Project

Due No Due Date **Points** 30

Project - Parallelizing a Particles Simulation
Total Points: 30

Deadline to find your group partner and create your group on MarkUs: Sunday, March 7th, at noon. Your group partners for part 1 and part 2 remain the same. DO NOT change partners between the two parts or you will be flagged for plagiarism because of shared design between the parts!

Due Date for Part 1: This part includes the serial and the OpenMP section should be submitted on Friday, March 26th at 10PM. You have to submit the code using the project-part1 MarkUs instance. Grace tokens will be deducted if you submit part 1 after this deadline. You have to submit a report called report-part1.pdf on this deadline along with your complete code for Part 1 of the project.

Due Date for Part 2: This part includes the MPI section should be submitted on Tuesday, March 30th at 10PM. You have to submit the code using the project-part2 MarkUs instance. Grace tokens will be deducted if you submit part 2 after this deadline. You have to submit a report called report-part2.pdf on this deadline along with your complete code for Part 2 of the project.

Important note for phase 2: MPI jobs can run for a very long time if you have created a deadlock. Do not leave your jobs running for long periods of time. Always monitor your jobs submitted to the compute nodes. Also, do not run large jobs on the login node! You are only allowed to run very short jobs on the login node. Jobs on the login node should only run for a few seconds at max! Always check that you do not have a long-running job on the login node. A code with a deadlock is considered a long job since it will never terminate. We will deduct 30% to 100% of your grade if you run long jobs on the login node. We do have the complete log of all jobs submitted to Scinet.

Scinet

We will use Scinet for this project. You should have already received an email with your Scinet login. In that email you see a link to The short intro to Scinet, document (the document is also uploaded in

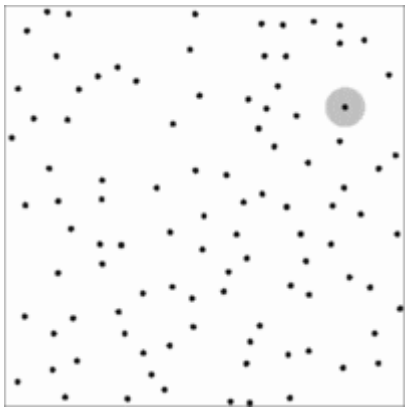
that email you see a link to "The short intro to Scinet" document (the document is also uploaded in Quercus). You have to read that document before starting this lab. While you can use your own machine or a lab machine to debug the code, your final code has to be performance engineered and executed on Scinet. We will do all grading on Scinet. For each part, we do provide scripts to help with running your code on Scinet.

Before you start!

Note that this is a project not an assignment. You will be evaluated on your efforts to understand the problem and to find answers to issues that come up as you start implementations. The project code, handout, and resources listed at the end of the handout provide you with all that is needed to understand the problem and even provide hints (read items 6 and 7 before you start the project, they are very helpful!). Being able to understand these and leverage them in your implementation is a part of your grade. To allow for a fair grading and to not give out answers, we might not be able to answer questions in Piazza that relate to questions that want us to explain the problem in detail or ask to resolve issues with your implementation. You are encouraged to ask questions from your classmates in Piazza, we will take note of students that actively help others and answer questions.

Overview

You have to work with a partner for this project. In this project, we will be parallelizing a toy particle simulation (similar simulations are used in **mechanics** (http://www.thp.uni-duisburg.de/~kai/index_1.html), **biology** (http://www.ks.uiuc.edu/Research/namd/), and **astronomy** (http://www.mpa-garching.mpg.de/gadget/clusters/index.html)). In our simulation, particles interact by repelling one another. A run of our simulation is shown here:



The particles repel one another, but only when closer than a cutoff distance highlighted around one particle in grey.

Asymptotic Complexity

Serial Solution Time Complexity

If we were to naively compute the forces on the particles by iterating through every pair of particles, then we would expect the asymptotic complexity of our simulation to be O(n^2).

However, in our simulation, we have chosen a density of particles sufficiently low so that with n particles, we expect only O(n) interactions. An efficient implementation can reach this time complexity. The first part of your assignment will be to implement this linear time solution in a serial code, given a naive O(n^2) implementation.

Parallel Speedup

Suppose we have a code that runs in time T = O(n) on a single processor. Then we'd hope to run close to time T/p when using p processors. After implementing an efficient serial O(n) solution, you will attempt to reach this speedup using two different programming models.

