Author: Zixuan Cheng
Student ID: 1165964

The problem at hand involves a multi-faceted computational challenge. We are tasked with simulating the gravitational behavior of objects in a D-dimensional space, coupled with clustering data obtained from a Gaussian mixture model (GMM). To tackle this problem, a highly parallelized approach is essential to efficiently process the large volume of data and complex calculations involved.

The process begins by generating data points from a GMM on multiple computational nodes. The initial data distribution is essential for the subsequent clustering and gravitational simulation. Each node generates its data points using its rank as a random seed, ensuring reproducibility while parallelizing the data generation.

Once the data points are generated on different nodes, they need to be centralized for clustering. The MPI (Message Passing Interface) communication paradigm is employed to gather all the data points from individual nodes into a global dataset. The usage of MPI_AllGather ensures efficient and scalable data aggregation across distributed memory systems.
This process is executed as in Algorithm 1:

Algorithm 1 Generate Points

```
for node in allnodes do
    for n = 1, 2, ..., N do
        Generate body according to the input c-Component GMM Model
        Store point in local body list
    end for
end for
Gather bodies from each node using MPI_ALLGather and store in global
bodies list
```

After gathering the data, the clustering phase is executed in parallel. The K-means++ algorithm is employed to cluster the points. This step significantly benefits from parallelization, as the calculation of cluster centroids and assignment of points to clusters can be independently distributed to different computational nodes.
The clustering and scattering process is executed as in Algorithm 2:

Algorithm 2 K-Means++

```
centroids ← []
Add a random body from the global bodies list into centroids
for k = 1, 2, ..., K do
    for node in all_nodes do
        for body in local_bodies do
            Compute min distance to any item in centroids and record as d(body)
        end for
    end for
    Root node choose one body at random with weight proportional to d(body)^2 and insert to centroids
end for
for body in local_bodies do
    Assign node to centroid in centroids
end for
Scatter bodies according to node according to centroid in belongs to
```

Upon clustering, the next challenge is the gravitational simulation of these data points. This involves modeling each point as a physical body, simulating the interaction of these bodies under gravity. According to the spec, the stimulation should terminate when the variance of point locations is reduced by a factor of 4, but after running several stimulation on different numbers of points, it is found out that the condition can not be achieved in most conditions. So the termination condition is changed to when variance of point locations increases by 2 times To optimize the gravitational calculations, the Barnes-Hut Algorithm is utilized. In this algorithm, a tree structure is constructed using the local data points on each node. This tree efficiently approximates the gravitational forces between any point and the cluster of points it contains, reducing the computational load[3].
The tree structure is constructed as in Algorithm 3:

Algorithm 3 construct tree

```
D ← dimension
for body in local_bodies do
    cell ← root
    while cell has children do
        cell ← child_body_belongs_to
    end while
    if cell has a body then
        old_body ← original_body_in_cell
        generate 2^D children
        old_child ← child_old_body_belongs_to
        store old_body in old_child
    end if
    child ← child_body_belongs_to
    store body in child
end for
```

To compute the forces applied by the Barnes-Hut tree on each cluster, the local computations on each node are distributed and aggregated. The forces on each cluster are summed to determine the net gravitational force acting on them.
The total force acting on each body is calculated as in Algorithm 4:

Algorithm 4 Compute_Force(*body*, *cell*)

```
body: The body need to compute force
cell: The tree structure representing a cluster of nodes
for node in all_nodes do
    if first time called then
        cell ← local_tree_root
    end if
    if cell has no children then
        compute force between body and cell
    else
        for child in children do
            force ← force + Compute_Force(body,cell)
        end for
    end if
end for
```

The movement of the data points under the influence of gravity is simulated using Euler's method. While the Euler method is simpler than more advanced numerical techniques, it can provide a reasonable approximation for the gravitational dynamics of data points.
The body location update by gravity stimulation is stimulated as in Algorithm 5:

Algorithm 5 Gravity Stimulation

```
for Every time step do
    for node in all_nodes do
        for body in global_bodies do
            body_force ← Compute_Force(body,cell)
        end for
        MPI_ALLReduce body_force from each node by sum
        for body in global_bodies do
            update body position using body_force
        end for
    end for
end for
```

In addition to MPI, OpenMP is employed to further enhance the parallelization of the project. OpenMP is used to compute forces in parallel, which is crucial for improving the speed and efficiency of the gravitational simulation. This parallelization allows for the distribution of computational workloads across multiple processor cores, making efficient use of multi-core architectures.

