The Chinese University of Hong Kong, Shenzhen

CSC 4005

Parallel Computing

Project 3
N-body simulation

Author:
Zijuan Lin

Student Number:
1210909327

November 16, 2022

Contents

1 How to run the implementations

To run the implementations, you can use `make all` to compile all the source code.

Sequential The sequential implementation has two executeble file: `seq` and `seqg`. Both of the executebles will record all points' coordinates in every iteration, one can get the replay by `./video ./<the folder of record>` To get a real-time animation, you can use command `./seqg <n_body> <n_iteration>`.
Result

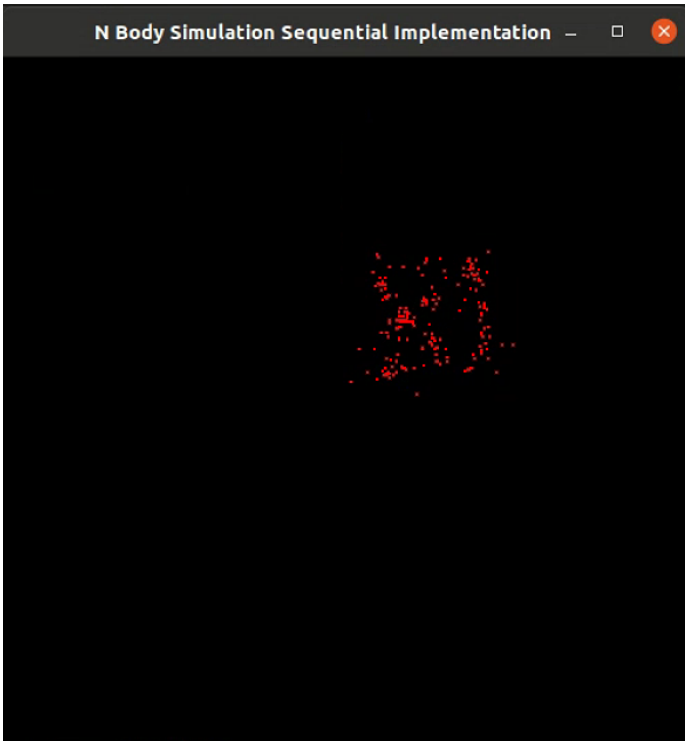

Figure 1: The result of sequential implementation when body=200,iteration=2000

MPI To run the MPI implementation, you can use command `mpirun -np <proc> ./mpig <n_body> <n_iteration>` to get a real-time animation as well as recording the coordinates; or use command `mpirun -np <proc> ./mpi <n_body> <n_iteration>` and only records of the coordinates can be obtained. The records can be used in the same way as mentioned in sequential implementation.
Result