Starter Code

The starter code is available on Scinet under:

/home/t/teachcsc367/CSC367Starter/project/starter.tgz

The starter code provides correct implementations of the particle simulation problem in serial/OpenMP/MPI, however, these implementations are not optimized for performance. Note that correctness does not mean optimized, it only means that the provided code correctly computes the particles interactions.

Read the below on the source code structure carefully:

- common.cpp, common.h ; these files provide an implementation of common functionality, such as I/O, numerics and timing. Do not change these files by any means. You will alter the physics if you do! We will use the default version (what we provide with the starter code) of these files for grading.

  You do not need to understand the underlying physics (such as ordinary differential equations and the Euler method) used in the common.cpp and common.h files since the movement of particles and how forces are applied are already correctly implemented for you in these two files.
- autograder.cpp; a code that helps calculate performance for both serial and parallel implementations. Do not change this file. We will use the default version (what we provide with the starter code) of this file for grading.

file for grading.

- Makefile; a makefile that compiles your code on Scinet. Do not create a make run command. Run your code using the sbatch files provided.
- serial.cpp: This file provides a correct serial implementation to the problem, however, the implementation is not optimal and is a naive O(n^2) implementation. Do not try to run this code for large particle sizes. It will not finish on-time and the sbatch files will timeout.
- openmp.cpp: This file provides a correct parallel implementation of the problem in OpenMP. However, the implementation is not optimized for performance or scalability!
- mpi.cpp: This file provides a correct MPI implementation to the problem. However, the implementation is not optimized for performance or scalability!
- setup-project.sh: This file loads the modules and compiles your code. The jobs scripts are not called with this file.
- job-teach-serial, job-teach-openmp, job-teach-mpi: are sbatch files that run your code with small number of particles and do not include the “-no” flag.
- auto-teach-*: are sbatch files that run your code with different number of particles/cores/processors and call the autograder. These runs are with the “-no” flag.

We recommend that you only change the serial.cpp, openmp.cpp, and mpi.cpp files. However, if you do decide to include other files or modify the Makefile (do not modify common.cpp, common.h, autograder.cpp), spell out the changes in your report. Also, spell out in your report what Makefile targets we are to build for the different parts of your report.

Correctness and Performance

A simple correctness check which checks the smallest distance between 2 particles during the entire simulation is provided (look inside the provided source code!). A correct simulation will have particles stay at greater than 0.4 (of cutoff) with typical values between 0.7-0.8. A simulation were particles don't interact correctly will have particles closer than 0.4 (of cutoff) with typical values between 0.01-0.05 . More details as well as an average distance are described in the source file.

We'd recommend keeping the correctness checker in your code (for the OpenMP and MPI codes the overhead isn't significant) but depending on performance desires it can be deleted as long as the simulation can still output a correct txt file.
The performance of the code is determined by doing multiple runs of the simulation with increasing particle numbers and comparing the performance with the autograder.cpp provided. This can be done automatically with the auto-* scripts.

There will be two types of scaling that are tested for your parallel codes:

- In strong scaling we keep the problem size constant but increase the number of processors

- In strong scaling we keep the problem size constant but increase the number of processors
- In weak scaling we increase the problem size proportionally to the number of processors so the work/processor stays the same

For more details on the options provided by each file you can use the -h flag on the compiled executables.

Important notes for Performance and the Autograder:

The scripts we are providing have small numbers of particles and large number of particles. We will use the same particles sizes to test the performance of your code! We will put great weight on your code running correctly and performing well for the large particles so don't just optimize for the small particles!

Remember to use the -no flag to “turn off all correctness checks and particle location outputs” for actual timing runs. Our auto-teach* scripts use the -no flag, but the job-teach-* does not include the -no flag. So use the job-teach-* for debugging and checking for correctness before moving to the auto-teach-* sbatch files that will help with actual timings and information on scaling and efficiency that the autograder provides.

More details on the autograder:

- Serial code performance will be tested via fitting multiple runs of the code to a line on a log/log plot and calculating the slope of that line. This will determine whether your code is attaining the O(n) desired complexity versus the starting O(n^2).
- OpenMP Memory performance and MPI Performance will be tested via calculating the strong and weak scaling average efficiencies for 1,2,4,8,16 processor/thread runs.
  - The strong and weak average efficiencies (effss , effws) are reported for your reference.

