Image Sharpening

Practical Introduction to HPC Exercise

(Cirrus version)

epcc

1 Aims

The aim of this exercise is to get you used to logging into an HPC resource, using the command line and an editor to manipulate files, and using the batch submission system. We will be using Cirrus for this exercise. Cirrus is one the UK's EPSRC-funded national Tier-2 HPC services, operated by EPCC at the University of Edinburgh, and is an SGI ICE XA system with a total of 10,080 cores (280 x 36-core nodes).

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

- http://www.cirrus.ac.uk/docs/

2 Introduction

In this exercise you will run a simple program, in serial and parallel, to sharpen the provided image. Using your provided guest account, you will:

1. log onto the Cirrus 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. run a serial job on the login node;
6. submit a serial job to the compute nodes using the PBS batch system;
7. submit a parallel job using the PBS batch system;
8. run the parallel executable using an increasing number of cores and examine the performance improvement.

Please do ask questions in the tutorials if you do not understand anything in the instructions.

3 Instructions

3.1 Log into Cirrus frontend nodes and run commands

You should use your Cirrus guest user name and password to log into Cirrus.

3.1.1 Procedure for Mac and Linux users

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

```
ssh -Y user@login.login.ac.uk
```

You should be prompted to enter your password.

3.1.2 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 Cirrus login address (login.cirrus.ac.uk).

When you connect you will be prompted for your username and password.

You can follow the instructions for setting up PuTTY by watching the introductory video here:

- https://www.youtube.com/watch?v=oVFQg1qFjKQ

By default, PuTTY does not send graphics back to your local machine. We will need this later for viewing the sharpened image, so you should “Enable X11 Forwarding” which is an option in the “Category” menu under “Connection -> SSH -> X11”. You will need to do this each time before you log in with PuTTy.

3.2 Running commands

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

input
```
ls
```
output
```
bin work
```

NB: The first time you do this there will be no files or directories so ’ls’ will return with an empty line.

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

input
```
ls -l
```
output
```
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*:

input
```
man ls
```
output
```
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, ‘u’ to move up, and ‘q’ to quit to the command line.

3.3 Download and extract the exercise files

Before you can compile and run programs on Cirrus you must load appropriate modules.

```
module load mpt
module load intel-compilers-17
```

Use *wget* (on Cirrus) to get the exercise files archive from the EPCC webserver. Material for the course is stored at:

http://www.archer.ac.uk/training/course-material/2018/03/intro-hw/

If you access the material with a web browser it will be downloaded to your laptop, which is not very useful if we want the files on Cirrus. It is easier to copy them straight from the website using the wget utility on Cirrus:

Input

```
wget http://www.archer.ac.uk/training/course-
material/2018/03/intro-hw/exercises/sharpen.tar.gz
```

output

```
--2014-06-27 16:15:42--
http://www.archer.ac.uk/training/course-material/...
Resolving www.archer.ac.uk... 193.62.216.12
Connecting to www.archer.ac.uk|193.62.216.12|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1754173 (1.7M) [application/x-gzip]
Saving to: 'sharpen.tar.gz'
100%[======================================>] 1,754,173   --.-
K/s   in 0.02s
2016-07-10 16:15:42 (107 MB/s) - 'sharpen.tar.gz.1' saved
[1754173/1754173]
```

To unpack the archive:

input

```
tar -xzvf sharpen.tar.gz
```

output

```
sharpen/C-SER/
sharpen/C-SER/filter.c
...
sharpen/F-OMP/dosharpen.f90
sharpen/F-OMP/Makefile
sharpen/F-OMP/fuzzy.pgm
```

If you are interested in the C examples move to the *C-SER* subdirectory; for Fortran, move to *F-SER*. For example:

```
cd sharpen/C-SER
```

