Shared Memory Programming with OpenMP - Exercise Notes

OpenMP on Cirrus

Hardware

For this course you will be compiling and running the code on Cirrus:
`login.cirrus.ac.uk`

Compiling using OpenMP

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

```
module load intel-20.4/compilers
```

To compile an OpenMP code, simply add the flag **`-qopenmp`**

You can also use the GNU compilers: to access an up-to-date version that supports the latest OpenMP features, type

```
module load gcc/10.2.0
```

For the GNU compilers (gcc) the flag is **`-fopenmp`**

Using a Makefile

Makefiles are supplied with most of the exercises to help you build the code.

To build the code in a directory, just type **`make`**, and to remove all object and executable files, type **`make clean`**

Job Submission

You can develop and test the exercise codes on the login nodes, but for doing accurate timing runs you *must* use the compute nodes. To do this, you should submit a batch job as follows.

```
sbatch myjob.slurm
```

where **`myjob.slurm`** is the shell script supplied with each exercise.

Note that batch jobs can only be submitted from the **`/work`** filesystem. If your home directory is, say, **`/home/tc069/tc069/tc069xy`** then you must submit batch jobs from **`/work/tc069/tc069/tc069xy`** (or subdirectories thereof).

`tc069` is also the budget code you should use in your submit scripts.

To change the number of threads, edit the script and change the value assigned to the **OMP_NUM_THREADS** variable. To run on different numbers of threads you can reassign this variable and re-run the executable multiple times in the same batch script. You can monitor your job's status with the **squeue** command and queued or running jobs can be cancelled with **scancel**.

Getting started

Copy the tar file containing the code to your account on Cirrus and unpack it with the commands:

```
cp /home/z04/shared/OpenMPSummerSchool.tar .

tar xvf OpenMPSummerSchool.tar
```

Exercise 1: Hello World

This is a simple exercise to introduce you to the compilation and execution of OpenMP programs. The example code can be found in **OpenMPSummerSchool/HelloWorld/**

Compile the code, using the supplied Makefile.
Before running it, set the environment variable **OMP_NUM_THREADS** to a number n between 1 and 4 with the command:

```
export OMP_NUM_THREADS=n
```

When run, the code enters a parallel region at the **#pragma omp parallel** directive. At this point n threads are spawned, and each thread executes the print command separately. The **omp_get_thread_num()** library routine returns a number (between 0 and n-1) which identifies each thread.

Extra Exercise

Incorporate a call to **omp_get_num_threads()** into the code and print its value inside and outside the parallel region.

Exercise 2: Area of the Mandelbrot Set

The aim of this exercise is to use the OpenMP directives learned so far and apply them to a real problem. It will demonstrate some of the issues which need to be taken into account when adapting serial code to a parallel version.

The Mandelbrot Set

The Mandelbrot Set is the set of complex numbers c for which the iteration $z = z^2 + c$ does not diverge, from the initial condition $z = c$. To determine (approximately) whether a point c lies in the set, a finite number of iterations are performed, and if the condition $|z| > 2$ is satisfied, then the point is considered to be outside the Mandelbrot Set. What we are interested in is calculating the area

of the Mandelbrot Set. There is no known theoretical value for this, and estimates are based on a procedure similar to that used here.

The Code

The method we shall use generates a grid of points in a box of the complex plane containing the upper of the (symmetric) Mandelbrot Set. Then each point is iterated using the equation above a finite number of times (say 2000). If within that number of iterations the threshold condition $|z| > 2$ is satisfied then that point is considered to be outside of the Mandelbrot Set. Then counting the number of points within the Set and those outside will lead to an estimate of the area of the Set.

The serial code is in **`OpenMPSummerSchool/Mandelbrot2/area.c`** Parallelise the serial code using the OpenMP directives and library routines that you have learned so far. The method for doing this is as follows:

1. Start a parallel region before the main loop, nest making sure that any private, shared or reduction variables within the region are correctly declared.
2. Distribute the outermost loop across the threads available so that each thread has an equal number of the points. For this you will need to use the OpenMP library routines **`omp_get_num_threads()`** and **`omp_get_thread_num()`** to explicitly calculate the range of iterations for each thread.

Once you have written the code try it out using 1, 2, 4, 8 and 16 threads. Check that the results are identical in each case, and compare the time taken for the calculations using different number of threads.

Extra Exercise

Is your solution well load balanced? Try different ways of mapping iterations to threads.

Exercise 3: Mandelbrot with worksharing

You can start from the code you have already, or another copy of the sequential code which can be found in **`OpenMPSummerSchool/Mandelbrot2/`**.

This time parallelise the outer loop using a **`parallel for`** directive. Don't forget to use **`default(none)`** and declare the shared, private and reduction variables. Add a schedule clause and experiment with the different schedule kinds.

Exercise 4: Synchronisation

Redo the Mandelbrot example using critical sections, or atomics or lock routines to prevent the race condition on **`numoutside`**, instead of a reduction clause. Which version is fastest/slowest?