Important note about the autograder: The strong and weak scaling numbers reported by the autograder should only be taken seriously when you have optimized the serial code. To clarify, if you run the starter code only, the scaling efficiency and speedups reported by the autograder might not be bad, but don't be fooled! Recall the discussions in class on “how to fool the masses with reporting performance numbers!”

Part 1: Serial and OpenMP

In this part, you will write two versions of our simulation. DO NOT change the headers in the provided sbatch scripts. First, you will write an O(n) serial code. Then, you will write a parallel version of this O(n) code for shared memory architectures using OpenMP.

There are two source files (serial.cpp and openmp.cpp) you need to modify. You need to create one serial code (serial.cpp) that runs in O(n) time and one shared memory implementation (openmp.cpp)

using OpenMP.

You must represent your results visually (graphs!) and analyze them in a short report (you must name this file: report-part1.pdf). You should describe your implementation, plot and analyze your results, draw conclusions and report your findings in a written report. Write a very technically sound report and it is important to show insight into what you are measuring and explain the results you are seeing. Below lists examples of what you can include in your report:

- A description of the design choices that you tried and how did they affect the performance.
- Plots: Serial: (1) A table that shows the running time of your optimized code for different particle sizes as well as the slope estimate for your optimized serial code. (2) A log-log plot of running time vs the number of particles to show that your optimized serial implementation runs in O(n) time.
- Plots: OpenMP: (1) The strong and weak scaling plots that show the running time of your optimized OpenMP implementation. (2) The strong scaling plots that show the speedup of your optimized OpenMP implementation. (3) The strong and weak scaling plots that show the efficiency of your optimized OpenMP implementation. (4) And a table that shows the average weak and strong scaling efficiency of your optimized OpenMP implementation.
- Analyze the speedup plots to show how closely your OpenMP code approaches the idealized p-times speedup and a discussion on whether it is possible to do better.
- Where does the time go? Discuss the synchronization and locking strategies used in your code and the optimization you have made to make the code more efficient.
- A discussion on your serial and OpenMP implementation.

Hint: Our solution gets to ~10s for 160000 particles for the OMP implementation. This is only a rough ballpark for the times we expect you to get with a decent optimization and is not a hint for efficiency and scaling. However, much better timings can be achieved, we will give the top three performing teams that achieve good speedups, efficiency, and scalability up to 10% bonus marks for Part 1.

Part 2: MPI

The MPI part of the project is set to be implemented on one Scinet node. DO NOT change the headers in the provided sbatch scripts. Note that you should not make assumptions of the underlying network and do not optimize for a specific topology. We are using a “logical view” of a distributed system in this section so refer to “cores” as “processors” in all your plots.

There is one source file (mpi.cpp) you need to modify. You need to create one distributed memory implementation (MPI) that runs in O(n) time with as close as possible to O(n/p) scaling.

You must represent your results visually (graphs!) and analyze them in a short report (you must name this file: report-part2.pdf. You should describe your implementation, plot and analyze your results, draw conclusions and report your findings in a written report. It is important to show insight into what you are measuring and explain the results you are seeing.

Here are some items you might add to your report:

- Plots: (1) The strong and weak scaling plots that show the running time of your optimized MPI implementation. (2) The strong scaling plots that show the speedup of your optimized MPI implementation. (3) The strong and weak scaling plots that show the efficiency of your optimized MPI implementation.
- A description of the communication you used in the distributed memory implementation.
- A description of the design choices that you tried and how they affect performance.
- Analyze the speedup plots to show how closely your MPI code approaches the idealized p-times speedup and a discussion on whether it is possible to do better.
- Where does the time go? Consider breaking down the runtime into computation time or communication time. You can plot the distribution or even comeup with a parameterized expression that estimates the amount of data communication vs the computation cost. How do they scale with p? If you have designed your partitioning to reduce communication costs, explain your strategy and why it works. What else comes into mind to improve your code?
- A discussion on your MPI implementation.

Hint: Our solution gets to ~2.6s for 160000 particles for the MPI implementation and we also get very good scaling and efficiency. This is only a rough ballpark for the times we expect you to get with a decent optimization. However, better timings might be achieved, we will give the top three performing teams that achieve good speedups, efficiency, and scalability up to 10% bonus marks for Part 2.

