PX425 Assignment 3

Introduction

The purpose of this assignment is to demonstrate your ability to parallelise a scientific computer code in a number of different ways using OpenMP.

First, visit the assignments section of the Moodle pages and download `perc.zip` to your space on the SCRTP file-system. Uncompress this file using the `unzip` command in a terminal window, or by using any of the decompression utilities installed on the SCRTP system. Do not be alarmed at the presence of a Fortran `.f90` file within `perc.zip`. This contains functions that your C program will need to call, but you do not need to modify (or even open) it. You will only need to edit `percolation.c`.

This program generates statistics on random graphs or networks. A *graph* in this context is simply a collection of N vertices (points) some of which are connected by lines (edges). Any vertex i may be connected, with probability P, to any other vertex j. If $P = 0$ there are no connections, and as this probability is increased, the resulting random graphs become increasingly highly connected. Two measures of overall connectivity are the number of vertices within the largest connected group or *cluster*, and the total number of clusters. For large enough P, a percolation threshold is reached and the graph consists of a single cluster of size N.

Code structure and compilation

Examine `percolation.c` in your editor. You will find definitions of the total number of vertices N (`Nvert`), the number of values of P to investigate (`Np`), and the number of graphs to average over for each value of P (`Ngraphs`). `Np` and `Ngraphs` are currently set to 1, and `Nvert`=5000, to ensure quick runs.

There are already a number of OpenMP statements in this program. Some of them relate to the random number generator which is used such that the sequence of random numbers generated by each thread will be different. When generating a random graph, the outer loop over the `Nvert` vertices has been parallelised with an OpenMP work-sharing construct.

To compile the code, a makefile has been provided. Currently the makefile does not include any OpenMP flags and will hence build a serial version of the code. Build the code by issuing the command `make` in your terminal. Run the code with `./perc` and note the output in your report. *You will be asked to make note of responses to several questions in your report so it is advisable to write this as you go along.* With both `Np` and `Ngraphs` equal to 1, the code will run sufficiently quickly that you can safely do this interactively on `godzilla`.

Parallelism with OpenMP

Now modify `Makefile` to add `-fopenmp` to the `FFLAGS` *and* `CFLAGS` definitions. Force a total rebuild with `make clean && make` to ensure that all files are re-compiled with the new settings.

Parallelism over vertices

If you examine the logic carefully, you might notice that there is a problem with the existing OpenMP parallelism. There is a potential *data race* if two different threads need to add connections to the same vertex j, or if a particular j being updated by one thread is currently being used as i by a different thread. Note in your report which lines of code are generating this data race, and describe the circumstances under which incorrect data will be produced.

You will likely find that this situation is very rare for the values of `Pcon` we use here, but we should still do something about it! By adding additional OpenMP pragmas *only*, correct the code to eliminate the data race entirely. The program output will vary with the number of threads used, but should be reproducible on a fixed number of threads with a static schedule. Confirm this, and note the corresponding program output in your report. Briefly discuss performance and reproducibility versus the unmodified code.

Your next task is to modify the existing `omp parallel` construct to optimise scheduling of iterations to threads. First, with 4 threads, establish a base line time (as printed by the program itself) with static scheduling and a chunk size of `Nvert`/4. This will likely be the same as when no particular chunk size was specified. State in your report if you expect this to be the optimal schedule for this loop. Experiment with alternative chunk sizes, and note the results of these experiments in your report. Record the chunk size at which the time taken to generate the graph is smallest on 4 threads. If you struggle with this part of the assignment then you might want to leave it until after the week 7 workshop.

Now change `Np` to 8, and `Ngraphs` to 20 to obtain statistics averaged over many graphs for several values of P. As this will now take substantially longer to run, we will be using the SCRTP job queue to run this. You *must not* run the code directly on `godzilla` for the rest of this assignment. Instead you submit the script `perc.slurm` to the batch system as we practised in workshops. Output will appear in files like `slurm-XXXXXX.out` where `XXXXXX` is a job number. The script `perc.slurm` sets `OMP_NUM_THREADS` for you based on the number of processors requested. Submit this script to the batch system, and note the time reported in the output.

Copy this version of your program to `percolation1.c` and keep it safe for submission later.

Parallelism over P

Remove the existing `omp parallel` directive from the loop over vertices, and any pragmas/directives you placed inside the inner loop.

Your task is now to parallelise the code over P, by adding appropriate pragmas or directives to the loop over the `Np` values considered. Pay careful attention to which variables should be shared, and which should be private.

Test and time the resulting code (by submitting the script to the queue) on 1,2,3 and 4 threads, all for `Np=8` and `Ngraphs=20`. This will require editing of `perc.slurm`. Note the resulting times in your report and assess whether or not this parallelism strategy is more or less effective than parallelisation over vertices.

Annoyingly, the average statistics will not be printed in a convenient order. Add an appropriate clause to your parallel directive, and contain the output statements within an appropriate OpenMP construct such that the program output is ordered as per the serial version. Now repeat your timings (by submitting the script to the queue) on 1,2,3 and 4 threads. Note these in your report and comment on any time penalty suffered by introducing the ordering.

Copy this version of your program to `percolation2.c` and keep it safe for submission later.

Parallelism over graphs

Your next task is to add parallelism over graphs, *instead* of parallelism over P. Remove any OpenMP pragmas or directives relating to parallelism over P. Then add appropriate pragmas or directives to the loop over the `Ngraphs` generated. Pay careful attention to which variables should be shared, which should be private and which should be reduction variables.

Run the resulting code (by submitting the script to the queue) on 1,2,3 and 4 threads and note the timings. How do these compare to parallelism over P? Discuss which of the two parallelism strategies is most efficient in your report, making reference to both parallel overheads (cost in CPU time for fork and join threads) and distribution of work to threads.

Copy this version of your program to `percolation3.c` and keep it safe for submission later.

Nested Parallelism

You should now combine your previous two versions of the code to generate one which is parallel over *both* P and graphs and maintains correct output ordering, by implementing nested parallelism as demonstrated in lectures. Set `Np=2`. By experimenting with the environment variables `OMP_NUM_THREADS`, `OMP_NESTED` or otherwise, your aim is to engineer a fork into **two** threads (one for each P), which then further fork into **two** threads which share the work involved in generating the twenty random graphs. No more than four threads should be running in total.

Note the environment variables or other adjustments used for this in your report, and note and comment on the time reported by the program under these circumstances. Copy this version of your program to `percolation4.c`.

Deadline and Submission

You have two weeks in which to complete this assignment. You should submit all four versions of your program saved above, and a brief report. Please check all the places in the script that suggest you note or discuss something in your report and check you have addressed all of them.

Compress all these files into a zip file with the same name as your SCRTP user code, for example like this

```
zip phuxyz.zip percolation[1-4].c report.pdf
```

where you use your SCRTP username instead of `phuxyz`. This should compress all versions of your code plus your report into a single .zip file. Then upload the zip file to the Moodle page for Assignment 3.