Time Spent: 70 Hours

Hardware Details: Codio

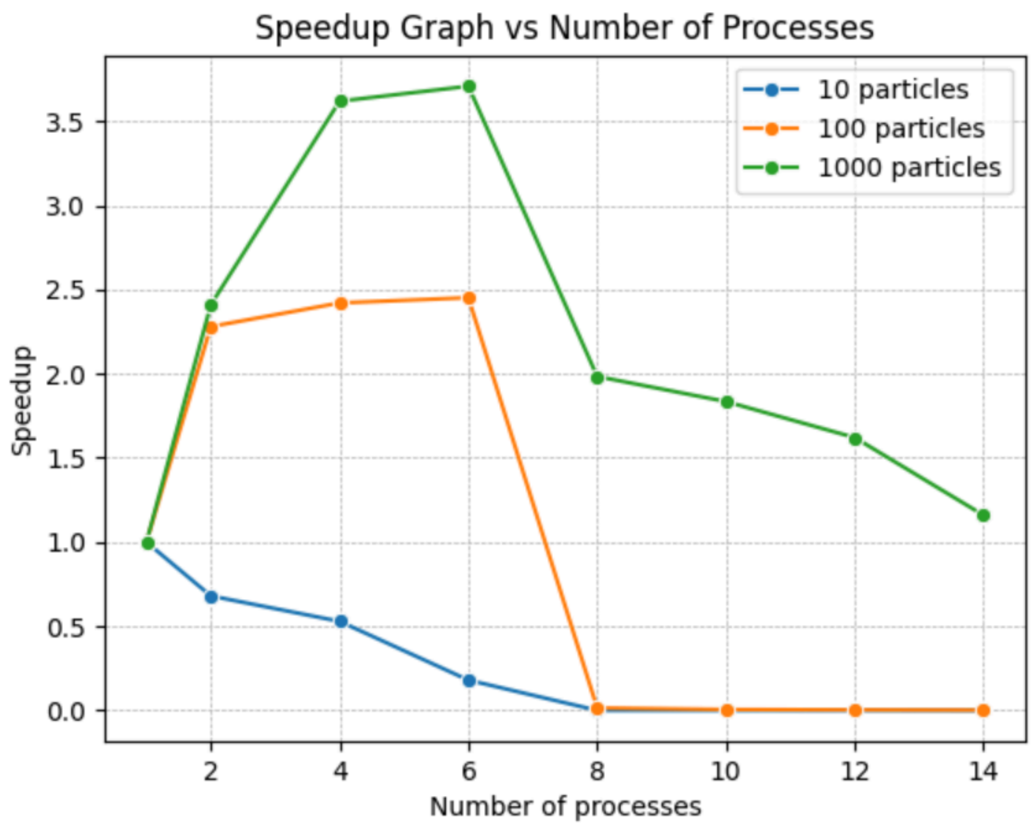
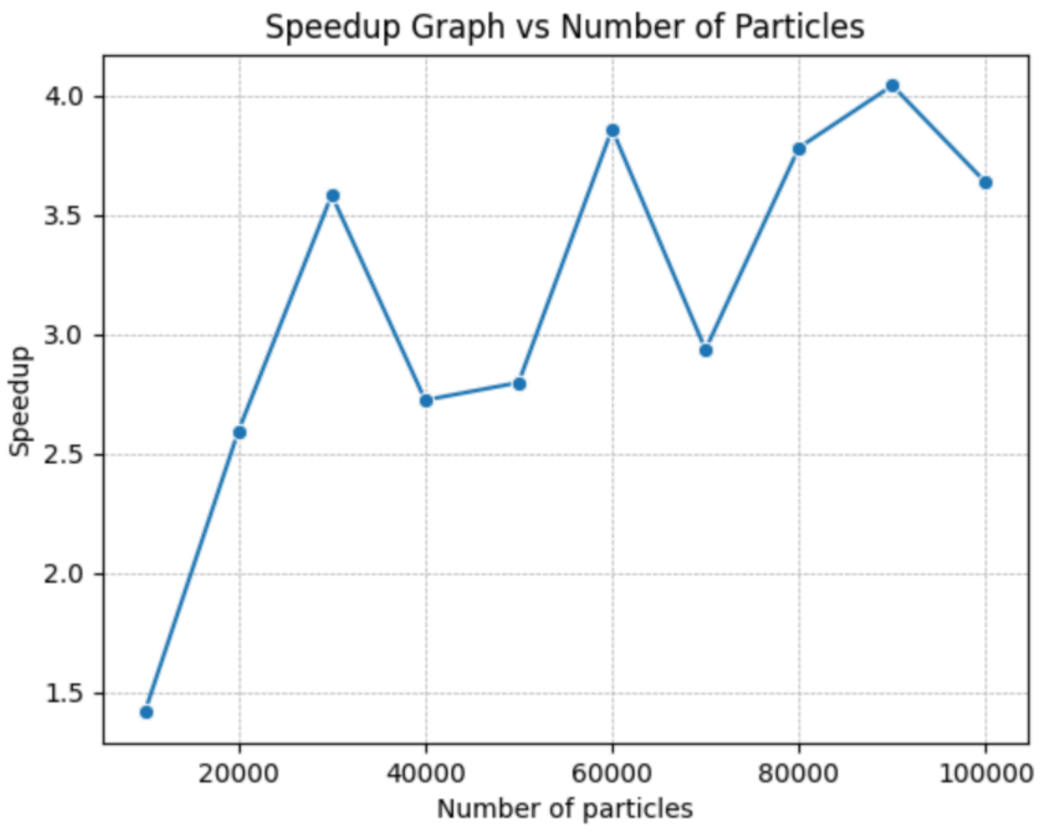


