Sharpen Exercise: Using HPC resources and running parallel applications

Andrew Turner, Dominic Sloan-Murphy, David Henty, Adrian Jackson

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 which is a Cray XC30 system with 4920 compute nodes each with 24 cores (a total of 118,080 cores). ARCHER uses 12-core 2.7GHz Intel E5-2697 v2 (Ivy Bridge) series processors; the network is the Cray Aries interconnect.

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 OpenSHMEM 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.**

Use *wget* (on ARCHER) to get the exercise files archive `sharpen.tar.gz` from the ARCHER webserver.

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-SHM Parallel C version using OpenSHMEM

F-SHM Parallel Fortran version using OpenSHMEM

We are most interested in the OpenSHMEM versions - the MPI versions are supplied for reference.

3.3 Compile the source code to produce an executable file

We will compile the C/SHM parallel version of the code for our example. Move to the *C-SHM* subdirectory and build the program: note that you have to *load the shmem module* every time you log on to ARCHER as it is not in the default setup.

```
guestXX@archer:~> module load cray-shmem
guestXX@archer:~> cd sharpen/C-SHM
guestXX@archer:~> make
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 OpenSHMEM which is dependent on being run on compute nodes.)

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

```
guestXX@archer:~> module load cray-shmem
guestXX@archer:~> cd sharpen/F-SHM
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.

If you are Fortran programmer, 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 and to provide a simple timer function.

3.4 Running a job

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.

3.4.1 The PBS job script

You are provided with a simple batch script to run your job:

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

#!/bin/bash --login

# PBS job options (name, compute nodes, job time)
#PBS -N sharpen
#PBS -l select=1
#PBS -l walltime=00:01:00
#PBS -A y14
```

```
# Ensure enough symmetric memory is available to OpenSHMEM
export XT_SYMMETRIC_HEAP_SIZE=100M

# Change to the directory that the job was submitted from
# (remember this should be on the /work filesystem)
cd $PBS_O_WORKDIR

# Launch the parallel job using 4 processes

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=00:01:00" sets the maximum job length to 1 minute; "-A y14" sets the budget to charge the job to "y14"; "-N sharpen" sets the job name to "sharpen".

These options can be specified either on the command line to *qsub* or via a job submission script as above. If you are submitting batch jobs it is often more convenient to add the options to the job script like this rather than type them on the command line every time you submit a job.

There is then a line that is specific to OpenSHMEM regarding symmetric memory - this concept will be explained in the lectures. 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 reserved course queue (the ID will be given to you on the day):

```
guestXX@archer:~> qsub -q RXXXXXX sharpen.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

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, use the *qdel* command with the jobID, e.g.

```
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 PE(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
```

```
Rank 0 on core 0 of node <nid00002>
Rank 1 on core 1 of node <nid00002>
Rank 3 on core 3 of node <nid00002>
Rank 2 on core 2 of node <nid00002>
Starting calculation ...

... finished

Writing output file: sharpened.pgm

... done

Calculation time was 2.688252 seconds
Overall run time was 2.992134 seconds
```

3.5 Viewing the images

To see the effect of the sharpening algorithm, view the images using for example the `display` program; type "q" in the window to close the program.

```
guestXX@archer:~> display sharpened.pgm
```

3.6 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 performance on different numbers of PEs. 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/s̃gtatham/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.

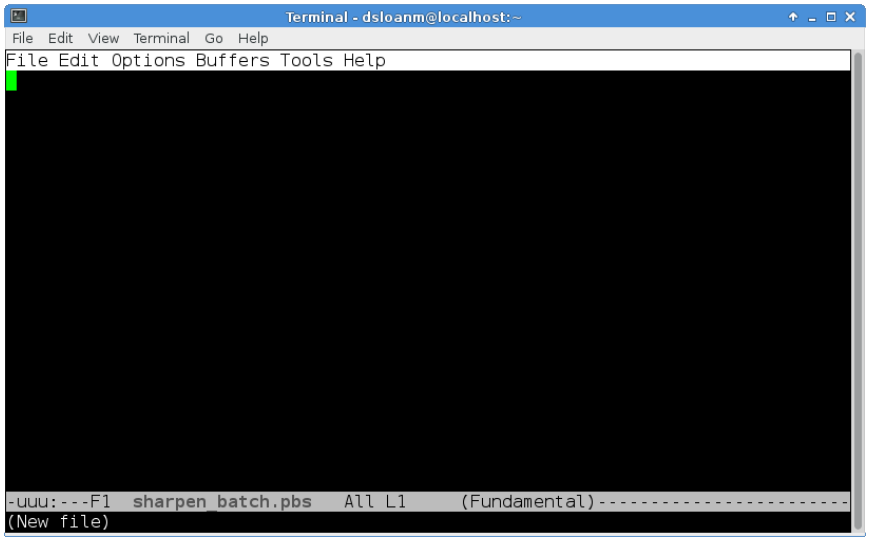
4.3 Using the Emacs text editor

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.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.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*

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

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