Sharpen Exercise: Using HPC resources and running parallel applications

Contents

1 Aims

The aim of this exercise is to get you used to logging into the ARCHER resource, using the command line and an editor to manipulate files, and using the batch submission system.

In this exercise we will be using ARCHER. ARCHER is a Cray XC30 system with a total of 72,192 cores. Although the overall software environment should be familiar to existing users of the HECToR Cray XE6 system, there are several important differences between ARCHER and HECToR. For example, ARCHER uses 12-core 2.7GHz Intel E5-2697 v2 (Ivy Bridge) series processors as opposed to HECToR's 16-core 2.3GHz AMD Opteron (Interlagos) processors. The network is the new Cray Aries interconnect, and ARCHER now supports the Intel compiler suite.

You can find more details on ARCHER and how to use it in the User Guide at:

- http://www.archer.ac.uk/documentation/user-guide/

2 Introduction

In this exercise you will run a simple MPI parallel program to sharpen the provided image.

Using your provided guest account, you will:

1. log onto the ARCHER frontend nodes;

2. copy the source code from a central location to your account;

3. unpack the source code archive;

4. compile the source code to produce an executable file;

5. submit a parallel job using the PBS batch system;

6. run the parallel executable on a compute node using a varying number of processors and examine the performance difference.

7. submit an interactive job.

Demonstrators will be on hand to help you as required. Please do ask questions if you do not understand anything in the instructions - this is what the demonstrators are here for.

3 Instructions

3.1 Log into ARCHER frontend nodes and run commands

You should have been given a guest account ID – referred to generically here as `guestXX` and password. (If you have not, please contact a demonstrator.)

These credentials can be used to access ARCHER by connecting to

```
ssh -X guestXX@login.archer.ac.uk
```

with the SSH client of your choice (`-X` ensures that graphics are routed back to your desktop). Once you have successfully logged in you will be presented with an interactive command prompt.

For more detailed instructions on connecting to ARCHER, or on how to run commands, please see the Appendix.

3.2 Download and extract the exercise files

Firstly, change directory to make sure you are on the "/work" filesystem on ARCHER.

```
guestXX@archer:~> cd /work/y14/y14/guestXX/
```

/work is a high performance parallel file system that can be accessed by both the frontend and compute nodes. **All jobs on ARCHER should be run from the /work filesystem.** ARCHER compute nodes cannot access the /home filesystem at all. Any jobs attempting to use /home will fail with an error.

Use *wget* (on ARCHER) to get the exercise files archive from the EPCC webserver:

```
guestXX@archer:~> wget tinyurl.com/archer030914/Exercises/sharpen.tar.gz
--2013-11-24 13:55:18--  http://tinyurl.com/archer030914/Exercises/sharpen.tar.gz
Resolving tinyurl.com... 195.66.135.249, 195.66.135.241, 195.66.135.248
Connecting to tinyurl.com|195.66.135.249|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www2.epcc.ed.ac.uk/~adrianj/archer030914/Exercises/sharpen.tar.gz [fol
--2013-11-24 13:55:18--  http://www2.epcc.ed.ac.uk/~adrianj/archer030914/Exercises/shar
Resolving www2.epcc.ed.ac.uk... 129.215.62.177
Connecting to www2.epcc.ed.ac.uk|129.215.62.177|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1921882 (1.8M) [application/x-gzip]
```

```
Saving to: 'sharpen.tar.gz'

100%[====================================>] 1,921,882   --.-K/s    in 0.02s

2013-11-24 13:55:18 (86.9 MB/s) - 'sharpen.tar.gz' saved [1921882/1921882]
```

To unpack the archive:

```
guestXX@archer:~> tar -xzvf sharpen.tar.gz
sharpen/
sharpen/C/
sharpen/C-MPI/
.
--snip--
.
sharpen/C/sharpen.c
sharpen/C/sharpen.h
sharpen/C/sharpen.pbs
```

