Individual Project

1 The Problem

Note that this is an **individual task**. In this final mandatory task in the course, you will work on an application or algorithm of your choice. The assignment must involve implementation of an algorithm, optimization of the code and parallelization either with Pthreads or OpenMP.

In Section 6 below you will find a list of projects to choose from.

2 The Report

The report should be written as a scientific paper of academic quality, written using either LaTeX (preferred) or a word processor (e.g. Word), in either English (preferred) or Swedish. Plots can be created using e.g. Matlab or Python Matplotlib. The report should include the following:

1. *Introduction*, providing a background and motivation.
2. *Problem description*, presenting the task.
3. *Solution method*, description of the algorithm, optimization and parallelization.
4. *Experiments*, showing correctness of code (software testing), evaluating the performance of your code both in serial and in parallel (using many cores, e.g. on `vitsippa.it.uu.se`), observations and comments on the results.
5. *Conclusions*, with explanations of the results and with ideas for possible optimizations or improvements. You should justify your observations and conclusions with objective and detailed reasoning.
6. *References*, listing all literature (including web pages) that were consulted in the project. If we find that you have copied text, code or code sections without a proper reference, it will be considered as plagiarism and reported to the disciplinary board.

3 Submission

The code should be portable to `fredholm.it.uu.se` or `arrhenius.it.uu.se` compiling and running without errors or warnings. Submit the code as a compressed archive `project.tar.gz` and the report separately as `report.pdf`. It is important that you submit the report itself and not include it in the tar-file, the report will be automatically uploaded to `Ourginal` for plagiarism check. The code will be separately checked with another software `MOSS` (*Measure Of Software Similarity*) which is an automatic system for determining the similarity of programs. MOSS was developed at Stanford University for detecting plagiarism in programming classes. During the Corona pandemic it use has increased dramatically world wide.

Note: The project is an official examination. You are given a lot of freedom to choose your own application in the project but with that comes a great responsibility to do the right thing and not get tempted to copy something that you have found on the internet or reusing a solution from a previous year student. Reading the report will in most cases reveal these attempts. Then also the automatic tools for checking will help us to detect plagiarism. If we even *suspect* plagiarism we are forced to report this further. The disciplinary board will then investigate and decide if it is plagiarism or not.

4 Oral presentation

After you have submitted and we have had time to read your report you can be asked to come and present your work orally and answer questions about your project. At this session we will ask you to further explain your work and discuss it with you. Especially, if your report is unclear or your code has some issues you will be asked to explain it further. It is only after this session that your project can be approved and graded.

5 Grading

The individual project will account for 60% of your final grade and the group assignments 1– 4 for 40%. When grading your work, we will take the following aspects into account:

- *Solution:* Choice of algorithm, serial and parallel efficiency.
- *Methodology:* Software testing, optimization and parallelization techniques, performance evaluation.
- *Code:* Design, robustness, correctness, efficiency, documentation, and general quality.
- *Report:* Disposition, presentation of results, reflection and reasoning on the results, language quality.
- *Oral presentation:* How well you can explain your work, answer questions and reason about different aspects in your work.

We will also take the complexity and difficulty of the problem into account when evaluating your work. It is also important that you meet the deadline for submission. The different grade requirements are as follows:

- **Grade 3:** You must demonstrate an understanding of the fundamentals of the course. Your code works correctly and you have made a reasonable attempt at optimizing and parallelizing important aspects of the code. The report is of minimal requirements where you only describe the problem, solution algorithm, show performance results and include used references.

- **Grade 4:** You have tried to optimize everything, and you're able to reason about the performance of your code at a high level and know why certain optimizations worked/didn't work. The report is well structured with appropriate references to theory and related work. In addition to grade 3 you also reflect on and discuss your results.

- **Grade 5:** The application needs to be of higher complexity. Then, in addition to very good optimizations and understanding of performance issues, you must also show a higher level of understanding of

the parallelized code. You must have optimized the code with respect to synchronization, data dependencies, load balance and parallel work. In your report, in addition to grade 4, you will also argue why you have chosen the specific method/technique and why your solution is optimal for the problem.

Note, it is important that you perform the task within the given time frame, late submission will affect negatively on the grade. Performing tasks within given time frames is a general program level goal that we also need to assess during your education.

6 Suggestions

Below follows a list of possible projects that you can choose from. The projects are grouped into two categories, less complex and more complex (challeging). If you are aiming for a high grade you should choose one of the more challenging projects.

1. Less complex

   - PDE solver, e.g., an explicit Finite Difference Solver in 2D.
   - Bucket Sort (with different random number sequences; normal, exponential, uniform)
   - Power method for eigenvalues
   - Conjugate gradient solver
   - Monte-Carlo simulation using Gillespie
   - LU-factorization (loop-parallelism)

2. More complex

   - Barnes-Hut algorithm for Galaxy Simulation (see BarnesHut.pdf)
   - Game of Life 3D (see GoL3D.pdf)
   - Parallel quicksort (see QuickSort.pdf)
   - Sudoku solver for large Sudokus (see SudokuSolver.pdf)
   - Matrix-matrix multiplication with Strassens algorithm (see https://en.wikipedia.org/wiki/Strassen_algorithm)
   - Block LU-factorization (task parallelism)

3. Project of your own choice

   - You can suggest a project of your own choice not listed above. If you want to do a project of your own choice you must submit a project description in advance and get it approved. Not approved projects will not be graded. (If you choose one of the listed projects above it will not have to be approved in advance.)