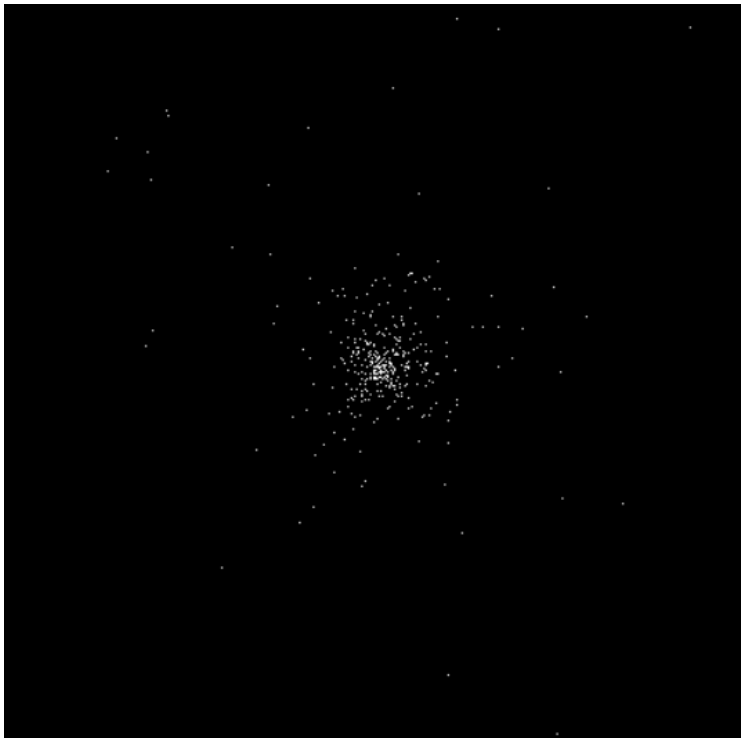
CSC4005 HW3: N-Body Simulation Report

Min Tian 116010168



Content

1. Introduction

1.1 N-Body System.

N bodies initially at rest. Their initial position and masses are to be selected randomly. The gravity between N-body should be described by the following equation:

$$F = G\frac{m_1 \times m_2}{r^2}$$

In which m_1 and m_2 represent two entities masses, r represents the distance between them, and the G is the gravitational constant with value of $6.67259 \times 10^{-11} N \cdot m^2/kg^2$. As Mr. Liu Haolin mentioned in the WeChat group, requirement of the collision and bouncing simulation are not strictly limited. The outcome can be accepted as long as it shows a tendency of shrinkage.

1.2 Goal

Implement

1. a sequential program,
2. a P-thread program,
3. an OpenMP program, and
4. an MPI program.

For two-dimensional N-Body simulation. Bonus (10 points): implement an MPI + OpenMP version.

2. Design

2.1 Data Structure

Each body will be considered as a particle, which indicates it does not have size. There are three attributes we need to describe a particle in 2-dimensional space: location, speed, and weight, which can be represented by a struct, containing x-coordinate, y-coordinate, x-velocity, y-velocity, and weight. A list will be used to contain all these particles.

2.2 Algorithm

Suppose we take time T as an interval in our computation. Our simulation of N-Body is not consistent, the force apply on each particle will give. It a Uniformly accelerated rectilinear motion in T. An iteration of the computation can be decomposed into these steps:

First, start with the leading particle, P_1, in the list, compute the gravitational force between each combination of two particles, from P_1P_2 to P_1P_n, by the law of universal gravitation.

$$F_{12} = G\frac{m_1 \times m_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Second, compute the speed change on x-axis and y-axis applied on P_1.

$$\Delta v_{1x} = v_x + T \times \frac{F_{12}}{m_1} = v_x + T \times G\frac{m_1 \times m_2}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \times \frac{1}{m_1}$$

We can find that the mass of particle 1 can be eliminated. So, we needn't take it into consideration in step one and step two. The equation above can be simplified into:

$$v_{1x} = v_x + T \times F_{12} \times \frac{(x_1 - x_2)}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}}$$

Third, the program will update the position for each particle. New location's x-coordinate can be computed as:

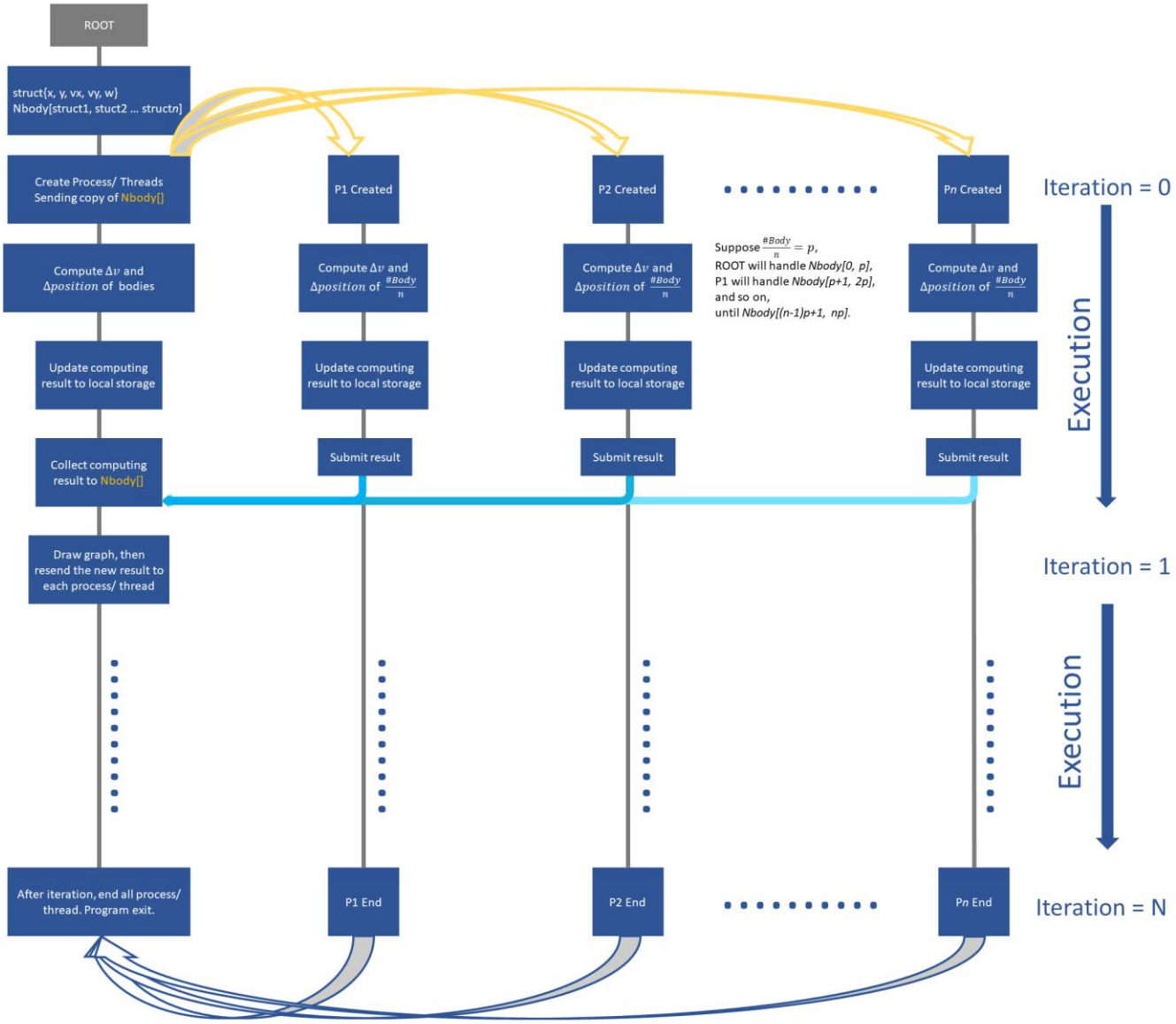
$$x_{1new} = x_{1origin} + v_{1x} \times T$$
$$y_{1new} = y_{1origin} + v_{1y} \times T$$

Above all is a simple description of one iteration. The complexity of the algorithm is expected to be $O(N^2)$, since there's n particles in the universe and in each iteration, it needs n times of calculations to

get the distance change of each particle.