This program takes a fuzzy image and uses a simple algorithm to sharpen the image. A very basic parallel version of the algorithm has been implemented which we will use in this exercise. There are a number of versions of the sharpen program available:

C-MPI Parallel C version using MPI

F-MPI Parallel Fortran version using MPI

C-HYB Parallel C version using a hybrid of MPI and OpenMP

F-HYB Parallel Fortran version using a hybrid of MPI and OpenMP

3.3   Compile the source code to produce an executable file

We will compile the C/MPI parallel version of the code for our example. Move to the *C-MPI* subdirectory and build the program:

```
guestXX@archer:~> cd sharpen/C-MPI
guestXX@archer:~> ls
Makefile  dosharpen.c  fuzzy.pgm   location.h  sharpen.h
cio.c     filter.c     location.c  sharpen.c   sharpen.pbs
guestXX@archer:~> make
```

4

```
cc -g -c sharpen.c
cc -g -c dosharpen.c
cc -g -c filter.c
cc -g -c cio.c
cc -g -c location.c
cc  -o sharpen sharpen.o dosharpen.o filter.o cio.o location.o
```

This should produce an executable file called *sharpen* which we will run on the ARCHER compute nodes. (Note: this executable will not work on the ARCHER frontend nodes as it requires MPI which is dependent on being run on compute nodes.)

For the Fortran MPI version, the process is much the same as above:

```
guestXX@archer:~> cd sharpen/F-MPI
guestXX@archer:~> ls
Makefile        filter.f90  fuzzy.pgm    sharpen.f90
dosharpen.f90  fio.f90      location.c   sharpen.pbs
guestXX@archer:~> make
ftn -g -c sharpen.f90
ftn -g -c dosharpen.f90
ftn -g -c filter.f90
ftn -g -c fio.f90
cc -g -c location.c
ftn  -o sharpen sharpen.o dosharpen.o filter.o fio.o location.o
```

As before, this should produce a *sharpen* executable.

Don't worry about the C file - here it is just providing an easy method for printing out the program's CPU bindings at run time.

3.4   Running a job

As with other HPC systems, use of the compute nodes on ARCHER is mediated by the PBS job submission system. This is used to ensure that all users get access to their fair share of resources, to make sure that the machine is as efficiently used as possible and to allow users to run jobs without having to be physically logged in.

Whilst it is possible to run interactive jobs (jobs where you log directly into the backend nodes on ARCHER and run your executable there) on ARCHER, and they are useful for debugging and development, they are not ideal for running long and/or large numbers of production jobs as you need to be physically interacting with the system to use them.

The solution to this, and the method that users generally use to run jobs on systems like ARCHER, is to run in *batch* mode. In this case you put the commands you wish to run in a file (called a job script) and the system executes the commands in sequence for you with no need for you to be interacting.

3.4.1 Write the PBS job script

Make sure you are logged onto ARCHER and not in an interactive job session. Using the editor of your choice, open a new file. For example:

```
guestXX@archer:~> emacs sharpen_batch.pbs
```

If you are unfamiliar with using a terminal for text editing, detailed instructions on the use of emacs, as well as general commands for examining files, are included in the Appendix.

Add the following lines to the file:

```
#!/bin/bash --login

#PBS -l select=1
#PBS -l walltime=00:05:00
#PBS -A y14
#PBS -N sharpen

# Change to directory that the job was submitted from
cd $PBS_O_WORKDIR

aprun -n 4 ./sharpen
```

The first line specifies which *shell* to use to interpret the commands we include in the script. Here we use the Bourne Again SHell (bash) which is the default on most modern systems. The –login option tells the shell to behave as if it was an interactive shell.

The #PBS lines provide options to the job submission system where "-l select" specifies that we want to reserve 1 compute node for our job - the minimum job size on ARCHER is 1 node (24 cores); the "-l walltime=00:05:00" sets the maximum job length to 5 minutes; "-A y14" sets the budget to charge the job to "y14"; "-N sharpen" sets the job name to "sharpen".