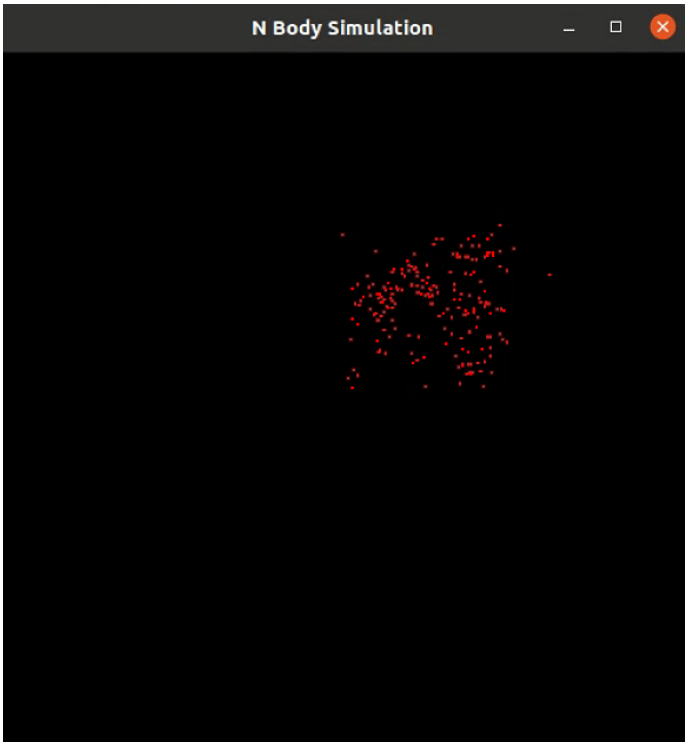

Figure 2: The result of MPI implementation when body=200,iteration=2000

Pthread To run the Pthread implementation, you can use command `./pthreadg <n_body> <n_iteration> <n_thread>` in order to get a real-time animation as well as recording the coordinates; or use command `./pthread <n_body> <n_iteration> <n_thread>` and only records of the coordinates can be obtained. The records can be used in the same way as mentioned in sequential implementation.
Result

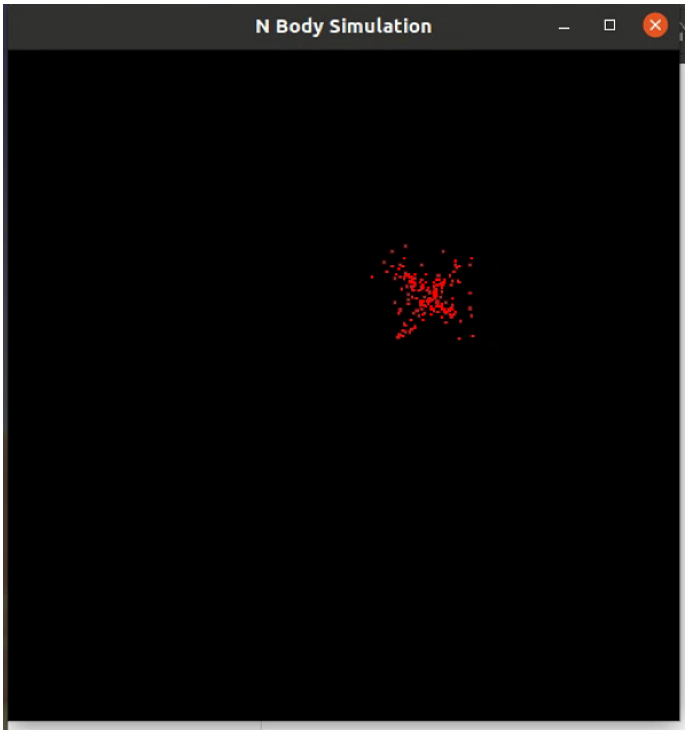

Figure 3: The result of Pthread implementation body=200,iteration=2000

OpenMP To run the OpenMP implementation, you can use command `./openmpg <n_body> <n_iteration> <n_thread>` in order to get a real-time animation as well as recording the coordinates; or use command `./openmp <n_body> <n_iteration> <n_thread>` and only records of the coordinates can be obtained. The records can be used in the same way as mentioned in sequential implementation.
Result

