HPC Lab

I. Moulitsas

Fall 2016

1 Working with the Astral Cluster

In order to use MPI and run on an HPC cluster, you will be working with Astral - Cranfield University-wide research HPC facility. Astral has 1280 user-accessible processors available, with two large shared-memory login nodes (front-ends) known as hpclogin-1 and hpclogin-2.
You will have access to the Astral for the duration of your MSc CSTE studies.
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 Astral and ensures that each of your MPI processes gets allocated to a separate CPU.

1.1 Accessing the Astral

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 the Astral from the command line:

```
ssh -X <s123456>@hpclogin-1.central.cranfield.ac.uk
ssh -X <s123456>@hpclogin-2.central.cranfield.ac.uk
```

Where <s123456> is your Cranfield University username.
Once logged on, you can go to your Astral working directory (in scratch space) by typing

```
cd $w
```

The full path to this storage is `/panfs/storage/$USER` or `/scratch/$USER`.

If you are accessing the Astral 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 directory `X` on the Astral Drive to current directory:

```
scp <s123456>@hpclogin-1.central.cranfield.ac.uk:/scratch/s123456/X/test.c ./
```

To copy all files from directory `X` on the Astral Drive to current directory:

```
scp <s123456>@hpclogin-1.central.cranfield.ac.uk:/scratch/s123456/X/* ./
```

To copy a file, for example `test.c`, from current directory to directory `X` on the Astral Drive:

```
scp ./test.c <s123456>@hpclogin-1.central.cranfield.ac.uk:/scratch/s123456/X/
```

To copy all files from current directory to directory `X` on the Astral Drive:

```
scp ./* <s123456>@hpclogin-1.central.cranfield.ac.uk:/scratch/s123456/X/
```

If you are accessing the Astral 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 the Astral

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

```
gedit &
```

Or edit files locally and copy to Astral 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.
A basic editor, called `nano` is also available. Please issue a `module load nano` to load the appropriate module.

2 MPI

2.1 Compiling MPI programs

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

To use Intel MPI

```
module load impi
```

To use Intel Fortran or C

```
module load icc
module load ifort
```

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

```
mpicc <intel c options> <file name>
```

or

```
mpif90 <intel fortran options> <file name>
```

mpicc and mpif90 are in fact wrappers for Intel's icc or ifort compiler, so all options which you can specify for icc or ifort are applicable. For more details

```
man icc
```

or

```
man ifort
```

2.2 Running MPI programs

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

```
mpirun -np <number of processes to start> ./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>
```

command where <script> is the job submission script which should correspond to the following template:

```
#!/bin/bash
##
## MPI submission script for PBS ASTRAL
## ----------------------------------
##
## Follow the 4 steps below to configure. If you edit this from Windows,
## *before* submitting via "qsub" run "dos2unix" on this file - or you will
## get strange errors. You have been warned.
##
## STEP 1:
## The following line contains the job name:
##
#PBS -N 16procs
##
## STEP 2:
##
```

```
## The select line below selects 1 chunk of 16 cpus
## Maximum values for ppn is 16.
##
#PBS -l nodes=1:ppn=16
##
## STEP 3:
##
## Select correct queue:
##
## express    - 2 hours
## short      - 8 hours
## medium     - 1 day
## long       - 3 days
## very_long  - 5 days
## extra_long - 10 days (by special arrangement)
## cismg      - 32 cpu nodes (by special arrangement)
##
#PBS -q express
##
## STEP 4:
##
##
## DO NOT CHANGE the following lines
##------------------------------------------------
#PBS -k oe
##
## Change to working directory
outfl=/panfs/storage/e102081/out.txt
cd $PBS_O_WORKDIR
##
## Find hosts
cat $PBS_NODEFILE > hosts
##
## Set up INTEL environment.
. /etc/profile.d/modules.sh

module load icc
module load ifort
module load impi

. iccvars.sh intel64
. ifortvars.sh intel64
. mpivars.sh

##
##-------------------------------------------------
##
## STEP 5:
##
```

```
## Put correct parameters in mpirun execution line
## below:
##
mpirun -f hosts -np 16 ./a.out 2>&1 $outfl
```

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` command to show the status of the queue and your job's position.

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
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
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

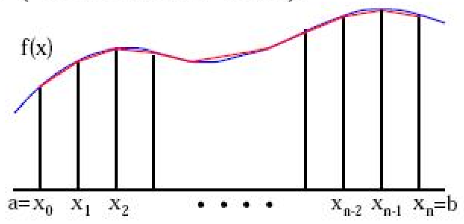
Parallelisation can be achieved based on the split of the integration interval into sub-intervals with each processor computing its part of the integral sum and returning it to the master processor which gathers the result. The sequential code is given by (or you can use the code which you have written during the introduction week) :

```
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):



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

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

Figure 1: 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 using 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 Astral master node
- Run the program through the scheduler

- Run the program using explicit hosts specification:

  ```
  mpirun -np <number of processors> <executable>
  ```

- 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.