N-body simulation using Barnes-Hut algorithm

Samyak Jain
Boston University
samyakj@bu.edu

Nihar Dwivedi
Boston University
ndwivedi@bu.edu

Project Report for
EC526 ~ Parallel Programming

taught by

Richard Brower
Professor
Electrical and Computer Engineering

BU College of Engineering

1.Introduction

Our goal in this project is to demonstrate that the N-body simulation task used to study the behavior of galaxies and their evolution is highly parallelizable on modern multi-core CPU platforms. Here we have implemented the Barnes-Hut algorithm (used for N-body simulations) on a multi-core CPU platform using the OpenMP API.

1.1 What is the N-body simulation?

In physics and astronomy, an N-body simulation is a simulation of a dynamical system of particles, usually under the influence of physical forces, such as gravity. N-body simulations are widely used tools in astrophysics, from investigating the dynamics of few-body systems like the Earth-Moon-Sun system to understanding the evolution of the large-scale structure of the universe. In a basic N-body simulation, every body exerts a force on every other body in the system. This means that in every step of the simulation a net force from all the other bodies has to be computed for each of the bodies. An N-body simulation approximates the motion of particles, often specifically particles that interact with one another through some type of physical forces (like gravity, electrostatic forces, etc.). In direct gravitational N-body simulations, the equations of motion of a system of N particles under the influence of their mutual gravitational forces are integrated numerically without any simplifying approximations.

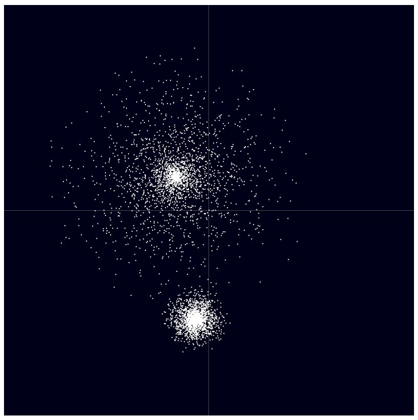


Fig 1. Particle distribution resembling two neighboring galaxies

Talking about gravitational N-body simulations, the most obvious way to perform this simulation is through direct integration of the

Newtonian gravitational force equation, where the total force on each particle is the superposition of the forces imparted by each of the other particles:

$$\sum_{i \neq n}^{N} \frac{G m_n m_i}{{r_{ni}}^2} \hat{r}_{ni}.$$

Also, it is worth noting that in this naive/brute-force approach used for solving N-body simulation, the interaction between two bodies in a system is independent from all other bodies present. This means that this approach is embarrassingly parallelizable. But as we discussed before, the complexity of computation of this approach is $O(n^2)$, where n is the number of bodies present and this becomes a huge computation issue when simulating large systems i.e. systems containing ~10000s of bodies.
Therefore, there are other methods that are employed to perform these simulations in order to get better computational performance. Some of these are:

A) Barnes-Hut algorithm
B) Fast multipole method
C) Parallel Tree Multipole Algorithm

But the most common one of these is the tree algorithm, the Barnes-Hut algorithm, which we are going to discuss in detail in the next section.

2.Barnes-Hut Algorithm

The Barnes Hut simulation is an approximation algorithm for performing an N-body simulation in which we build a spatial tree to form a hierarchical clustering of bodies, so that during the acceleration computation phase, each body can treat far away clusters as a single large body to reduce total number of computations. Since the gravitational force decreases as $1/r^2$, bodies that become substantially far enough from other bodies become significantly less important to its subsequent motion. The Barnes-Hut algorithm provides a systematic way to define "*far-away*" along with providing a method for approximating the force due to far-away bodies. Thus, we get an algorithm which has a time complexity of O(n log n) in comparison with $O(n^2)$ of the direct sum algorithm.

