Simulating Thermal Equilibrium with Lennard-Jones Potential

Paul Yang, Minh Le, Nikhil Datar, Dhrubhagat Singh

May 5, 2024

Abstract

Understanding molecular dynamics, and more broadly particle interaction, is important in a variety of fields including drug discovery [1], material science, biological mechanisms, and more. This project simulates and visualizes the dynamics of particles in a two-dimensional space using the Lennard-Jones potential to model inter-particle forces. The simulation calculates forces between particles based on the Lennard-Jones potential, which comprises attractive and repulsive components. Particle collisions are detected and resolved to ensure realistic motion, and the system's kinetic energy is tracked over time to monitor equilibrium. Importantly, this research is aimed at understanding the suitability of this problem scope to high performance computing (HPC). We evaluate 3 implementations, namely, a baseline sequential implementation, shared memory implementation, and a combined shared and distributed memory implementation, for various particle problem sizes. Overall, we find a performance speedup of 2.1 for our shared memory model and 6.2 for our combined shared and distributed memory model relative to the baseline, showing the benefit of applying HPC to this setting, and supporting the ability for future research at larger problem scales to be conducted.

1 Background and Significance

As the number of particles increases, accurately simulating the interactions and collisions through the use of the Lennard Jones Potential and Verlet Velocity becomes computationally expensive. By employing parallel computing strategies, we aim to significantly reduce computation times and improve the efficiency of the simulation that models thermal equilibrium.

To investigate and compare the effectiveness of our parallel strategies in particle dynamics, we implemented three models:

1. **Baseline Sequential Model**: This sequential model processes particle interactions on a single CPU core. It serves as a baseline to measure the speedup gained from parallelization.
2. **Shared Memory (SM) Model**: This model uses OpenMP for shared-memory parallelism, enabling multi-core processing on a single machine. Each thread computes a portion of the particle interactions concurrently, reducing the overall simulation time compared to the baseline.
3. **Shared and Distributed Memory (SDM) Model**: This combination of distributed and shared model employs Message Passing Interface (MPI) and OpenMP for multi-node parallelism. Each node computes interactions for a subset of particles, enabling the simulation to scale across multiple machines for handling larger number of particles and compute their interactions through the Lennard Jones Potential.

The significance of the project lies in the use of multi-threading and distributed computing to significantly speed up particle simulations, which is essential for handling large-scale systems that may require many computations. The use of the Lennard Jones Potential also relates to natural phenomena and systems that are striving to reach thermal equilibrium which can be used to simulate different molecular dynamics and assist with material science research and help develop strong and more durable material under different conditions. While many particle simulations use specialized hardware like GPUs or high-end computing clusters, not all focus on directly comparing different parallelization strategies. This project differentiates itself by providing a clear comparison between single-core, shared-memory (SM), and distributed/shared-memory (SDM) models within the same simulation framework with various number of simulated particles.

2 Scientific Goals and Objectives

The primary scientific goals and objectives of this project revolve around simulating a large-scale particle system to understand dynamic behaviors, collision effects, and energy interactions in high-density environments. By comparing different computational models—sequential, SM, and SDM—we aim to evaluate their performance in terms of speedup and efficiency in simulating large particle counts. Furthermore, we intend to assess the scalability of each model with increasing particle numbers and computational resources. A significant objective of the project is to ensure resource efficiency by developing and benchmarking computational models that minimize energy and resource consumption while maximizing throughput. Lastly, we aim to explore optimization strategies in multi-threading and distributed/shared computing to determine the best approach for maximizing computational efficiency in large-scale particle simulations. The need for compute hours on an HPC architecture is crucial for several reasons. First, the problem involves simulating thousands and hundred of thousands of particles, which requires significant computational power for realistic simulation speeds. HPC systems offer the necessary computational resources to handle such massive parallel workloads efficiently. Moreover, the project requires exploring both shared-memory (SM) and distributed/shared-memory (SDM) parallelism, necessitating a multi-core and multi-node environment which HPC architectures can provide. In addition to computational power, HPC systems provide the necessary tools, libraries, and infrastructure support for high-performance simulations at scale, which is difficult to replicate on standard local systems.

