1 Introduction

In this assignment we worked to improve a sequential algorithm for the simulation of particle forces, and to parallelize the algorithm using OpenMP.

2 Sequential Algorithm Improvements

The configuration of the original sequential algorithm was to compute the forces of one particle on all particles per each time step, thus requiring $O(n^2)$ time. We have worked to linearize this solution, to reduce the complexity to $O(n)$. In order to achieve this goal, we first approached the problem by considering the grid space and worked to divide the space into a finite number of bins, each with a size of $(totalgriddimension/cutoff)$. Upon splitting the grid into bins, our algorithm traverses each bin in row-major order and computes the forces on all particles in the bin from its neighbor bins. By reducing the size of the computation space to one in which only the neighbor points within the cutoff range are accounted for, the time complexity reduces to $O(n)$ (as opposed to computing the potential force of all points on a single point which costs $O(n^2)$). Consider the following:

c = cutoff
A = area of computation space = $9c^2$, where 9 is the number of bins in the space
d = density of computation space = $|P|/A$ for P points

Therefore, $A = |P|/d = 9c^2$ and so $|P| = 9c^2d$, which represents the number of comparisons per point. In order to linearize the time complexity, our goal is to achieve $9c^2dn$ total comparisons for n points. Thus, so long as $9c^2d < n$, this goal will be met. Given that the density of the grid is evenly distributed on average, we would not expect the density of the computation space to ever reach size n.

Figure 1: Mock grid with bin indexing

	0.4	0.8	1.2	1.6	2
0.4	0	1	2	3	4
0.8	5	6	7	8	9
1.2	10	11	12	13	14
1.6	15	16	17	18	19
2	20	21	22	23	24

Figure 1 represents a mock-up of the particle space. Highlighted in yellow is an example of a computation space of $9c^2$ where c is the cutoff (e.g. 0.4). For a point at (1,1), all of its potential neighbors exist in the highlighted area. In order to account for all areas in the grid, the total number of bins = $(1 + n/c)^2$, where n is size of grid, c is cutoff, and $1 + n/c$ is number of columns. We included one extra column to account for edges in areas where the cutoff does not perfectly divide the grid space. For example, the index of the bin at point (1,1) would be $(x/\text{cutoff}) * \#\text{columns} + (y/\text{cutoff}) = int(1/0.4) * 5 + int(1/0.4) = 12$, as shown in Figure 1. By indexing each point with linear bin notation, we can define the computation space for each point to consist of only the 8 surrounding boxes and its own box, rather than all boxes in the grid space. After binning each block, we created a linked list of bin addresses. Knowing that for i in [-1,1] and j in [-1,1], $z + (i * \ \#\text{columns}) + j$ represents the index of all 9 bins, or neighbors (e.g. bins 6-18 in Figure 1), in a computation space relative to a bin with index z, we defined the computation space as such, and computed 'apply_force' for each point with all points in this space.

For the cache complexity, we leveraged the fact that our bins are ordered by row, and so by accessing and

storing data using row-major, we could optimize our cache using spatial locality. To build our bins, we used a linked list of particles ordered by bin index. In our linked list, we defined all points in the computation space for the corresponding bin and set all other nodes to be null. Our algorithm then traverses the linked list and computes the force of all points on all other points in the list while ignoring nodes that are null. After computing the applied force for a 9-bin computation space, we then unbin the particles and repeat the process for each time step. By doing so, we reuse our memory allocation in our linked list with each time step.

As shown in our results below, we were able to achieve a time complexity of $O(n)$, and sometimes even better.

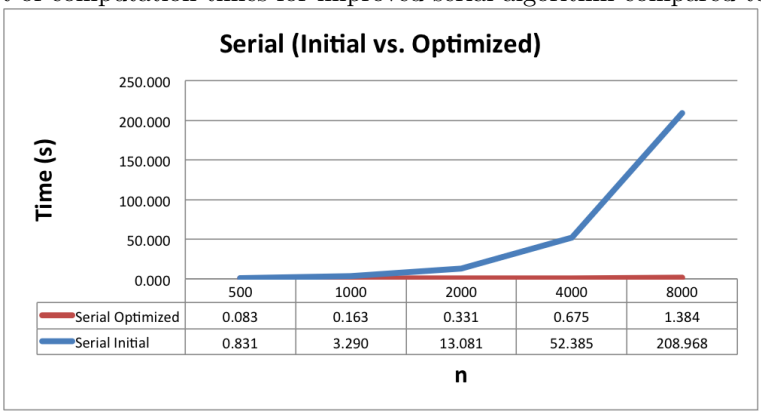
3 Parallel Algorithm

To parallelize the particle simulation algorithm, we used OpenMP pragmas for the for loops that traverse the bins in the grid and compute the forces between neighbors. Since the underlying data structure uses linked lists, the binning process must be done more or less sequentially as adding to any linked list or vector requires mutually exclusive writes.

