Srinivas Prasad Prabhu

UT EID - sp55629

CS380P – Lab 5 – MPI Barnes-Hut

Platform Details -

CPU - AMD EPYC 7413 8 core processor
RAM - 128 GB RAM
OS - Ubuntu 22.04.2 LTS

MPICH Version: 4.1.2

MPICH Release date: Wed Jun 7 15:22:45 CDT 2023

MPICH ABI: 15:1:3

MPICH Device: ch4:ofi

MPICH configure: --disable-fortran

MPICH CC: gcc -O2

MPICH CXX: g++ -O2

MPICH F77:

MPICH FC:

The message passing interface (MPI) is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.

In parallel computing, multiple computers – or even multiple processor cores within the same computer – are called nodes. Each node in the parallel arrangement typically works on a portion of the overall computing problem. The challenge then is to synchronize the actions of each parallel node, exchange data between nodes, and provide command and control over the entire parallel cluster. The message passing interface defines a standard suite of functions for these tasks. The term *message passing* itself typically refers to the sending of a message to an object, parallel process, subroutine, function or thread, which is then used to start another process.

Benefits of the message passing interface

The message passing interface provides the following benefits:

- **Standardization.** MPI has replaced other message passing libraries, becoming a generally accepted industry standard.
- **Developed by a broad committee.** Although MPI may not be an official standard, it's still a general standard created by a committee of vendors, implementors and users.
- **Portability.** MPI has been implemented for many distributed memory architectures, meaning users don't need to modify source code when porting applications over to different platforms that are supported by the MPI standard.
- **Speed.** Implementation is typically optimized for the hardware the MPI runs on. Vendor implementations may also be optimized for native hardware features.
- **Functionality.** MPI is designed for high performance on massively parallel machines and clusters. The basic MPI-1 implementation has more than 100 defined routines.

For this project I am using MPICH implementation of MPI.

Barnes-Hutt Algorithm

The Barnes–Hut simulation (named after Josh Barnes and Piet Hut) is an approximation algorithm for performing n-body simulation.

Typically, n-body simulation takes $O(n^2)$ by brute force method. In this method the force by every body on every other body in the system is calculated leading to n^2 order of growth.

Barnes-Hut provided a variant of this - approximation of this force simulation problem which grows in the order of O(nlogn).

The BH method consists of 3 steps:

1) Construct the BH tree – Using Orthogonal Recursive Bisection
2) Compute the forces on each body.
3) Update the location and velocity of the body using leapfrog vertlet integration.

Construct the BH Tree –

Here a quad tree structure is used to store the points. Quad-Trees serve as a powerful tool in spatial data indexing. Quadtrees are the two-dimensional trees and are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions.

Benefits of Quad-Tree

- **Speed and Efficiency:** Quad-Trees facilitate fast region queries and updates as they reduce the search space considerably with each level.
- **Adaptive Resolution:** Quad-Trees offer an adaptive resolution as different regions of space can be represented at different levels of detail. For example, a region with a high concentration of data points can be subdivided more than a region with few data points, offering a higher resolution where needed.
- **Intuitive and Easy to Implement:** Quad-Trees follow a simple rule for subdivision, and hence are relatively easier and more intuitive to implement than other spatial data structures.

