Project Summary:
We are going to implement the N body simulation, where we simulate the movement of N bodies/particles in a 2 dimensional space under the influence of gravitational force using Pthreads, OpenMP and MPI. We will compare the Speedup achieved using these 3 approaches.

Project Background:
The N-body simulation is a classic computational physics problem in which the objective is to predict the individual motions of a group of celestial objects interacting with each other gravitationally. Each body exerts a force on all others, influencing their acceleration, velocity, and ultimately, their position over time. This problem scales quadratically with the number of bodies (N^2), as every body interacts with every other body, making it computationally intensive, especially for large values of N.

The simulation involves calculating the gravitational forces between all pairs of bodies, updating their velocities and positions, and iterating this process over discrete time steps. The computationally demanding nature of this task makes it an ideal candidate for parallelization.

Parallelism can be exploited in several ways: by distributing the computation of forces among processors, by parallelizing the update of body attributes, or by doing both concurrently. The goal of parallelizing the N-body simulation is to reduce the overall computation time by dividing the work among multiple processing elements. In shared-memory systems, threads can be created via Pthreads or OpenMP to perform calculations concurrently, with OpenMP providing a higher-level, often simpler, abstraction for parallelism. Message Passing Interface (MPI) is used for distributed-memory systems where data is passed between processes running on different nodes of a cluster. Hybrid approaches can combine MPI with OpenMP to leverage both distributed and shared memory models, and for highly parallel architectures like GPUs, CUDA can be utilized to perform massive amounts of calculations in parallel.

The N-body simulation's inherent lack of data dependency during the force calculation phase makes it particularly amenable to parallel processing. Each body's new state can be computed independently based on a common, unchanging snapshot of the system. This characteristic allows for the distribution of the simulation across multiple computational units without the need for synchronization during the calculation phase, thus providing a significant opportunity for performance gains through parallelism. The speedup and efficiency of parallel implementations can be measured by comparing the execution time to that of a sequential algorithm, particularly as the number of bodies and the computational resources vary.

Work done so far:

Dataset generation:
For the N-body simulation, dataset is a important part of it. We used a python script to generate particles using certain constraints:

Particles: 10k, 100k, 1mil, 10million
Size: ranging from 1kg to 10kg
Velocity: 1-10kmph along the x and y axis.
We are planning to use the same dataset across all the runs to measure accurate changes in our application code.

Serial Implementation:
In the initial approach to the N-body simulation described, each particle within a designated bounding box is subjected to force calculations emanating from every other particle contained within the same spatial limits. This method, fundamentally rooted in Newton's law of universal gravitation, requires the computation of gravitational forces as an inverse square function of the distance between any two particles. For every cycle of the simulation—each representing a discrete time step—the forces are calculated afresh based on the most current particle positions. This iterative recalibration is essential to accurately model the dynamic, continuously interacting system. The force exerted on each particle is then used to update its velocity and position, thereby simulating its motion over time.

However, this straightforward approach carries a significant computational drawback, particularly evident as the number of particles increases. The required calculations grow quadratically with the number of particles, described mathematically as O(n^2). This quadratic increase means that each addition of a particle nearly doubles the computational workload, leading to a rapid escalation in total processing time. This exponential rise in computational demand makes the method inefficient for large systems, such as those typically encountered in astrophysical simulations or complex molecular dynamics. As a result, while the method's simplicity and directness provide clear, step-by-step computational procedures, its practical application is severely limited by its scalability, prompting the need for more sophisticated algorithms like the Barnes-Hut, which significantly reduce computational complexity.

Improved Serial Implementation:
The naive implementation had a runtime of O(n^2). Improvements could be made by relying on approximation, and this could be done by usage of Barnes-Hut algorithm. The key idea behind the algorithm is to reduce the number of calculations needed to compute the gravitational forces between all pairs of bodies, which naively would be proportional to N^2. Instead, the Barnes-Hut algorithm groups nearby bodies together and approximates them as a single entity when calculating gravitational forces on a body that is sufficiently far away. This is achieved by organizing all the bodies into a hierarchical tree structure, typically a quadtree (in 2D) or an octree (in 3D).