In summary, this project leverages parallel techniques facilitated by both MPI and OpenMP to efficiently handle data generation, clustering, and the Barnes-Hut Algorithm for gravitational simulation. These parallelized computations allow us to address a complex problem involving the interaction of data points under simulated gravitational forces while optimizing for both speed and accuracy. The implementation of this approach is expected to provide insights into the clustering behavior and gravitational dynamics of data points in D-dimensional space.

Methodology

In evaluating the performance of the computational approach designed for this complex problem, two primary measures are of paramount importance: speedup and accuracy. These measures provide essential insights into the efficiency and precision of the implemented techniques.

1. MPI Speedup Measurement:

To gauge the impact of increasing the number of computational nodes on overall execution speed, we will employ the MPI Speedup measurement. This metric is calculated by varying the number of nodes while keeping the computational cores per node constant. Speedup, denoted as S(N), is calculated using Amdahl's Law:

$$S(N) = T(1) / T(N)$$

Where:

S(N) is the speedup achieved with N nodes.
T(1) is the execution time with a single node.
T(N) is the execution time with N nodes.
This measurement allows us to evaluate the scalability of the MPI-based solution when distributed across multiple nodes.

2. Accuracy Measurement:

To evaluate the accuracy of the gravitational simulation algorithms and the effectiveness of different accuracy parameter settings within the Barnes-Hut Algorithm, we will perform a direct comparison of data point representations. This comparison will assess how well the Barnes-Hut Algorithm approximates gravitational interactions and whether the chosen accuracy parameter (θ) strikes an appropriate balance between computational efficiency and accuracy. The variance is calculated with:

$$\text{Variance} = \sum_i(\sum_d(x_{i_d} - x_{mean_d})^2)/N$$

Where:
x_{i_d} is the coordinate of point i at dimension d.
x_{mean_d} is the mean coordinate of all points at dimension d.

3. Time Measurements:

For a comprehensive assessment of performance, we will measure the time taken by various parts of the program:

Point Generation Time: Measure the time taken to generate data points.
K-Means++ Time: Measure the average time taken to perform K-Means++ clustering.
Gravity Simulation Time: Measure the average time spent on the gravitational simulation.
Variance Checking Time: Measure the average time taken to check variances between data point locations.

To utilize these measurements, we will employ a series of experimental methodologies, each tailored to specific measurements and observations. Our experimentation plan encompasses four crucial aspects to comprehensively assess the performance and accuracy of our implemented computational approach. These experiments aim to provide insights into the impact of varying computational resources, accuracy parameter settings, and the execution times of different phases of the simulation.

Experiment 1: Execution Time Observation (Scaling with Data Points):
This experiment observes the distribution of computational time across different phases of the simulation. Its goal is to identify potential performance bottlenecks and assess the time percentage consumed by each phase as the number of generated data points varies.

Experiment 2: Scalability Assessment (MPI Speedup):
This experiment focuses on evaluating the scalability of our solution by varying the number of computational nodes while keeping the number of computational cores per node constant. Its primary aim is to assess the MPI-based speedup, shedding light on how the solution scales with an increasing number of nodes.

Experiment 3: Accuracy vs. Speed Assessment (Barnes-Hut Algorithm):
We assess the accuracy and speed of the gravitational simulation and the impact of different accuracy parameter(θ) settings in this experiment. Accuracy will be evaluated by comparing data point locations using changes in variance, helping us understand the trade-offs between accuracy and computational efficiency. Execution speed will be observed by measuring the time spent on simulation.

These experiments will provide valuable insights into the solution's performance and accuracy characteristics, guiding optimization and fine-tuning efforts for efficiency and precision. Detailed experimental settings and results will be discussed in subsequent sections.

Experiments:

To ensure the fairness and accuracy of our performance evaluations, it is imperative to address the variations in point generation caused by altering the number of points (N) or the number of nodes used for generation. These variations stem from the inherent differences in both the quantity of points and the seed values for number generation.

Invariably, these disparities can lead to distinct starting variances and subsequent changes in variance for different experimental settings. These variations may skew the number of time steps executed before reaching a terminal condition, rendering observations inequitable and unreliable.

To mitigate this potential bias and facilitate a more meaningful assessment of execution speed across different settings, we have enforced a fixed number of time steps. This approach allows us to effectively isolate and analyze the performance impact of varying parameters without the interference of unpredictable initial conditions.

Experiment 1: Execution Time Observation

Objective:
Assess the time allocation for each phase of the program as the number of generated points varies. This experiment lays the foundation for a deeper understanding of the factors that impact subsequent measurements.