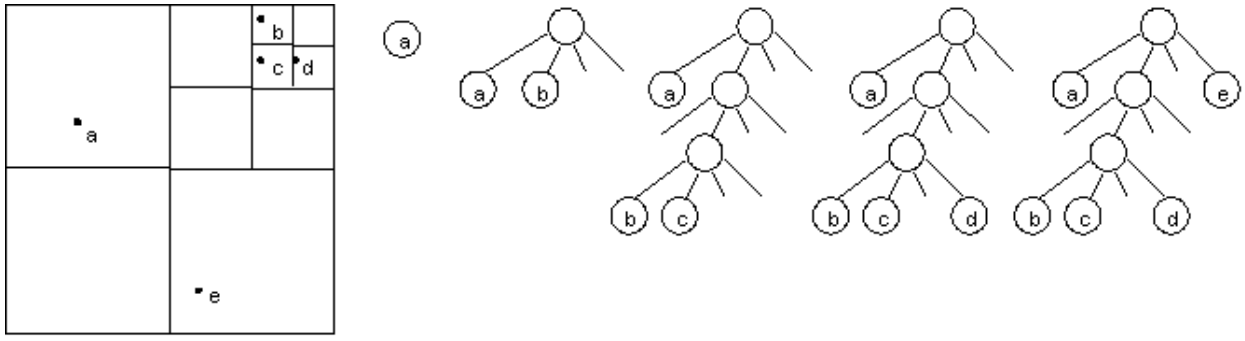
Limitations of Quad-Tree

- **Inefficient with Irregular Data:** Quad-Trees may not be efficient when data is distributed irregularly, leading to trees that are deeper on one side. This could potentially result in longer query times.
- **Difficulty Handling Dynamic Data:** Quad-Trees are not particularly well-suited for handling data that changes frequently (dynamic data). Adding and removing points can cause significant changes to the structure of the tree, potentially leading to a high computational cost.
- **Not Optimal for High-Dimensional Data:** Quad-Trees are ideal for two-dimensional data. However, as the number of dimensions increases, the number of children for each node increases exponentially, making the tree highly inefficient.

To construct the Barnes-Hut tree, insert the bodies one after another into a quad tree. To insert a body b into the tree rooted at node x, the following recursive procedure is used:

- If node x does not contain a body, put the new body b here.
- If node x is an internal node, update the center-of-mass and total mass of x. Recursively insert the body b in the appropriate quadrant.
- If node x is an external node, say containing a body named c, then there are two bodies b and c in the same region. Subdivide the region further by creating four children. Then, recursively insert both b and c into the appropriate quadrant(s). Since b and c may still end up in the same quadrant, there may be several subdivisions during a single insertion. Finally, update the center-of-mass and total mass of x.

As an example, with 5 bodies in the diagram below. In these examples, it is convention to call the branches, from left to right, as northwest, northeast, southwest, and southeast quadrants, respectively. The tree gets constructed as shown below.



The root node contains the center-of-mass and total mass of all five bodies. The two other internal nodes each contain the center-of-mass and total mass of the bodies *b*, *c*, and *d*.

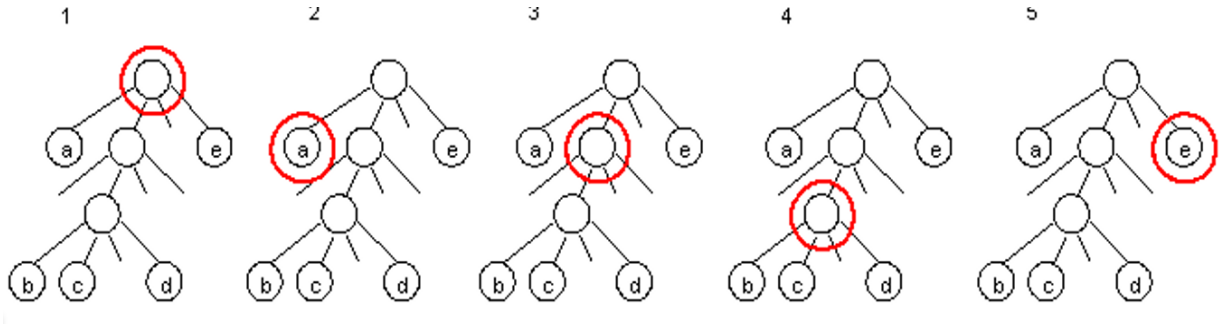
Compute the forces on each body

Calculating the force acting on a body. To calculate the net force acting on body *b*, the following recursive procedure, starting with the root of the quad-tree is used:

1. If the current node is an external node (and it is not body *b*), calculate the force exerted by the current node on *b*, and add this amount to *b*'s net force.
2. Otherwise, calculate the ratio s / d. If $s / d < \theta$, treat this internal node as a single body, and calculate the force it exerts on body *b*, and add this amount to *b*'s net force.

3. Otherwise, run the procedure recursively on each of the current node's children.

As an example, to calculate the net force acting on body *a*, we start at the root node, which is an internal node. It represents the center-of-mass of the five bodies *a*, *b*, *c*, *d*, and *e*, which have masses. The force calculation proceeds as follows:



Update the location and velocity of the body

The location and velocity of the bodies are updated using the leapfrog vertlet integration.

When you have the total force on a body you can compute the new position (Px', Py') and velocity (Vx', Vy') of the body given its current position (Px, Py) and velocity (Vx, Vy) as follows

ax = Fx/M0
ay = Fy/M0
Px' = Px + Vx*dt + 0.5*ax*dt^2
Py' = Py + Vy*dt + 0.5*ay*dt^2
Vx' = Vx + ax*dt
Vy' = Vy + ay*dt
where dt is the timestep.