The remaining lines are the commands to be executed in the job. Here we have a comment beginning with "#", a directory change to

$PBS_O_WORKDIR (an environment variable that specifies the directory the job was submitted from) and the aprun command (this command tells the system to run the jobs on the compute nodes rather than the frontend nodes).

3.4.2 Submit the script to the PBS job submission system

Simply use the qsub command with the reservation ID (i.e. (replace $<$resID$>$ in the command below with the reservation ID provided by the trainer) and the job submission script name:

```
guestXX@archer:~> qsub -q <resID> sharpen_batch.pbs
58306.sdb
```

The jobID returned from the *qsub* command is used as part of the names of the output files discussed below and also when you want to delete the job (for example, you have submitted the job by mistake).

3.4.3 Monitoring/deleting your batch job

The PBS command *qstat* can be used to examine the batch queues and see if your job is queued, running or complete. *qstat* on its own will list all the jobs on ARCHER (usually hundreds) so you can use the "-u $USER" option to only show your jobs:

```
guestXX@archer:~> qstat -u $USER

sdb:
                                                        Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
58306.sdb       guest01  standard sharpen       --   1  24     -- 00:15 Q   --
```

if you do not see your job, it usually means that it has completed.

If you want to delete a job, you can use the *qdel* command with the jobID. For example:

```
guestXX@archer:~> qdel 58306.sdb
```

3.4.4 Finding the output

The job submission system places the output from your job into two files: <job name>.o<jobID> and <job name>.e<jobID> (note that the files are only produced on completion of the job). The *.o<jobID> file contains the output from your job *.e<jobID> contains the errors.

```
guestXX@archer:~> cat sharpen.o58306

Image sharpening code running on 4 processor(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Process  0 is on cpu  0 on node nid01133
Process  1 is on cpu  1 on node nid01133
Process  3 is on cpu  3 on node nid01133
Process  2 is on cpu  2 on node nid01133
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 1.552566 seconds
Overall run time was 1.614773 seconds
Application 742518 resources: utime ~6s, stime ~0s, Rss ~24192, inblocks ~13696, outbl
```

3.5 Submit an interactive job and wait for it to start

As well as using the batch system to schedule and run your jobs, it is also possible to submit an *interactive job*. An interactive job allows us to run executables on the ARCHER compute nodes directly from the command line using the *aprun* command as we did in the batch script. This mode of use is extremely useful for debugging and code development as it allows you

to get instant feed back on your executable on the compute nodes rather than having to wait for the end of the job (as is the case for non-interactive or *batch* jobs). It has the disadvantage that you have to be physically logged into the machine to issue the commands.

Submit an interactive job using the command (replace <resID> with the reservation ID provided by the trainer):

```
guestXX@archer:~> qsub -IV -l select=1,walltime=3:0:0 -A y14 -q <resID>
qsub: waiting for job 57939.sdb to start
```

(Note that there is no space between the select and walltime options above as they are both arguments to the "-l" option.) The meanings of the various options are:

-I Interactive job

-V Make the job environment match my current session

-l select=1 Reserve 1 node (24 cores) for this job.

-l walltime=3:0:0 Set a maximum wallclock time of 3 hours for this job

-A y14 Charge the job to the y14 budget

-q <resID> Submit the job in the specified reservation

The job may take a minute to start, when it starts you will be returned to a command prompt in your home directory.

Once you have finished running executables on the compute nodes you can exit the interactive job using the *exit* command (do not do this right now):

```
guestXX@mom3:~> exit
logout

qsub: job 57939.sdb completed
```

If you type exit by mistake you can simply resubmit the job using the qsub command above again.