3 Algorithms and Code Parallelization

Algorithms: The computational study primarily involves the simulation of particle dynamics where each particle's movement is influenced by Newtonian mechanics. The interactions between particles are managed through collision detection algorithms and force calculations based on Lennard-Jones potential and updating the velocities with the Velocity Verlet algorithm which both together models the realistic particle interactions. The updating functions for these equations are listed below, where (1) calculates the total potential energy of the system, (2) updates the particle positions, (3) is an intermediate velocity calculation, (4) calculates the acceleration, and (5) updates the particle velocities.

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1}$$

$$r(t + \Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{2}$$

$$v\left(t+\frac{1}{2}\Delta t\right) = v(t) + \frac{1}{2}a(t)\Delta t \tag{3}$$

$$\text{Calculate } a(t+\Delta t) \text{ based on } r(t+\Delta t) \tag{4}$$

$$v(t+\Delta t) = v\left(t+\frac{1}{2}\Delta t\right) + \frac{1}{2}a(t+\Delta t)\Delta t \tag{5}$$

Setup: For our three different models, we generated initial particle positions and velocities for 675, 1250, 2500, 5000, 10000, 25000, 50000, and 100000 particles, and wrote the positions and velocities to separate files depending on the number of particles. The code for this step is present within `generate_particles.cpp` and this code essentially generates the particle positions using a uniform random distribution, as the rand function in C++ by default uses a uniform distribution. For each of our models, we initialize the particles by reading the particle positions and velocities from the input file and store it in a vector. By reading in the same particles randomly distributed by our script, we ensure that our simulation experiments are standardized across the different parallelization methods (SM, SDM). The most important function in our simulation is our `updateParticles()` function, which updates positions and velocities based on the Lennard-Jones potential, as this function has to iterate through all of the different pairwise interactions between particles. Based on a profiling of our baseline simulation, where we used `gprof`, a program profiling tool, it was revealed that the `updateParticles()`, which was the function that updates the particle positions and velocities is the most computationally intensive, comprising 47.8% of the computations. Our Baseline Model simply applied the Lennard-Jones Potential and Velocity Verlet formulas/algorithms to calculate the accelerations of the particles based on each pairwise interaction, and use the accelerations to update the velocities and positions of each of the particles.

Simulation Stopping Condition: Upon running initial experiments of our simulation with different particle sizes, we realized that our initial goal of having to wait for thermal equilibrium, which occurs when there is a stabilization of kinetic energies across particles, was too restrictive of a condition to be able to effectively simulate larger particle simulations, namely the $100,000$ size simulation for the baseline model. This is because a bulk of the simulation time was spent trying to ensure that the kinetic energy fluctuations met a certain threshold. Upon reconsideration, we rethought our stopping condition to instead capture the first 100 steps within the $100,000$ particle size simulation, which took over 3.5 hrs to run using the baseline algorithm, and then use that as the stopping condition for all the simulation sizes less than $100,000$. This condition was also favorable due to the fact that it was able to capture most of the exponential decay in kinetic energies across the different particle size simulations less than $100,000$, and it was this particular decay that we were looking into parallelizing and speeding up for more standard comparison.

SM Model: We utilize `pragma omp parallel for` to run the pairwise computations, in addition to the velocity and position updates, on multiple threads. It is important to note that a pairwise interaction between two particles is independent of a pairwise interaction between another two particles, unless one of the particles is in both pairwise interactions, which may lead to race conditions. Thus, we decided to implement our parallelization using dynamic scheduling to ensure that the workload is balanced across threads. To account for the race conditions, we declare the Lennard-Jones Potential parameters, in addition to the acceleration parameters, to be shared.