Figure 1. Speedup graph vs processors using 100 steps.

My MPI implementation involves using processor 0 for initialization steps. The input is read into every processor to avoid communication overhead. In addition, we have every processor build their own trees and contain their local copy of all particles. Each processor will compute forces and update positions on their local copy. When they are finished, each processor will broadcast their chunk of the array to the other processors. Every processor will have an updated array of particles and the next iteration begins. Figure 1 shows that the implementation is scalable to the number of cores the system has. We see a sharp increase with the increasing processor count up to 6 processors. Since my Codio environment was limited to 7 processors, we can observe a sharp decrease starting at 8 processors. This is likely due to multiple processes running on the same core, which reduces the amount of time each process gets to run on the cpu core and incurs context switch overhead. The main point is to show increasing speedup with increasing core count. We can also observe the amount of increase increasing with more coarse workloads. In Figure 1, we can see the slightly higher speedup increase gained with 100,000 particles compared to the relatively plateaued speedup with 100 particles. This is likely due to the amount of time spent in communication being relatively lower in 100,000 particles compared to 100 particles. With 10 particles, the time spent in communication dominates the total execution time, leading to a slowdown when compared to sequential. Figure 2 further supports that communication overhead lowers as work granularity becomes more coarse. The amount of speedup with 10,000 particles starts below 1.5. However, as we decrease work granularity by adding more particles, we can observe a steady increase in speedup.