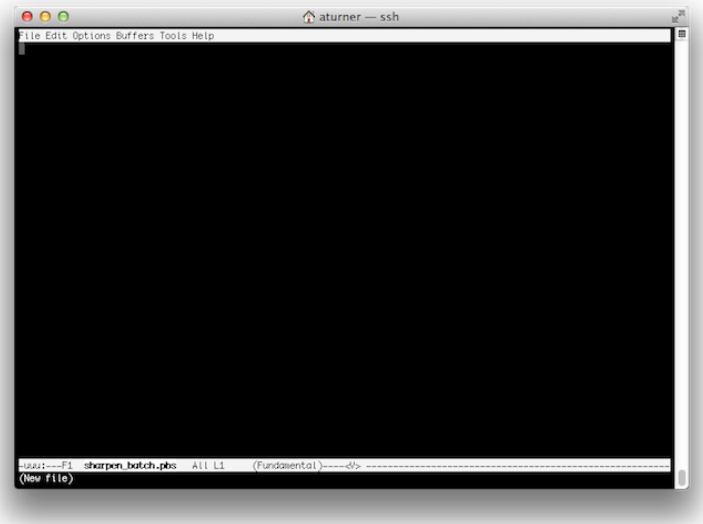
3.4 Using the Emacs text editor

Running interactive graphical applications on Cirrus can be very slow. It is therefore best to use Emacs 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, in this mode there are no menus that can be accessed using the mouse).

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

```
emacs -nw t2sharpen.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 ./t2sharpen.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.

3.5 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:

```
cat t2sharpen.pbs
...
```

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:

```
less Makefile
```

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.

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-SER Serial C version
- F-SER Serial Fortran version
- C-MPI Parallel C version using MPI
- F-MPI Parallel Fortran version using MPI
- C-OMP Parallel C version using OpenMP
- F-OMP Parallel Fortran version using OpenMP

3.6 Compile the source code to produce an executable file

Remember that you must have loaded appropriate modules.
The default compilation settings in the Makefile are for ARCHER – on Cirrus, edit the Makefile and uncomment the lines for Cirrus by deleting the leading "#".

We will first compile the serial version (using the *make* command) of the code for our example.

input
```
ls
```
output
```
cio.c         filter.c   Makefile   sharpen.h    utilities.c
dosharpen.c   fuzzy.pgm  sharpen.c  sharpen.pbs  utilities.h
```

input
```
make
```
output
```
icc -g -DC_SERIAL_PRACTICAL -c sharpen.c
icc -g -DC_SERIAL_PRACTICAL -c dosharpen.c
icc -g -DC_SERIAL_PRACTICAL -c filter.c
icc -g -DC_SERIAL_PRACTICAL -c cio.c
icc -g -DC_SERIAL_PRACTICAL -c utilities.c
icc -g -DC_SERIAL_PRACTICAL -o sharpen sharpen.o dosharpen.o
filter.o cio.o utilities.o
```

This should produce an executable file called *sharpen* which we will run on Cirrus. For the Fortran version, the process is exactly the same as above, except you should move to the *F-SER* subdirectory and build the program there:

input
```
cd sharpen/F-SER
make
```

As before, this should produce a *sharpen* executable. Don't worry about the C file utilities.c – it is just providing an easy method for printing out various information about the program at run time, and it is most easily implemented in C rather than Fortran.

3.7 Running a serial job

You can run this serial program directly on the login nodes, e.g.:

input
```
./sharpen
```
output
```
Image sharpening code running in serial
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 ...
On core 0-31
... finished
Writing output file: sharpened.pgm
... done
Calculation time was 5.579000 seconds
Overall run time was 5.671895 seconds
```

3.8 Viewing the images

To see the effect of the sharpening algorithm, you can view the images using the display program from the ImageMagick suite.

input

```
display fuzzy.pgm
display sharpened.pgm
```

Type “q” in the image window to close the program.

To view the image you will need an X window client installed. Linux or Mac systems will generally have such a program available, but Windows does not provide X windows functionality by default. There are many X window systems available to install on Windows; we recommend Xming available at:

- http://sourceforge.net/projects/xming/

3.9 Running on the compute nodes

As with other HPC systems, use of the compute nodes on Cirrus 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 Cirrus and run your executable there) on Cirrus, 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 Cirrus, 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.

3.9.1 Using PBS job scripts

The file “sharpen.pbs” is for ARCHER; on the Tier-2 Cirrus system use “t2sharpen.pbs”

We will first run the same serial program on the compute nodes. Look at the batch script:

input

```
emacs –nw t2sharpen.pbs
```

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 line *-l select=[nodes:ncpus=72]* is used to request the total number of compute nodes required for your job (1 in the example above).

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 Cirrus is 1 node (36 cores); the "-l walltime=00:01:00" sets the maximum job length to 1 minute; "-A y15" sets the budget to charge the job to "y15"; "-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 then the sharpen executable is run.

3.9.2 Submitting scripts to PBS

Simply use the qsub command:

input