Each cell (or node) in the tree represents a specific region of space and possibly contains several bodies. Each non-leaf node in the tree represents the combined center of mass and the total mass of all the bodies within its region. When calculating the gravitational force exerted on a particular body, the algorithm traverses this tree. For each node, it checks if the node is sufficiently far from the body in question; if so, the group of bodies in that node is treated as a

single body with their combined mass located at their center of mass. If not, the algorithm recursively examines finer subdivisions of the node.

This approach significantly reduces the number of interactions that need to be considered, lowering the computational complexity. The complexity of building the tree is O(N log N), and so is the complexity of computing the forces, leading to a total complexity of O(N log N) for each timestep of the simulation. This makes the Barnes-Hut algorithm substantially faster than a direct O(N^2) pairwise force calculation, especially for large N, allowing simulations of large systems to be computed more efficiently and realistically.

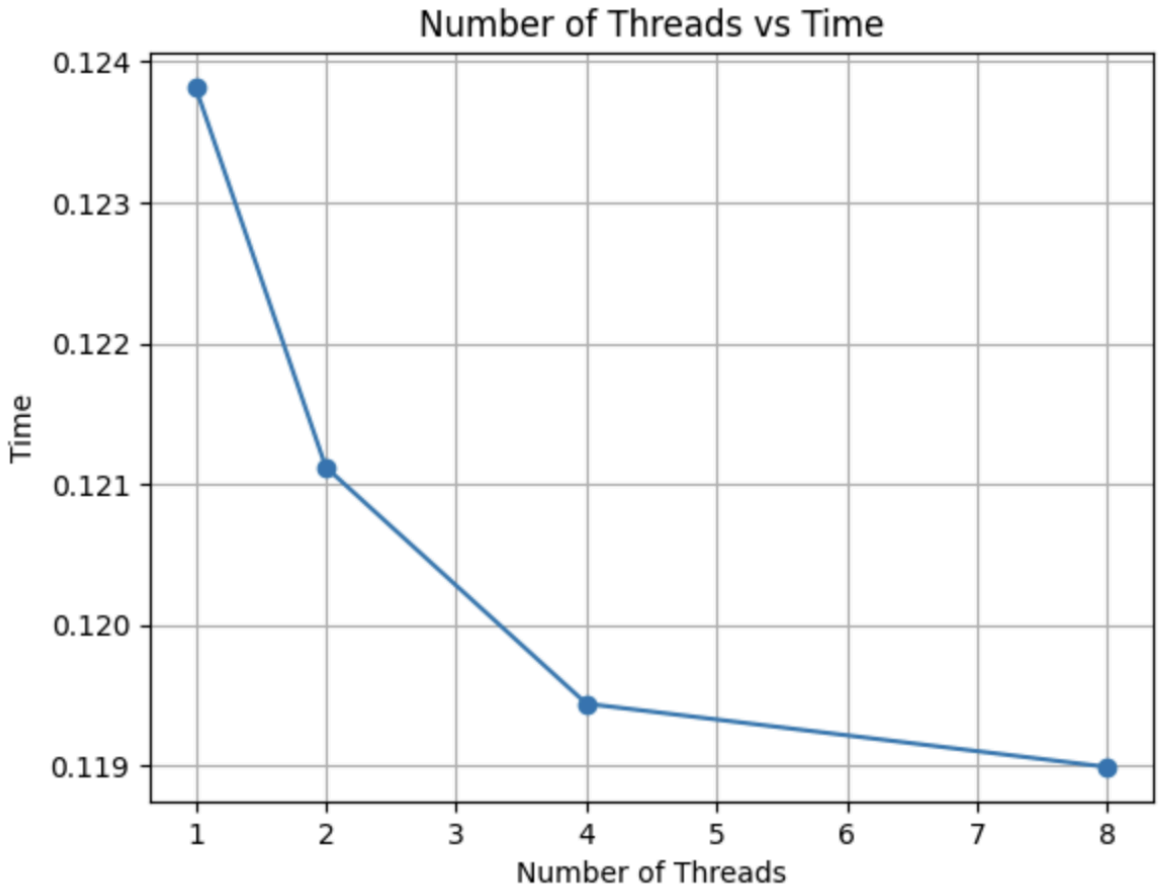
OpenMP Implementation:
In the simulation phase, there are two steps happening in a given cycle: **quadTree creation**, where a quadtree is generated for all the particles in the bounding box, which is needed for approximation. And the **force calculation**, where a force is calculated on a given object based on an approximation constant(theta).