Testing tips

For this assignment, you are required to use the teach cluster (SciNet) to get accurate performance results. You will be graded on the teach cluster only. You are free to use C or C++.

job-teach* files: These files will run one instance of your serial (or OpenMP, or MPI) implementation for small number of particles (see inside the job-teach-* file for more information) with the -no flag disabled to print outputs related to particles locations and correctness issues. Run these files (one at a time) using sbatch job-teach-*

auto-teach-* files: These files will run your code for different particles sizes and/or a different number of particles and then run the autograder to report scaling and efficient numbers. Note that the -no flag is included to report accurate timings. Also note that these jobs will take longer to run and should not be used for checking code correctness. The auto-teach-* files have two versions: *-small and *-large. The small files run your code for small particles. Please note that auto-teach-serial-small has a different

number of particles compared to auto-teach-openmp-small and auto-teach-mpi small. This is because the unoptimized serial code runs very slow so we provide smaller particle sizes for the serial-small script. DO NOT use auto-teach-serial-large on an unoptimized serial code since it will timeout and never finish. Always optimize your code and test with the *-small scripts first then move to *-large files. Also, if you have not created efficient data structures or allocated memory properly, your code might crash for *-large files or you might get a segmentation fault. So optimize your code properly and remember that the

files or you might get a segmentation fault. So optimize your code properly and remember that the optimizing your code for *-large files will be harder because of the large particle sizes! Run these files (one at a time) using sbatch auto-teach-*

Report

You must write two reports (report-part1.pdf for part 1 and report-part2.pdf for part 2) documenting your implementation, displaying your results in a meaningful way, and analyzing your findings, for each part of the assignment. In your report, you should present your implementation, the experimental setup and results. You should discuss what you noticed, draw conclusions and analyze the tradeoffs, if any. The reports should be written in a scientific manner (clear structure, clear description of your approach, results, findings, etc., and should use technical writing instead of colloquial terminology or phrases). Keep in mind that presenting your experimental findings and observations to a technical audience is an important skill to develop as a computer scientist.

Two-Phase Submission

It is critical to read the below to submit your project properly. The project should be submitted in two parts:

- Part 1 of the project that includes the serial and the OpenMP section should be submitted on Friday, March 26th at 10PM. You have to submit the code using the project-part1 MarkUs instance. Grace tokens will be deducted if you submit part 1 after this deadline. You have to submit a report called report-part1.pdf on this deadline along with your complete code for Part 1 of the project. Your submission for part 1 will be the only instance that we will use to grade for serial and OpenMP implementations. Please read the below on "*common submission rules for both parts"*.

- Part 2 of the project that includes MPI section should be submitted on Tuesday, March 30th at 10PM. You have to submit the code using the project-part2 MarkUs instance. Grace tokens will be deducted if you submit part 2 after this deadline. You have to submit a report called report-part2.pdf on this deadline along with your complete code for Part 2 of the project. Your submission for part 2 will be the only instance that we will use to grade for MPI implementations. DO NOT assume that the course staff will look at your submission from part 1 to grade this section. Two different course staff might grade parts 1 and part 2. So include anything needed for this part to be fully stand alone. Please read the below on "*common submission rules for both parts"*.

Common submission rules for both parts: You must keep the same structure in your repository as the

starter code. You must submit all the files required to build and run all your programs (including any header files and the makefile, etc.). You shouldn't need to add any new files to the starter code, but if you do, make sure that you update the makefile accordingly. Make sure your code compiles and runs correctly on Scinet. Be sure to make it clear in the report how to run your code! Do not submit any executables or object files! (do a "make clean" before making your final commit).

IMPORTANT: make sure to keep the optimization options (-O3) in the makefile in your final submission, otherwise your code will be too slow, even if your algorithms are efficient.

Aside from your code, you must submit two reports (named report-part1.pdf and report-part1.pdf) with the required content as described above. When discussing your approach, feel free to also describe any problems encountered and workarounds, what isn't fully implemented (or doesn't work fully), any special design decisions you've taken or optimizations you made (as long as they conform to the assignment specs!), etc.

Additionally, you must submit an INFO.txt file, which contains as the first 2 lines the following:

your name(s)
your UtorID(s)

If you want us to grade an earlier revision of your project for whatever reason (for example, for saving some grace tokens if you had a stable submission before the deadline, tried to add new functionality after the deadline but broke your submission irreparably), then you may specify the git hash for the earlier revision you want marked.

As a general rule, by default we will always take the last revision before the deadline (or last one after the deadline, up to your remaining unused grace tokens), so you should not be including a line with the git commit hash, except in the exceptional circumstances where it makes sense. So in general, please avoid using this option and just make sure that the last revision (either before the deadline if you submit on time, or up to a subset of your remaining grace tokens if you submit late) is the one you want graded.

