Advanced OpenMP
Exercise Notes

Getting started

Logging on to Cirrus

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

```
ssh -X trainXXX@login.cirrus.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.

Copy and extract the exercise files

Copy the tar file and unpack it with the commands

```
cp /home/shared/advomp.tar .
tar xvf advomp.tar
```

Compiling using OpenMP

The Intel compilers (icc/ifort) are available on Cirrus. To access them type

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

To compile an OpenMP code, simply add the flag `-qopenmp`
You can also use the GNU compilers (gcc/gfortran) instead: to access an up-to-date version that supports the latest OpenMP features, type

```
module load gcc
```

For the GNU compilers the OpenMP flag is `-fopenmp`

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.
You can monitor your jobs status with the `qstat` command, and jobs can be deleted with `qdel` .

Exercise 1: Mandelbrot with and without worksharing

First, remind yourself how to write some OpenMP by parallelising the code in `AdvOMP/*/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 `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 by submitting the supplied batch script with the command `qsub -q <resnum> coh.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 values of COREA and COREB in the script so that the code runs on different pairs of cores.

Exercise 4: NUMA effects

The example code can be found in AdvOMP/*/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: Molecular Dynamics performance

The AdvOMP/*/MolDyn/ directory contains a parallel version of a simple molecular dynamics code. Run the code using the script supplied with the script supplied to obtain a VTune profile. This should create a directory called r000hs which contains a .amplxe file (among other things). Subsequent runs create directories r001hs etc. To examine the profile, load the VTune module with

```
module load intel-vtune-17
```

and start the VTune GUI with the amplxe-gui command. Click on ”Open Result”, navigate to the r000hs directory and open the .amplxe file.
Try changing the loop schedule to improve load balance and profile again. Now modify the code so that it uses atomic update instead of CRITICAL — does the performance improve, and how does the profile change?

Exercise 6: 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.
Before compiling the code, load the MPI module with

```
module load mpt
```

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. 36 processes and 1 threads, 18 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 7: Target offload

This exercise is only available in C: apologies to Fortran programmers! The source code is in `AdvOMP/C/Laplace`. To build the code, first load the clang compiler on Cirrus with

```
module load clang
```

Then type `make` to compile the code.
The supplied code is parallelised for the CPU. You may wish to keep a copy of this version, so you can use it to measure the performance on the CPU. Add the appropriate directives to offload the two parallel loops to the GPU instead. Use the supplied batch script to run on the GPU nodes. Note that the error file should contain output from the nvprof profiling tool. Try using target data directives to reduce the amount of data movement - does the performance improve?