Thus the force on each particle can be given by the general equation:

```
Force = external_force + nearest_neighbor_force +
                 far_field_force
```

In Barnes-Hut algorithm, the simulation volume is usually divided up into cubic cells via the construction of a spatial tree (an *octree* for 3D space; or a *quadtree* for 2D space) so that only particles from nearby cells need to be treated individually, and particles in distant cells can be treated as a single large particle centered at the cell's center of mass (or as a low-order multipole expansion). A *quadtree* divides the 2D space into four equal squares (*subtrees*) and continues doing so until each leaf node in the tree has only a single body. This can dramatically reduce the number of particle pair interactions that must be computed. The figure below shows how the Barnes-Hut algorithm works:

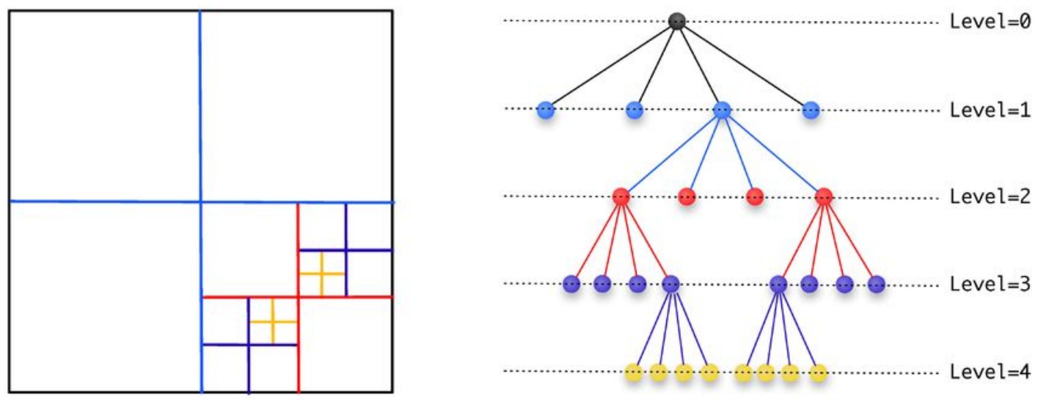
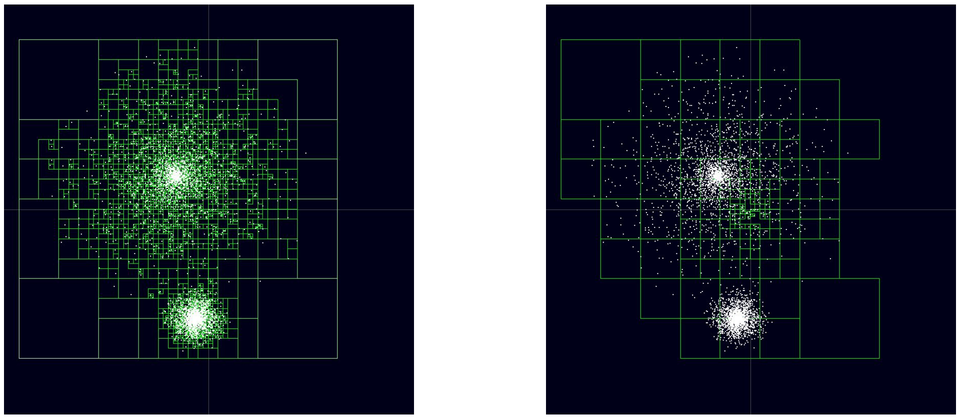


Fig 2. Quadtree construction in Barnes-Hut

Once the tree is populated, we have to calculate the force on each body. For this, we take a body and a node. Then we take the ratio of the side of the node's region to the distance between the body and center of mass of the node. This ratio is then compared to a predefined number *theta* (Θ). The larger this number, the less accurate is the simulation but with a smaller value of *theta*, the simulation takes longer to run and for *theta* = 0, this turns into the brute-force method.



a) b)

Fig 3. a) Complete Barnes-Hut tree (nodes that do not contain bodies are not drawn)
b) Nodes of Barnes-Hut tree for calculating force on a particle