Finally, you must submit a plagiarism.txt file (in the top directory of the assignment), with the following statement:

"All members of this group reviewed all the code being submitted and have a good understanding of it. All members of this group declare that no code other than their own has been submitted. We both acknowledge that not understanding our own work will result in a zero on this assignment, and that if the code is detected to be plagiarised, severe academic penalties will be applied when the case is brought forward to the Dean of Arts and Science."

A missing INFO.txt file will result in a 10% deduction (on top of an inherent penalty if we do not end up grading the revision you expect). Any missing code files or Makefile will result in a 0 on this assignment! Please reserve enough time before the deadline to ensure correct submission of your files. No remark requests will be addressed due to an incomplete or incorrect submission!

Again, make sure your code compiles without any errors or warnings.
Code that does not compile will receive zero marks!

Marking Scheme

Your grade will depend on the performance and efficiency sustained by your codes on Scinet and will be broken into 3 parts. We will be marking based on performance and efficiency (65%), coding style (5%), and report (30%). This grading criteria will be applied to different parts as follows: Serial - 30%, OpenMP - 30%, MPI - 40%.

Code that does not compile or computes the particle interactions incorrectly will receive 0 marks (for the part that applies)!

For grading your performance, we will consider multiple factors such as the running time of the implementation as well the strong and weak scaling performances and efficiency. It is our discretion on how to distribute the performance grade between these metrics. For example, an unoptimized serial code has not merit and it will also invalidate all the weak and strong scaling numbers in the OpenMP and MPI parts!

Negative deductions (please be careful about these!):
Code does not compile: -100% for *any* mistake, for example: missing source file necessary for building your code (including Makefile, header files, etc.), typos, any compilation error, etc

Changes to common.h, common.c: -100%, you are not allowed to change these two files!
No plagiarism.txt file: -100% (we will assume that your code is plagiarised and that you wish to withdraw your submission, if this file is missing)
Missing or incorrect INFO.txt: -10%
Warnings: -10%
Extra output: -20% (for any output other than what is required in the handout)

Code placed in other subdirectories than indicated: -20%
Submitted unnecessary files (compiled code, test reports, data files, etc.): -10%

We will deduct 30% to 100% of your grade if you run long jobs on the login node.

Copyright

Resources

1. Shared memory implementations may require using locks that are available as **omp_lock_t (https://www.openmp.org/spec-html/5.0/openmpse31.html)** in OpenMP (requires omp.h).
2. Distributed memory implementation may benefit from overlapping communication and computation that is provided by **nonblocking MPI routines (http://www.mpi-forum.org/docs/mpi-1.1/mpi-11-html/node46.html)** such as MPI_Isend and MPI_Irecv.
3. Other useful resources: **OpenMP tutorial (http://openmp.org/wp/2008/11/sc08-openmp-hands-on-tutorial-available/)** , **OpenMP specifications (http://www.openmp.org/specifications/)** and **MPI specifications (http://mpi-forum.org/docs/)** .
4. Programming in shared and distributed memory models are introduced in Lectures.
5. If you want a parallel debugging tool, Scinet has DDT installed, however, if you plan to use DDT on Scinet, remember the ask for a dedicated node (***https://docs.scinet.utoronto.ca/index.php/Niagara_Quickstart*** **(https://docs.scinet.utoronto.ca/index.php/Niagara_Quickstart)**) and you do need to ssh using X server! **TAU (Tuning and Analysis Utilities) (http://www.cs.uoregon.edu/Research/tau/home.php)** is a source code instrumentation system to gather profiling information; this system can profile MPI, OpenMP and mixtures, but it has a learning curve. Tau is not installed on Scinet so if you decide to use it you need to run outside of Scinet.
6. Hints on getting O(n) serial and Shared memory and MPI implementations (**pdf (https://drive.google.com/open?id=11vjRefkcRA3a8DlH1t0QoVYw_TMx9RTl)**). Please note that while these slides provide very useful hints on describing the problem and general approaches that you might want to take, they are not necessarily providing you with the most efficient solution. So do not limit your creativity!

7. Useful reading: **https://docs.scinet.utoronto.ca/index.php/Introduction_To_Performance#Strong_Scaling_Tests (https://docs.scinet.utoronto.ca/index.php/Introduction_To_Performance#Strong_Scaling_Tests)**