CSC4005

Distributed and Parallel Computing

Assignment #3

N-body Simulation

Hajun Lee

117010437

Introduction

In assignment3, we should implement 'N-body simulation' by using sequential, pthread, MPI and openmp. Those are to simulate N-body simulation to compare the efficiency of all those methods what have mentioned.

Method

Basically, we need to simulate and calculate about body collision, status and boundary.

1. All bodies should bounce when they collision.
    A. For this, when dot touch line, velocity will change to – and change direction as well.
2. Assume there have 2 bodies in all bodies which have large mass.
    A. In 1 and 2, I used two of the physics laws called 'conservation of energy' and 'conservation of momentum'.

$$F = G\frac{m_1 m_2}{r^2}, \qquad G = 6.67 \times 10^{-11}$$

There must 2 dots which are mass_A and mass_B.

G will be gravity which is 6.67.

After define, when it runs, it will initialize mass of dots and keep update.

Design

1. sequential.cpp

Before main function:

1. Include Header files
2. Modified variables.

Main function:

1. Initialized dots
2. total will be 0
3. Configuration of Xlib drawing
    A. Initialize related variables
    B. Get screen size and error condition
    C. Set window size
    D. Set window position
    E. Create opaque window
    F. Create graphics context
4. Timer begin
5. Total ++
6. Calculate speed, boundary, force
7. Simulate movement
8. Draw.

2. MPI.cpp

Before main function:

1. Include Header files
2. Modified variables.

Main function:

1. Initialized dots
2. total will be 0
3. When id == 0, MPI broadcast variables to each process
4. Calculate speed
5. Calculate boundary
6. Calculate force
7. Simulate movement
8. MPI gather
9. Timer finish
10. Print output

3. pthread.cpp

Before main function:

1. Include Header files and defined namespace.
2. Modified constant variables.

Main function:

1. Initialized dots
2. total will be 0
3. pthread start
4. Calculate speed
5. Calculate boundary
6. Calculate force
7. Pthread join
8. Finish time
9. Print output

4. openmp.cpp

Before main function:

1. Include Header files
2. Modified variables.

Main function:

1. Initialized dots
2. total will be 0
3. openmp, begin parallel
4. check collision
5. Calculate speed
6. Calculate boundary
7. Calculate force
8. Openmp finish parallel
9. Draw
10. Print output

Performance Analysis:

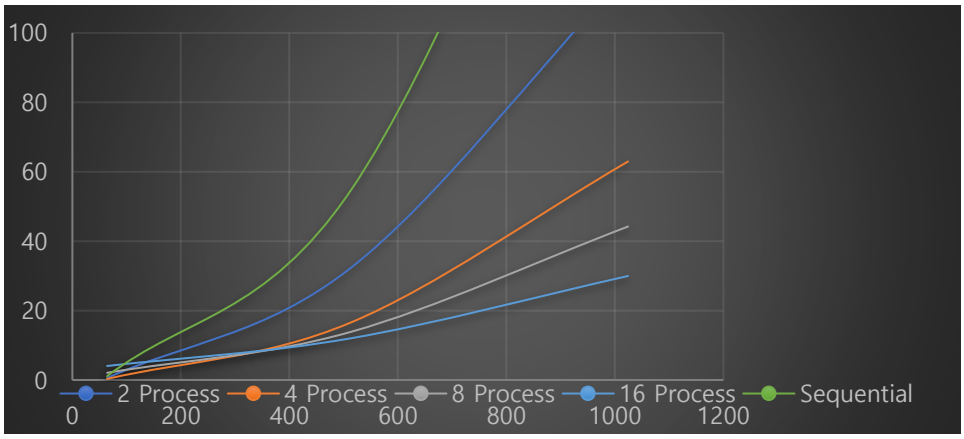
As analysis, MPI and openmp have better performance than pthread and sequential. Since, openmp has better performance when it has large threads and large data size, for MPI, has better performance when it has small process and small data size.

For pthread, threads won't matter about when data size is small enough, but when data size is getting bigger, efficiency of parallel computing will be depending on amount threads. When the number of threads is 2, parallel computing efficiency is lower than pthread cost but 8 or over 8 threads, parallel computing efficiency is high enough. Which means, when dots are smaller, pthread will be have better performance.