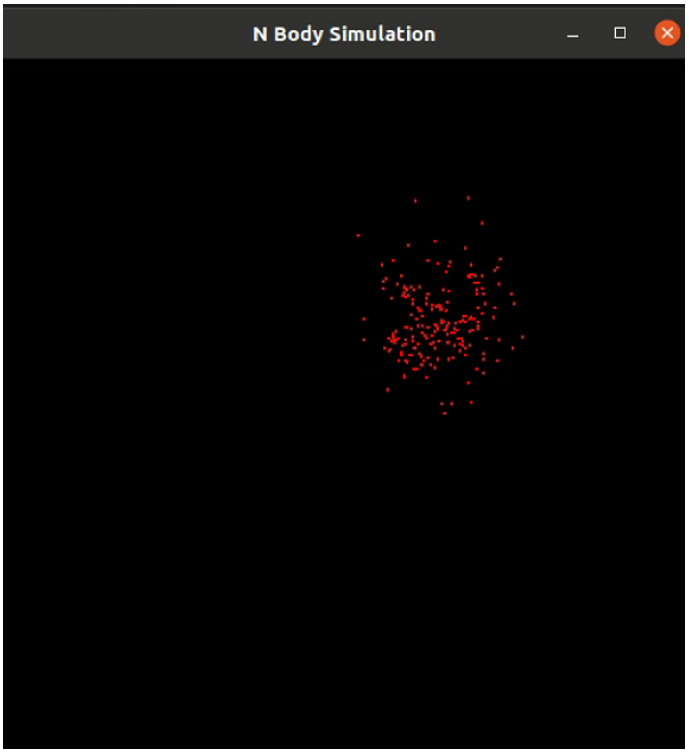

Figure 4: The result of OpenMP implementation body=200,iteration=2000

CUDA To run the CUDA implementation, you can use command `./cudag <n_body> <n_iteration>` in order to get a real-time animation as well as recording the coordinates; or use command `./cuda <n_body> <n_iteration>` and only records of the coordinates can be obtained. The records can be used in the same way as mentioned in sequential implementation.
Result

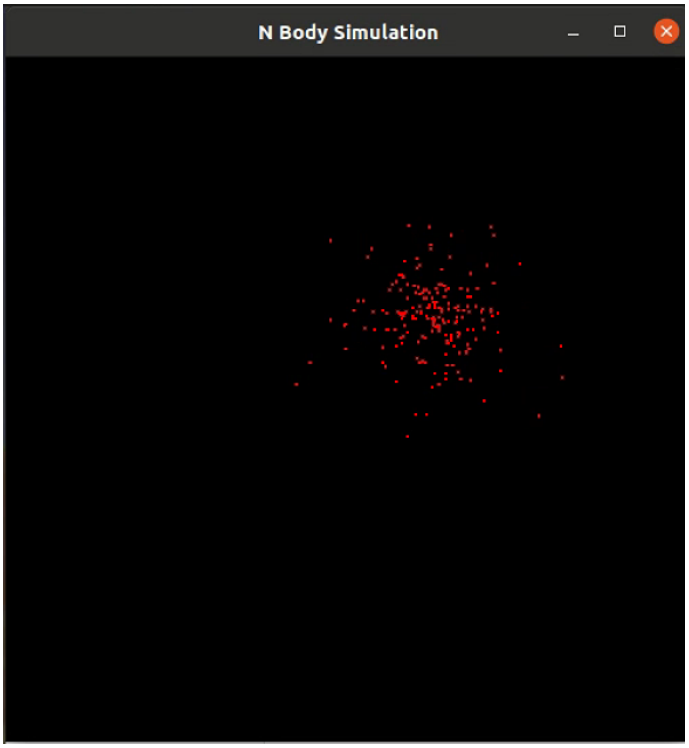

Figure 5: The result of CUDA implementation body=200,iteration=2000

MPI+OpenMP To run the MPI+OpenMP implementation, you can use command `mpirun -np <proc> ./bonusg <n_body> <n_iteration>` to get a real-time animation as well as recording the coordinates; or use command `mpirun -np <proc> ./bonus <n_body> <n_iteration>` and only records of the coordinates can be obtained. The records can be used in the same way as mentioned in sequential implementation.
Result