2.3 Parallelism

Parallelism design for the N-Body simulation need to pay special attension on data synchronization. In parallel program with “local storage” and “global storage”, data must be distributed, gathered, synchronized, and then distributed again into the processes or threads. These steps must be executed in every iteration to make sure the compliance. However, in shared-memory parallel program, we only need to operate the pointer in that we could operate on the same piece of data. A simple schematic diagram describing the parallel process has been shown below.



2.4 Timing

Time measurement result in parallel programming can be vary greatly between different measurement methods. To avoid this problem, *MPI_Wtime()* and *clock_gettime()* are used instead of *clock()*. Because *clock()* is the timing function for the CPU running time rather than the program execution time. The result measure by it will significantly larger than the actual value due to the fact that *clock()* takes the sum of all CPU’s running time into account.

3. Program Execution

3.1 Testing Environment

5 Intel(R) Xeon(R) CPU E5−2699 v4 @ 2.20GHz

{1 physical id : 0; 1 address sizes : 46 bits physical, 48 bits virtual} ×5

3.2 Programs Execution Command

For Sequential Version:

Parameters: *Number of Bodies(int), Number of Iteration(int)*

```
g++ hw3-seq.cpp -o seq.out -lX11 && ./seq.out 200 1000
```

For Pthread Version:

Parameters: *Number of Bodies(int), Number of Iteration(int), Number of Threads(int)*

```
g++ hw3-pthread.cpp -o pthread.out -lX11 -lpthread && ./pthread.out 200 1000 2
```

For OpenMP Version:

Parameters: *Number of Bodies(int), Number of Iteration(int), Number of Processes(int)*

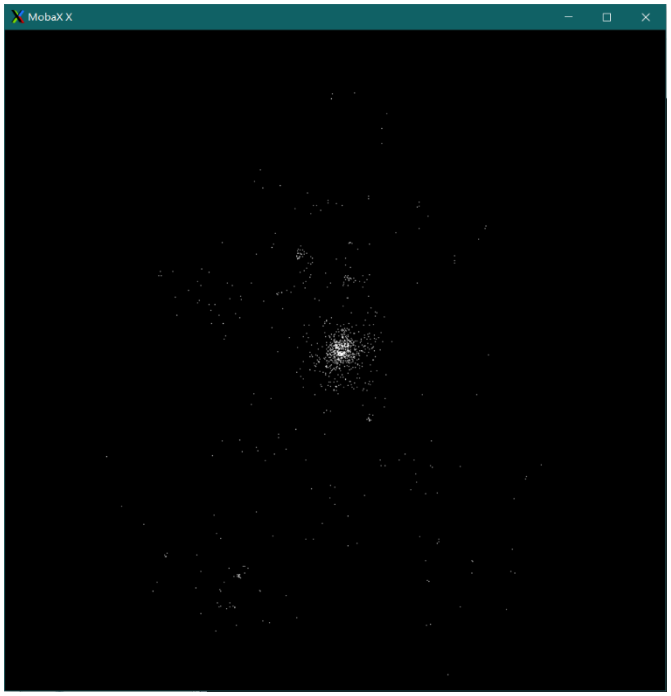
```
g++ hw3-openmp.cpp -o openmp.out -lX11 -fopenmp && ./openmp.out 200 1000 2
```

For MPI Version:

Parameters: *Number of Bodies(int), Number of Iteration(int)*

```
mpicxx hw3-mpi.cpp -o mpi.out -lX11 && mpirun -n 2 ./mpi.out 200 1000
```

