

**HIGH PERFORMANCE PROGRAMMING  
UPPSALA UNIVERSITY  
SPRING 2017  
ASSIGNMENT 5: USING PTHREADS**

**Relation to previous assignment:** This assignment is a continuation of 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.

1. ASSIGNMENT

In this assignment, you should take your code from Assignment 4 and parallelize it using Pthreads. 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, see below.

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

- (1) Analyse your code from Assignment 4 to find out where most time is spent, for large  $N$ .
- (2) Use Pthreads to parallelize that part of your code.
- (3) Write a report where you show the effectiveness of your parallelization, including plots of speedup when running on different numbers of cores.

Here follows a description of how your program should handle its input arguments. Note that the input is very similar to the previous assignment, but now there is one additional parameter specifying the number of threads to use.

**Input to your program:** The program that you create should be called `galsim` and it should accept seven input arguments as follows:

```
./galsim N filename nsteps delta_t theta_max graphics n_threads
```

where the input arguments have the following meaning:

`N` is the number of stars/particles to simulate

`filename` is the filename of the file to read the initial configuration from

`nsteps` is the number of timesteps

---

*Date:* February 23, 2017.

`delta_t` is the timestep  $\Delta t$

`theta_max` is the threshold value  $\theta_{\max}$  to be used in the Barnes-Hut algorithm

`graphics` is 1 or 0 meaning graphics on/off.

`n_threads` is the number of threads to use.

If the number of input arguments is not seven, the program should print a message about the expected input arguments and then stop.

Note that your program should expect the number of threads to use as an input argument. If the user of your program wants to get the best possible performance, then the user will probably run it with the same number of threads as there are cores available on the computer, but the user may also choose some other number of threads, e.g. to investigate what speedup the parallelization gives, or if the user for other reasons does not want all cores to be used by the program. The user may also choose to run with a larger number of threads than the number of cores available; that is probably not the best choice for performance but your program should work correctly anyway.

**Output from your program:** As before, the program `galsim` should simulate the stars/particles for the given number of timesteps. In the end, the final positions and velocities of the stars should be saved to a file called `result.gal` using the same binary file format as for the input file. The `result.gal` file should be created in the directory where the program is running.

Note that the results of your program now should be the same as for the Assignment 4 code, so you can easily use your code from Assignment 4 to verify that your multi-threaded program gives correct results.

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 Pthreads.

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 understands what you have done and how each part of your code works.

## 2. INPUT DATA AND REFERENCE OUTPUT DATA

The `Assignment5.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 `A5.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 `A5.tar.gz` should give a directory `A5` and inside that there should be a makefile so that simply doing “make” should produce your executable file “`galsim`”. The `A5` 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 `A5.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 A5.tar.gz
cd A5
make
./galsim 500 ../ellipse_N_00500.gal 200 1e-5 0.2 0 2
```

Then your 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, see above in Section 1.

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