Parallel Particles

Patrick Williams and Gabriel Marcus

October 2019

1 Introduction

Standard molecular dynamics algorithms rely on calculations of interactions between particles or objects mediated by a force which simulates actual physical laws. At each time step, positions, velocities and accelerations are determined for each particle. These values are continuously updated so that particles can experience distance dependent forces and move accordingly. In a naive algorithm, forces are computed by the interactions of every particle with every other particle. Ultimately, with n particles, this yields a time complexity of $O(n^2)$. The purpose of the following assignment was to redesign the MD algorithm in order to achieve a linear complexity of $O(n)$.

2 Revised Serial Algorithm

In order to reduce the required simulation time, each particle was allowed to interact with only a small subset of particles rather than every other particle. For this assignment, a linked cell list method as discussed in multiple papers and sourcess we researched was used to achieve a linear completion time. This approach subdivides the particle box into 2-dimensional cells with equal width and height. The length of a square cell for a given cutoff distance c and total space dimensions s by s is $s/(\lfloor s/c \rfloor)$. At each timestep, a particle will be located in one cell and interact with only those particles in its own cell or in the immediate neighbor cells sharing a side or corner. Pseudocode for the algorithm used is provided in Algorithm 1.

The linked cell list algorithm consists of two parts: Constructing two arrays whose values represent indices linking one particle to other particles in a cell followed by a scan to determine the nearest particles in neighboring cells which will interact and exert forces on a given particle. For the first part, two arrays designated *head* and *lscl* are populated to determine the members of a particular cell via index mapping. In the *head* array, the index of each position corresponds to a cell while the value in that position represents the number of the highest indexed particle in that cell. This value then links to an index of the *lscl* array whose value is the next listed member of that cell. By continuing to search in this manner through the *lscl* array, all particles in a cell can be enumerated until an empty cell is reached.

Forces acting on a specific particle in a specific cell are computed in the second portion of the algorithm which runs through a series of for loops that traverse each cell and its neighbors. Two scalar cell indices are computed in order to determine the first particle in cell i to interact with the first particle in cell j from the *head* array. Once this is done, the *lscl* array is used to find the other members of cell j that will exert forces on the particle in cell i. After all forces are calculated, the algorithm moves on to the next member of cell i using the *lscl* array mapping and repeats the process. At each timestep, this procedure is undertaken for each cell so that the positions, velocities and accelerations of all particles can be found for one iteration using only the necessary interactions from particles in neighboring cells.

3 Results

Our optimized code successfully changed the complexity of the algorithm from $O(n^2)$ to $O(n)$. The results of our scaling experiments are shown in in Figure 1 and Table 1. However, we were unable to make the code accurately predict particle movements. We adjusted different parameters, specifically the cutoff distance that affected the cell dimensions, but we were unable to run a serial simulation that properly visualized particle collisions and did not provide warnings about particle interactions. After debugging the code, we confirmed that each timestep of the simulation was properly calling the apply_force() function only for particles in neighboring cells of the chosen cell, and that no pair of particles was being computed more than once per timestep.

We also struggled in parallelizing the code. Because looping through all cells and all neighbor cells requires several nested loops working with the *head* and *lscl* arrays, we had difficulty determining how the data can be shared without issues.

Algorithm 1 Prepare *head* and *lscl* arrays for given timestep

```
for each item in $head$ do
    set value to -1
end for
for each particle $p$ do
    $cell_x = \lfloor p.x/cell\_edge\_length \rfloor$
    $cell_y = \lfloor p.y/cell\_edge\_length \rfloor$
    $head\_index = cell_x * \sqrt{num\_cells} + cell_y$
    $lscl[i] = head[head\_index]$
    $head[head\_index] = i$
end for
```

Algorithm 2 Calculate forces of particles on particles in neighboring cells

```
for each cell do
    calculate scalar index of cell $c$
    for each neighboring cell do
        calculate scalar index of neighbor $c1$
        $i$ = head[$c$]
        while $i$ != -1 do
            $j$ = head[$c1$]
            while $j$ != -1 do
                if $i < j$ then
                    apply_force(particles[$i$], particles[$j$])
                end if
                $j = lscl[j]$
            end while
            $i = lscl[i]$
        end while
    end for
end for
```

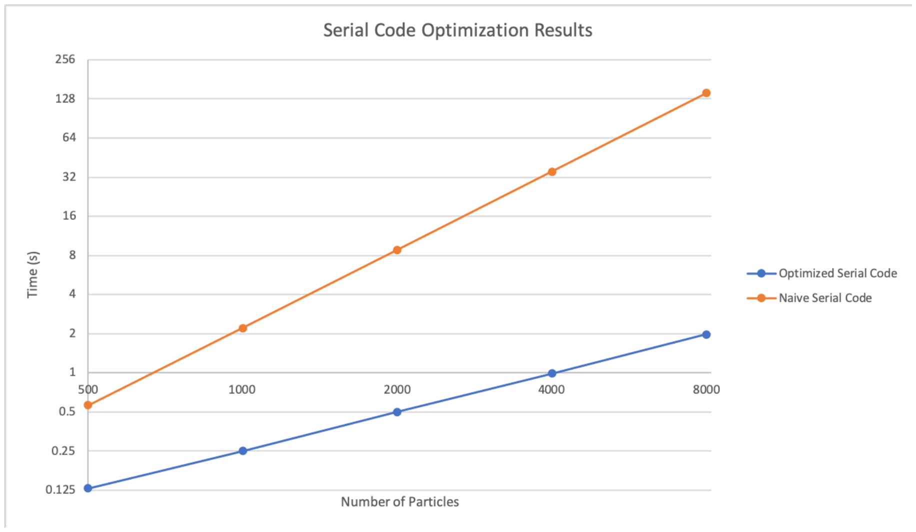
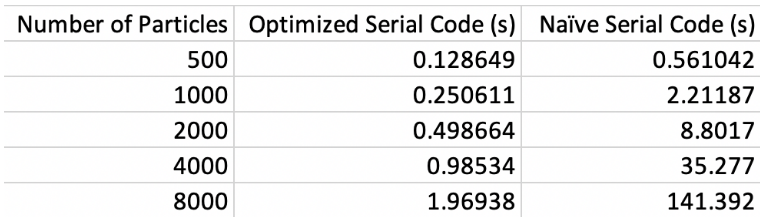


Figure 1: Results of linear time optimization of serial code



Number of Particles	Optimized Serial Code (s)	Naïve Serial Code (s)
500	0.128649	0.561042
1000	0.250611	2.21187
2000	0.498664	8.8017
4000	0.98534	35.277
8000	1.96938	141.392

Table 1: Results of linear time optimization of serial code