# HIGH PERFORMANCE PROGRAMMING
# UPPSALA UNIVERSITY
# SPRING 2022
# ASSIGNMENT 4: PARALLELIZATION

**Relation to previous assignment:** This assignment is a continuation of the previous Assignment 3; you are supposed to use your code from Assignment 3 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 3 works properly. If there are problems with your code for Assignment 3 you should fix that first, and then do this assignment.

## 1. Assignment

In this assignment, you should take your code from Assignment 3 and parallelize it using Pthreads and OpenMP. Your final codes 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 3 to find out where most time is spent.

(2) Use Pthreads to parallelize that part of your code.

(3) Use OpenMP to parallelize your code (i.e. re do the parallelization).

(4) Write a report where you show the effectiveness of your parallelizations, including plots of speedup when running on different numbers of threads.

Here follows a description of how your program should handle its input arguments. Note that the input is very silimar 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 six input arguments as follows:

```
./galsim N filename nsteps delta_t 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*: March 12, 2022.

`delta_t` is the timestep $\Delta t$
`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 six, 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 3 code, so you can easily use your code from Assignment 3 and the `compare_gal_files` function to verify that your multi-threaded program gives correct results.

**Portability:** You can choose freely which computer you want to use for your performance testing in this assignment — you can use one of the university's Linux computers, or your own computer, or some other computer you have access to as long as it has a moderate number of cores (at least 6 physical cores). However, even if you have used a different computer for your performance testing, you must still ensure that the code is portable so that it can be compiled and run on the university's Linux computers, `fredholm@it.uu.se` or `arrhenius@it.uu.se`. This is necessary to make sure that your teachers can test your code.

**Report:** 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 and with OpenMP. Use your previous report as a starting point, and add the parallelization discussion to that so that the final report you submit includes both serial optimization and parallelization aspects.

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.

## 2. Deliverables

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

You should package your code and your report into a single `A4.tar.gz` file that you submit in Studium. There should be a makefile so that issuing "make" produces the executable `galsim`.

Unpacking your submitted file `A4.tar.gz` should give a directory `A4` with two subdirectories `Pthreads` and `Openmp`. Inside these there should be a makefile so that simply doing "make" should produce your executable file "`galsim`". The `A4` directory should also contain your report, as a file called `report.pdf`.

In addition to the report, your `A4` directory should also include a text file called `best_timing.txt` that should contain four lines: on the first line, just one number giving the fastest time for Pthreads that you were able to achieve for the input case `ellipse_N_05000.gal` (5000 particles) with $\Delta t = 10^{-5}$ and 100 timesteps, and the number of threads that gives the best performance on the computer you were using. On the second line the best time for OpenMP using the same setup. On the third line, the name of the CPU model used. On the fourth line, the used number of threads.

Apart from `report.pdf` and `best_timing.txt` files, your submission should only contain C source code files and makefiles. 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.

Before submitting, make sure that your submission has the correct form, compiles and runs on the Linux system `fredholm.it.uu.se` or `arrhenius.it.uu.se`, and produces correct output as in Assignment 3 (use compare_gal_files).

The report shall be in pdf format (called `report.pdf`) 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. Describe optimizations you have used, or tried to use, and the measured effect of those optimization attempts. 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.

When you are done, upload your final `A4.tar.gz` file in Studium.