SDM Model: We build off of our SM Model with the strategy of essentially splitting our 2D particle space into 4 distinct quadrants as shown in the *Figure 1*, where each quadrant is run on a separate node and each node is responsible for the computations of its assigned quadrant. Essentially, the implementation for updating the particle positions and velocities based on the Lennard-Jones Potential and Velocity-Verlet algorithm is kept the same, except after a given iteration of updates is completed for a quadrant, we check to see if the positions of any particles are outside the boundaries for its respective quadrant. Afterward, we send the number of particles that should be in a neighboring quadrant, as well as the position and velocity data for these respective particles, to the corresponding neighboring quadrant through non-blocking communication (`MPI_Isend` and `MPI_Irecv`). To ensure computations for pairwise interactions are synchronized for the entire 2D space, we utilize the `MPI_Waitall` command to essentially wait for all of the sends and receives for a given iteration to occur before proceeding to the next iteration. In addition, we also handle collisions that occur at the boundaries, which we determine between two neighboring quadrants by sending the particle position and velocity to the neighboring quadrant/node, which is also done using `MPI_Isend` and `MPI_Irecv`, and synchronized using `MPI_Waitall`. More specifically, we implement the collisions by allocating a dynamic buffer that scales with how many particles are "near" the boundary regions of our quadrants, where we define the proximity threshold to be if any particle is positioned a particle radius distance away from the boundary region.

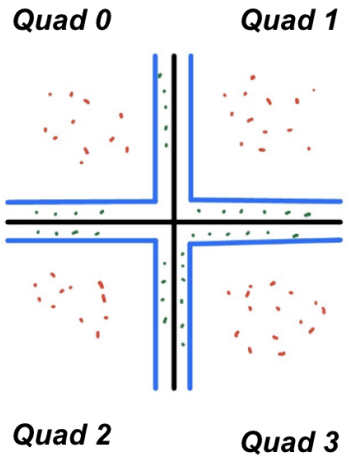

Figure 1: SDM Implementation using MPI

Validation/Verification: As per our methodology and implementation, particles approach equilibrium through a noticeable decrease in kinetic energy, as shown in the results in *Figure 3*. This phenomenon is inherently connected to the Lennard-Jones potential, where particles at close distances exhibit attractive forces, leading to clustering and gradual stabilization. The significant decrease in kinetic energy over time for two different particle counts. This decrease indicates that particles are losing kinetic energy, resulting in reduced movement and rearranging themselves into more stable configurations. This process ultimately leads to the system reaching thermal equilibrium, where particles no longer have enough energy. The clustering observed in *Figure 2* corresponds with the decrease in kinetic energy aligns with the theoretical expectations, indicating that as particles lose kinetic energy, they become more prone to interactions governed by the Lennard-Jones potential, pulling them into clusters due to the strong attractive forces at shorter distances. The clustering further stabilizes the particles, leading to a state of reduced kinetic energy. These results are reasonable because as discussed in [2], the equilibrium distance in the

Lennard-Jones potential is critical for accurately modeling adhesive contact which leads to clustering and the decrease in kinetic energy as it reach thermal equilibrium.