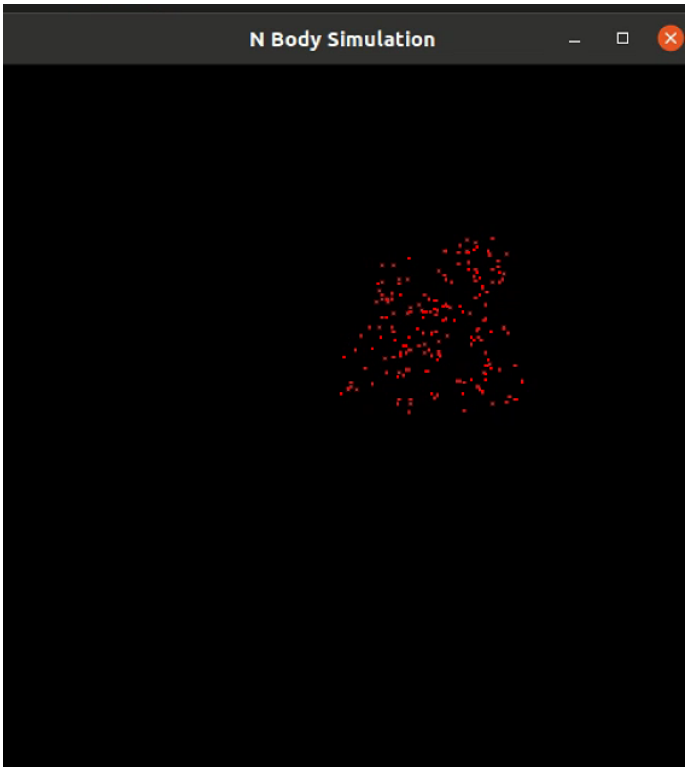

Figure 6: The result of MPI+OpenMP implementation body=200,iteration=2000

2 Problem statement

Mathematical background The N-body simulation is a classic implementation of parallel computing. It is a simulation of a dynamical system of particles, usually under the influence of physical forces such as gravity. Every particle is influenced by gravitational force of other particles. In a 2D stage, the gravitational force of particle i and j can be calculated by

$$r = \sqrt[2]{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$$F_x = \frac{Gm_i m_j}{r^2}(\frac{x_j - x_i}{r})$$

$$F_y = \frac{Gm_i m_j}{r^2}(\frac{y_j - y_i}{r})$$

Thus a particle i will have a total gravitational force of

$$F = \sum_{j=1}^{n}(F_x + F_y)$$

After obtaining the force, we can calculate Δv and update the position of the particles.

Remark:
In this project, collision is not considered, and particles will bounce back when they hit the boundaries. Basic physical data is as follows:

```
int bound_x = 4000;
int bound_y = 4000;
int max_mass = 400;
double error = 1e-9f;
double dt = 0.0001f;
double gravity_const = 100000.0f;
double radius2 = 2.0f;
```

3 Algorithm analysis

Sequential In the calculation process, we need to traverse all other particles in order to get the total force on one particle, thus the time complexity of the algorithm in sequential implementation is $O(N^2)$.

MPI In MPI implementation, the calculation process is distributed into several processes. The detailed procedure is as follows:

1. After generating the particle, the array recording the mass of each body is broadcasted to every process.
2. The velocity and coordinate of each particle is broadcasted.
3. Each process calculate the update of their own distribution of particles, and store the resulting coordinate, velocity in local arrays.
4. The local arrays are scattered and merged, all particles are updated. One iteration is over.
5. GOTO 2, until reaches the max iteration time.

Structure:

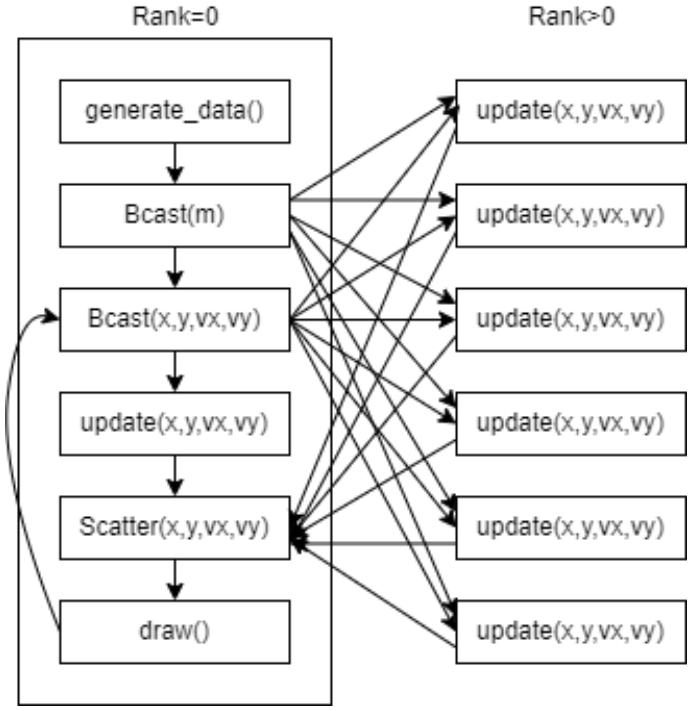

Figure 7: The structure of MPI implementation

MPI+OpenMP The structure of this implementation is quite similar to the MPI implementation, but it makes the processing even faster by distributing the particles calculated by each process into more threads.

The structure is as follows:

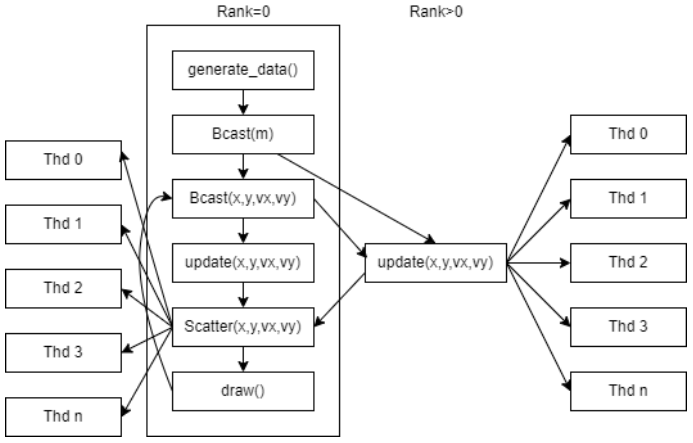

Figure 8: The structure of MPI+OpenMP implementation

CUDA In CUDA implementation, each particle is assigned to one thread in GPU, and they communicate via shared memory by using
`update_velocity<<<n_block, block_size>>>()` and
`update_position<<<n_block, block_size>>>()`.

Pthread & OpenMP Since they are all concurrency methods, their structures is similar.
In an iteration, particles are assigned to numerous threads and update the velocity first. A barrier is set in order to wait for every thread to finish their calculation. After every thread has finished the calculation, they will use the updated velocity to update the position of particles.

4 Performance

In this part, we evalute the acceleration effect of different implementations.

Parallel comparison In this part, we can see the acceleration effect of MPI, OpenMP, Pthread, with a comparison with sequential implementation.