3.6 Run the parallel executable on a compute node

Firstly, change to the directory where you compiled the code. For example:

```
guestXX@mom3:~> cd /work/y14/y14/guestXX/sharpen/C-MPI
guestXX@mom3:/work/y14/y14/guestXX/sharpen/C-MPI> ls
Makefile  dosharpen.c  filter.o    location.h  sharpen.c  sharpen.pbs
cio.c     dosharpen.o  fuzzy.pgm   location.o  sharpen.h
cio.o     filter.c     location.c  sharpen     sharpen.o
```

Use the *aprun* command to run the sharpen executable using 4 parallel tasks - the '-n' option to *aprun* specifies how many parallel tasks to use:

```
guestXX@mom3:/work/y14/y14/guestXX/sharpen/C-MPI> aprun -n 4 ./sharpen

Image sharpening code running on 4 processor(s)

Input file is: fuzzy.pgm
Image size is 564 x 770

Using a filter of size 17 x 17

Reading image file: fuzzy.pgm
... done

Starting calculation ...
Process  0 is on cpu  0 on node nid02206
Process  1 is on cpu  1 on node nid02206
Process  2 is on cpu  2 on node nid02206
Process  3 is on cpu  3 on node nid02206
... finished

Writing output file: sharpened.pgm

... done

Calculation time was 1.541406 seconds
Overall run time was 1.602274 seconds
Application 738180 resources: utime ~6s, stime ~0s, Rss ~24192, inblocks ~13693, outblo
```

If you try to run the executable using *aprun* outwith an interactive job then you will see an error that looks like:

```
guestXX@archer:~> aprun -n 4 ./sharpen
apsched: request exceeds max nodes, alloc
```

3.7   Viewing the images

To see the effect of the sharpening algorithm, you can view the images using
the `eog` Eye of Gnome program, e.g.

```
guestXX@archer:~> eog fuzzy.pgm
guestXX@archer:~> eog sharpened.pgm
```

Type "q" in the image window to close the program.

3.8   Additional Exercises

Now you have successfully run a simple parallel job on ARCHER, here are
some suggestions for additional exercises.

1. The sharpen program comprises some purely serial parts (all IO is
   performed by rank 0) and some purely parallel parts (the sharpening
   algorithm operates independently on each pixel). Measure the per-
   formance on different numbers of processes. Does the computation
   time decrease linearly? Does the total time follow Amdahl's law as
   expected?

2. Compile the program using different compilers (you will need to type
   `make clean` then `make` to force the program to be rebuilt). Do you
   see any differences in performance?

3. Use the `-N` option to `aprun` to control the number of processes on each
   node of ARCHER, and set it to a value less than 24. Look at the
   log file to see how the processes are allocated to the cores on different
   nodes – is it as you expected? We will see how to control the placement
   of processes more precisely in subsequent lectures.

4 Appendix

4.1 Detailed Login Instructions

4.1.1 Procedure for Mac and Linux users

Open a command line *Terminal* and enter the following command:

```
local$ ssh -X guestXX@login.archer.ac.uk
Password:
```

you should be prompted to enter your password.

4.1.2 Procedure for Windows users

Windows does not generally have SSH installed by default so some extra work is required. You need to download and install a SSH client application - PuTTY is a good choice:

- http://www.chiark.greenend.org.uk/˜sgtatham/putty/

When you start PuTTY you should be able to enter the ARCHER login address (login.archer.ac.uk). When you connect you will be prompted for your user ID and password.

4.2 Running commands

You can list the directories and files available by using the *ls* (LiSt) command:

```
guestXX@archer:~> ls
bin  work
```

You can modify the behaviour of commands by adding options. Options are usually letters or words preceded by '-' or '−'. For example, to see more details of the files and directories available you can add the '-l' (l for long) option to *ls*:

```
guestXX@archer:~> ls -l
total 8
drwxr-sr-x 2 user z01 4096 Nov 13 14:47 bin
drwxr-sr-x 2 user z01 4096 Nov 13 14:47 work
```

If you want a description of a particular command and the options available you can access this using the *man* (MANual) command. For example, to show more information on *ls*:

```
guestXX@archer:~> man ls
Man: find all matching manual pages
 * ls (1)
   ls (1p)
Man: What manual page do you want?
Man:
```

In the manual, use the spacebar to move down a page, 'u' to move up, and 'q' to quit and exit back to the command line.