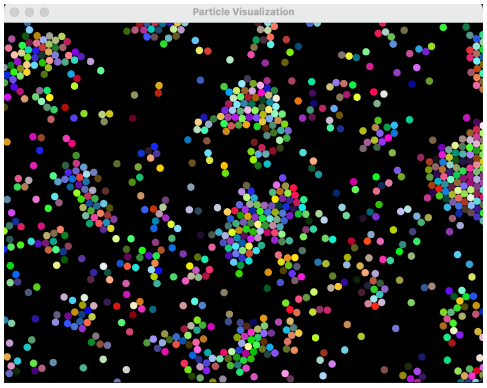

Figure 2: Particle Visualization

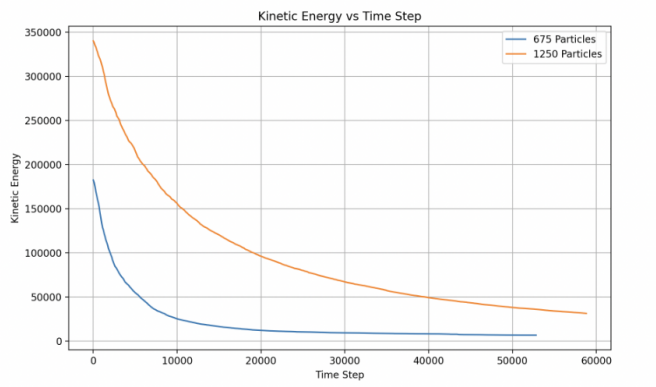

Figure 3: Kinetic Energy vs Time

4 Performance Benchmarks and Scaling Analysis

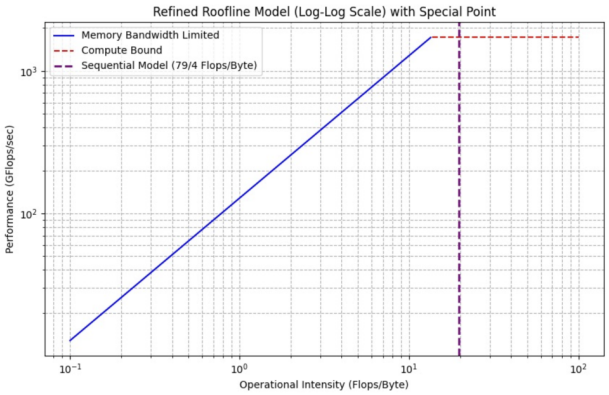

Figure 4: Roofline Analysis

Roofline Analysis: We conducted a roofline analysis of our hardware architecture and measured the operational intensity of our baseline simulation, as can be seen in *Figure 4*. We base our roofline model on one Skylake compute node of the Intel 8124M processor, and we compute the nominal peak arithmetic performance to be 1728 Gflops/s and the nominal peak memory performance to be 128 GB/s. Thus, we calculate that the ridge point is 1728/128 = 13.5 Flops/Byte, as we can see in *Figure 4*. Based on the memory accesses and operations of our overall simulation, we compute that the operational intensity of our baseline simulation is 19.75 Flops/Byte, which places our baseline simulation under the compute-bound based on our roofline model. This makes logical sense, as we noticed in our profiling of the baseline simulation that the majority of the computations occur through calculating the Lennard-Jones potentials from the pair-wise interactions between all particles, and so we are essentially bottlenecked by the number of computations that are necessary for each iteration of our simulation to be able to update the positions and velocities of the particles.

We ran each of the baseline, SM, and SDM implementations for 10 trials and recorded the average runtimes for each of the programs on the 675, 1250, 2500, 5000, 10000, 25000, 50000, and 100000 input particle files. We get the following graphs for our simulations:

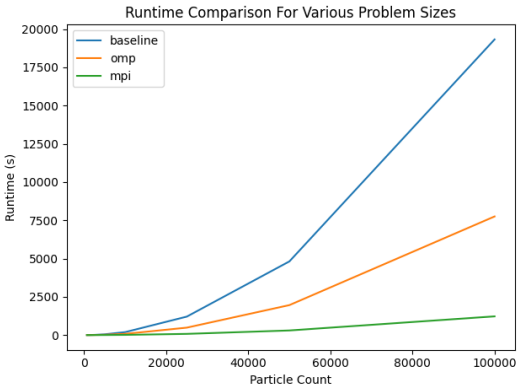

Figure 5: Runtime Across Particle Counts

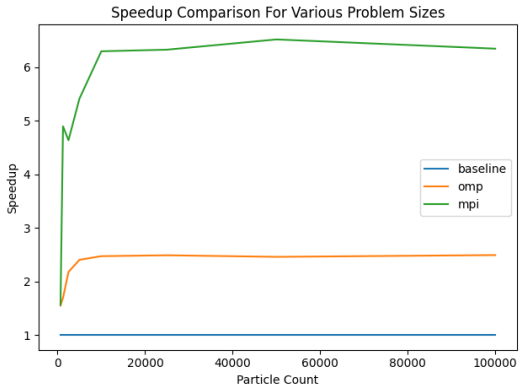

Figure 6: Speedup Across Particle Counts

Runtime Analysis: As we can see from *Figure 5* and *Figure 6*, we can clearly observe that for the runtime across the various particle counts, the baseline model has the highest runtime followed by the SM implementation followed by the SDM implementation. This makes logical sense given the segment of our code containing the $O(n^2)$ pairwise operations required to update particle positions and velocities scale, which scale intensely in the sequential baseline mode, leading to greater runtimes. By a similar line of reasoning, we can therefore see that the runtime is indeed reduced in the parallel implementations, as it more effectively distributes the workload in those velocity and acceleration computations, with the SM and SDM implementations.