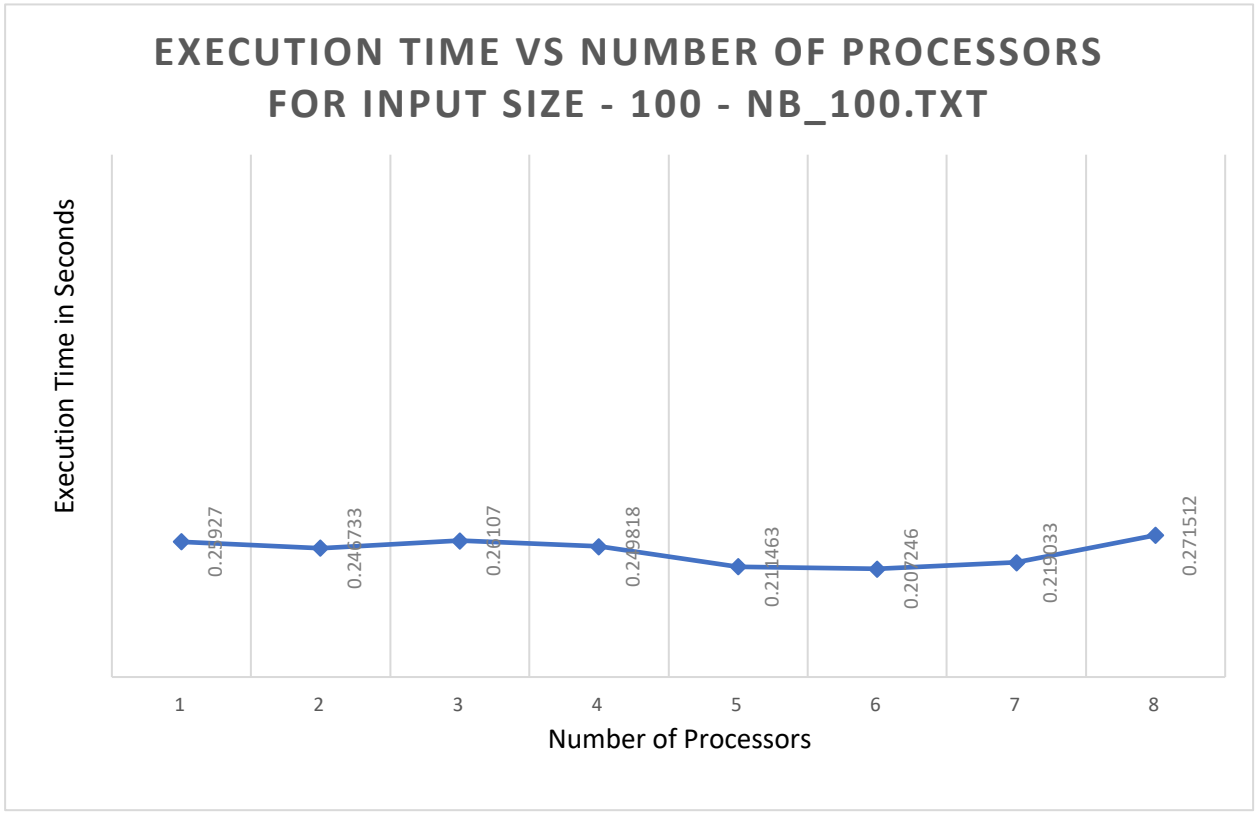
In the current implementation, we can see that we are able to achieve parallel scaling using MPI to speed up the BH algorithm. For different number of processors, we can see below that we obtain parallel scaling.

Based on Amdahl's law, we can see, that we don't get ideal speedup. There will always be some part which can't be parallelized, and which runs in a sequential manner.

