HIGH PERFORMANCE PROGRAMMING
UPPSALA UNIVERSITY
SPRING 2017
ASSIGNMENT 6: USING OPENMP

Relation to previous assignments: This assignment is again connected to the previous Assignment 4; you are supposed to use your code from Assignment 4 as a starting point.

Remember that getting correct results is more important than any optimization and parallelization efforts. Therefore, before starting with this assignment you must make sure that your code from Assignment 4 works properly. If there are problems with your code for Assignment 4 you should fix that first, and then do this assignment.

This assignment is very similar to the previous Assignment 5, the main difference is that now you are supposed to do the parallelization using OpenMP instead of Pthreads.

1. Assignment

In this assignment, you should take your code from Assignment 4 and parallelize it using OpenMP. Your final code should be as efficient as possible, both regarding serial optimization and parallelization, so that it uses the multi-core computer it is run on as efficiently as possible. The number of threads to use should be specified as an input parameter to the program.

To make sure you are focusing your parallelization efforts on the most important part of your code, follow these steps:

(1) Use OpenMP to parallelize the most time-consuming part of your code.

(2) Since OpenMP makes it more convenient to parallelize code with only small changes, consider if there are some other parts of your code that you can also easily parallelize using OpenMP. For example, can you parallelize the code for updating the particle positions?

(3) Write a report where you show the effectiveness of your parallelization, including plots of speedup when running on different numbers of cores.

(4) In your report, include a comparison of the performance of your OpenMP-parallelized code compared to the Pthreads parallelization you did in the previous assignment. Which code performs best?

Your program should handle its input and output in precisely the same way as in Assignment 5; see those instructions for details.

Date: February 28, 2017.

In your report for this assignment, you should include both a discussion about serial optimization techniques used, as in previous assignments, and a discussion about the parallelization you have done with OpenMP.

About the parallelization, your report should include an investigation of the speedup you get when using different numbers of threads. Present this as speedup plots with the number of threads on the x-axis and the achieved speedup on the y-axis. For comparison, the plot should also show the ideal speedup, e.g. as a dashed or dotted line. Include at least two such speedup plots, for different problem sizes. Usually it is easier to get close to ideal speedup for larger problem sizes – is that true also in your case?

When writing your report, remember that *reproducibility* is important: whenever you write a report that includes some timing measurements, or other computational experiments of some kind, you should make sure that the report includes all information necessary so that the reported results become reproducible. The reader of your report should be able to reproduce your results. Your report should make it clear exactly what it is you are reporting, if you have some timings it must be explained what you were measuring: how was the measurement made, was it for the whole program run or for some specific part of the code, what parameters were used, how many timesteps, etc.

Note that timing results should not include calls to graphics routines. When measuring timings, run your code with graphics turned off to be sure that graphics routines are not disturbing your timings.

You may work in groups of up to three. You need to formally form a group in the Student Portal to be able to submit. If you work alone, you anyway need to form a "group" of 1 person in the Student Portal before you can submit.

The group should decide on a distribution of roles during the exercise and very shortly describe it in the final report. For example, who did the most programming and debugging, measuring performance and generating figures/tables, writing a report. If the group members have contributed equally to everything, then write that. If you focus on different things, make sure everyone in the group still undestands what you have done and how each part of your code works.

2. Input data and reference output data

The `Assignment6.tar.gz` file includes some input files, and some reference output data that you can use, those files are the same as for Assignment 4. Note however that provided that your code from Assignment 4 works correctly, you can use that to generate your own reference output data to use when verifying results of your new multi-threaded program.

3. Deliverables

It is important that you submit the assignment in time. **See the deadline in the Student Portal.**

You should package your code and your report into a single `A6.tar.gz` file that you submit in the Student Portal. There should be a makefile so that issuing "make" produces the executable `galsim`.

Unpacking your submitted file `A6.tar.gz` should give a directory `A6` and inside that there should be a makefile so that simply doing "make" should produce your executable file "galsim". The `A6` directory should also contain your report, as a file called "report.pdf".

Apart from your pdf report, your submission should only contain C/C++ source code files and makefile(s). No object files or executable files should be included, and no input/output (.gal) or other binary files.

Part of our checking of your submissions will be done using a script that automatically unpacks your file, builds your code, and runs it for some test cases, checking both result accuracy and performance. For this to work, it is necessary that your submission has precisely the requested form.

To be sure that your submission has the correct form, you can test it yourself as follows. Start by creating a separate directory for your test, and copy into that directory two files: your `A6.tar.gz` file and the `ellipse_N_00500.gal` input file. Then cd into that directory so that if you do "ls" you just see those two files listed. Then issue *exactly* the following commands:

```
tar -xzf A6.tar.gz
cd A6
make
./galsim 500 ../ellipse_N_00500.gal 200 1e-5 0.2 0 2
```

Then you program should run a simulation for the given number of timesteps (200) using $\theta_{\max} = 0.2$, graphics should be turned off and the program should use 2 threads. In the end, a `result.gal` file should be created containing the final result.

The report shall be in pdf format and shall contain the following sections:

- The Problem (very brief)
- The Solution (Describe the data structures, the structure of your codes, and how the algorithms are implemented. How did you do the parallelization? Are there other options, and why did you not use them?)
- Performance and discussion. Present experiments where you investigate the performance of the algorithm and your code. Include both a figure showing the scaling behavior as a function of N, and a couple of speedup plots showing how your parallelization works. Also compare the performance you get now to what you got in the previous assignment, when Pthreads was used, see above in Section 1.

If there are any questions, e-mail to `elias.rudberg@it.uu.se`.