Figure 2. Speedup Graph vs Particles using fixed 100 steps and 6 MPI Nodes



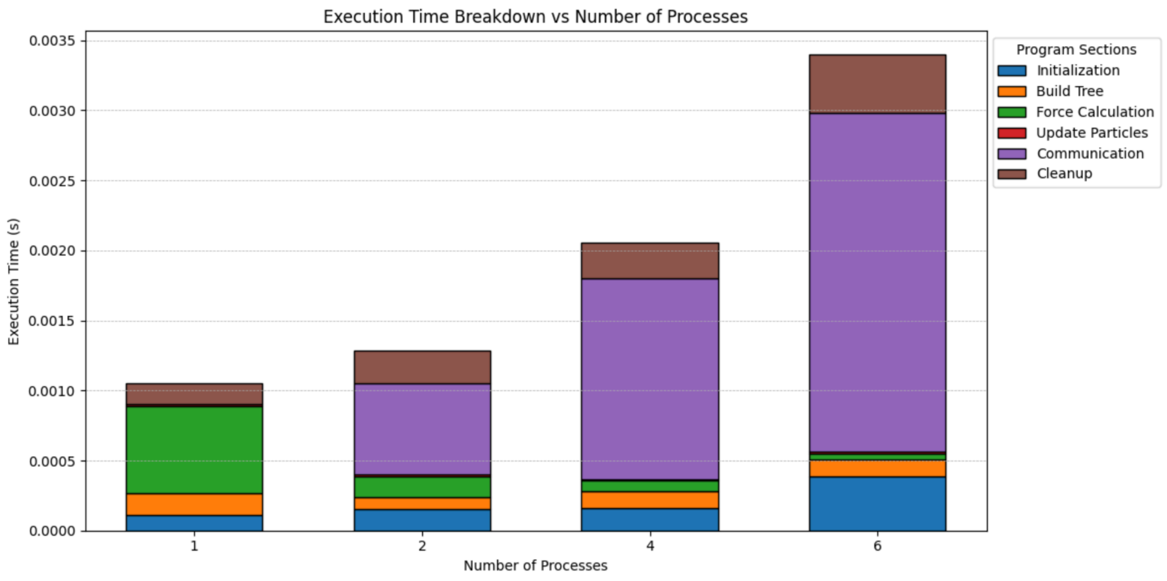


Figure 3, Execution Time Breakdown Graph for 10 particles 100 steps. CPU core counts higher than 6 were not used for visual clarity, Initialization refers to any overhead before the program can begin looping through the simulation. This includes initializing data structures and reading input.

Communication refers to the total amount of time spent on CPU cores broadcasting information. Cleanup includes freeing dynamically allocated memory and writing the output to a file. Note that some categories may be missing or appear missing; these categories are either not used or make up a very tiny portion of the execution time.

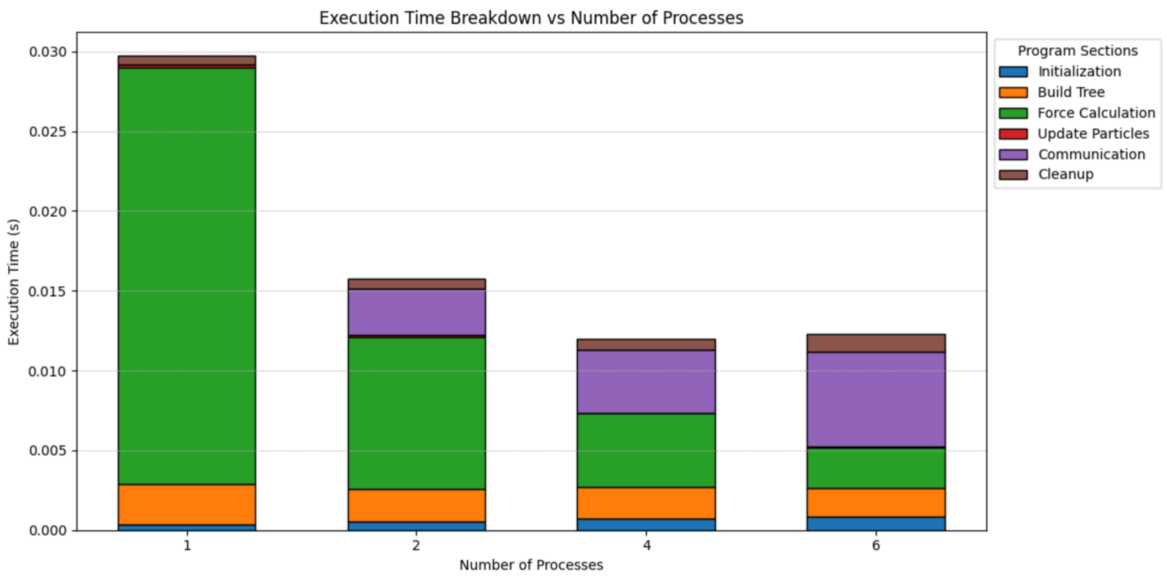


Figure 4. Execution Time Breakdown Graph for 100 particles 100 steps. CPU core counts higher than 6 were not used for visual clarity. Initialization refers to any overhead before the program can begin looping through the simulation. This includes initializing data structures and reading input. Communication refers to the total amount of time spent on CPU cores broadcasting information. Cleanup includes freeing dynamically allocated memory and writing the output to a file. Note that some categories may be missing or appear missing; these categories are either not used or make up a very tiny portion of the execution time.