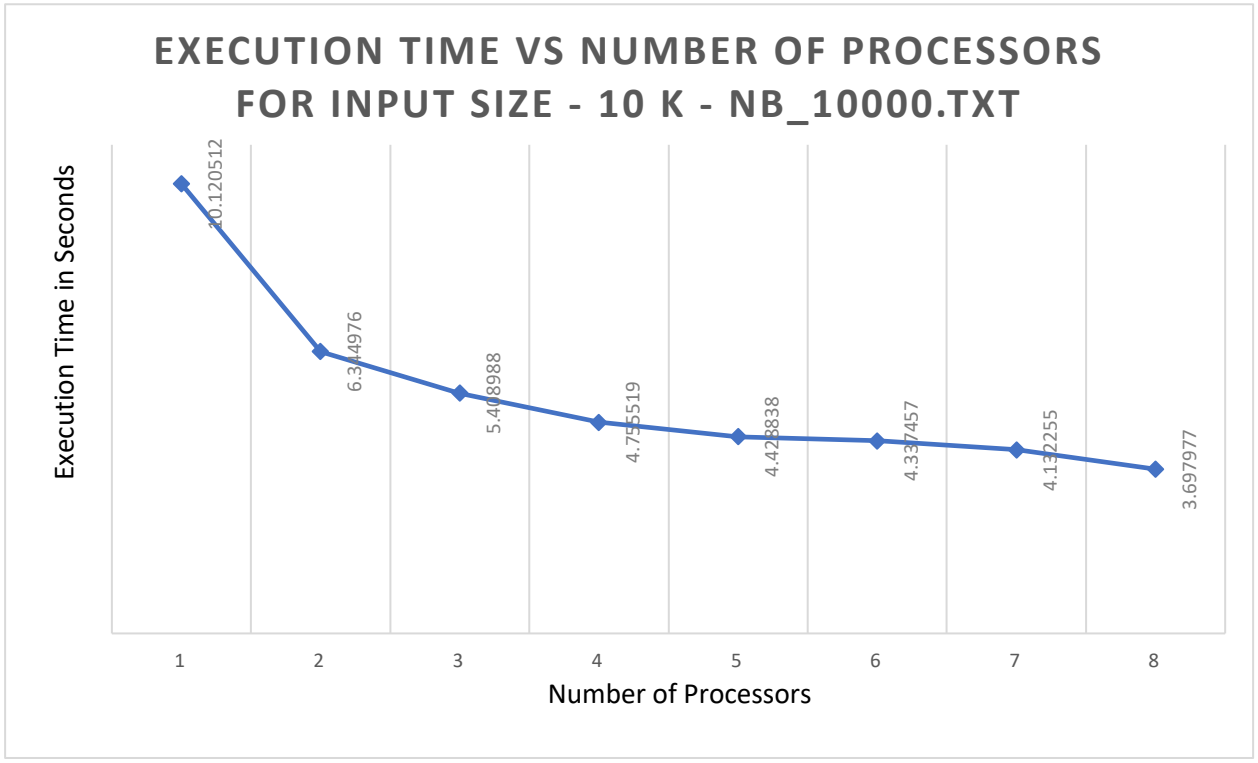
In the case of BH algorithm, even though we can parallelize the calculation of the forces, eventually all the bodies must be shared between all the processors for

successive iterations. This is blocking and prevents the realization of ideal speedup.