3.3 Expected Result

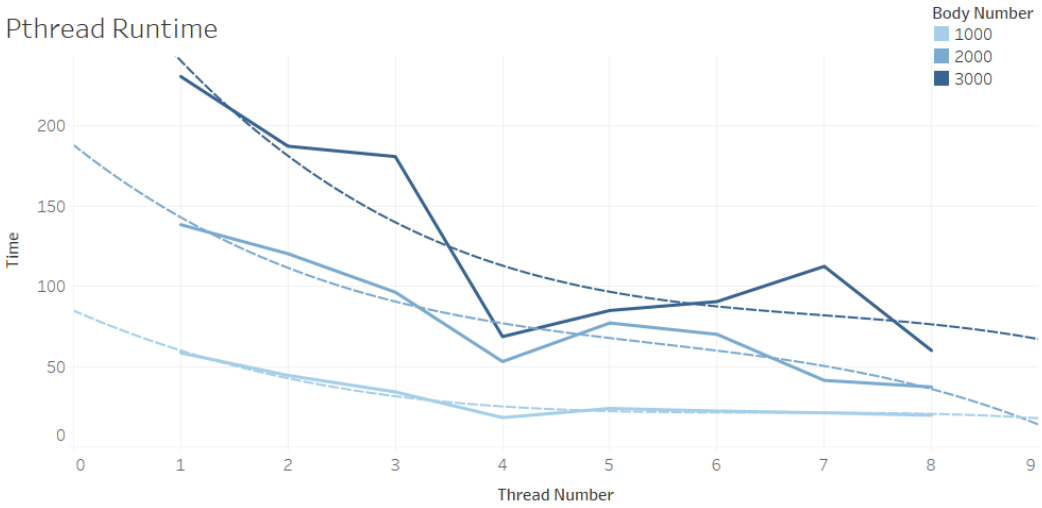


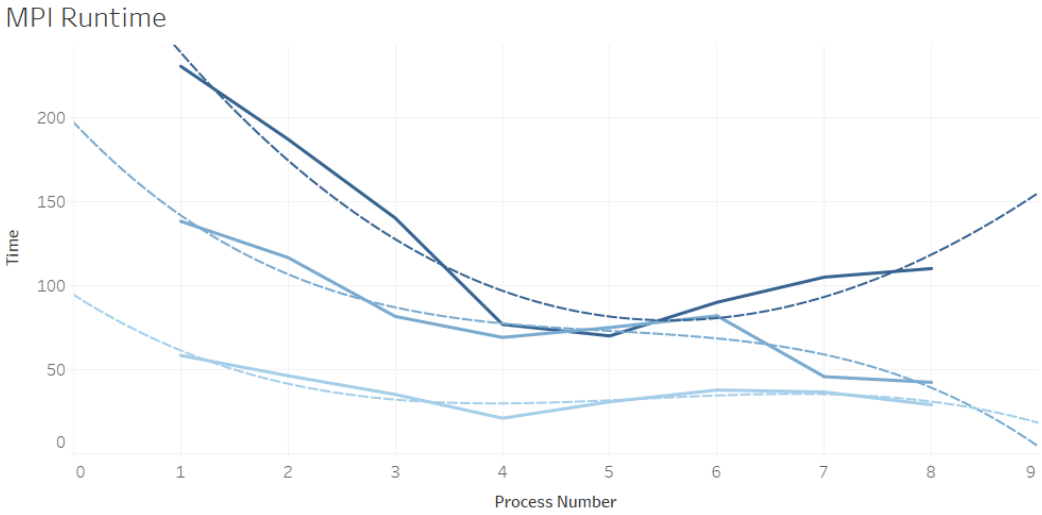
Terminal output would also contain the assignment information and the running time. We can clearly see from the image that all particles tend to shrinkage into the centre by heavier particles. The explosions that I cannot control may blow some particles to coordinates like (NaN, NaN), which looks like the they disappeared, however they actually still on the canvas.

