COMP90025 Parallel and Multicore Computing Project 3

Parallel All-Pair Algorithm and Barnes-Hut Algorithm for N-Body Problem

Chuanru Wan
Information Technology
The university of Melbourne
Victory Australia
chuanruw@student.unimelb.edu.au

ABSTRACT

This report discusses about the MPI+OpenMP parallel methods for both All-Pairs algorithm and Barnes-Hut algorithm in N-Body problems. This report is divided into four parts, Introduction, Parallel Algorithm, Experiments and Conclusion. The Introduction part will present overall description for N-Body problem and motivation for parallelization. The Parallel Algorithm will discuss the design of parallelization strategies for both All-Pairs algorithm and Barnes-Hut algorithm. The Experiments part will demonstrate the experiment setup, processes, and results. Finally, the Conclusion part will show analysis of experiments results and further investigated. According to the final experiment results, the effectiveness of parallel algorithms grows dramatically as the increasement of computing nodes.

KEYWORDS

All-Pairs Algorithm, Barnes-Hut Algorithm, N-Body Problem, MPI, OpenMP

1 Introduction

1.1 Problem Description

The aim of this project is using both MPI and OpenMP to parallelize two-dimensional N-Body problems.

The N-Body problem is a common physical simulation problem. In the N-Body system, each particle interacts with the rest of the other particles, resulting in corresponding physical phenomena [7]. For example, celestial bodies system is a very classic N-body system. According to Newton's law of universal gravitation, different celestial bodies in the universe will have interactive force, which varies according to the mass and distance between the two celestial bodies. Similarly, the trajectory of a celestial body ultimately depends on the combined force of the gravitational forces of all remaining celestial bodies in the celestial systems.

There are many algorithms of solving N-Body problems, two algorithms, All-Pairs algorithm and Barnes-Hut algorithm, will be discussed in this report. We will implement parallelization version of these two algorithms and analyze how the parallelization strategy raise the effectiveness compare to the sequential one.

1.2 Algorithms for Solving N-Body Problem

This part will discuss the main ideas of both All-Pair Algorithm and Barnes-Hut Algorithm. And demonstrate formulas which will be used.

1.2.1 All-Pairs Algorithm

This is the most basic algorithm of N-Body problem. If we regard each celestial body as a particle, the formula for computing force between $particle_i$ and $particle_j$ is shown below:

$$F_{ij} = \frac{Gm_i m_j (p_j - p_i)}{\|p_j - p_i\|^3} \quad (1)$$

In formula (1), G present the gravity, m_i and m_j present the mass of $particle_i$ and $particle_j$ and the $(p_j - p_i)$ is a vector which present the distance from $particle_i$ to $particle_j$.

And the formulation for computing combined force from all particles except $particle_i$ itself is below:

$$F_i = \sum_{0 \le j \le N, j \ne i} F_{ij} = Gm_i \sum_{0 \le j \le N,\ j \ne i} \frac{m_j (p_j - p_i)}{\|p_j - p_i\|^3} \quad (2)$$

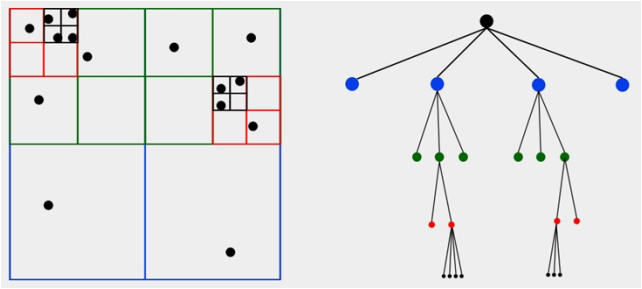
In formula (2), just add all force between $particle_i$ and each of the rest particles, respectively. And this formula can be used to construct All-Pair algorithm for solving N-Body problem.

1.2.2 Barnes-Hut Algorithm

Compare to All-Pairs algorithm, Barnes-Hut algorithm reduces the complexity to $n \log n$ while calculate a less accurate but accepted estimated value of force [9].

Using 2D solution as example, this algorithm divided all particles into cubic cells via quadtree, as you can see in Figure 1. When calculating the force of $particle_i$, the basic formula is similar with formula (2), but only particles from nearby cells will be calculated individually. For those cells which are far away from $particle_i$, each cell will be regard as an entity large particle to

calculate with $particle_i$ [7]. So, this algorithm can dramatically reduce the number of particle pair interactions which should be computed. More specifically, the number of F_{ij} computation will be reduced dramatically from n to $\log n$.



