

High Performance Technical Computing: Lab Sessions

Irene Moultsas

November 2018

1 Working with the Crescent Cluster

In order to use MPI and run on an HPC cluster, you will be working with Crescent - Cranfield University-wide small-scale research HPC facility. Crescent has a mixture of node types, standard SGI rack mount 2U chassis, with 4 nodes per 2U chassis, an individual SGI 1U node, and individual HP 1U or 2U single nodes. There are 20 SGI chassis in total spread over four racks, 1 SGI 1U node, and 6 HP 1U nodes spread over four racks; and 3 HP 2U nodes in the central rack. The SGI Chassis provide a total of 77 compute nodes, as three of these SGI nodes are used for cluster management. Each chassis based SGI node has two Intel Sandy Bridge CPUs giving 16 CPU cores and 64GB of shared memory, with 8 of these nodes having 128GB of memory. The individual SGI node has two Haswell CPUs giving 32 cores and 256GB of shared memory. Each HP 1U node has two Intel Sandy Bridge CPUs giving 16 CPU cores and 128GB of shared memory. Each HP 2U node has four Sandy Bridge CPUs giving 32 CPU cores and 256GB of shared memory. Taken together these give a total of 1488 available cores.

You will have access to Crescent for the duration of your MSc CSTE studies.

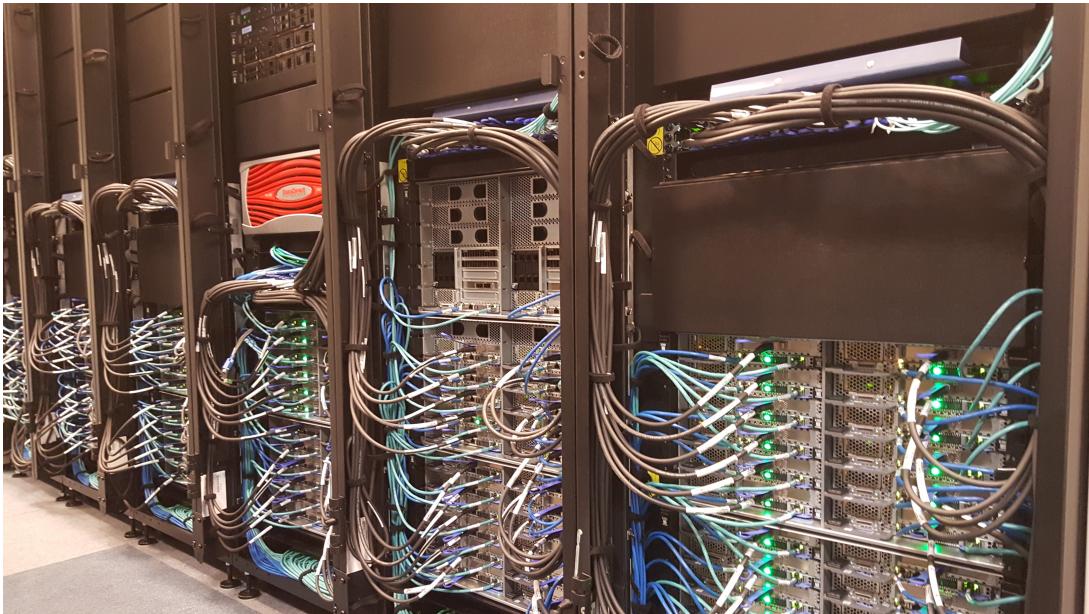


Figure 1: Delta HPC Cluster at Cranfield University

When developing/testing MPI programs some users tend to run small MPI programs interactively on the front node. This is not good practice and should be avoided. In order to run computations which

measure execution time, it is only advisable that you use *PBS* - a batch system which manages jobs on the Crescent and ensures that each of your MPI processes gets allocated to a separate CPU.

1.1 Accessing Crescent

A substantial amount of information regarding research computing at Cranfield University can be found at <https://intranet.cranfield.ac.uk/it/Pages/HighPerformance.aspx>
In order to login to Crescent from the command line:

```
ssh -X s123456@crescent.central.cranfield.ac.uk
```

Where s123456 is your Cranfield University username.

If you are accessing Crescent from a Windows machine you can use putty, or some other similar program.

1.2 Some useful commands

Transferring files

To copy a file, for example `test.c`, from Crescent to current directory:

```
scp s123456@crescent.central.cranfield.ac.uk:test.c ./
```

To copy all files from Crescent to current directory:

```
scp s123456@crescent.central.cranfield.ac.uk:/* ./
```

To copy a file, for example `test.c`, from current directory to Crescent:

```
scp ./test.f s123456@crescent.central.cranfield.ac.uk:..
```

To copy all files from current directory to Crescent:

```
scp ./* s123456@crescent.central.cranfield.ac.uk:..
```

If you are accessing Crescent from a Windows machine you can use winscp3, or some other similar program.

