding options specified in the man pages, you can have SGE send you messages when your job begins and ends. To receive messages about your job use:

```
#$ -m b sends a message when the job begins
```

```
#$ -m e sends a message when the job ends
```

```
#$ -m be sends messages when the job begins, and when it ends
```

You will need to specify your email address using the command

```
#$ -M email@address
```

You should also estimate the cpu time needed to run the job. To find the time needed for a job, run the job overnight with an eight-hour cpu time, then check the output files for the cpu time taken. Time needed is expressed as hours:minutes:seconds, for a job expected to complete within one-hour cpu time you would add the line

```
#$ -l h_cpu=01:00:00
```

The following SGE script file runs the program `prog1`, expected to complete within an hour, and sends job begins and job ends messages to `b.booth@sheffield.ac.uk`.

```
#
#$ -l h_cpu=01:00:00
#$ -m be
#$ -M b.booth@sheffield.ac.uk
prog1 < prog1.dat > prog1.res
```

Commonly used options with the SGE `qsub` command are shown in the table below.

Option	Description
<code>-l h_rt=hh:mm:ss</code>	The wall clock time. This parameter must be specified, failure to include this parameter will result in the error message: "Error: no suitable queues".
<code>-l h_vmem=memory</code>	Sets the limit of virtual memory required (for parallel jobs per processor).
<code>-help</code>	Prints a list of options.
<code>-pe mpich-gm np</code>	Specifies the parallel environment to be handled by the Score system. <code>np</code> is the number of nodes to be used by the parallel job. Please note: this is always one more than needed as one process must be started on the master node, which, although does not carry out any computation, is necessary to control the job.
<code>-cwd</code>	Execute the job from the current working directory; output files are sent to the directory from which the job was submitted, not to the user's home directory.
<code>-m be</code>	Send mail at the beginning and at the end of the job to the owner.
<code>-S shell</code>	Use the specified <code>shell</code> to interpret the script rather than the C shell (default).
<code>-masterq iceberg.q</code>	Specifies the name of the master scheduler as the master node ( <code>iceberg</code> ).
<code>-v</code>	Export all environment variables to all spawned processes.

---

## 4.6 Multiprocessor Jobs Using Job Arrays

SGE provides a particularly useful way of utilising multiple processors using so called task arrays. This approach is particularly useful for high throughput tasks where there is very little coupling between each of the processors.

This approach is beneficial when it is necessary to conduct parametric studies to optimize various parameters in experiments. Computing the optimum solution involves repeated execution of same set of operations with a range of values for the input data. The Sun Grid Engine (SGE) has a job array option, enabling the submission of all the jobs with one command file and lets the scheduler execute the jobs as and when resources are available. It is necessary to insert an instruction of the form:

```
#$ -t lower-upper:interval
```

Into the SGE command file. The `-t` option defines the task index range, where lower is the lower index, upper the higher index and interval the interval of the jobs. SGE runs your executable once for each number specified in the task index range and it generates a variable, `$SGE_TASK_ID`, which your executable can use to determine its task number each time it is run. It can select input files or other options to run according to its task index number. Below is an example, which demonstrates the usage of the job array facility. In this example, the executable is a script but it could be a C or Fortran program which uses the `getenv` function to get the value of the variable `$SGE_TASK_ID`.

```
#!/bin/sh
# -N TestJobArray
# -cwd
# -t 1-4:2
echo "Task id is $SGE_TASK_ID"
if [ -e $HOME/JOBARRAY/data$SGE_TASK_ID.in ]; then
while read file
do
echo "hello $file"
done < $HOME/JOBARRAY/data$SGE_TASK_ID.in
fi
C Program Example:
#include <stdio.h>
int main(int argc, char *argv[])
{
char* TestID = new char [400];
sprintf(TestID, "%s ", getenv("SGE_TASK_ID"));
printf(" Test ID is %s \n " , TestID);
return 0;
}
```