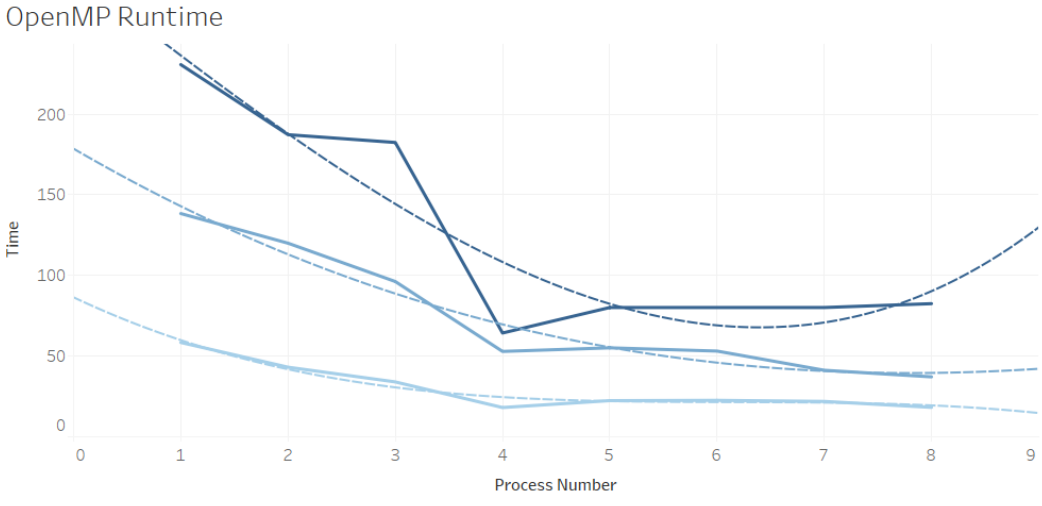
4. Performance Analysis

4.1 Execution Time

The experiment was designed as follow: 1) Set a fix iteration number of 200. 2) Run MPI, Pthread, OpenMP program separately. 3) Record the execution time of process/ thread number from 1 to 8. The graph below shows the examination outcome.