Proportion of Parallelizable Code: An important consideration for our scaling analysis and speedup analysis is that given our baseline implementation of the particle simulation, there are certain functions in our simulation that are not necessarily parallelizable. For example, checking if particles are placeable in the 2D simulation space and also resolving collisions and overlaps between particles are functions that are not necessarily parallelizable and are not computationally exhaustive based on our profiling. Thus, we estimate that approximately 70% of our baseline simulation code is parallelizable, and we utilize this metric when analyzing our speedup and calculating the ideal strong and weak scaling in our analysis.

Speedup Analysis: Prior to implementation, we decided to determine rough upper bounds on the expected speedup we would see after implementing SM and SDM. For the SM implementation, using Amdahl's Law, and our estimate of 70% parallelizable code, we see that the theoretical speedup is

$$S_{Ideal} = f + \frac{1-f}{p} = .3 + \frac{.7}{18} = 2.95$$

From *Figure 6*, we can see that the speedup for our SM code was roughly 2.1. While this is less than the ideal speedup, it is still quite close, and can be explained by the fact that utilizing multiple processors can result in increased execution time due to the need for load balancing, cache coherence, and other overheads, which can increase the execution time and cause deviations from the ideal.

To place an upper bound on our SDM implementation, we noted that we split up the particles across 4 nodes. Since our algorithm is $O(n^2)$ time complexity, we expect that each node runs 16 times faster

than running on one node. Since we previously estimated that the `updateParticles()` function, our only $O(n^2)$ function, comprises 47.8% of the computations, multiplying this by the decreased workload gives a rough upper bound on the ideal speedup of

$$S_{Ideal} = (\frac{n}{n/4})^2 * .478 \approx 7.648$$

From *Figure 6*, we can see that the speedup for our SDM implementation was roughly 6.2. While this is slightly less than the ideal, the deviation can be explained by not only the problems with thread load balancing mentioned previously, but the additional overhead of synchronization across multiple nodes. Overall, both implementations are slightly lower than the ideal speedup, which is understandable given the previously mentioned practical considerations.

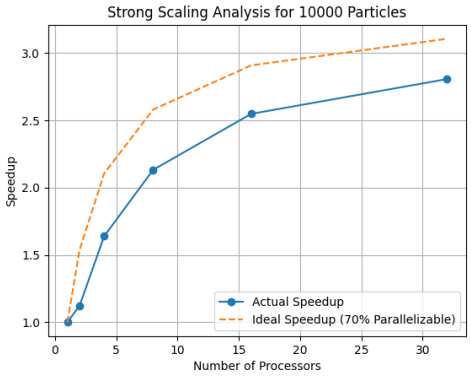

Figure 7: Strong Scaling Analysis

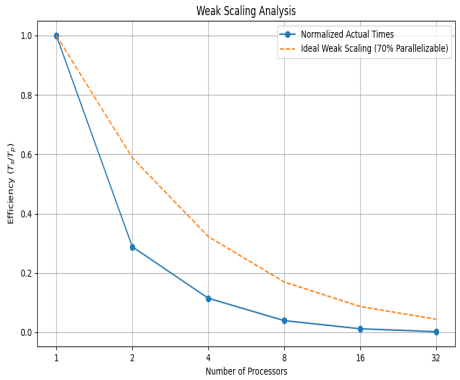

Figure 8: Weak Scaling Analysis

Scaling Analysis: We can notice in the strong and weak scaling analysis above that our simulation follows both the ideal strong scaling and ideal weak scaling trends, although it is slightly off from the ideal in both analyses. This makes sense because in the strong scaling analysis, we evaluate how much of a speedup we can observe from our implementation given a fixed workload, this case being 10000 particles, from utilizing more threads (processors). Meanwhile in the weak scaling analysis, we similarly evaluate speedup but this time we proportionally increase the workload (particle count) with the number of processors. Thus, the differential in ideal speed up for strong scaling and efficiency for weak scaling can be explained due to the overhead and potential thread imbalance that comes from increasing the number of threads as we run our parallelized simulation in a real-world scenario with limited computational resources. However, because we observe that our parallelized simulation exhibits both strong and weak scaling very close to the ideal, we can conclude that our parallelization technique utilizes computational resources quite efficiently and manages the workload across different threads in a balanced manner. This also suggests that we may be able to scale to larger problem sizes, such as increasing the number of particles in our simulation, given a proportional increase in computational resources as well.