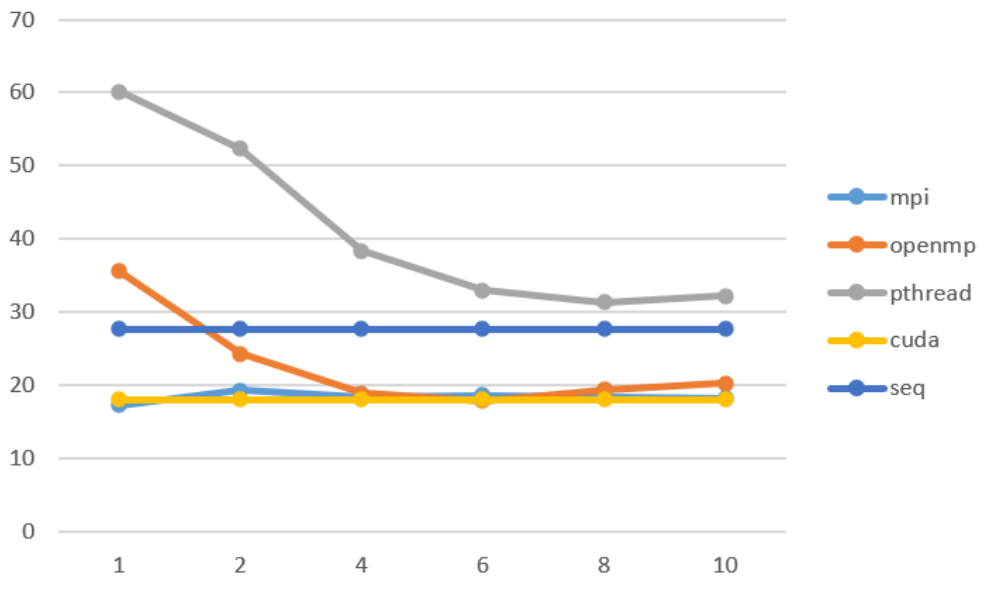

Figure 9: Execution time when body=200,iteration=4000

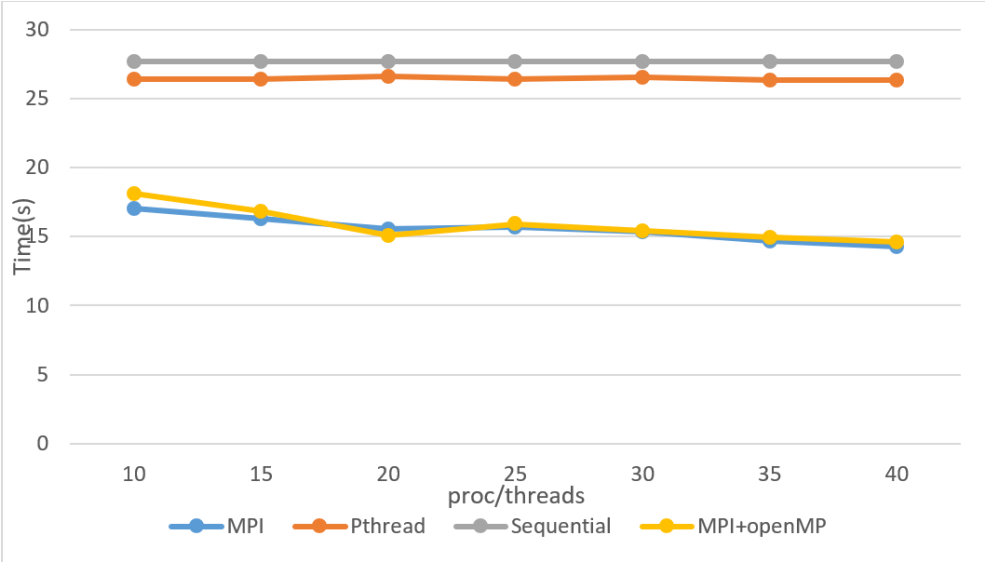

Figure 10: Execution time when body=200,iteration=4000, thd in MPI+openMP is 10

We can see that the MPI implementation is not so effective since $n_{proc} \cdot t_{para} > t_{seq}$. The multi-threaded implementations also have a poor Performance. This is because the rapid memory R/W and exchange. The velocity and position of each particle have to be updated in every process/thread, which brings a significant time cost, even for implementations with shared memory models. What's more, creating and joining threads is are time-consuming, which makes Pthread implementation much slower.

Growth of body & Growth of Iteration Take MPI implementation as example, we can see that the growth of iteration will bring more effect on execution time since the time spent to update the velocity and position of each particle occupies a bigger portion. In comparison, the growth of number of particles only makes the task inside a single cell heavier, thus the time spent in inter-process communication will not have too much influence.