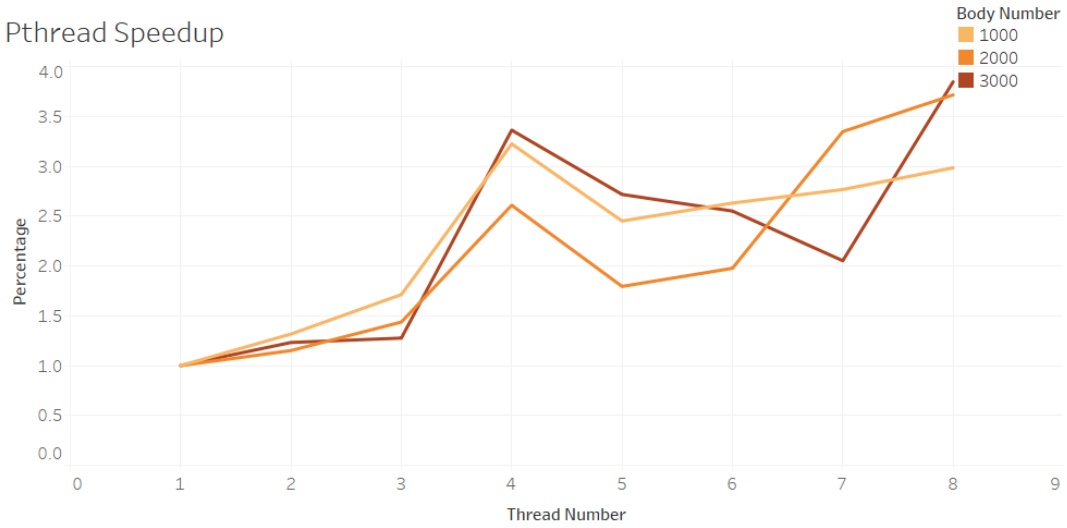


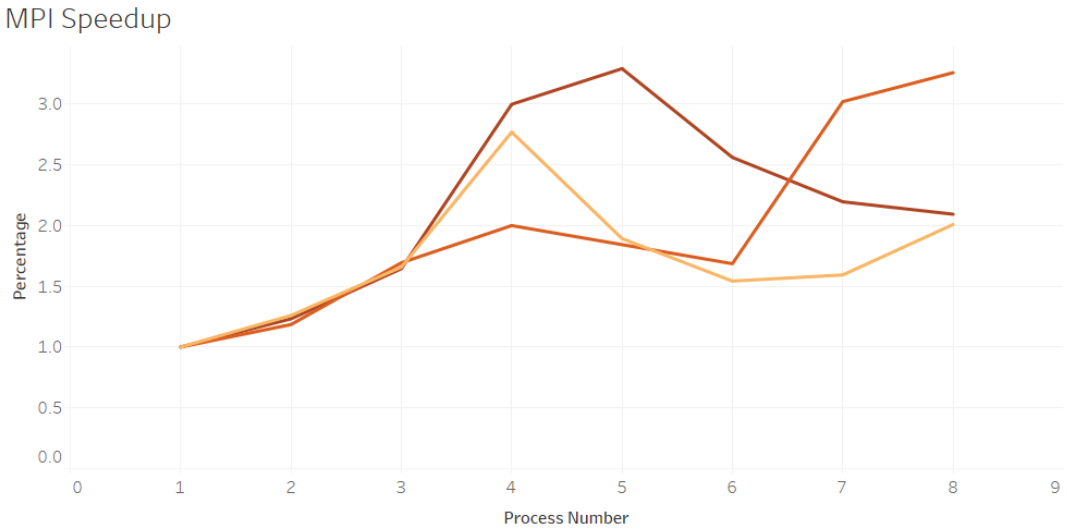


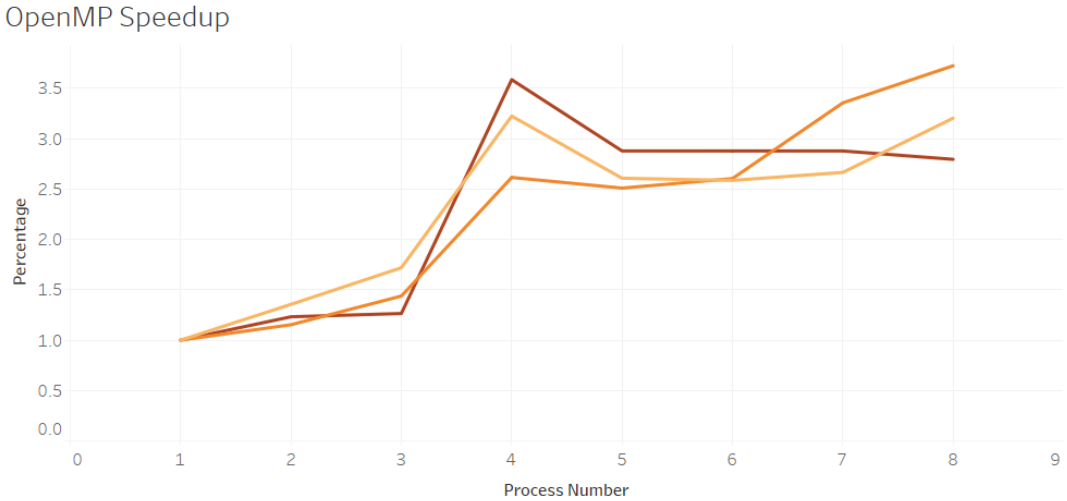


4.2 Speedup Factor

According to definition, we can calculate the speedup factor by $S(n) = \frac{Sequential\ Time}{Parallel\ Time\ with\ n\ processor}$

