ARCHER2 Advanced OpenMP
Practical Notes

Getting started

Logging on to ARCHER2

You should already have an account on ARCHER2, and should be able to log on to it using

```
ssh -X accountname@login-4c.archer2.ac.uk
```

(replacing accountname with the name of your account on the system) or 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 ARCHER2, or on how to run commands, please see the Appendix.

Download and extract the exercise files

Firstly, change directory to make sure you are on the `/work` filesystem on ARCHER2.

```
cd /work/ta041/ta041/accountname/
```

where ta041 is the name of the project being used for this course. `/work` is a high performance parallel file system that can be accessed by both the frontend and compute nodes. **All jobs on ARCHER2 should be run from the `/work` filesystem.** ARCHER2 compute nodes cannot access the `/home` filesystem at all: any jobs attempting to use `/home` will fail with an error.
Use the following commands (on ARCHER2) to get the exercise files archive from the web and unpack it:

```
cp /work/z19/shared/advomp.tar .
tar xvf advomp.tar
```

Exercise 1: Mandelbrot with worksharing

First, remind yourself how to write some simple OpenMP by parallelising the code in
`AdvOMP/*/Mandelbrot`, where `*` is either `C` or `Fortran90`, using parallel and worksharing loop constructs. Run the code using the script supplied to measure the performance on 1, 2, 4, 8, 16 and 32 threads.

Exercise 2: Mandelbrot with nesting, collapse and tasks

Start from the worksharing loop version of the Mandelbrot example. Try exploiting the parallelism in *both* outer loops using first nested parallel regions, and then the collapse clause.
Now rewrite this example using OpenMP tasks. To begin with, make the computation of each point a task, and use one thread only to generate the tasks. Once this is working, measure the performance. Now modify your code so that it treats each row of points as a task. Modify your code again, so that all threads generate tasks. Which version performs best? Is the performance better or worse that using a loop directive? Note that reduction variables cannot be accessed in tasks, so you will need to find an alternative solution.

Exercise 3: Performance

The `AdvOMP/*/MolDyn/` directory contains a (not very efficient) OpenMP parallel version of a simple molecular dynamics code. Run the code using the script supplied with the script supplied to measure the performance on 1, 2, 4, 8 and 16 threads.
Try to identify the performance bottlenecks and fix them! You can try using the CrayPAT profiling tool as described at `https://docs.archer2.ac.uk/user-guide/profile/` (use CrayPAT rather than CrayPAT-lite). Use the following sequence of module commands to get the environment set up correctly before (re-)building the code:

```
module restore -s PrgEnv-gnu
module restore -s PrgEnv-cray
module load perftools-base
module load perftools
```

Exercise 4: OpenMP + MPI

In this exercise, we will use a 1-D cellular automaton example which models the flow of cars on a road in a very simple way, and implement a mixed OpenMP/MPI version. A working MPI implementation can be found in `AdvOMP/*/Traffic`.
Add parallel loop directives to the two loops inside the main iteration loop: the one which applies the cellular automaton rule, and the one which copies the new state of the road to the old one.
Use the script provides to run different combinations of threads/processes on the same number of processors (e.g. 128 processes and 1 thread, 64 processes and 2 threads, etc.). The number of MPI processes is set in the batch script by `--tasks-per-node` and the number of threads by both `--cpus-per-task` **and** `OMP_NUM_THREADS`
Which combination gives the best performance? How does this compare to the MPI only version?

Extra exercise

Try implementing the code in different hybrid styles (Funneled, Serialized and Multiple).

Exercise 5: Bandwidth and NUMA

The example code can be found in `AdvOMP/*/NUMA`. This is the well-known STREAM benchmark for measuring memory bandwidth. Use the Makefile to compile the code, and run it using different numbers of threads using the supplied batch script.
Does the bandwidth scale linearly with processors? Now try removing the OpenMP loop directive from the initialisation of the arrays. How does the performance change? You can also try using the "wrong" schedule for the loop, or selecting different sets of cores to run on.

Exercise 6: Cache Coherency

The code for this exercise is in `AdvOMP/*/Coherency/` where `*` is either `C` or `Fortran90`.
First of all, take a look at the code `coherency.[f90|c]` and work out what it is doing. Use the Makefile to compile the code. Execute it using two threads on different pairs of cores by submitting the supplied batch script. Try to explain the observed results!

Appendix

Detailed Login Instructions

Procedure for Mac and Linux users

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

```
local$ ssh -X username@login.archer2.ac.uk
Password:
```

you should be prompted to enter your password.

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 ARCHER2 login address:

`login.archer2.ac.uk`

When you connect you will be prompted for your user ID and password.

Running commands

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

```
username@archer2:~> 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*:

```
username@archer2:~> 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*:

```
username@archer2:~> 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.

Using the Emacs text editor

As you do not have access to a windowing environment when using ARCHER2, 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:

```
username@archer2:~> 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.

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:

```
username@archer2:~> 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:

```
username@archer2:~> 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.

Hardware

Each node of ARCHER2 consist of two sockets, each containing a 64-core AMD Epyc Rome processor.

Compiling

The default compilers are the Cray compilers for Fortran 90 and C. To use the AMD or GNU compilers:

```
username@archer2:~> module restore PrgEnv-aocc
```

or

```
username@archer2:~> module restore PrgEnv-gnu
```

The compiler is always invoked with `ftn` or `cc`, but you will need to modify the flags for the different compilers.

Job Submission

To run codes, you should submit a batch job as follows:

```
sbatch --reservation <resnum> scriptfile.sh
```

where `resnum` is the reservation name for the session.
You can monitor your jobs status with the `squeue` command, and jobs can be deleted with `scancel`.