4 Results

By incorporating binning, bin force computations, and row-major traversal, we were able to reduce the time complexity from $O(n^2)$ to $O(n)$, as shown in Figure 2. Compared to the slope estimate for the initial line fit of 1.994, the slope estimate for our improved serial algorithm is 1.018.

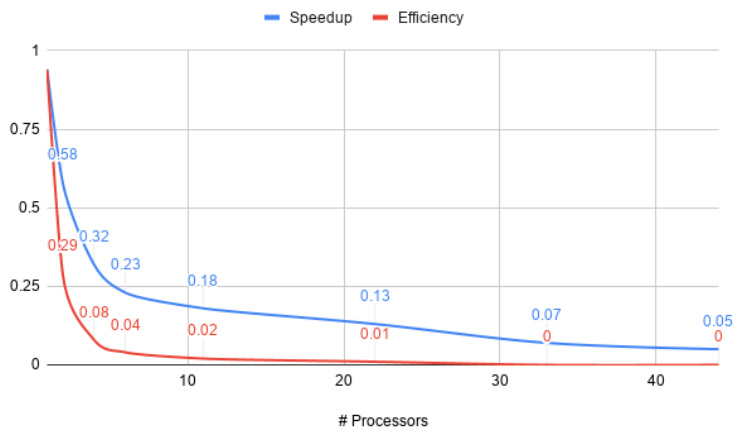
Figure 2: Plot of computation times for improved serial algorithm compared to initial serial



	500	1000	2000	4000	8000
Serial Optimized	0.083	0.163	0.331	0.675	1.384
Serial Initial	0.831	3.290	13.081	52.385	208.968

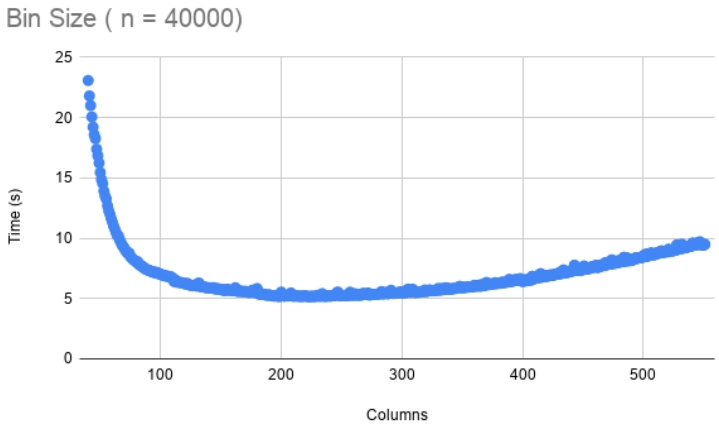
For the OpenMP parallel optimization, we weren't able to achieve any speedup and actually received parallel slowdown instead. There may be some hidden optimizations to be had with the binning data structure that would better take advantage of multiple processors. As mentioned earlier, our binning approach hopes to take advantage of temporal cache locality by repeatedly updating and accessing the same particles in adjacent bins. With a block distribution of bins across the grid, we would expect this cache complexity to not be such a major factor.

Figure 3: Parallel Speedup and Efficiency for OpenMP



To investigate the parallel slowdown, we ran some experiments with varying bin size. We found that the bin size that corresponded to the best performance of the OpenMP algorithm corresponded to about twice as long as the cutoff (225 instead of 450). With further experimentation with other varying parameters, we can find a correlation that causes this curve and replicate it explicitly in the algorithm.

Figure 4: Bin Size Experiments



5 Conclusion

In conclusion, using a binning strategy and optimizing spatial locality, we achieved linear time complexity for the serial algorithm. However, this same binning strategy seemed to cause a parallel slowdown in the OpenMP algorithm. There may be some more optimizations that we haven't yet explored with this.