Workflow Parameters and Results: *Table 1* and *Table 2* show the workflow parameters and raw results we observed in our experiments. Overall, the wall clock time scales in a quadratic manner with the number of particles, the memory scales linearly with the number of particles, while all other parameters are kept constant.

	675 Particles	1250 Particles	2500 Particles	5000 Particles
Typical wall clock time (seconds)	0.368567	0.363718	1.19731	3.70316
Typical job size (nodes)	4	4	4	4
Memory per node (KB)	21.5	39.8	79.8	159
Maximum number of input files in a job	1	1	1	1
Maximum number of output files in a job	1	1	1	1
Library used for I/O	fstream	fstream	fstream	fstream

Table 1: Workflow parameters of the first 4 experiments used during project development.

	10000 Particles	25000 Particles	50000 Particles	100000 Particles
Typical wall clock time (seconds)	12.3917	76.5249	300.238	1221.45
Typical job size (nodes)	4	4	4	4
Memory per node (KB)	319	795	1550	3100
Max. number of input files in a job	1	1	1	1
Max. number of output files in a job	1	1	1	1
Library used for I/O	fstream	fstream	fstream	fstream

Table 2: Workflow parameters of the last 4 experiments used during project development.

5 Resource Justification

The optimal job size based on our experiments from Section 4 is 4 nodes. For our largest number of particles being 100000 particles, the wall clock time was 1221.45 seconds on average, which is equivalent to ~ 1.357 node hours, as a result of the following product:

$$1.357 \text{ node hours} = 4 \text{ nodes} \times \frac{1221.45s}{3600\frac{s}{\text{hour}}}$$

Since the simulation took 1.357 node hours for 100 iterations, that means it takes 0.01357 node hours for 1 iteration. However, this benchmark only represents 100 iterations, which is not enough iterations for the particle simulation to reach thermal equilibrium based on how we define it. In a real production run, we will need to extend our simulation to run for greater iterations in order to reach thermal equilibrium, which we define as when the change in kinetic energy is small enough below our threshold.

To estimate how many iterations should be necessary to complete a simulation in production, we can refer to *Figure 3*, in which we can see that the turning point for when the simulations for both particle sizes 675 and 1250 approach equilibrium after approximately 5000 iterations.

Additionally, we may wish to run our simulation for a larger number of particles in order to model more complex physical systems, and so another experiment that we would like to run would be a simulation using 200000 particles. From *Figure 5*, we can notice that our SDM implementation follows an increasing trend and we can extrapolate from the visualization that the runtime for 200000 particles would likely be approximately 4800 seconds on average, which would be equivalent to ~ 5.333 node hours, as a result of the following product:

$$5.333 \text{ node hours} = 4 \text{ nodes} \times \frac{4800s}{3600\frac{s}{\text{hour}}}$$

Since the simulation with 200000 particles is estimated to take 5.333 node hours for 100 iterations, that means it would take 0.05333 node hours for 1 iteration. Again, we would like to run this experiment for 5000 iterations in order to reach the point in which the simulation reaches thermal equilibrium.

	100000 Particles	200000 Particles
Simulations per task	10	10
Iterations per simulation	5000	5000
node hours per iteration	0.01357	0.05333
Total node hours	678.500	2666.667

Table 3: Justification of the resource request

Thus, based on the total node hours that we see in the table above, we request a total of $678.5 + 2666.667 = 3345.167$ node hours to cover both simulations with 100000 and 200000 particles.

6 Limitations

Particle Motion Across Quadrants: The main limitations of our simulation stem from the assumptions that we made in order to proceed with our various parallelization strategies. The main assumption we made was that for our SMD model, we assumed that the particles mostly interacted with other particles in their quadrant, and that the particles can only cross through into their neighboring quadrants and cannot migrate to diagonally opposing quadrants. This assumes a very specific type of motion for all the particles we ideally simulate within our simulation. Although our simulation is flexible in the sense that it takes in different parameters for the Lennard-Jones potential and particle radius size, this specific restriction on motion might not represent or accurately model how real world physics and molecular dynamics works.

