ECE1747H Assignment 1: CPU Parallelism with Pthreads and MPI

A Range-Limited Electrostatic N-Particle Simulation

You are presented with a system of charged particles (point charges). The solution to the electrostatic N-particle problem involves calculating the electrostatic force between every pair of particles and summing these forces to determine the net force on each particle. However, this is a computationally expensive problem, as the number of calculations grows quadratically with the number of particles. One method to reduce complexity and approximate the solution is to limit the range of interaction between particles. In this **range-limited N-particle simulation**, we assume forces between particles further apart than a given *cutoff radius* are negligible, and we do not calculate them. This allows for creative methods that can significantly reduce the number of pairwise force calculations, and lends itself well to parallelization, which you will implement in this assignment.

Consider the following system of particles:

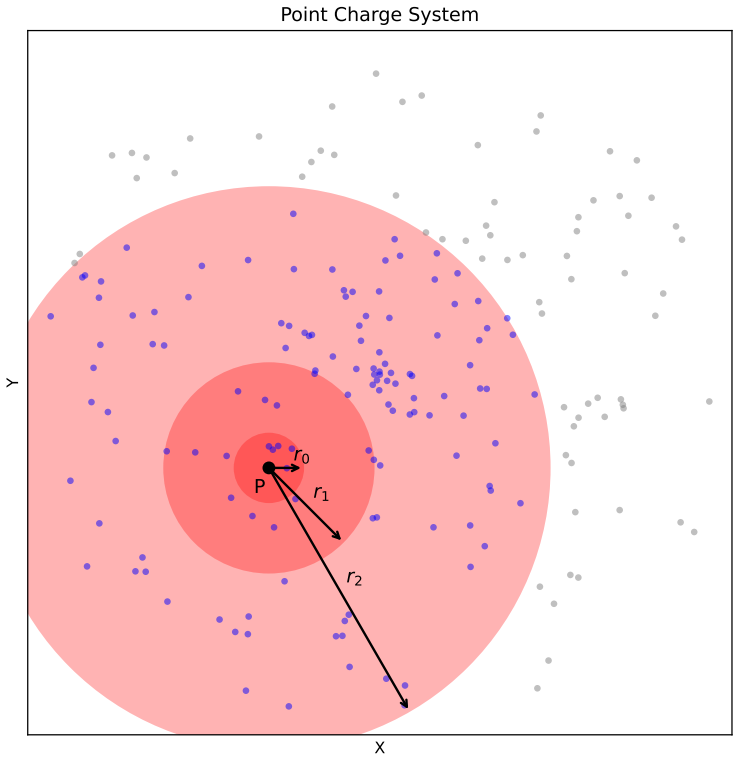

Figure 1: Visualization of different cutoff radii for a particle P in a system

In the above system, given cutoff radius $r = r_0$, if we focus on particle P you would be required to calculate the forces between P and each of the five particles that lie within the radius, ignoring the particles outside, and then sum the forces to determine the net force on P. This process is then repeated for the remaining particle in the system.

Problem Setup

The particles lie stationary in a 2D plane, and each particle is either a proton or an electron. The force between two particles is calculated using Coulomb's Law (See **Appendix A: Coulomb's Law**). This force is attractive if the particles have opposite charges and repulsive if they have the same charge. In a typical N-body problem, the forces between particles are represented as vectors, and the net force on each particle is found by summing these vectors. However, to simplify the problem, we only consider the **magnitude and sign** of the force between each pair of particles. So, the "net force" is instead a **signed scalar sum** of the forces on each particle.