2.1 Algorithm used

```
Algorithm 1 Sequential implementation of Barnes-Hut
for step in simulation steps do
    for body in total bodies do
        update body position and half time step velocity
    end for
    initialise quadtree
    for body in total bodies do
        insert body into quadtree
    end for
    traverse quadtree to assign masses and center of masses to nodes
    for body in total bodies do
        update body acceleration
    end for
    for body in total bodies do
        update body velocity
    end for
end for
```

The above pseudo-code shows the serial Barnes-Hut algorithm which gives a computational complexity of O(n log n). As we can see in this algorithm, there are a number of for loops which can be parallelized without changing the functionality of the code. So the parallel implementation of this code using OpenMP is shown below:

Algorithm 2 Parallel implementation of Barnes-Hut

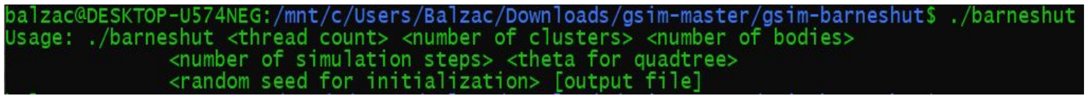
```
for step in simulation steps do
    #pragma omp parallel for schedule(static)
    for body in total bodies do
        update body position and half time step velocity
    end for
    initialise quadtree
    #pragma omp parallel for schedule(static)
    for body in total bodies do
        insert body into quadtree
    end for
    traverse quadtree to assign masses and center of masses to nodes
    partition bodies between threads
    #pragma omp parallel for schedule(static)
    for thread in thread count do
        for body in total bodies do
            update body acceleration
        end for
    end for
    #pragma omp parallel for schedule(static)
    for body in total bodies do
        update body velocity
    end for
end for
```

Here, we have used the OpenMP pragmas for the outer *for* loops in order for them to run parallelly with static scheduling. Also, we have distributed the bodies among the CPU threads based on the amount of work to be done while traversing the tree instead of random distribution. This helps in achieving better performances.

3.Implementation

For running our code, we used an Intel i7 8700k processor with 6 cores (can have a total of 12 threads), 5 GHz clock speed, and 16 GB RAM.
We ran our code on 1, 2, 4, 8, and 12 threads to get a better understanding of the behavior of this algorithm with an increasing number of threads.

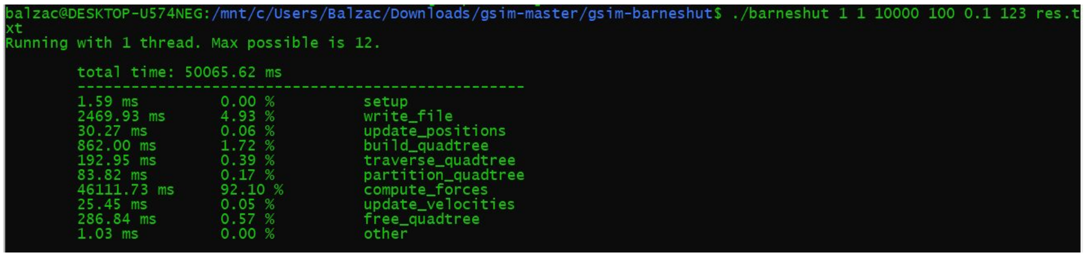
We have simulated a system with 1 cluster containing 10000 bodies and 100 steps of simulation. We have used four different values of *theta* (0.1, 0.3, 0.5, 0.6) for simulation purposes in order to see the behavior of the N-body simulator with varying values of *theta*.
Following usage options are required to run the program-



```
balzac@DESKTOP-U574NEG:/mnt/c/Users/Balzac/Downloads/gsim-master/gsim-barneshut$ ./barneshut
Usage: ./barneshut <thread count> <number of clusters> <number of bodies>
            <number of simulation steps> <theta for quadtree>
            <random seed for initialization> [output file]
```