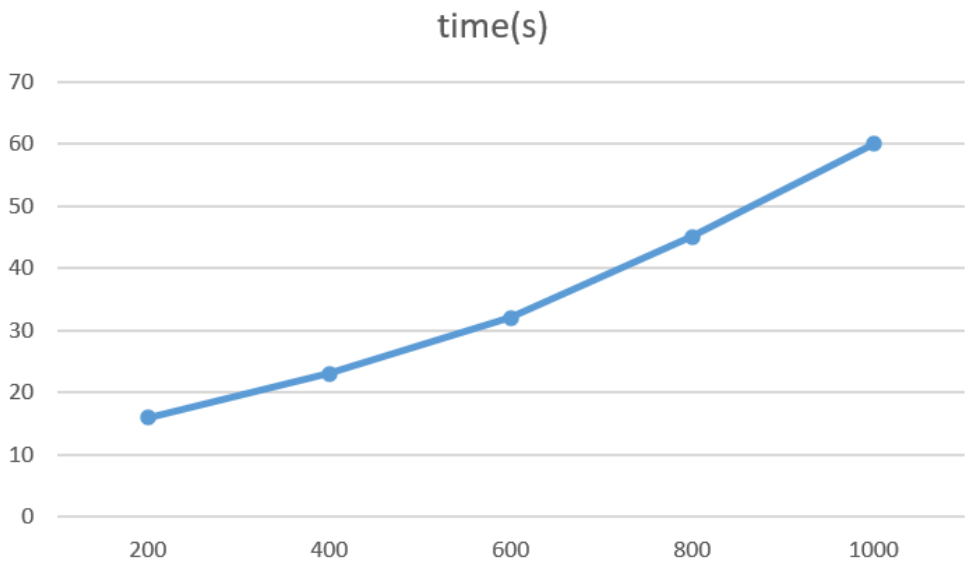

Figure 11: Execution time when proc=20,iteration=4000

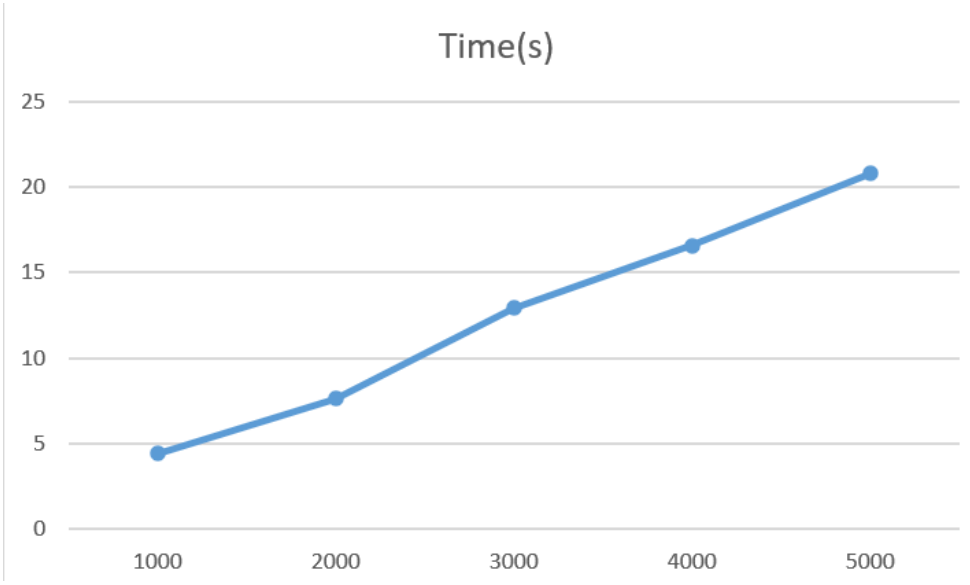

Figure 12: Execution time when proc=20,body=200

5 Conclusion

Even though there are some problems which make the parallel implementation seem less effective, designing a parallel solution to the N-body simulation is still worth-

while. During the experiment, the exchange, broadcast, and collection of information is the most time-consuming. A better memory dispatch strategy may help a lot in increasing the execution speed.