Consider the following worked example of aggregation of scalar forces in a system with four particles: P_1 (Electron at (1, 1)), P_2 (Proton at (4, 5)), P_3 (Electron at (7, 1)), and P_4 (Electron at (8, 2)). The unit of distance is $1 \times 10^{-10}\,\text{m}$. We will calculate the net force on P_1 with a cutoff radius $r_0 = 6 \times 10^{-10}\,\text{m}$:

1. **Distance Calculation:**
   - $r_{12} = \sqrt{(4-1)^2 + (5-1)^2} = \sqrt{9+16} = 5 \times 10^{-10}\,\text{m}$
   - $r_{13} = \sqrt{(7-1)^2 + (1-1)^2} = \sqrt{36+0} = 6 \times 10^{-10}\,\text{m}$
   - $r_{14} = \sqrt{(8-1)^2 + (2-1)^2} = \sqrt{49+1} = 7.07 \times 10^{-10}\,\text{m}$
   - Since $r_{14} > r_0$, P_4 is outside the cutoff radius and excluded from force calculations.
2. **Force Calculation:**
   - $F_{12} = k\dfrac{|q_1 q_2|}{r_{12}^2} = 8.99 \times 10^9 \dfrac{(1.60 \times 10^{-19})^2}{(5 \times 10^{-10})^2} = 9.21 \times 10^{-10}\,\text{N}$

     Since P_1 and P_2 have opposite charges, the force is attractive and thus negative:

     $$F_{12} = -9.21 \times 10^{-10}\,\text{N}$$
   - F$F_{13} = k\dfrac{|q_1 q_3|}{r_{13}^2} = 8.99 \times 10^9 \dfrac{(1.60 \times 10^{-19})^2}{(6 \times 10^{-10})^2} = 6.39 \times 10^{-10}\,\text{N}$

     Since P_1 and P_3 have the same charge, the force is repulsive and thus positive:

     $$F_{13} = 6.39 \times 10^{-10}\,\text{N}$$
3. **Signed Scalar Aggregation:**

   $$F_{\text{net}} = F_{12} + F_{13} = -9.21 \times 10^{-10} + 6.39 \times 10^{-10} = -2.82 \times 10^{-10}\,\text{N}$$

   The resultant scalar sum of forces on P_1 is $-2.82 \times 10^{-10}\,\text{N}$.

The brute force solution to this problem would require calculating the force between every pair of particles, resulting in $O(N^2)$ runtime complexity. However, by limiting the range of interaction between particles, we open the problem to numerical methods that can approximate the solution. These approximation methods include neighbour lists (Verlet lists), spatial partitioning, multipole methods, etc. It's up to you to decide which method to use, to achieve high performance.

The Particle Dataset

The particle dataset is located in `particles.csv`. The unit of distance is 1×10^{-10} m. Each line in the dataset represents a particle, with the following format:

```
x  y  q
```

`x, y` are the coordinates of the particle
`q` is either "p" or "e" for proton or electron

You are provided with the *oracle* solution, which is the exact solution to the problem, located in `oracle.csv`. This will be used to compute the error between the numerical solution and the exact solution. You are also provided with a secondary, bonus dataset titled `particles_skewed.csv` and its associated oracle solution `oracle_skewed.csv`. See the Code Description section for further details.

Your Code Implementation

Code Logistics

The core implementation must be written in C++, using Pthreads (or `std::threads`) and MPI. You may use other programming languages for accessory scripts, such as graph generation. You may use the compute resources available at UofT (see Accessing Computing Resources) or your own machine. MPI Resource: [1] Pthread/thread Resources: [2], [3], [4].

Code Description

Now that we have gone through the core calculation that your code will be executing, let's take a look at how the code should be implemented. Your program will have 3 distinct modes; each mode represents a different implementation, but the functionality remains the same. The output of your program should be a **single-column csv** with one force value per particle. The three modes of your application should be configurable as an input parameter. In other words, the user can decide which mode they want to use when they run the executable – re-compiling should not be necessary to change modes. There is also an opportunity for bonus marks if you optimize your program for the skewed dataset.

Mode 1: Sequential Computation

This implementation should be entirely serial. No multithreading, just the approximation method you applied to compute the signed scalar force sums on every particle. The cutoff radius must be an input parameter for this mode.