Communication Overhead: Furthermore, we could see from our performance scaling analysis that there is some amount of overhead in our SM and SDM model implementations, which would limit performance as we scale the simulation to larger particle sizes. A key source of this overhead for the SDM model, for example, can most likely be explained by potentially redundant particle position and velocity computations we have to perform for any particles that are at the boundary regions of the quadrants in our program.

Visualization at Scale: Another key limitation for our program is that the visualization of the simulation was only practically feasible for a very small amount of particles due to the overhead needed to render the graphical user interface. To visualize larger particle simulations, we would also need to figure out how to more effectively parallelize and handle graphics rendering for our simulation.

7 Future Work

For the future scope of our project, we would ideally like to explore more advanced parallelization techniques like GPU acceleration using CUDA [3]. This could particularly be very helpful in accelerating the graphics rendering, and also can potentially make the pairwise interaction updates quicker [4]. Additionally, we could work on parallelizing our I/O file, namely parallelizing the random generation of particles. For our SDM implementation, we worked with only 4 nodes, but we could potentially experiment with more nodes to have a more efficient distribution of the workloads.

References

[1] Jacob D. Durrant and J. Andrew McCammon, *Molecular Dynamics Simulations and Drug Discovery*, BMC Biology, Volume 9, Issue 1, 2011, Pages 71-71, https://doi.org/10.1186/1741-7007-9-71.

[2] Ning Yu and Andreas A. Polycarpou, *Adhesive contact based on the Lennard–Jones potential: a correction to the value of the equilibrium distance as used in the potential*, Journal of Colloid and Interface Science, Volume 278, Issue 2, 2004, Pages 428-435, ISSN 0021-9797, https://doi.org/10.1016/j.jcis.2004.06.029.

[3] Weiguo Liu, et al., *Accelerating Molecular Dynamics Simulations Using Graphics Processing Units with CUDA*, Computer Physics Communications, Volume 179, Issue 9, 2008, Pages 634-641, https://doi.org/10.1016/j.cpc.2008.05.008.

[4] Mario Santos Camillo and Wu Shin-Ting, *Accessing CUDA Features in the OpenGL Rendering Pipeline: A Case Study Using N-Body Simulation*, 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), IEEE, 2017, Pages 315-322, https://doi.org/10.1109/SIBGRAPI.2017.48.

8 Contributions

Paul Yang took the lead the Baseline component of the project, namely creating the first iteration of the Lennard-Jones model, while the other team members, Minh Le, Dhrubhagat Singh, and Nikhil Datar, contributed support in various capacities that led to the successful very first baseline model. Minh Le assumed a leading role in the SM model aspect of the project, particularly implementing our scheduling approach, with contributions from Dhrubhagat, Nikhil, and Paul in debugging and providing theoretical support to help identify parallelization opportunities in critical segments of the code. Dhrubhagat and Nikhil led the development of the SDM segment. Nikhil particularly worked on the first idea of distributing the workload across 4 nodes, and Dhrubhagat worked on adding logic to handle collisions of particles at the boundary regions of the newly assigned quadrants. The team also supported in helping migrate existing techinques from the SM model and adopt it to the SDM approach.

For the presentation, Paul took the lead on the Motivation, Mathematical Model, and Roofline and Profiling slides. Dhrubhagat took the lead on the Sequential Baseline, Testing/Comparison Method, and OpenMP Parallelization slides. Nikhil took the lead on the OpenMP + MPI Implementation, Results/Speedup, and Performance scaling slides. Minh took the lead on the Assumptions and Limitations and What's Next slide and provided overall refinement and editing on our presentation.

For the final paper, Paul took the lead on the Background and Significance and Scientific Goals and Objectives section. Minh took the lead on Algorithms and Code Parallelization. Nikhil took the lead on the Performance Benchmarks and Scaling Analysis. Dhrubhagat took the lead on the Resource Justification and Limitations. The team all worked together on the Future work segment and overall editing and refinement of the paper.