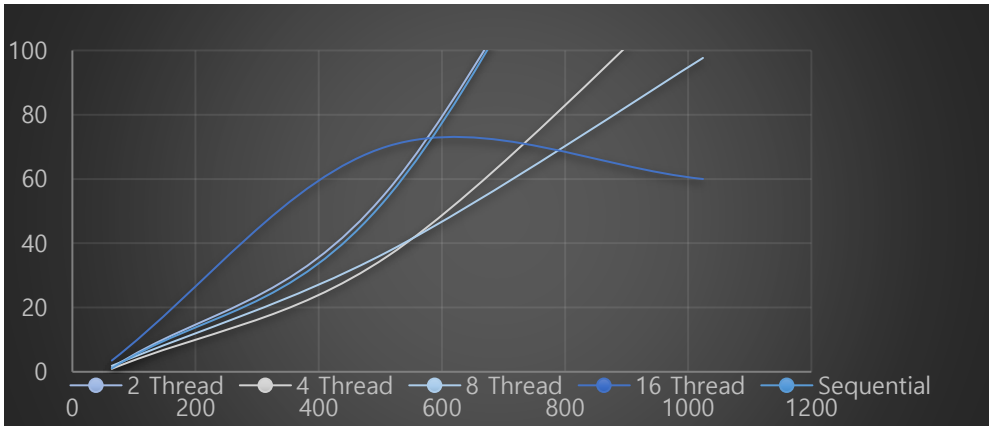
For MPI, it has really good performance in most of the cases when dots are higher and better performance than sequential for sure. When process of parallel is larger, performance will be higher as well.

For openmp, it has better performance than sequential as well and when threads are larger and larger, dots are larger and larger, performance will be better like MPI.

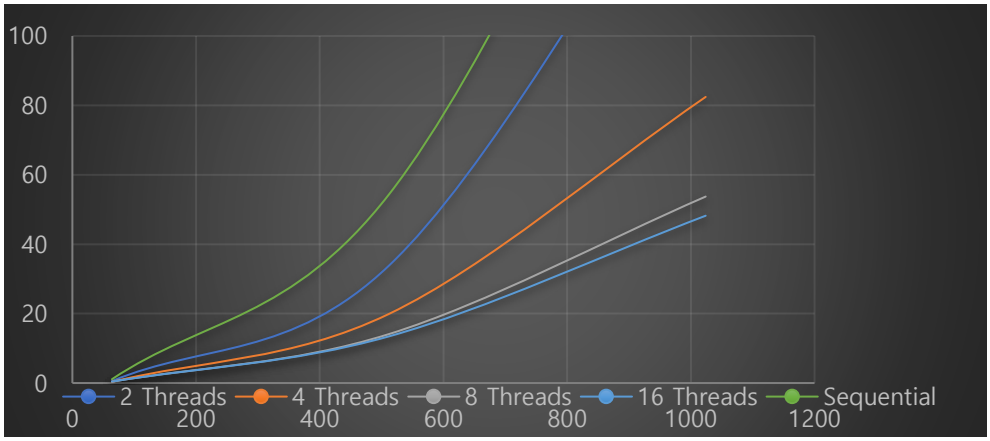
MPI.cpp:



Pthread.cpp:



Openmp.cpp:



Output:

	small	medium	large
sequential	0.23	4.4	34.32
pthread	small	medium	large
2	1.24	34.33	80
4	0.91	17.31	50
8	1.84	21.78	46.72
16	3.44	53	40
MPI	small	medium	large
2	0.56	24.16	120
4	0.3	16.48	62.35
8	2.07	13.49	44.11
16	3.09	12	29.31
openmp	small	medium	large
2	0.56	14.85	90
4	0.33	19.92	60.24
8	0.38	14.11	40.71
16	0.41	13.23	36.05

Conclusion:

In this experiment, we needed to do N-body simulate by using different methods. Normally, when threads and processes go higher, execution time will be decrease but it's depending on parallel computing.

From this experiment, I learned about number of threads and processes. Only keep adding threads and processes won't be necessary to decrease execution time.

How to run a code (sequential.cpp):

To run the sequential version,

1. Compile it with below.
   - g++ -std=c++11 sequential.cpp -o sequential.out -lX11
2. Run it with below.
   - ./sequential.out 100 100

How to run a code (MPI.cpp):

To run the MPI version,

1. Compile it with below.
   - mpic++ MPI.cpp -o mpi -lX11
2. Run it with below.
   - mpirun -n 4 mpi 100 100

How to run a code (pthread.cpp):

To run the pthread version,

1. Compile it with below.
   - g++ -std=c++11 pthread.cpp -o pthread -lX11 -lpthread
2. Run it with below.

- ./pthread 100 100 4

How to run a code (openmp.cpp):

To run the openmp version,

1. Compile it with below.
    - g++ -std=c++11 openmp.cpp -o openmp -fopenmp -lX11
2. Run it with below.
    - ./openmp 100 100 4