---

## 5. Running & Building Multiprocessor Applications

### 5.1 Shared Memory Programming with OpenMP

Source code containing OpenMP compiler directives can be compiled on symmetric multiprocessor nodes such as the dual processor Sun V20Z and quad processor V40Z boxes that make up iceberg. SMP source code is compiled using the PGI compilers with the `-mp` option. To compile C, C++, Fortran77 or Fortran90 code, use the `mp` flag as follows

```
pgf77 [compiler options] -mp filename
pgf90 [compiler options] -mp filename
pgcc [compiler options] -mp filename
pgCC [compiler options] -mp filename
```

A simple makefile example is shown here,

```
# Makefile for the molecular dynamics code

#
# C compiler and options
#
CC=    pgcc -fast -mp
LIB=   -lm

#
# Object files
#
OBJ=   main.o \
      dfill.o \
      domove.o \
      dscal.o \
      fcc.o \
      forces_p.o \
      mkekin.o \
      mxwell.o \
      prnout.o \
      velavg.o

#
# Compile
#
md:    $(OBJ)
       $(CC) -o $@ $(OBJ) $(LIB)

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

#
# Clean out object files and the executable.
#
clean:
       rm *.o md
```

The number of parallel execution threads at execution time is controlled by setting the environment variable `OMP_NUM_THREADS` to the appropriate value. For the `cs`h or `tcsh` shell this is set using,

```
setenv OMP_NUM_THREADS=2
or for the sh or bash shell use,
export OMP_NUM_THREADS=2
```

To start an interactive shell with `NPROC` processors enter,

```
qsh -pe openmp NPROC -v OMP_NUM_THREADS=NPROC
```

Note that although the number of processors required is specified with the `-pe` option, it is still necessary to ensure that the `OMP_NUM_THREADS` environment variable is set to the correct value.

To submit a multi-threaded job to sun grid engine it is necessary to submit it to a special parallel environment that ensures that the job occupies the required number of slots. Using the `SGE` command `qsub` the `openmp` parallel environment is requested using the `-pe` option as follows,

```
qsub -pe openmp 2 -v OMP_NUM_THREADS=2 myjobfile.sh
```

The following job script, `job.sh` is submitted using, `qsub job.sh`  
Where `job.sh` is,

```
#!/bin/sh
#$ -cwd
#$ -pe openmp 4
#$ -v OMP_NUM_THREADS=4
./executable
```

---

## 5.2 Parallel Programming With MPI

Iceberg is designed with the aim of running MPI (message passing interface ) parallel jobs, the sun grid engine is able to handle MPI jobs. In a message passing parallel program each process executes the same binary code but executes a different path through the code this is SPMD (single program multiple data) execution. Iceberg uses the mpich-gm implementation provided by myricom for the myrinet fast interconnect at 4GigaBits/second. A simple MPI hello world program is shown below.

```
#include <mpi.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
int rank; /* my rank in MPI_COMM_WORLD */
int size; /* size of MPI_COMM_WORLD */
/* Always initialise mpi by this call before using any mpi
functions. */
MPI_Init(& argc , & argv);
/* Find out how many processors are taking part in the
computations. */
MPI_Comm_size(MPI_COMM_WORLD, &size);
/* Get the rank of the current process */
MPI_Comm_rank(MPI_COMM_WORLD, & rank);
if (rank == 0)
printf("Hello MPI world from C!\n");
printf("There are %d processes in my world, and I have rank
%d\n",size, rank);
MPI_Finalize();
}
```

When run on 4 processors the above program produces the following output,

```
Hello MPI world from C!
There are 4 processes in my world, and I have rank 2
There are 4 processes in my world, and I have rank 0
There are 4 processes in my world, and I have rank 3
There are 4 processes in my world, and I have rank 1
```

To compile C, C++, Fortran77 or Fortran90 MPI code using the portland compiler, type,

```
mpif77 [compiler options] filename
mpif90 [compiler options] filename
mpicc [compiler options] filename
mpiCC [compiler options] filename
```