Mode 2: Evenly-Distributed Parallel Computation

In this implementation, you will use the Pthread/thread execution model to create multiple threads and divide the computation among the threads. You must divide the dataset among the threads at the point in time when the thread is created, so that once it finishes its given portion of work, it returns. Divide the work as evenly as possible among the threads. The number of threads and the cutoff radius must be input parameters for this mode.

Mode 3: Load-Balanced, Leader-Based Parallel Computation

In this implementation, you will begin by creating *leader* processes using MPI. Each *leader* must be given an equal partition of the dataset. Each *leader* creates a pool of *worker* threads in the form of Pthreads/threads. Each *leader's* partition of the data must be further partitioned into smaller chunks and placed into a queue that can be accessed by all of its *worker* threads. Worker threads must take one small chunk of data at a time, execute the necessary computation, and then return to the queue to take more work. Threads only return once the queue is empty and all work is done. The number of *leader* processes and *worker* threads and the cutoff radius must be input parameters for this mode.

Data Parsing

You are given a dataset in the form of a `.csv` file. In your code, you must parse this data and save it into a data structure **before** force calculation begins. You should not be reading directly from the file when executing calculations. When dividing the data among processes and threads in Mode 2 and 3, exercise caution; if you need to partition the data such that it is not a perfectly equal partition, you can also do that (but you must partition as evenly *as possible*).

Tips

- **Profiling:** As you develop your code, measure the execution times of different stages of your program, as well as the total execution time. This will help you identify bottlenecks, and will be useful for your report.
- **Thread Management:** Some things you can consider when managing threads:
  - Pinning threads to cores.
  - Tracking which cores threads get assigned to when they are not explicitly pinned.
  - Looking at more detailed hardware monitoring tools, such as:
    - **htop** to view CPU and memory usage in real-time (Linux).
    - **perf** to gather insights about your program, which are given to you after execution (Linux).
    - These and many more good Linux tools and explanations can be found here.
    - If you are using your own machine and OS, please feel free to explore the hardware monitoring tools available for your system.

Report

Along with your code, you will write and submit a report. This report is strongly coupled with the code in that you will need to write and execute test cases to show the performance of your code. All of your test cases must be submitted along with clear instructions on reproducibility in your `README.md`.

Report Format

- Page limit: 7 (not including title page), but you are encouraged to keep it concise.
- Include a title page with the name of the assignment, your name, and student number.
- Use 12pt font, single spaced, 1" margins with page numbers in the bottom right corner.

Report Content

1. **Hardware Resources**: Provide hardware specs for the machine that the program is executed on (number of cores, any other relevant configurations or constraints, etc.).

2. **Program Architecture**: For each mode, provide a figure that shows how your program threads and processes execute the required calculation. Provide an explanation for each figure. In particular, for Modes 2 and 3, explain how multiple software threads/processes and multiple hardware threads/cores are used in the problem, and explain what parts of the code are parallelized and which parts execute as serial code.

3. **Mode 1: Sequential Computation**: In this mode, you will sweep the cutoff radius from 1 until performance plateaus (in intervals of your choice) and compare your results against the oracle (the exact solution). Evaluate how the accuracy of the numerical approximation changes with the cutoff radius.

4. **Mode 2: Evenly-Distributed Parallel Computation**: In this mode you have two independent variables: cutoff radius and number of threads. Complete the following:

   (a) Fix a value for the cutoff radius that achieves a $< 5\%$ error margin compared to the oracle solution. Sweep the number of threads by starting with 1 thread, and increase one thread at a time up to double the amount of cores your machine has. Provide a brief explanation and analysis of your results, including a value for your implementation's speedup in relation to Mode 1.

   (b) Now, experiment with both the cutoff radius and the number of threads until you find the best combination that gives you the fastest execution time for a $< 5\%$ error margin. Provide a chart(s) (table, graph, or diagram) to show the results. Discuss the results, including how you tuned parameters and any tradeoffs made.