4.3   Using the Emacs text editor

As you do not have access to a windowing environment when using ARCHER, Emacs will be used in *in-terminal* mode. In this mode you can edit the file as usual but you must use keyboard shortcuts to run operations such as "save file" (remember, there are no menus that can be accessed using a mouse).

Start Emacs with the *emacs* command and the name of the file you wish to create. For example:

```
guestXX@archer:~> emacs sharpen_batch.pbs
```

The terminal will change to show that you are now inside the Emacs text editor:
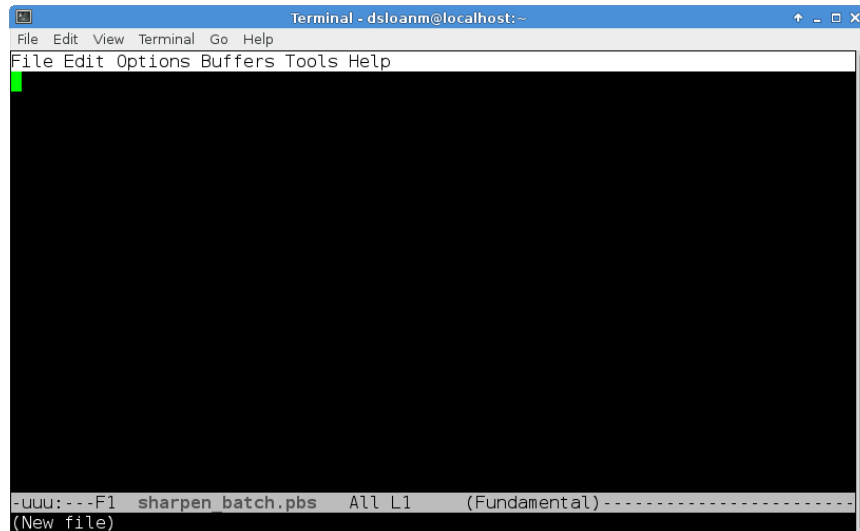
Typing will insert text as you would expect and backspace will delete text. You use special key sequences (involving the Ctrl and Alt buttons) to save files, exit Emacs and so on.

Files can be saved using the sequence "Ctrl-x Ctrl-s" (usually abbreviated in Emacs documentation to "C-x C-s"). You should see the following briefly appear in the line at the bottom of the window (the minibuffer in Emacs-speak):

```
Wrote ./sharpen_batch.pbs
```

To exit Emacs and return to the command line use the sequence "C-x C-c". If you have changes in the file that have not yet been saved Emacs will prompt you (in the minibuffer) to ask if you want to save the changes or not.

Although you could edit files on your local machine using whichever windowed text editor you prefer it is useful to know enough to use an in-terminal editor as there will be times where you want to perform a quick edit that does not justify the hassle of editing and re-uploading.

4.4   Useful commands for examining files

There are a couple of commands that are useful for displaying the contents of plain text files on the command line that you can use to examine the contents of a file without having to open in in Emacs (if you want to edit a file then you will need to use Emacs). The commands are *cat* and *less*. *cat*

14

simply prints the contents of the file to the terminal window and returns to the command line. For example:

```
guestXX@archer:~> cat sharpen_batch.pbs
aprun -n 4 ./sharpen
```

This is fine for small files where the text fits in a single terminal window. For longer files you can use the *less* command. *less* gives you the ability to scroll up and down in the specified file. For example:

```
guestXX@archer:~> less sharpen.c
```

Once in *less* you can use the spacebar to scroll down and 'u' to scroll up. When you have finished examining the file you can use 'q' to exit *less* and return to the command line.