```
qsub –q RXXXXXX t2sharpen.pbs
```

output

```
58306.indy2-login0
```

You will be given the number of the reserved course queue RXXXXXX. 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).

3.9.3 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 Cirrus (usually hundreds) so you can use the "-u $USER" option to only show your jobs:

input

```
qstat -u $USER
```

output

```
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - ----
58306.sdb       user     standard sharpen       --    1  36    --  00:01 Q   --
```

"Q" means the job is queued and "R" that it is running.

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

input

```
qdel 58306
```

3.9.4 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 sharpen.o<jobID> file contains the output from your job sharpen.e<jobID> contains any errors.

input

```
cat sharpen.o58306
```

output

```
Image sharpening code running in serial
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 ...
On core 0
... finished
Writing output file: sharpened.pgm
... done
Calculation time was 5.400482 seconds
Overall run time was 5.496556 seconds
```

4 Running in Parallel

4.1 Running a parallel job on the compute nodes

Repeat the same procedure as above but use the parallel MPI version of the code in *C-MPI* or *F-MPI*. By default, the batch script is set up to run on 4 cores. You should see that that the parallel code is faster than the serial one.

4.2 Timings

If you examine the log file you will see that it contains two timings: the total time taken by the entire program (including IO) and the time taken solely by the calculation. The image input and output is not parallelised so this is a serial overhead, performed by a single processor. The calculation part is, in theory, perfectly parallel (each processor operates on different parts of the image) so this should get faster on more cores.

You should run a number of jobs and fill in Table 1 below: the IO time is the difference between the calculation time and the overall run time; the total CPU time is the calculation time multiplied by the number of cores.

- If you want to run on more than 36 cores then you will need to request more than one node with the ”select=” option to PBS – each Cirrus node only has 36 CPU-cores.

Once you have completed Table 1, you should have the information required to complete Table 2 to compute the speedup of the parallel code overall, of the calculation part and of the IO part. Remember that the speedup is the ratio of runtime at 1 core compared to the runtime at *P* cores.

Look at your results – do they make sense? Given the structure of the code, you would expect the IO time to be roughly constant, and the performance of the calculation to increase linearly with the number of cores: this would give a roughly constant figure for the total CPU time. Is this what you observe?

4.3 Plotting the speedup

You should also produce a speedup plot where you with number of cores (on the x-axis) versus speedup (on the y-axis). Plot all three speedup curves on the same graph.

To aid you with plotting graphs we provide a small Python plotting utility, plotxy.py that you can download from the ARCHER website with:

```
wget http://www.archer.ac.uk/training/courses/2016/PracIntroHPC/util/plotxy.py
```

If you put your data in CSV (comma-separated value) form in a file then it will plot them for you. For example, to plot the speedup data you would create a file with five values per line: the number of cores followed by each of the speedup values for that core count:

```
<cores>, <ideal>, <overall>, <calculation>, <io>
```

You could create this file using Emacs:

```
emacs –nw speedup.csv
```

and the first couple of lines may look like:

```
1,   1.000,  1.000,  1.000, 1.000
2,   2.000,  1.877,  1.920, 1.014
```

Once you have the data in a CSV format you can plot it with the commands:

```
module load anaconda
python plotxy.py speedup.csv speedup.png
```

(The Anaconda packagecontains many useful Python libraries, including those that can be used for plotting data: matplotlib). The utility will save a PNG image containing the plot in the second file name you specify on the command line. You can view the plot using the "display" command as you used for the image:

```
display speedup.png
```

This is a very simple plot to allow you to quickly preview results – if you were plotting for including in a report or paper you would produce something more elaborate (including axis labels, proper series labels,etc.).

4.4 OpenMP code

If you are interested, you can try running the OpenMP code in *C-OMP* or *F-OMP*.

Note that to change the number of cores that the program uses you must set the environment variable OMP_NUM_THREADS. This is already done for you in the PBS script, although you will need to change the actual number yourself.

Note that you can run using multiple threads on the login nodes – you must set OMP_NUM_THREADS before you run, e.g.:

```
export OMP_NUM_THREADS=4
./sharpen
```

The maximum number of cores you can use with OpenMP is 36.

5 Tables

# Cores	Overall run time	Calculation time	IO time	Total CPU time
1				
2				
4				
7				
10				
36				
72				
108				
144				

Table1: Time taken by parallel image processing code

# Cores	Ideal Speedup	Overall speedup	Calculation Speedup	IO Speedup
1				
2				
4				
7				
10				
36				
72				
108				
144				

Table2: Speedup for parallel image processing code