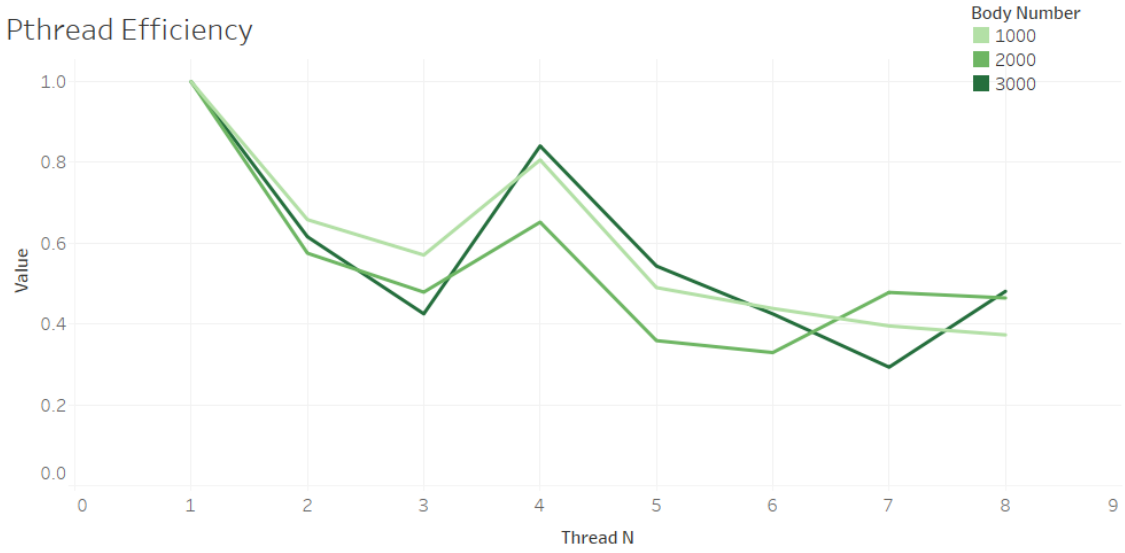


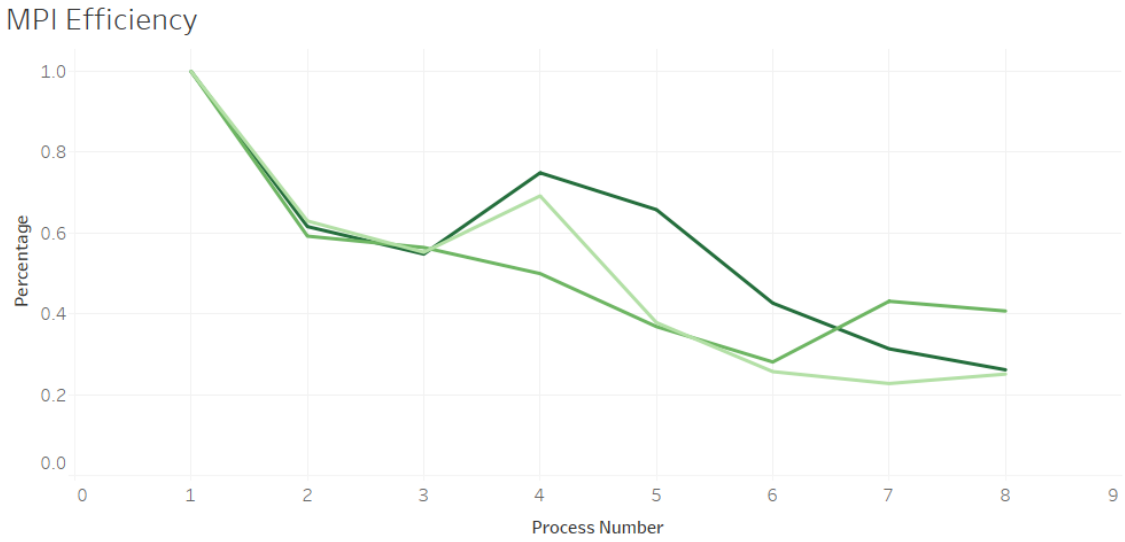


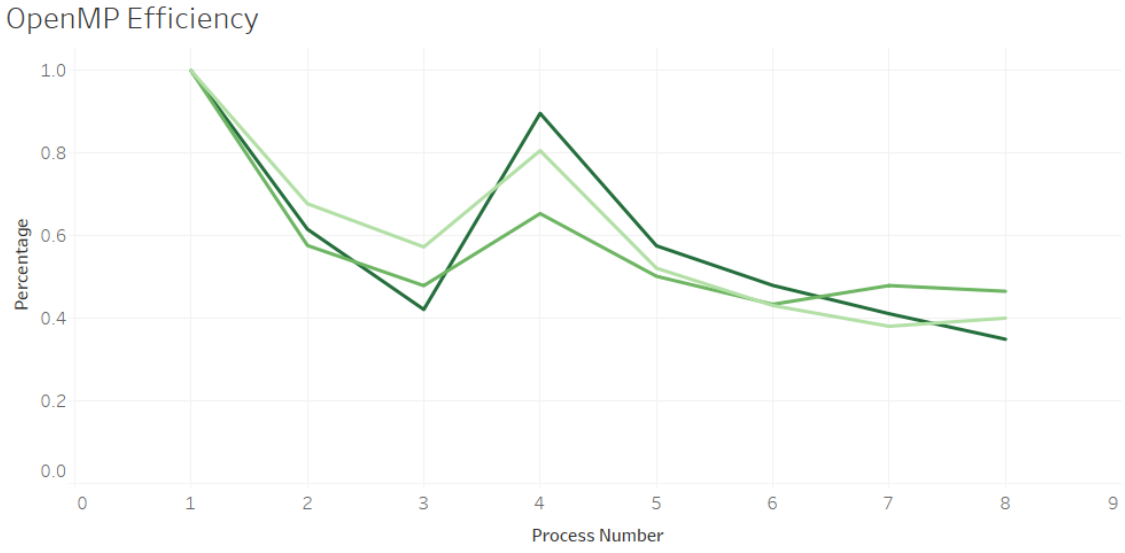


4.3 Efficiency

$$Efficiency = \frac{Execution\ Time\ with\ one\ processor}{Execution\ Time\ with\ multi-processor \times \#processor} = \frac{T_s}{T_p \times n} = \frac{S(n)}{n} \times 100\%.$$







4.4 Analysis

As the results shown above, MPI communication take up a great partition of execution as test size increases. Also, it needs more scheduling time, as the number of processes increases. In the graph of efficiency, it clearly shows that as the number of processors goes up, efficiency drops steeply. As for Pthread and OpenMP, they're in a kind of similar performance level, because the memory sharing mechanism, they run slightly faster than the MPI version.

Due to the time limitation, the experiment did not cover process/ thread number to 16. The data gathered looked a little messy, especially from the speedup factor graph (Figure 1). All three version shows a local maximum at $n = 4$, before a moderate drop on $n = 5$.

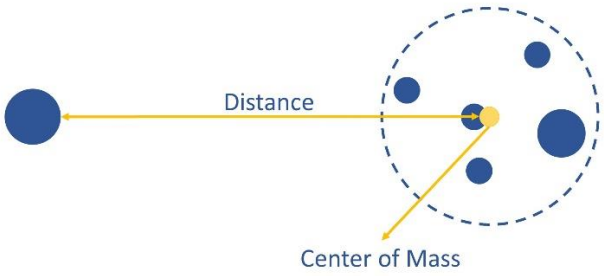
5. Conclusion & Improvement

5.1 Conclusion

In this assignment, we implemented three main parallel programming tools: MPI, Pthread, and OpenMP. Due to the time limitation, I didn't implement the MPI + OpenMP version. I think the most challenging part is the MPI version. Because I spent lots of time handling debugging the data synchronization. What's more, X11 is irritating, causing errors that misleading me into finding algorithm problems. As mentioned in the analysis part, the efficiency of Pthread looks always better than MPI's, which is because communication among processes is more consuming than accessing shared memory within threads.