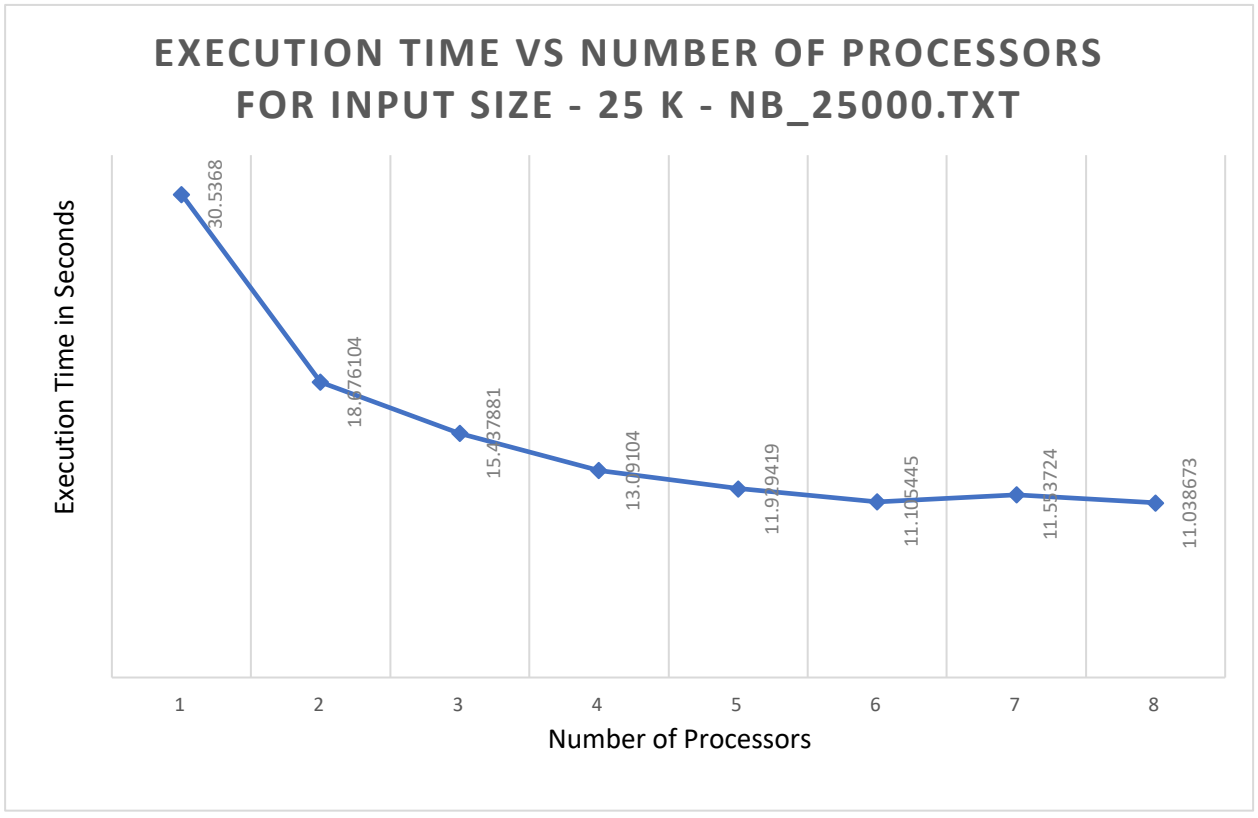
mpiexec -n <proc> ./nbody -i input/nb-100.txt -o output.txt -s 1000 -t 0.5 -d 0.005



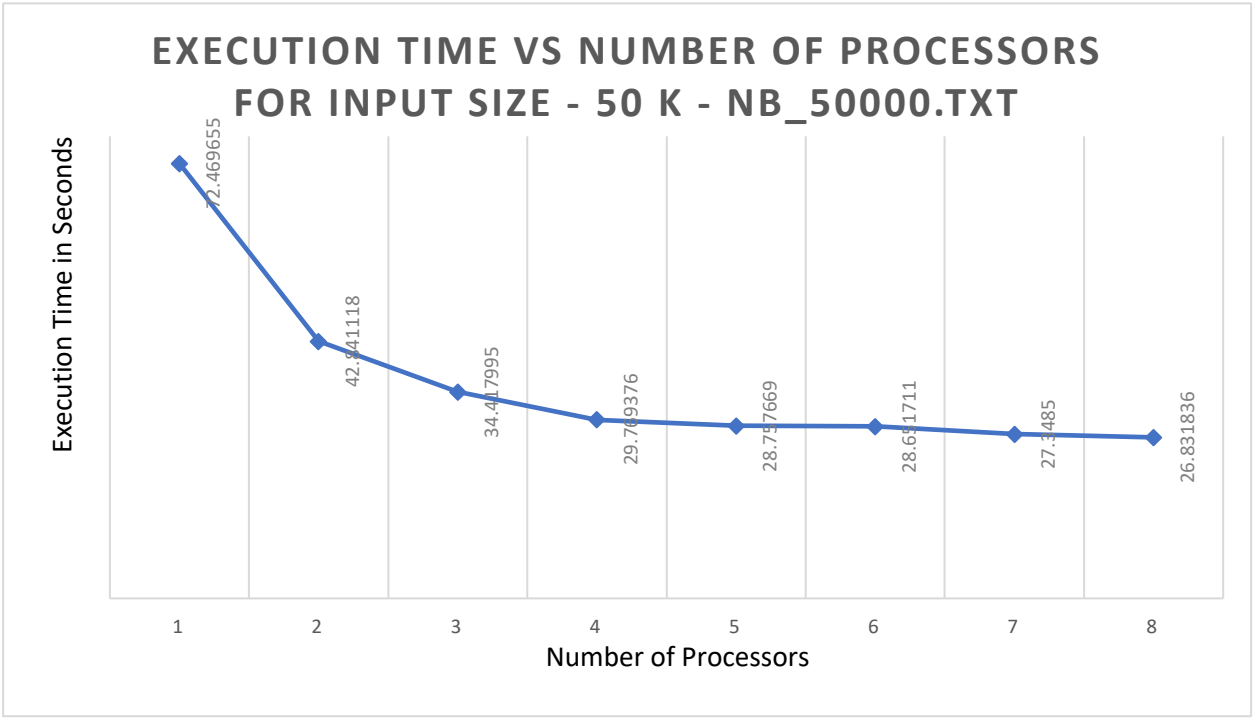
mpiexec -n <proc> ./nbody -i input/nb-10000.txt -o output.txt -s 100 -t 0.5 -d 0.005