From the 2 steps, building a quadtree can be inherently sequential because each decision about where to place a particle in the tree structure depends on the current state of the tree. Furthermore, the dynamic nature of particle movement means the tree must be reconstructed from scratch in each cycle, maintaining the integrity of the data structure but also adding to the computational overhead. Due to these complexities, the quadtree creation is typically performed as a serial operation within the simulation, as parallelizing this step involves significant challenges that could complicate the implementation without a clear guarantee of performance improvement.

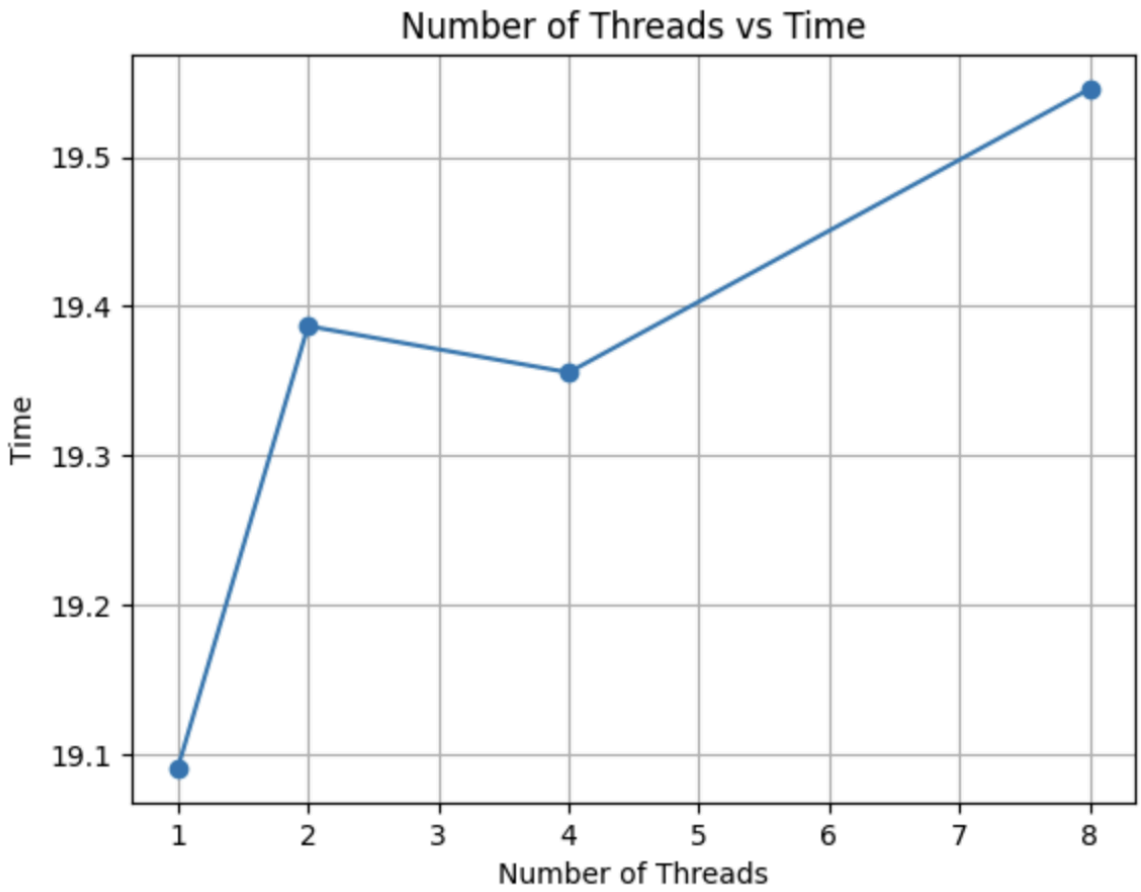
As for the force calculation, this deals with read-only items of the tree, and hence this step could be parallelised. By using OpenMP, the force calculations are distributed among multiple threads. This parallelization is managed using a dynamic scheduler, which dynamically assigns chunks of the workload to different threads as they become available. This approach helps in maximizing the utilization of computational resources, reducing the time required to compute the forces across all particles.