Inputs:
The experiment involves generating input data with N values set to 1000, 5000, 10000, 100000. Each data point is generated using a 4-component Gaussian Mixture Model (GMM) with identical mean and deviation. The dimension (D) is set to 4, accuracy parameter (θ) to 0.9, and the program will run on 16 nodes with 4 cores per node. The gravity simulation will be conducted for a fixed 1000 iterations to ensure consistent observations.

Results:
Table 1: Execution detail for different number of points

N points	Total Execution Time (seconds)	generation time	K-Means++ Time	Gravity Simulation Time	Variance Checking Time
1000	23.0	0.02	0.04	20.9	0.02
5000	116	0.02	0.04	114.1	0.10
10000	239	0.02	0.05	236.8	0.11
100000	3528	0.04	0.05	3521.8	1.4
1000000 (50 iteration)	528	0.06	0.1	517.9	0.35
1000000 (estimated)	10365	0.06	0.1	10358	7.0

The execution time of 1000000 points is estimated using the execution time of 50 iterations because real execution time is expected to be too long for the Slurm job to be queued. It is calculated as:
Estimation time = generation time + K-Means++ Time + Gravity Simulation Time / 50 * 1000 + Variance Checking Time / 50 * 1000
The detailed execution time does not add up to total execution time because there are some parts such as initialization and I/O not included.

Experiment 2: MPI Speedup Evaluation

Objective:
Evaluate the parallel speedup achieved by implementing MPI in the program, focusing on performance when running gravitation stimulation on different numbers of nodes as the number of points increases.

Inputs:
The program will be executed on 1, 2, 4, 8, and 16 nodes with N set to 100, 1000, 5000, 10000. Similar to Experiment 1, the input data is generated using a 4-component GMM with identical mean and deviation. The dimension (D) is set to 4, the accuracy parameter (θ) is 0.7, and each

node has 2 cores. The gravity simulation will be conducted for a fixed 1000 iterations to ensure consistent observations.

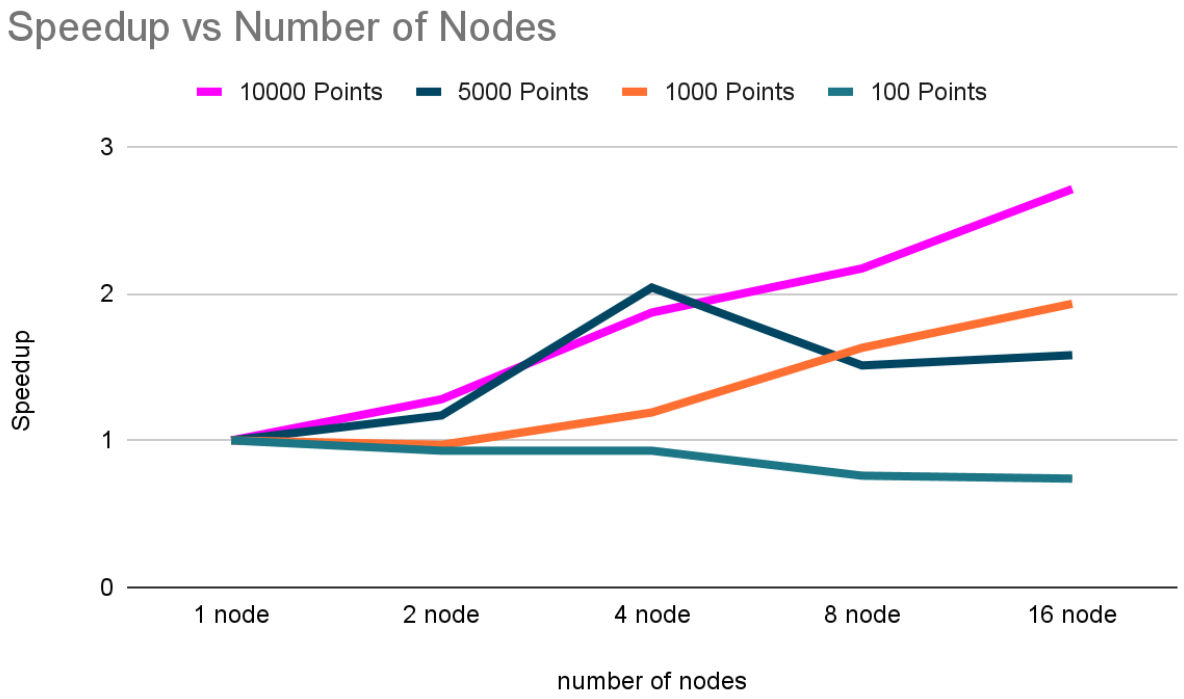
Results:

Table 2: Total Execution Time(seconds) on different numbers of nodes when stimulating different numbers of points.