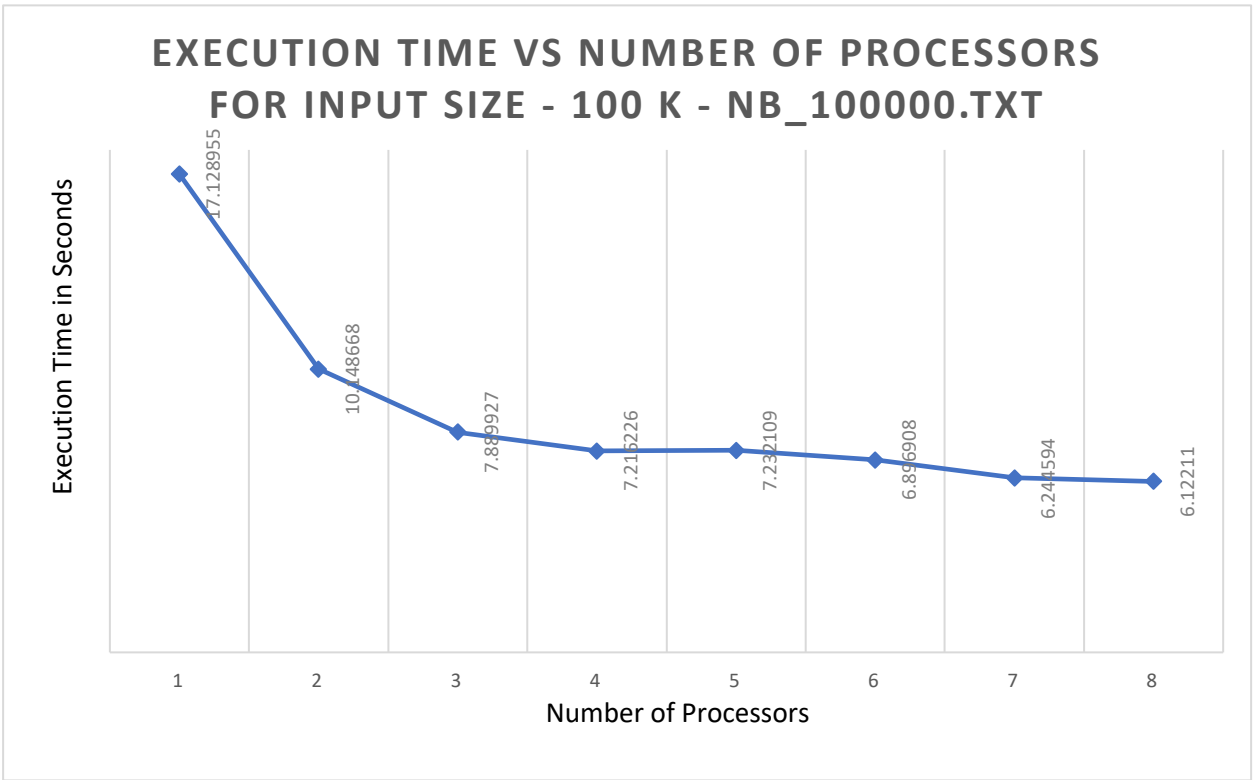
mpiexec -n <proc> ./nbody -i input/nb-25000.txt -o output.txt -s 100 -t 0.5 -d 0.005



mpiexec -n <proc> ./nbody -i input/nb-50000.txt -o output.txt -s 100 -t 0.5 -d 0.005