Standard Commands

Some Unix commands you might find useful:

- `ls` - list the files in the current directory
- `ls -laF` - list the files in the current directory including details such as modification date and file size
- `cd A` - change directory to A
- `cd ..` - change directory - go one level up
- `mkdir` - create a directory
- `cp pathA/file.A pathB` - copy file.A from directory given by pathA to directory given by pathB
- `rm file.A` - remove file file.A
- `rmdir A` - remove directory A
- `cat file.A` - print the contents of file.A on the screen

- **less file.A** - open file.A in a viewer (q to exit the viewer)
- **history** - print last commands executed
- **pwd** - print working directory

In general you have access to further information and the arguments of a Unix command by:

- **man command_name**

Editing files on Crescent

You can either run the editor on Crescent remotely, i.e. after logging in via ssh:

```
gedit &
```

Or edit files locally and copy to Crescent using **scp**.

There are more advanced editors for Unix based environments, such as **vi** or **xemacs**, but one needs to get acquainted to the command interface of such editors.

2 MPI

2.1 Compiling MPI programs

In order to use MPI one first needs to set their environment variables accordingly.

To use the latest stable Intel MPI

```
module load intel
```

In order to compile an MPI C++, C, or FORTRAN program, run:

```
mpicxx intel_compiler_options file_name.cpp
```

or

```
mpicc intel_compiler_options file_name.c
```

or

```
mpif90 intel_compiler_options file_name.f90
```

mpicxx, **mpicc** and **mpif90** are in fact wrappers for Intel's compilers. Therefore, all Intel compiler options that you can specify for **icpp**, **icc** or **ifort** are also applicable for the corresponding MPI Intel compilers. For more details

```
man icpp
```

or

```
manicc
```

or

```
mani fort
```

2.2 Running MPI programs

To run an mpi program, say `a.out`, the command is:

```
mpirun -n number_of_processes_to_launch ./a.out
```

As discussed, the scheduling system should be used in order to run a program. To submit an MPI program to the scheduler you will need to use

```
qsub script_name
```

command where `script_name` is the job submission script which should correspond to the following template:

```
#!/bin/bash
##
## MPI submission script for PBS on CRESCENT
## -----
##
## Follow the 6 steps below to configure your job
##
## STEP 1:
##
## Enter a job name after the -N on the line below:
##
#PBS -N mpi_example
##
## STEP 2:
##
## Select the number of cpus/cores required by modifying the #PBS -l select line below
##
## Normally you select cpus in chunks of 16 cpus
## The Maximum value for ncpus is 16 and mpiprocs MUST be the same value as ncpus.
##
## If more than 16 cpus are required then select multiple chunks of 16
## e.g. 16 CPUs: select=1:ncpus=16:mpiprocs=16
## 32 CPUs: select=2:ncpus=16:mpiprocs=16
## 48 CPUs: select=3:ncpus=16:mpiprocs=16
## ..etc..
##
#PBS -l select=1:ncpus=16:mpiprocs=16
##
## STEP 3:
##
## Select the correct queue by modifying the #PBS -q line below
##
## half_hour - 30 minutes
## one_hour - 1 hour
## half_day - 12 hours
## one_day - 24 hours
## two_day - 48 hours
## five_day - 120 hours
## ten_day - 240 hours (by special arrangement)
##
```

```

#PBS -q half_hour
##
## STEP 4:
##
## Replace the hpc@cranfield.ac.uk email address
## with your Cranfield email address on the #PBS -M line below:
## Your email address is NOT your username
##
#PBS -m abe
#PBS -M hpc@cranfield.ac.uk
##
## =====
## DO NOT CHANGE THE LINES BETWEEN HERE
## =====
#PBS -j oe
#PBS -W sandbox=PRIVATE
#PBS -k n
ln -s $PWD $PBS_O_WORKDIR/$PBS_JOBID
## Change to working directory
cd $PBS_O_WORKDIR
## Calculate number of CPUs
export cpus='cat $PBS_NODEFILE | wc -l'
## =====
## AND HERE
## =====
##
## STEP 5:
##
## Load the default application environment
## For a specific version add the version number, e.g.
## module load intel/2016b
##
module load intel
##
## STEP 6:
##
## Run MPI code
##
## The main parameter to modify is your mpi program name
## - change YOUR_EXECUTABLE to your own filename
##

mpirun -machinefile $PBS_NODEFILE -np ${cpus} YOUR_EXECUTABLE

## Tidy up the log directory
## DO NOT CHANGE THE LINE BELOW
## =====
rm $PBS_O_WORKDIR/$PBS_JOBID
#

```

Where you specify your values for the job name, output/error file names, number of processors and executable name. Once the job has been submitted you can use

```
qstat -a
```

command to show the status of the queue and your job's position. A sample script can also be found at
`/apps/examples/pbs/mpi.sub`

3 Tasks

In these lab sessions you will write four MPI programs:

- Simple Hello World Send-Receive test
- Parallel Dot Product computation
- Parallel Pi computation
- Parallel Ping-Pong network timing

You may find the following online MPI references useful:

<http://www.netlib.org/utk/papers/mpi-book/mpi-book.html>
<http://www.llnl.gov/computing/tutorials/mpi/>