A simple makefile example is shown here,

```
# MPI Makefile for intrompi examples

.SUFFIXES: .f90 .f .o

MPI_HOME      = /usr/local/mpich-gm2_PGI
MPI_INCLUDE   = $(MPI_HOME)/include

ARCH          =
COMM          =

LIB_PATH      =
LIBS          =

#
# C compiler and options
#
CC            = ${MPI_HOME}/bin/mpicc
CLINKER       = ${CC}
COPTFLAGS    = -O -fast

F90           = ${MPI_HOME}/bin/mpif90
FLINKER       = $(F90)
FOPTFLAGS    = -O3 -fast

LINKER=${CLINKER}
OPTFLAGS=${COPTFLAGS}

#
# Object files
#
OBJ=          ex1.o
#dfill.o \
#domove.o \
#          dscal.o \
#
#velavg.o

#
# Compile
#
ex1:          $(OBJ)
              $(CC) -o $@ $(OBJ) $(LIBS)

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

#
# Clean out object files and the executable.
#
clean:
              rm *.o ex1
```

To submit an MPI job to sun grid engine it is necessary to use the mpich-gm parallel environment that ensures that the job occupies the required number of slots. Using the SGE command `qsub` the mpich-gm parallel environment is requested using the `-pe` option as follows,

```
qsub -pe mpich-gm 4 -q parallel.q myjobfile.sh
```

---

The following job script, `job.sh` is submitted using, `qsub job.sh`  
Where `job.sh` is,

```
#!/bin/sh
#$ -cwd
#$ -pe mpich-gm 4
#$ -q parallel.q
# SGE_HOME to locate sge mpi execution script
#$ -v SGE_HOME=/usr/local/sge6.0
$SGE_HOME/mpi/myrinet/sge_mpirun mpiexecutable
```

The MPI execution script provided by SGE sets the number of processors and a machinefile used to specify that machine configuration the MPI job.

The SGE script uses the PGI mpirun implementation for the mpich-gm parallel environment. Using this executable directly the job is submitted using `qsub` in the same way but the scriptfile `job.sh` is,

```
#!/bin/sh
#$ -cwd
#$ -pe mpich-gm 4
#$ -q parallel.q
# MPIR_HOME from submitting environment
#$ -v MPIR_HOME=/usr/local/mpich-gm2_PGI
$MPIR_HOME/bin/mpirun -np 4 -machinefile $TMPDIR/machines
mpiexecutable
```

The following points should be noted:

1. Number of slots required and parallel environment must be specified using `-pempich-gm nslots`
2. The correct SGE queue set up for parallel jobs must be specified using `-q parallel.q`
3. The job must be executed using the PGI mpich-gm implementation of mpirun. Note also:
  - 3.1. Number of processors is specified using `-np nslots`
  - 3.2. Specify the location of the machinefile used for your parallel job, this will be located in a temporary area on the node that SGE submits the job to.

The downside to message passing codes is that they are harder to write than scalar or shared memory codes. The system bus on a modern cpu can pass in excess of 4Gbits/sec between the memory and cpu. A fast ethernet between PC's may only pass up to 200Mbits/sec between machines over a single ethernet cable and this can be a potential bottleneck when passing data between compute nodes.

The solution to this problem for a high performance cluster such as iceberg is to use a high performance network solution, such as the 4Gbit/sec interconnect provided by myrinet. The availability of such high performance networking makes possible a scalable parallel machine.

For example, using 10 cpus, a well written piece of code would expect to run approximately 10 times faster. However, it is important to take into account Amdahls law. This law states that the speedup of a parallel algorithm is

---

effectively limited by the number of operations which MUST be performed sequentially, i.e its Serial Fraction.

## 6. Running Applications

For a current software list with details see the CiCS web pages at:  
[www.shef.ac.uk/cics/services/network/hpcs](http://www.shef.ac.uk/cics/services/network/hpcs)

To run the modelling applications use the `qsh` command to open an xterm and then run the software interactively. For example, `qsh` opens an xterm window, then type `fluent` to run Fluent. Build the model interactively in GUI mode, and save the solving commands into a text file. When you are ready to solve your model, submit the file of solve commands to SGE using the `qsub` command.