Number of nodes	10000 points	5000 points	1000 points	100 points
1	801	259	31	2.8
2	628	222	32	3.0
4	428	127	26	3.0
8	369	171	19	3.7
16	296	164	16	3.8

Table 3: Largest cluster size for different numbers of points

Number of nodes	10000 points	5000 points	1000 points	100 points
1	10000	5000	1000	100
2	6514	3687	934	99
4	5103	3008	502	39
8	3247	2355	275	52
16	2542	1453	316	52



Experiment 3: Accuracy vs. Speed Evaluation with Respect to Accuracy Parameter θ in Barnes-Hut Algorithm

Objective:
Investigate the trade-off between accuracy and execution speed when using the Barnes-Hut Algorithm by adjusting the accuracy parameter (θ). The gravity simulation will be conducted for a fixed 1000 iterations to ensure consistent observations.

Inputs:
The program will be executed on 8 nodes, each with 2 cores. The input data is generated with N set to 5000, using a 4-component GMM with identical mean and deviation. The dimension (D) is 4. The accuracy parameter (θ) will be varied between 0, 0.3, 0.5, and 0.7. This experiment aims to provide insights into how changing θ affects both accuracy and computational efficiency.
Results:

Table 4: Change of variance over time steps with different θ settings

θ	0	0.3	0.5	0.7	0.9	1.1	1.3
Execution time(seconds)	1675	734	492	311	242	220	191
Speedup	1	2.28	3.40	5.38	6.92	7.61	8.76
Start variance	8.4	8.4	8.4	8.4	8.4	8.4	8.4
Variance after 100 seconds	7.3	7.3	7.3	7.3	7.3	7.2	7.2
Variance after 200 seconds	5.0	5.0	5.0	5.0	5.0	5.0	5.0
Variance after 300 seconds	3.3	3.3	3.3	3.3	3.3	3.3	3.3
Variance after 400 seconds	3.0	3.0	3.0	3.0	3.0	3.0	3.0
Variance after 500 seconds	3.6	3.6	3.6	3.6	3.6	3.6	3.5
Variance after 600 seconds	4.7	4.7	4.7	4.6	4.6	4.6	4.4
Variance after 700 seconds	6.2	6.2	6.3	6.1	6.1	6.1	5.8
Variance after 800 seconds	7.8	7.8	8.0	7.8	7.8	7.7	7.2
Variance after 900 seconds	9.3	9.3	9.6	9.4	9.4	9.2	8.5
Variance after 1000 seconds	10.5	10.5	10.9	10.6	10.6	10.4	9.4

Evaluation:

Experiment 1: Execution Time Observation

In Experiment 1, we aimed to understand how the execution time is distributed across different phases of the program as the number of generated points (N) varies. The results provide valuable insights into the program's performance, its time complexity, and the dominance of specific computational components.

The analysis showed that the time spent on point generation and K-means clustering becomes trivial as N increases. This is in line with expectations, as both processes exhibit linear time complexity, O(N/P), where P is the number of nodes(processors) available. Point generation is particularly straightforward, involving N*D calls to the Gaussian Mixture Model to generate coordinates, while the time complexity for K-means++ clustering is O(N/P*D*K) due to its iterative nature, but it tends to be linear when the cluster and dimension sizes remain constant.

However, the time measurements for point generation and K-means++ clustering indicate that their times do not significantly increase as N scales up. There is a potential explanation for this behavior: the time taken by these phases may be too short to be significantly influenced by the growth in N, and it could be predominantly occupied by overhead introduced by MPI and other operations, such as data structure initialization.

In contrast, the time spent in gravity simulation adheres to the expected time complexity of the Barnes-Hut Algorithm, which is O(N/PlogN/P)[2]. This time increases by about 14 times when N increases by 10 times, from 10,000 to 100,000. The increase is under the anticipated O(NlogN) growth, and it highlights the dominating role of gravity simulation in the overall execution time, especially when N is substantial. This behavior results in an overall program time complexity of O(NlogN) for large N or extensive iteration counts.

Experiment 2: MPI Speedup Evaluation

Experiment 2 primarily focused on assessing the speedup brought by using MPI in the program. The results revealed a nuanced relationship between the number of nodes and speedup, shedding light on the role of point distribution.

In general, the program exhibited an increase in speedup as the number of nodes (N) increased, aligning with the expectations of parallelization. However, the speedup didn't always linearly correspond to the number of nodes, and sometimes, it even decreased. This deviation from the expected trend can be attributed to the distribution of points across nodes.

