Advanced OpenMP
Exercise Notes

Getting started

Logging on to ARCHER

You should have been given a guest account ID – referred to generically here as `guestXX` and password.
These credentials can be used to access ARCHER using

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

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 ARCHER, 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 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.
Copy the tar file and unpack it with the command

```
tar xvf advomp.tar
```

Exercise 1: Mandelbrot with and without worksharing

First, remind yourself how to write some OpenMP by parallelising the code in `Advanced/*/Mandelbrot`, where `*` is either `C` or `Fortran90`, using parallel and

loop constructs. Run the code using the script supplied to measure the performance on 1, 2, 4, 8, 12 and 24 threads.
Now try writing a version which does not use worksharing loop constructs or reduction variables.

Exercise 2: Mandelbrot with nesting, collapse and tasks

Go back to 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: Cache Coherency

The code for this exercise is in `Advanced/*/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 by submitting the supplied batch script with the command `qsub -q <resnum> coherency.pbs` where `resnum` is the reservation code for the session. Try to explain the observed results, and use them to compute the cost of a coherency miss.

Extra exercise

Try changing the `-cc` argument to `aprun` in the script so that the code runs on different pairs of cores.

Exercise 4: NUMA effects

The example code can be found in `Advanced/*/NUMA/`. This is the well-known STREAMS 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.

Extra exercise

Try reducing the array size `N` by a factor of 100 or 1000 (and increase the repetition count `NTIMES` by the same amount).

Exercise 5: 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 `Advanced/*/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. 48 processes and 1 threads, 24 processes and 2 threads, etc.). 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 6: Molecular Dynamics performance

The `Advanced/*/MolDyn/` directory contains a 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, 12 and 24 threads. Now modify the code so that it uses atomic update instead of `CRITICAL` — does the performance improve?
Next modify the code so that it uses an array of locks instead. Write your code so that you can associate more than one particle with a lock.

Extra exercise (Fortran only)

Use a reduction clause for `f` instead. Compare the performance with the other versions.

Appendix

Detailed Login Instructions

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.

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.

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.

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.

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_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.

Hardware

Each node of ARCHER consist of two sockets, each containing a 12-core Intel Ivy Bridge processor.

Compiling using OpenMP

The OpenMP compilers we use are the Cray compilers for Fortran 90 and C. To compile an OpenMP code, simply compile with either ftn (for Fortran) or cc (for C): OpenMP is enabled by default: to disable it add the flag `-h noomp`.

Job Submission

You can run OpenMP codes on the login nodes in the usual way (set `OMP_NUM_THREADS` and execute).
For accurate timings, you should submit a batch job as follows:

```
qsub -q <resnum> scriptfile.pbs
```

where `resnum` is the reservation number for the session.
Alternatively, you can reserve an interactive session with

```
qsub -q <resnum> -IVl select=1,walltime=1:0:0 -A y14
```

which reserves one node for 1 hour. Note that the third character in the `-IVl` argument is the letter "l" and not the number "1".
When the node is allocated, you are effectively logged into it and you will get back a command prompt like this:

```
guestXX@mom1:~>
```

Note that **you will now be back in your home directory** so you will need to change to the work filesystem: `cd /work/y14/y14/guestXX/`.
You can now run parallel jobs directly from the command line as you have interactive access to one of the compute nodes. For example, to run on 4 threads:

```
export OMP_NUM_THREADS=4
aprun -n 1 -d $OMP_NUM_THREADS ./hello
```

Do not log out from this session! You should leave this interactive window open all the time so you can run jobs whenever you want. You cannot compile in this window, though: you will need to open another window on ARCHER to compile your code.
You can monitor your jobs status with the `qstat` command, and jobs can be deleted with `qdel` .