To write and execute your own programs use the `qsh` command to access an Iceberg. Develop and compile your program on the Iceberg. To run the compiled program, submit it to SGE using the `qsub` command as outlined above. The following section gives an example SGE script file.

### 6.1 Fluent Example

Fluent jobs must ran on the execution nodes rather than on the head node. Fluent users must start Fluent up in one of the following two ways depending on whether they need interactive or batch operation.

#### Interactive Fluent Users

Type `qsh` to start up an interactive session on one of the execution nodes.

Type `fluent` to start up fluent.

#### Batch Fluent Users

Use the `runfluent` command to submit your fluent job onto one of the execution nodes.

Usage:

```
runfluent [2D,2DDP,3D or 3DDP] fluent_journal_file -time  
hh:mm:ss [-nq] [-t nprocs][fluent_params]
```

Where: *fluent\_journal\_file* is the file containing the fluent6 commands.

*hh:mm:ss* is the cpu time needed in hours:minutes:seconds

`-nq` is an optional parameter to submit without confirming

`-t nprocs` optionally only needed for parallel jobs to specify the number of processors.

*fluent\_params* (optional) any parameter not recognised will be passed onto the fluent startup script.

**Example:**

```
runfluent6 3D nozzle.jou -time 00:30:00
```

nozzle.jou is essentially a sequence of Fluent Commands you would have entered by starting fluent in non-gui mode

Here is an example (.jou) journal file:

```
/file/read-case test.cas  
/file/read-data test.dat  
/solve iter 200  
/file/write-data testv5b.dat  
yes  
/exit  
yes
```

Note that there can be no graphics output related commands in the journal file as the job will be run in batch mode. To help you prepare a Fluent Journal file (during an interactive `qsh` session) you can startup an interactive fluent session in non-gui mode by typing

```
fluent -g
```

## 7. Resources

There are three areas of filestore available on Iceberg.

- A permanent, secure, backed up area in your home1 directory
- data directory
- temporary, secure, but not backed up area in the scratch directory.

Storage allocations for each area are as follows:

- Your home1 directory has a filestore of 950 MB, but you can store 1 GB for short periods.
- If you change directory to /data or /scratch you will see a directory labelled by your username.
- In /data or /scratch you can store 950 MB of files and up to 1 GB of files for short periods.

Note:

- The scratch area is not backed up.
- Any files in the scratch area will be automatically deleted if they have not been accessed for thirty days.
- This thirty-day limit may change during holiday periods, check the news for details.

### 7.1 Using FUSE to Mount File Systems on Iceberg

FUSE is a kernel module and library enabling users to create "user-level" filesystems. This means that iceberg users can now mount filesystems from their local machine (or other White Rose Grid accounts) onto iceberg using SSH. Any machine you wish to mount using fuse must have an SSH server which supports SFTP. Windows users can set up SSH using [cygwin](#). Below are the instructions for mounting and unmounting remote file systems.

#### Mounting the File System

The username for my maxima account is wrsnod and I wish to mount the home directory of that account. From the command line on iceberg the following commands should be executed. Note that the remote machine will request the password for the requested user.

```
[cslnod@iceberg cslnod]$ mkdir maxima
[cslnod@iceberg cslnod]$ sshfs wrsnod@maxima.leeds.ac.uk:
maxima
Password:
```

Executing the command:

```
ls maxima
```

---

Will list the contents of my remote directory (in this example on maxima).

## Unmounting the File System

To unmount the remote directory simply execute the command:

```
[cslnod@iceberg cslnod]$ fusermount -u maxima
```

## Comments on FUSE

FUSE mounts are only available to iceberg itself and not the worker nodes. They do not get re-exported by NFS. If you need data from your local machine accessible on a worker you will have to copy it to the iceberg head node first. This is a limitation of NFS. It is also important to note that if a remote connection is unused then that connection will be dropped. In this case it is necessary to issue `fusermount` to unmount the connection the user will then need to issue the `sshfs` connection to re-establish the mount point.