(Figure 1: Subdivision in 2-Dimension (cells and quadtree))

1.3 Motivation

1.3.1 All-Pairs Algorithm

If there are n particles in the N-Body system, we can figure out that the computation of force respect to each particle need n times computation. As there are $n$particles, n times of force computation are needed, the total algorithm complexity is n^2 [8]. As the growth of n, the required computation time is also raising dramatically. After analyzing the computation processes of N-Body problem, we can aware that the computation of $F_i$of each $particle_i$ is independent. What is more, the computation of each F_{ij} inside F_i computation is also independent. So, we can regard N-Body problem as a loosely synchronous problem. We can implement a parallelization strategy for All-Pairs Algorithm to raise effectiveness and save execution time.

1.3.2 Barnes-Hut Algorithm

The complexity of Barnes-Hut Algorithm is $\mathrm{nlog}\, n$. Same with All-Pair Algorithm, the computation of $F_i$of each $particle_i$ is independent. But the complexity of computation of F_i is reduced to $\log n$, but a recursion method will be used here to traverse the quadtree [2]. Even the recursion method is not appropriate for parallel, we still can regard this algorithm as a loosely synchronous algorithm because the outside loop is easy for parallelization. So, a parallelization strategy can be implemented for this algorithm to raise effectiveness and save execution times.

2 Parallel Algorithm

2.1 Parallel Methods for All-Pairs Algorithm

This section will present the sequential and parallel pseudocodes of All-Pair Algorithm and Barnes-Hut Algorithm, respectively.

2.1.1 Sequential All-Pairs Algorithm

According to formula (2) in section 1.2.1, we can construct a pseudocode for All-Pairs sequential algorithm as below:

Algorithm 1 All-Pair Sequential Algorithm

```
Input: particles
Output: result
function CaculateNextLocation
    for each particle in particles do
        Fi ← 0
        for each particle in particles do
            Fij ← Formula(1)
            Fi+ = Fij ← Formula(2)
        end for
        update location
        update velocity
    end for
    return location
end function
```

In Algorithm 1, firstly calculate and update the current force of $particle_i$, and then use the updated force, last location and last velocity to calculate the new location. At last, update the velocity for next round of computation. This pseudocode just shows one round of computation which means each particles only move one time.

2.1.2 Parallel All-Pairs Algorithm

For construct parallel algorithm, we consider using both MPI and OpenMP. Firstly, we only look at the computation of F_i, we can get below formula.

$$F_i = \sum_{1 \le j \le N, i \ne j} F_{ij} \tag{3}$$

Each F_{ij} is independent. So, we can compute them parallelly. We will use OpenMP to parallelize this loop here. We have set $\#pragma\ omp\ parallel\ for\ schedule(guided,4)$ to allocate tasks for each thread. We assume that the number of threads is T and computation tasks is N here. Theoretically the allocated tasks of each thread will be N/T if the execute time of each task is same.

And then, considering F_i computation of each particles, they are also independent with others. For example, F_i and F_{i+1} can be computed at the same time. We use MPI to allocate and execute these F_i tasks. For example, if the number of tasks which may be same with the number of particles is N and the number of processes is T, each process will execute N/T tasks. MPI_SCATTER is used here to allocate tasks equally by sending particles to each process. All the computation of F_i need masses and position of all particles. So, MPI_BROCAST is used to send masses and position of all particles to all processes.

Combine both MPI and OpenMP, we can construct below pseudocode:

Algorithm 2 All-Pair Parallel Algorithm

Input: particles
Output: *result*

```
1: function CaculateNextLocation
2:     rank,size
3:     MPI BROCAST(particles masses)
4:     MPI BROCAST(particles positions)
5:     MPI SCATTER(particles forces)
6:     !MPI Do allocated tasks
7:     for each particle in allocated particles do
8:         Fi ← 0
9:         !omp parallel do schedule(guided,4)
10:        for each particle in particles do
11:            Fij ← Formula(1)
12:            Fi+ = Fij ← Formula(2)
13:        end for
14:    end for
15:    MPI GATHER(particles forces)
16:    update position
17:    update velocity
18:    return location
19: end function
```

s

MPI_GATHER will collect all computed forces and update velocity and position of all particles.