5. **Mode 3: Load-Balanced, Leader-Based Parallel Computation**: Repeat the second analysis from Mode 2, but with the number of leader processes, worker threads, and cutoff radius as independent variables.

6. **Execution Time vs. Mode**: Create a chart(s) (table, graph, or diagram) to compare the execution times of each mode. Your chart must show the execution times of specific stages of your program (i.e. parsing data stage, data partition stage, force calculation, etc.). Choose a number of threads to use for mode 2 and 3 (they must use the same number of processes/threads total). Explain your experimental setup and other relevant parameters. Explain which mode is the fastest/slowest and why. Discuss how the configuration parameters affect the modes' performance.

7. **Speedup**: Explain what superlinear speedup means in this case. Was superlinear speedup achievable? Justify your response.

8. **Re-usability**: Which code parts were optimized to be calculated/executed once and results used by many afterwards?

9. **Bonus: Optimizing for Skewed Dataset**: If you have optimized your code for the skewed dataset, explain how you did it and provide a chart(s) (table, graph, or diagram) to show the results. Discuss the results and the tradeoffs you made.

Submission Details

You are required to submit **two** files:

1. The report, in the form of one PDF file named `firstname_lastname_A1_report.pdf`.
2. The code artifacts, in the form of one compressed file in `.zip` or `.tar.gz` format named `firstname_lastname_A1_code.{zip, tar.gz}`. These consist of:
   - All source code required by the assignment.
   - All test cases and plotting scripts used to provide required graphs in the report.
   - A `README.md` file that clearly outlines how to compile and run your code, and how to reproduce all of the test cases.

Rubric

Criteria	**Pts**
Hardware Resource Description	0.5 pts
Program Architecture	3 pts
Mode 1: Sequential Computation	1 pts
Mode 2: Time vs. Number of Threads	3 pts
Mode 3: Execution Time vs. Data Size	3 pts
Execution Time vs. Mode	3 pts
Speedup	1 pts
Re-usability	0.5 pts
Bonus: Optimizing for Skewed Dataset	Up to +1.5
Total (without bonus)	**15 pts**
Total (with bonus)	**16.5/15 pts**

Due Date: October 27, 2024 at 11:59PM EST

Appendix A: Coulomb's Law

At the core of this assignment, you are implementing a program that calculates the force between particles (specifically, point charges). Coulomb's Law tells us that given two charges, the force between them can be found using the equation:

$$F = k\frac{|q_1 q_2|}{r^2}, \quad \text{where:} \quad \begin{array}{l} k = 8.99 \times 10^9\,\mathrm{N}\frac{\mathrm{m}^2}{\mathrm{C}^2} \text{ (Coulomb's constant)} \\ r \text{ is the distance between the particles} \\ q \text{ represents the charge of a particle} \end{array}$$

You will have either a proton or an electron, both of which have a charge of $q = 1.60 \times 10^{-19}\,\mathrm{C}$. We denote a proton as a positive charge, and an electron as a negative charge. Thus, if you have two of the same particles, this creates a repulsive force, and if you have two different particles, this creates an attractive force.

Example: Given two particles and their coordinates in space, we will calculate the magnitude of the force between them. Assume that one unit of distance in this coordinate space is equal to $1 \times 10^{-10}\,\mathrm{m}$. Particle 1 is an electron at (3, 4) and Particle 2 is a proton at (3, 2).

1. Calculate the distance between the points using the Pythagorean theorem:

$$r = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = 2 \times 10^{-10}\,\mathrm{m}$$

2. Plug in the value of k and the charge of each particle into the equation:

$$F = 5.7536 \times 10^{-9}\,\mathrm{N}$$

3. Since the charges are opposite, we know that this is an attractive force. So,

$$F = -5.7536 \times 10^{-9}\,\mathrm{N}$$