Fig 4. Usage options

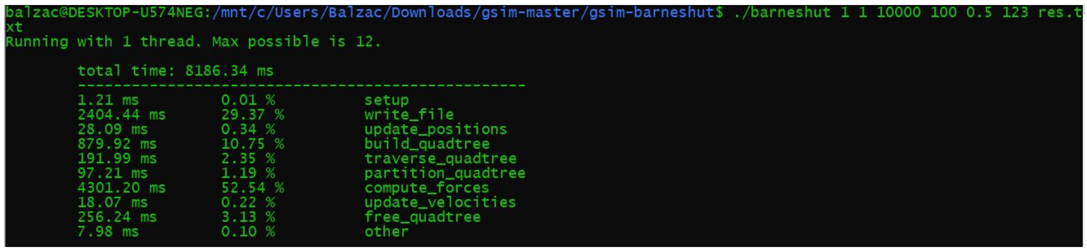
Following images show the simulation outputs for various numbers of threads and values of theta.



```
balzac@DESKTOP-U574NEG:/mnt/c/Users/Balzac/Downloads/gsim-master/gsim-barneshut$ ./barneshut 1 1 10000 100 0.1 123 res.t
xt
Running with 1 thread. Max possible is 12.

        total time: 50065.62 ms
        --------------------------------------------------
        1.59 ms         0.00 %          setup
        2469.93 ms      4.93 %          write_file
        30.27 ms        0.06 %          update_positions
        862.00 ms       1.72 %          build_quadtree
        192.95 ms       0.39 %          traverse_quadtree
        83.82 ms        0.17 %          partition_quadtree
        46111.73 ms     92.10 %         compute_forces
        25.45 ms        0.05 %          update_velocities
        286.84 ms       0.57 %          free_quadtree
        1.03 ms         0.00 %          other
```

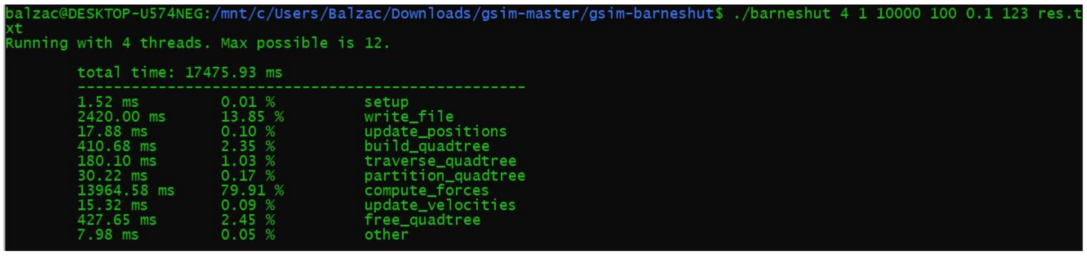
Fig 5. Number of Threads - 1, Theta = 0.1



```
balzac@DESKTOP-U574NEG:/mnt/c/Users/Balzac/Downloads/gsim-master/gsim-barneshut$ ./barneshut 1 1 10000 100 0.5 123 res.t
xt
Running with 1 thread. Max possible is 12.

        total time: 8186.34 ms
        --------------------------------------------------
        1.21 ms         0.01 %          setup
        2404.44 ms      29.37 %         write_file
        28.09 ms        0.34 %          update_positions
        879.92 ms       10.75 %         build_quadtree
        191.99 ms       2.35 %          traverse_quadtree
        97.21 ms        1.19 %          partition_quadtree
        4301.20 ms      52.54 %         compute_forces
        18.07 ms        0.22 %          update_velocities
        256.24 ms       3.13 %          free_quadtree
        7.98 ms         0.10 %          other
```