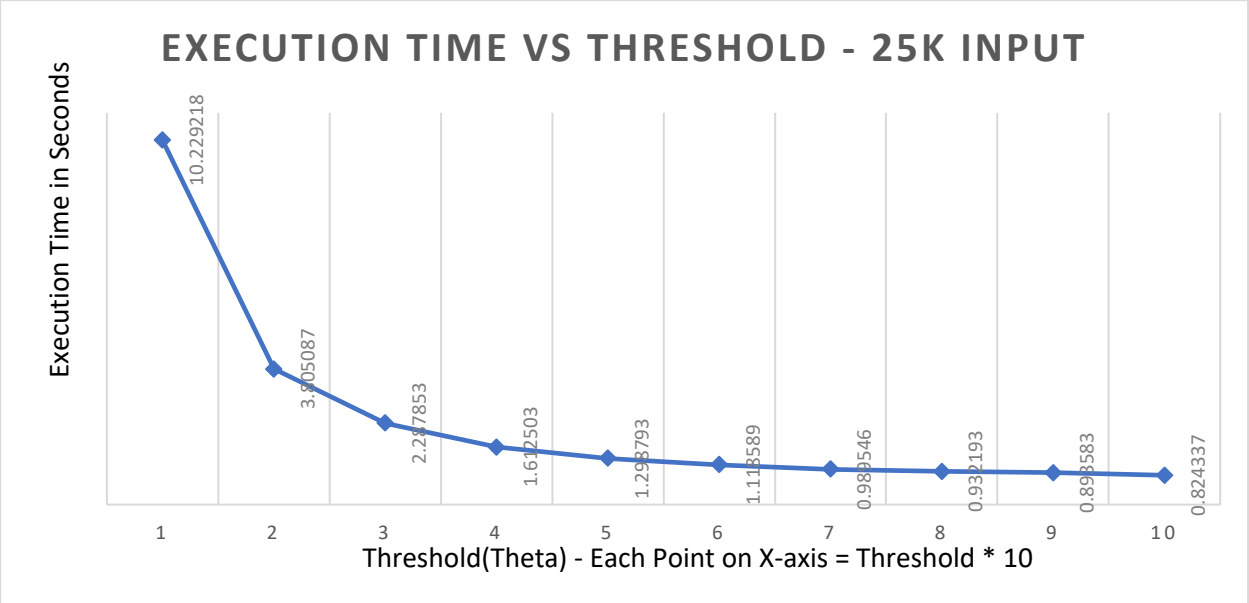
mpiexec -n <proc> ./nbody -i input/nb-100000.txt -o output.txt -s 10 -t 0.5 -d 0.005



Threshold Variation

Another important factor in BH algorithm is the threshold. As we vary the threshold, we can see that the time changes for a fixed number of processors. This is because with a very high threshold, most of the tree need not be walked and the results will be highly approximated. With a very low threshold, we must walk through the entire tree for every force calculation leading to longer execution times. For this experiment, I took 25K input, 4 processors and varied the threshold between 0.1 to 1.

mpiexec -n 4 ./nbody -i input/nb-25000.txt -o output.txt -s 10 -t <0.1-1> -d 0.005



References –

1) https://www.cs.utexas.edu/~rossbach/cs380p/lab/bh-submission-cs380p.html
2) http://arborjs.org/docs/barnes-hut
3) https://www.cs.princeton.edu/courses/archive/fall03/cs126/assignments/barnes-hut.html
4) https://opensourcegisdata.com/quad-tree-geospatial-data-structure-functionality-benefits-and-limitations.html
5) [1] M. S. Warren and J. K. Salmon. "Astrophysical N-body simulations using hierarchical tree data structures," In Proceedings of the 1992 ACM/IEEE conference on Supercomputing (Supercomputing '92), Robert Werner (Ed.). IEEE Computer Society Press, Los Alamitos, CA, USA, 570-576.
6) [2] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy, "Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multipole, and Radiosity," Journal of Parallel and Distributed Computing, Volume 27, Issue 2, 1995, Pages 118-141.
7) [3] M. S. Warren and J. K. Salmon, "A parallel hashed Oct-Tree N-body algorithm," In Proceedings of the 1993 ACM/IEEE conference on Supercomputing (Supercomputing '93). ACM, New York, NY, USA, 12-21.
8) [4] Ananth Y. Grama, Vipin Kumar, and Ahmed Sameh, "Scalable parallel formulations of the barnes-hut method for N-body simulations," In Proceedings of the 1994 ACM/IEEE conference on Supercomputing (Supercomputing '94). IEEE Computer Society Press, Los Alamitos, CA, USA, 439-448.