When a cluster size on a node becomes significantly larger than others, the construction of the tree in Barnes-Hut Algorithm becomes more substantial. Consequently, it requires more time for force calculations, potentially slowing down the simulation. The variations in speedup observed in Experiment 2 are closely tied to the balance of point distribution across nodes. For example, when running stimulation of 5000 points on 8, one node contains over 50% of points, which is likely to be the cause of decrease in speedup. In cases where clusters on nodes grow disproportionately, the construction of the tree becomes larger and can result in non-linear relationships between node count and speedup. These insights underscore the critical role of balanced point distribution in optimizing the performance of the program when using MPI.

Experiment 3: Accuracy vs. Speed Evaluation with Respect to Accuracy Parameter θ in Barnes-Hut Algorithm

Experiment 3 explored the trade-off between accuracy and execution speed by varying the accuracy parameter (θ) in the Barnes-Hut Algorithm. The results offered a clear understanding of how different θ values impact both accuracy and speedup.

When θ is set to 0, the Barnes-Hut Algorithm essentially functions like an O(N^2) approach, as it calculates forces between each pair of points. The accuracy benchmark set by this approach demonstrates that accuracy is maintained when θ is less than 1.1. However, as θ exceeds 1.3, accuracy starts to degrade significantly.

The relationship between speedup and θ reveals an interesting pattern. Speedup increases as θ approaches 1.1, indicating a sweet spot where points that are sufficiently distant from each other can be accurately approximated. As θ continues to increase, accuracy may decline, particularly for points that are close to each other. Nonetheless, the computational steps do not decrease significantly, as most of the distant points are already neglected with smaller θ values. This implies that a balance can be struck between accuracy and computational efficiency by setting θ within the range of 0.7 and 1.1.

Challenges:

During the implementation and testing of the program, an unexpected behavior of point movements was identified. After a few iterations, the variance of point locations started to increase rapidly, and this increase occurred faster and with larger magnitude than anticipated. Research and testing revealed that the issue was related to the fixed time step used in the simulation. In some cases, two bodies would approach each other closely, resulting in a massive force due to the inverse square relation(F is proportional to 1/d^2). Consequently, their velocities increased significantly, causing them to travel long distances away from other bodies. In subsequent iterations, the force acting on these bodies became small, and they continued to travel away. This phenomenon negatively impacted the accuracy and stability of the simulation.

To address this issue, a smoothing parameter, EPSILON, was introduced to set a minimum distance that any two bodies must maintain. With this parameter, the maximum force between two bodies was restricted to a specific value. A constant value of 0.01 was chosen for EPSILON, roughly equivalent to 1/50 of the mean distance between generated points. This setting strikes a balance between accuracy and stability, helping to overcome the issue and maintain the program's accuracy[1].

Possible Improvements:

Several potential improvements were considered but not implemented during the course of this work. The first one involves enhancing the clustering strategy to balance the sizes of clusters. This could be achieved through a technique that runs k-means++ iteratively until the overall clustering sizes are more evenly distributed. However, challenges emerge when points naturally form unbalanced clusters. In such cases, additional clustering time may outweigh the benefit of a more balanced distribution.

Another potential improvement that was not implemented is dynamic clustering updates during simulation. The current implementation relies on a static cell size for the Barnes-Hut Algorithm. When points travel far from their original clusters, the construction of the tree becomes unbalanced, with most points concentrated in one child node in the first few layers of the tree. To counteract this, dynamic clustering updates during the simulation could be introduced. However, this approach comes with significant challenges and overhead, including determining the trigger for re-clustering and reorganizing data storage structures. Implementing this technique without extensive experimentation might inadvertently reduce the program's performance.

In conclusion, these experiments shed light on the performance characteristics of the implemented computational approach and provide insights into how different parameters and techniques impact execution time, speedup, and accuracy. Understanding the interplay of these factors is essential for optimization and fine-tuning efforts in computational simulations.

Reference:

[1]*Numerical Methods Handbook.* CampusPress. https://bpb-us-e1.wpmucdn.com/sites.northwestern.edu/dist/2/77/files/2017/01/numerical_methods.compressed-1jnsi3e.pdf

[2] Barnes Josh and Hut Piet. 1986. *A hierarchical O(N log N) force-calculation algorithm. Nature* 324, 4 (1986), 446–449. https://doi.org/10.1016/j.nmd.2014.06.252

[3] Martin Burtscher and Keshav Pingali. 2011. *An efficient CUDA implementation of the tree-based barnes hut n-body algorithm.* In GPU computing Gems Emerald edition. Elsevier, 75–92. https://doi.org/10.1016/B978-0-12-384988-5.00006-1