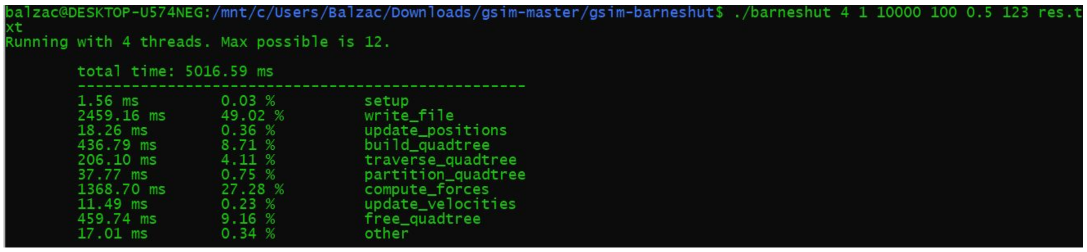
Fig 6. Number of Threads - 1, Theta = 0.5



```
balzac@DESKTOP-U574NEG:/mnt/c/Users/Balzac/Downloads/gsim-master/gsim-barneshut$ ./barneshut 4 1 10000 100 0.1 123 res.t
xt
Running with 4 threads. Max possible is 12.

        total time: 17475.93 ms
        --------------------------------------------------
        1.52 ms         0.01 %          setup
        2420.00 ms      13.85 %         write_file
        17.88 ms        0.10 %          update_positions
        410.68 ms       2.35 %          build_quadtree
        180.10 ms       1.03 %          traverse_quadtree
        30.22 ms        0.17 %          partition_quadtree
        13964.58 ms     79.91 %         compute_forces
        15.32 ms        0.09 %          update_velocities
        427.65 ms       2.45 %          free_quadtree
        7.98 ms         0.05 %          other
```

Fig 7. Number of Threads - 4, Theta = 0.1



```
balzac@DESKTOP-U574NEG:/mnt/c/Users/Balzac/Downloads/gsim-master/gsim-barneshut$ ./barneshut 4 1 10000 100 0.5 123 res.t
xt
Running with 4 threads. Max possible is 12.

        total time: 5016.59 ms
        --------------------------------------------------
        1.56 ms         0.03 %          setup
        2459.16 ms      49.02 %         write_file
        18.26 ms        0.36 %          update_positions
        436.79 ms       8.71 %          build_quadtree
        206.10 ms       4.11 %          traverse_quadtree
        37.77 ms        0.75 %          partition_quadtree
        1368.70 ms      27.28 %         compute_forces
        11.49 ms        0.23 %          update_velocities
        459.74 ms       9.16 %          free_quadtree
        17.01 ms        0.34 %          other
```

Fig 8. Number of Threads - 4, Theta = 0.5

4.Results

After implementing the parallelized version of Barnes-Hut algorithm for different values of *theta*, it was observed that when the number of threads used was more than 8, the time performance of the system became worse. This might be because of the increasing amount of communication overhead as the number of threads increased. Also, it is known that in the N-body simulation problem the node allocation in quadtree is very dynamic.
The below plots show the performance results of the parallel code for different values of *theta*.