2.2 Parallel Methods for Barnes-Hut Algorithm

2.2.1 Sequential Barnes-Hut Algorithm

Algorithm 3 Barnes-Hut Sequential Algorithm

Input: *bodies*, *treenode*
Output: *result*

```
1: function Main
2:     treenode ← ConstructQuadtree
3:     for each body in bodies do
4:         force[i] ← CaculateForce(body, treenode)
5:     end for
6:     update locations and velocities
7: end function
8:
9: function CaculateForce(body, treenode)
10:    for each particle in particles do
11:        if treenode is a leaf then
12:            return body-body interaction (body,treenode)
13:        else
14:            if body is distant enough from treenode then
15:                return body-cell interaction (body,treenode)
16:            else
17:                for all treenode' in nonemptyCildren(treenode) do
18:                    accumulate CaculateForce(body,treenode')
19:                    return accumulated interaction
20:                end for
21:            end if
22:        end if
23:    end for
24:    return location
25: end function
```

Above Algorithm 3 is the pseudocode of Sequential Barnes-Hut Algorithm. The variables 'bodies' contain the masses, positions and velocities of all particles. The function CaculateForce will do recursion in quadtree to sum up all F_{ij} between bodies or body and cell. The 'body-body interaction' means calculate the force using the masses of bodies and distance between them. The 'body-cell interaction' means calculate the force using the mass of body and mass of cell and distance between body and center of cell. The force will be accumulated through the recursion in the quadtree.

2.2.2 Sequential Barnes-Hut Algorithm

The calculation of F_i is still independent but the inside loop of each F_i is using recursion which is hard to do parallelization. So, we decided using both MPI and OpenMP in the outside loop.

If there are totally N tasks, P processes and T threads inside each process. We will equally assign tasks into each process. So, the allocated tasks of each process will be N/P . And then, we can equally assign tasks inside a process into each thread. Finally, the allocated tasks of each thread will be $N/(PT)$. The pseudocode is shown below:

Algorithm 4 Barnes-Hut Parallel Algorithm

Input: *bodies*, *treenode*
Output: *result*

```
1: function Main
2:     !MPI do allocate tasks
3:     MPI BROCAST(bodies)
4:     treenode ← ConstructQuadtree
5:     !omp parallel do schedule(guided,4)
6:     for each body in allocated tasks do
7:         force[i] ← CaculateForce(body, treenode)
8:     end for
9:     update locations and velocities
10: end function
11:
12: function CaculateForce(body, treenode)
13:     if treenode is a leaf then
14:         return body-cell interaction (body,treenode)
15:     else
16:         if body is distant enough from treenode then
17:             return body-cell interaction (body,treenode)
18:         else
19:             for all treenode' in nonemptyCildren(treenode) do
20:                 record Fij ← CaculateForce(body,treenode')
21:             end for
22:             Fi ←combine all record Fij
23:             return Fi
24:         end if
25:     end if
26:     return location
27: end function
```

3 Experiments

3.1 Environment

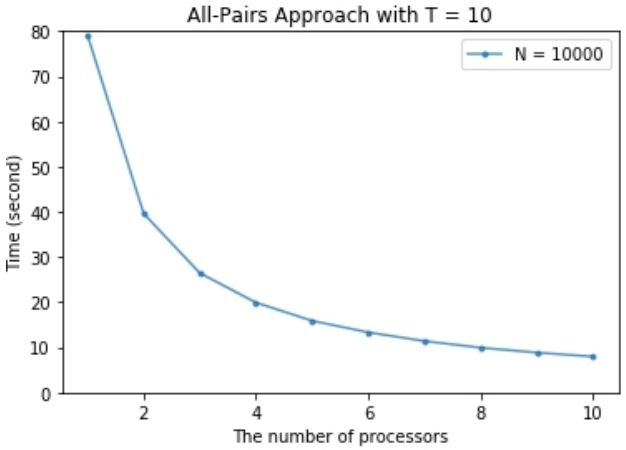
In this experiment, all algorithms were executed on Spartan platform of UniMelb. The node in this system is consisted by four Intel(R) Xeon(R) Gold 6254 processors which contains 16 physical cores. And all experiments execute on snowy partition.

3.2 Setup

For All-Pairs Algorithms, the number of particles is 10000, the number of rounds is 10 which means all particles will move 10 times and the experiment range of the number of nodes is [1,2,3,4,5,6,7,8,9,10]. For Barnes-Hut Algorithm, the number of particles is 100000, the number of rounds is still 10 but the range of the number of nodes is [1,2,3,4,5,6,7,8,9,10,11,12].