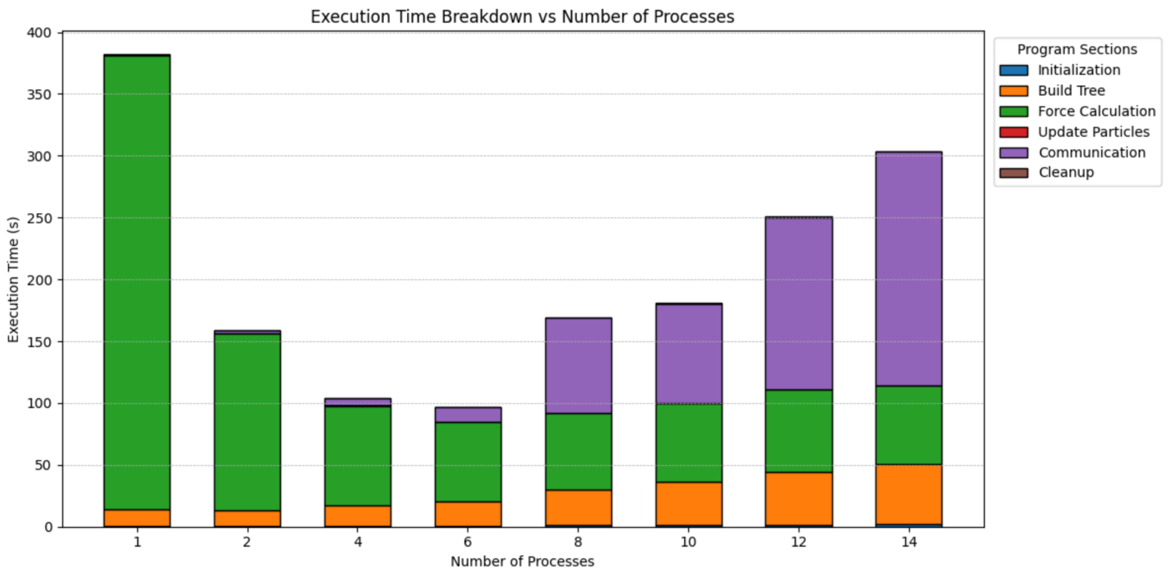


Figure 5. Execution Time Breakdown for 100,000 particles 100 steps. CPU core counts higher than 6 were included here for more information.(My Codio environment supplied me with 7 cores so higher execution times are expected) Initialization refers to any overhead before the program can begin looping through the simulation. This includes initializing data structures and reading input. Communication refers to the total amount of time spent on CPU cores broadcasting information. Cleanup includes freeing dynamically allocated memory and writing the output to a file. Note that some categories may be missing or appear missing; these categories are either not used or make up a very tiny portion of the execution time.

Figures 3, 4, and 5 show detailed breakdowns for the execution times at 10, 100, and 100,000 particles respectively. In Figure 3, we see drastic decreases in force calculation times as the CPU core counts increase. However, as the number of cores increases, the communication costs also increase more rapidly than the decrease in force calculation. The cost of communication alone using 4-6 cores is greater than the entire execution time of the serial program. In Figure 4, because we are using more particles, there is more work for each processor per iteration, leading to a more coarse workload and better speedup. With 100 particles, we still have the problem of communication costs making up ~50% of the total execution time for 6 cores. Figure 5 shows that using a very coarse workload yields better speedup results because the communication time is proportionally small. Using 6 cores, communication costs make up ~10% of the total execution time, which is much better than the ~50% for 100 particles and ~80% for 10 particles.