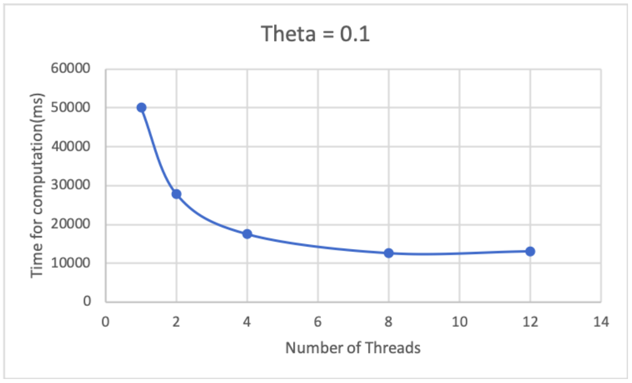


Fig 9. Parallel BH for theta = 0.1

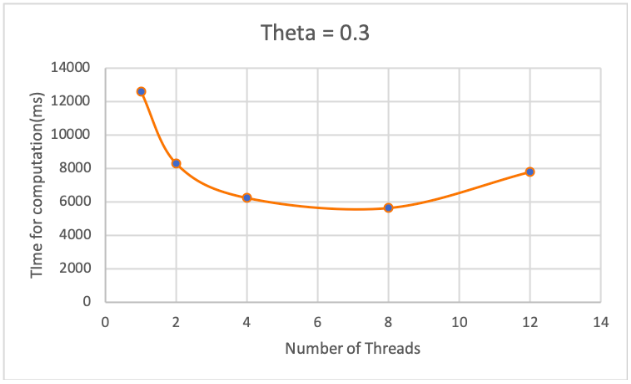


Fig 10. Parallel BH for theta = 0.3

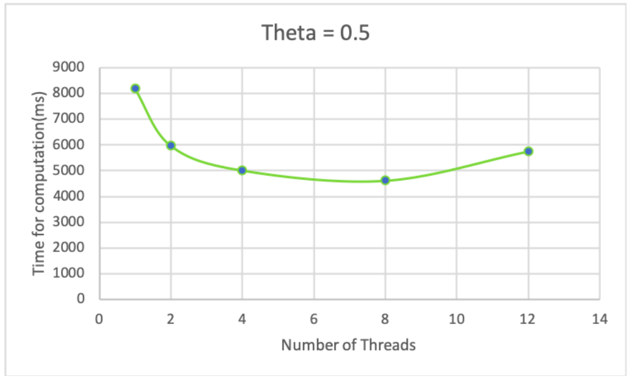


Fig 11. Parallel BH for theta = 0.5

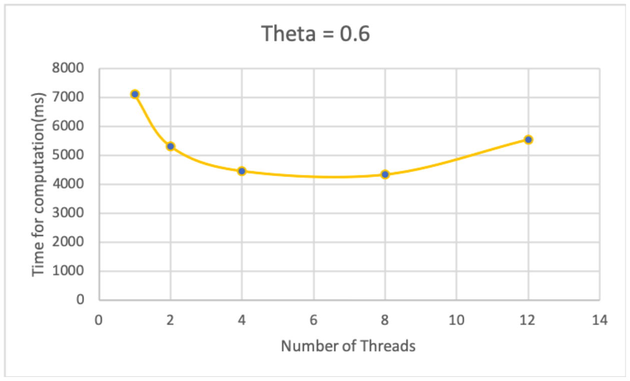


Fig 12. Parallel BH for theta = 0.6

The following observations can be made from the above plots:

1. As we can see from the above plots, the computation time increases as the *theta* decreases and time for *theta* = 0.1 is almost 7x as compared to *theta* = 0.6.
2. Secondly, for all *thetas* except 0.1 (here computation time for 12 threads is almost same as that for 8 threads), the time performance worsens when more than 8 threads are used.
3. Lastly, it can be observed that for *theta* = 0.3, the most optimal parallel Barnes-Hut algorithm can be implemented.

5.Challenges faced during the project

The algorithm presented a few challenges that needed to be overcome for successful parallelization-

1. The cost of building and traversing a quad-tree can increase significantly when divided among processes.
2. Traversing a quad-tree can increase significantly when divided among processes.
3. The irregularly structured and adaptive nature of the algorithm makes the data access patterns dynamic and irregular, and the nodes essential to a body cannot be computed without traversing the quad-tree.
4. Building a quad-tree needs synchronization.
5. The value of the fixed accuracy parameter (θ) plays a prominent role and must be optimized. Higher values of θ imply that fewer nodes are considered in the force computation thus increasing the window for error; while lower values of θ will bring the Barnes-Hut approximation time complexity closer to that of the AllPairs algorithm.

6.Conclusion

We have implemented an N-body simulator using the Barnes-Hut algorithm to simulate gravitational interactions between a number of bodies. We showed performance gains possible by parallelizing the code through multithreading using the OpenMP API on modern multi-core CPUs. The project also explored the practicality of parallelizing code. We showed performance is only gained if the code is parallelized efficiently and that the gains are not scalable beyond a number of threads, and in fact start decreasing beyond a certain number of threads.

7.References

- http://physics.princeton.edu/~fpretori/Nbody/intro.htm
- http://www.cs.cmu.edu/~guyb/real-world/nbody/index.html
- http://arborjs.org/docs/barnes-hut
- https://beltoforion.de/article.php?a=barnes-hut-galaxy-simulator
- https://jheer.github.io/barnes-hut/