3.1 Hello World

- Write an MPI program which:
 - Initialises MPI
 - Obtains the rank of the process
 - For each process prints “Hello World” message and the rank of the process
 - Do not forget to check the return value of every MPI operation to detect errors
- Run this program interactively on 2 processors.
- Run the program through the scheduler on 16 processors

3.2 Send-Receive Test

- Modify your Hello World program so that:
 - Process 0 Sends each process a message with a real array with n variables, where n is the user input
 - Each of the other processes receives the array and prints “Hello World” message, rank of the process and the first element of the array received
 - Process 0 measures the **overall send** time using MPI_WTIME function
 - Do not forget to check the return value of every MPI operation to detect errors
- Run this program interactively on 2 processors.
- Run the program through the scheduler on 16 processors

3.3 Dot Product Computation

Based on the following serial implementation of the dot product computation, write an MPI program to implement the same tasks. Vectors will be distributed among the processes and use MPI_Allreduce when necessary.

```
c  serial_dot.f -- compute a dot product
c          serially -- on a single processor.
c
c  Input:
c      n: vector size
c      x, y: vectors
c
c  Output:
c      the dot product of x and y.
c
PROGRAM serial_dot
implicit none
integer MAX_ORDER, i, n
parameter (MAX_ORDER = 20000)
real dot, x(MAX_ORDER), y(MAX_ORDER)
double precision t1, t2
c
real serial_dot_product
c
call timing(t1)

print *, 'Enter the vector size'
read *, n
call read_vector('the first vector', x, n)
call read_vector('the second vector', y, n)

do i=1, 1000000
    dot = serial_dot_product(x, y, n)
enddo
write (*,*) 'The dot product is ', dot

call timing(t2)
write (*,*) 'user+system time=',t2-t1

stop
end
c
c ****
subroutine read_vector(prompt, v, n)
implicit none
integer i, n
character *20 prompt
real v(n)
c
print *, 'Enter ', prompt, 'data (return after each entry): '
do i = 1, n
    read (*,*) v(i)
enddo
```

```

    return
end

c ****
real function serial_dot_product(x, y, n)
implicit none
integer i, n
real x(n), y(n)
real sum
c
sum = 0.0
do i = 1, n
    sum = sum + x(i)*y(i)
enddo
serial_dot_product = sum

return
end

c ****
subroutine timing (tnow)
double precision tnow
real*4 d(2)

rtnow=etime(d)
tnow=d(1)+d(2)

c      write (*,*) 'rtnow=', rtnow

return
end

```

3.4 Computation of PI

Parallelisation can be achieved based on the split of the integration interval into sub-intervals with each processor computing its part of the integral sum and returning it to the master processor which gathers the result. The sequential code is given by :

```

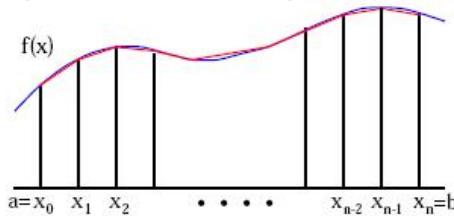
program compute_pi
    integer n, i
    double precision w, x, sum, pi, f, a
! function
    f(a) = 4.d0 / (1.d0 + a*a)
    print *, ' Enter number of intervals: '
    read *,n
! interval size
    w = 1.0d0/n
    sum = 0.0d0
    do i = 1, n
        x = w * (i - 0.5d0)
        sum = sum + f(x)
    end do

```

- Well-known formula:

$$\int_0^1 \frac{4}{1+x^2} dx = \pi.$$

- Numerical integration (Trapezoidal rule):



$$\int_a^b f(x) dx \approx h \left[\frac{1}{2}f(x_0) + f(x_1) + \dots + f(x_{n-1}) + \frac{1}{2}f(x_n) \right].$$

$$x_i = a + ih, \quad h = (b - a)/n, \quad n = \# \text{ of subintervals}.$$

Figure 2: Computation of π

```

pi = w * sum
print *, 'computed pi = ', pi
stop
end program compute_pi

```

- Time the execution of the sequential code using CPU_TIME FORTRAN subroutine (see Intel's FORTRAN reference on the Blackboard).
- Parallelise the algorithm based on the split of the integration interval into n subintervals where n is the number of processes
- Write the following two parallel versions of this program:
 - Version based on the reduction operation MPI_REDUCE.
 - Version based on point-to-point communications MPI_SEND, MPI_RECV.
 - Each version should calculate the execution time using MPI_WTIME function.
- Run these programs interactively on 2 processors.
- Run these programs through the scheduler on 16 processors

3.5 Ping-Pong

- Generate a ping-pong code which exchanges 1000 messages of double arrays of size n , where n is user input, between two processes and measures the total communication time.
- Run the program locally on the Crescent master node
- Run the program through the scheduler

- Run the program using explicit hosts specification:

```
mpirun -n number_of_processors executable_file
```

- Based on the comparison of the communication time difference between different nodes and on the same node comment on the network speed versus the memory speed.