Results:
We ran the n-body simulations on a dataset with:
10k particles and 1mil particles.
Masses with range from 1kg to 10kg
Velocity of 1kmph - 10kmph on the horizontal and vertical axis

The results on 10k particles:



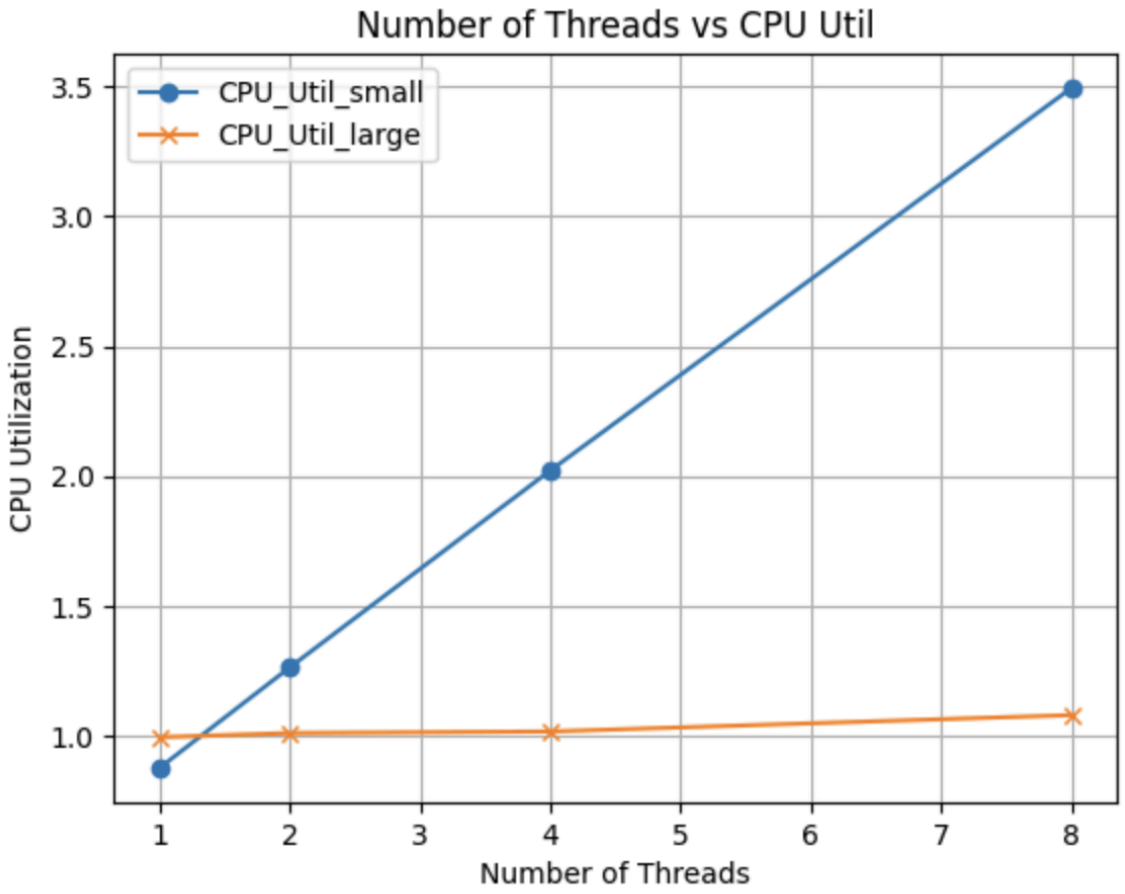
The results on 1million particles:



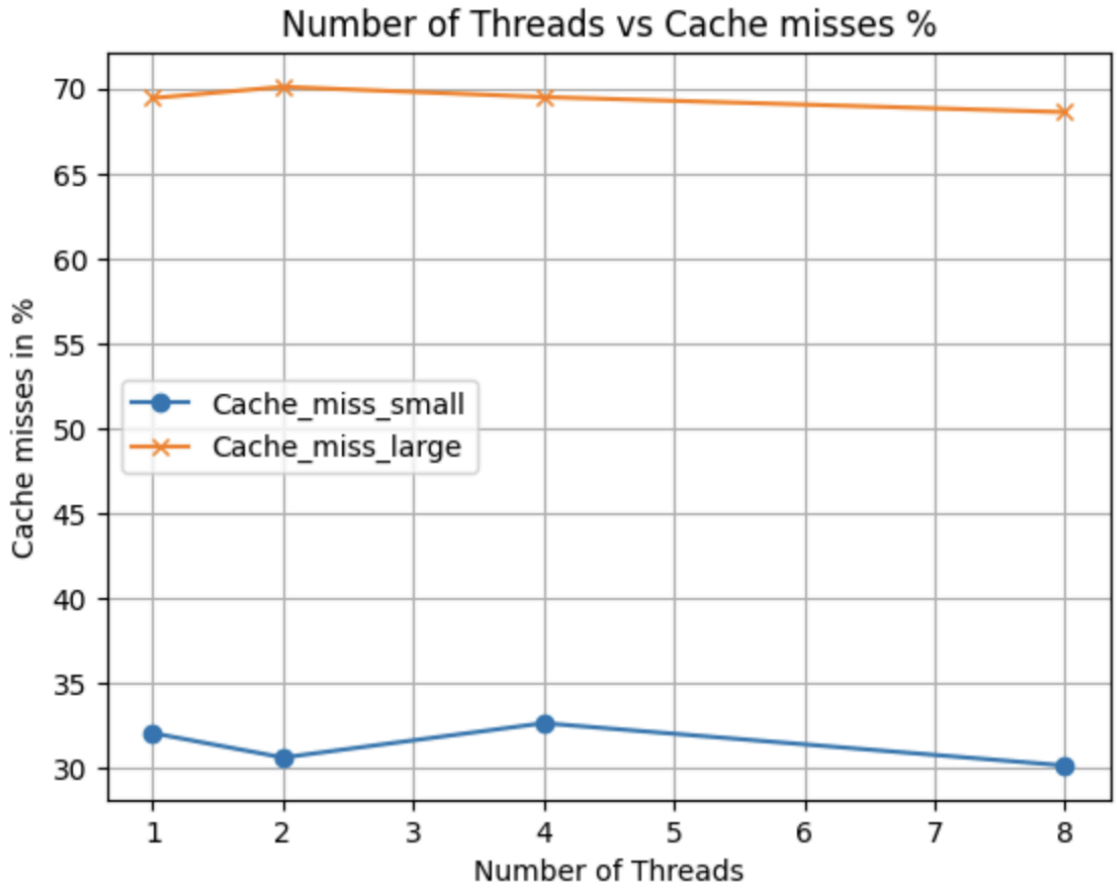
With 10k particles, we noticed some speedup. However, with 1million particles, we noticed there was a slowdown.

Issues:

The CPU Utlization remains small with the large dataset:



The cache misses are high in the large dataset as well:



These issues might be caused due to false sharing, more data being having to moved in and out through the bus etc.

These issues have to be addressed moving forward.

Timeline:
03/27 - 03/30: Exploring the references and check various approaches [DONE]
04/01 - 04/07: Implement the sequential version of the algorithm [DONE]
04/08 - 04/15 (includes Milestone report): Improve on the sequential version and work on the OpenMP version [DONE]

Upcoming Timeline:
04/16 - 04/23:
Make fixes to the OpenMP version(Ashwin)
Implement the MPI version of the parallel algorithm (Akhil)

04/24 - 04/31:
Implement the pthreads version of the algorithm (Akhil)
Perform optimizations on the openMP and MPI versions (Ashwin)

05/01 - 05/05: Final report with detailed analysis of speedup with different problem sizes.

Overall Sentiment:
Although a bit of hiccup on the performance of OpenMP version, we are overall still on track and will reach all the objectives set.