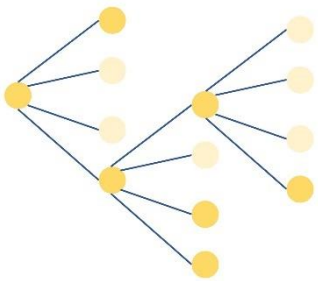
5.2 Barnes-Hut Algorithm

As I looked into numerous resources related to N-Body simulation, I saw Barnes-Hut Algorithm. It can be briefly interpreted as: when calculating force between a body and some others, if the distance between them are considerably large, and others bodies can be seen as a whole. We could calculate multiple bodies' center of weight, then use virtual body to compute force. The complexity can be improved to $O(Nlog(N))$ on ideal situation.



5.3 Quad-Tree

A Quad-Tree can be used to determinate ways to define a cluster, in order to implement Barnes-Hut algorithm. Basically, we could iteratively divide the 2-D space into four parts, until these only one particle in each sub-square box. Then, we could form a tree with four leaves on each node. This may help improve the data query and computation performance.



6. Reference

Stackoverflow, N-Body Simulation in C++: https://stackoverflow.com/questions/27480095/n-body-simulation-in-c

Dorns N-Body: https://github.com/drons/nbody/tree/38ab8a13ad1474ba778ddb68f8c5ffde9ebde757

Bab178 PP-Project3: https://github.com/bab178/PP-Project3/blob/3e1ded35bec1a24af620d40b1d9c3196e71bbf94/NBodySimulation/nbody_pthread.sh

hhLeo Parallel-Programming: https://github.com/hhLeo/Parallel-Programming

Wikipedia N-Body Simulation: https://en.wikipedia.org/wiki/N-body_simulation