3.3 Result

3.3.1 All-Pairs Algorithm



(Figure 2: All-Pairs Algorithm experiment results)

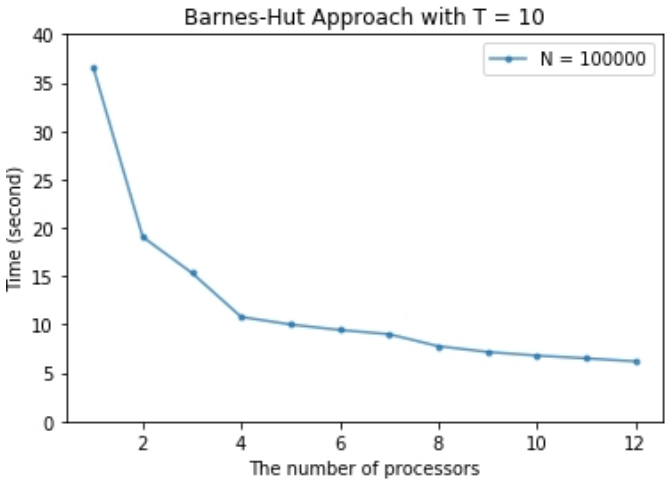
According to Figure 2, the experiment result is under expectation. As we increase the number of nodes from 1 to 2, the execute time also is reduced around 1/2. And the execute time is reduced to around 1/4 since we increase the number of nodes to 4. Because the increasement of nodes will also increase the time of communication time.

If we assume that the execute time with one node as $time\_1$, the execute time with two nodes as $time\_2$ and time which may be caused by unavoidable facts like communication, poor load balance or different runtime environment as $unexpected\_time$. We can get a formula:

$$time\_2 = \frac{time\_1}{2} + unexpected\_time \quad (4)$$

And therefore, the chart in Figure 2 is not a straight line. But overall, this result is acceptable.

3.3.2 Barnes-Hut Algorithm



(Figure 3: Barnes-Hut Algorithm experiment results)

According to Figure 3, the overall trends in the chart is expected. It is similar with the chart in Figure 2. The formula (4) can also be used here to explain why the chart is not a straight line. As the number of particles is 100000 which is 10 times than the number of particles in All-Pairs algorithm, the communication time should be larger. And this should be why the efficiency increment is slight when the number of nodes goes to high.

4 Conclusion

According to results we demonstrated above, both All-Pairs algorithm and Barnes-Hut algorithm pass the speed up test. So, the parallelization implementation is successful. If we ignore the unexpected time which are discussed in section 3.3.1, theoretically, the effectiveness of programs linearly grows as the increment of nodes. Through this project, I have gain experience of using parallelization to solve real problems. I have learned how to analyze sequential algorithm to figure out independent computations in loops and using MPI or OpenMP to parallelize these computations.

For further investigation, I will do more research and test on using OpenMP to parallelize computations. I have used OpenMP in this project, but the result is undesirable. The increasement of processes in MPI can dramatically raise the effectiveness while the increasement of threads in each process contribute less to the effectiveness. As I know, the physical cores of each processor on spartan is 16. So, I have increased the threads to 16, but the result is not desirable. More experiments may be needed to figure out issues.

REFERENCES

[1] Guy Blelloch and Girija Narlikar. A practical comparison of n-body algorithms. Parallel Algorithms: Third DIMACS Implementation Challenge, October 17-19, 1994, 30:81, 1997.

[2] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. Journal of computational physics, 73(2):325–348, 1987.

[3] Leslie Greengard. The Numerical Solution of the N-Body Problem. Computers in Physics 4, 142 (1990); doi: 10.1063/1.4822898

[4] P. H. J Barnes. A hierarchical o(n log n) force calculation algorithm. Nature, 324:446–449, 1986.
[5] S.Aarseth. Gravitational n-body simulations. Cambridge University Press, 2003.
[6] K.W.Tom, Ventimiglia. Barnes-hut galaxy simulator. http://www.cs.princeton.edu/courses/archive/ fall03/cs126/assignments/barnes-hut.html, 2003.
[7] ScalaBlitz team, Parallel Barnes-Hut Simulation, https://scala-blitz.github.io/home/documentation/examples//barneshut.html
[8] Aaron Harwood, Tuck Leong Ngun (2020)," Parallel algorithm techniques ", School of Computing and Information Systems, The University of Melbourne, unpublish.
[9] Tushaar Gangavarapu (2020)," OpenMP Case Study: The Barnes-Hut N-body Algorithm", unpublish.