CSC367H1: Parallel Programming **Instructor: Maryam Mehri Dehnavi**

Project - Part 1

Name: Yifei Tang(1003815437), Zezhong Pan(1002057748)

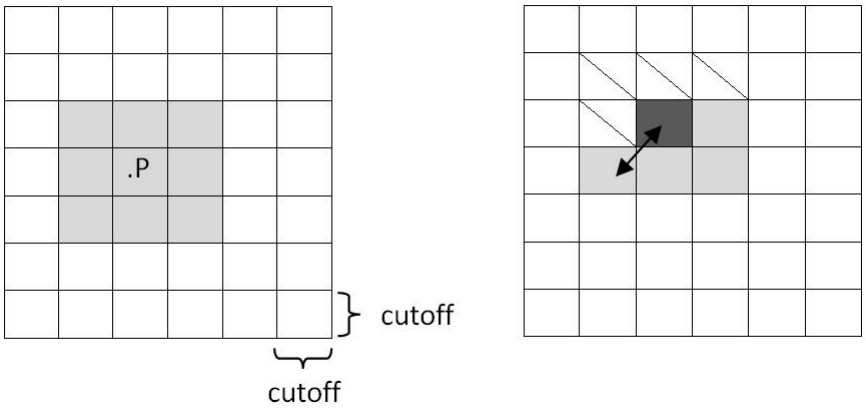
1 Introduction

In this project, we trying to optimize a piece of codes for particle simulation. In the simulation, n particles live in a plane, and two particles can interact with each other if their distance is closer than *cutoff*. The simulation was originally implemented in a naive way of checking all pairs of particles at each step, and make interaction if the distance is close enough, which has complexity $O(n^2)$. Given that the density of the particles is low enough with $O(n)$ expected interactions each time, our goal is to write an efficient serial implementation that run in $O(n)$, and then parallelize the new implementation with OpenMP.

2 Implementations

2.1 Optimized Serial Implementation

Since particles can only interact if their distance is less than cutoff, we don't really need to consider another particle that is very far away. Suppose we divide the plane into small squares of size $cutoff \times cutoff$, then a particle p in square S can only interact with other points in same square S or the 8 squares around it, and points in the same center square. Since the density of the particles is low, the expected number particles in the vicinity is constant.



If we iterate through all the squares, from left to right and row by row, and do two-way interactions instead of one-way interaction, then each time for all points in a square, we only have to consider at most 5 squares. The interactions from above and left should be already done from previous iterations.

With the analysis above, we design the optimized serial implementation this way. In each step, we first determine the square each particles is located and insert the particles to the linked list corresponding. Then we loop through the grid row by row, and compute the interactions. And finally move the particles according to their updated information.

2.2 Parallel Implementation

Based on the optimized serial implementation, we parallelized our codes using OpenMP. In each iteration of the simulation, there are 3 parts that can be considered for parallelizing.

The first part is inserting the points to the linked list of the square they belong to. If we parallel this part by

partitioning the particles, then locks will be needed because two particles held by two different threads may try to update the same linked list. And these locks will slow down the program significantly. So we partition the grid instead, which means each thread will loop through all particles but will only be responsible for updating part of the grid, and thus no lock is needed.

The second part is computing the particle interactions. We partition the grid in row-major way, and each thread will do the same computation as in serial implementation with (instead of the whole gird) a number of consecutive rows of grid assigned to it.

The third part is moving the particles. We simply distribute the particles to thread, because no interaction is involved in this part.

3 Experiments

3.1 Serial Implementation Performance

Result

Number of Particles	Runtime (s)
500	0.030393
1000	0.05855
1500	0.088162
2000	0.11771
3000	0.178473
4000	0.239568
6000	0.365514
8000	0.491129
10000	0.619183
20000	1.25964
40000	2.60111
80000	5.10243
160000	10.4678

Table 1: Performance of Serial Implementation

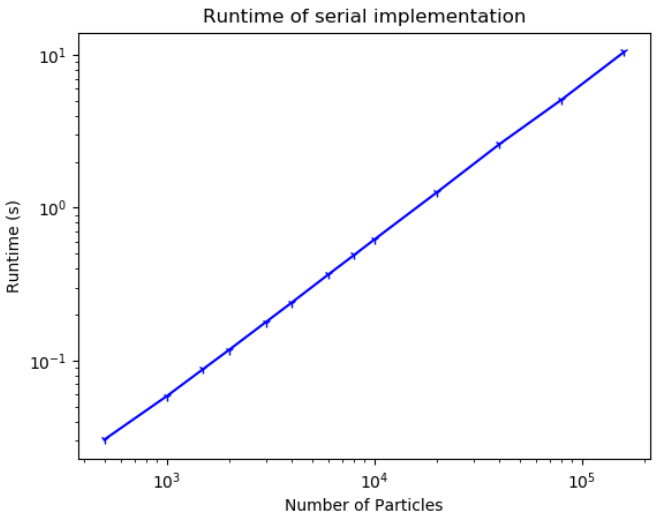


Figure 1: Running Time of Serial Algorithm

Analysis

In Plot 1 and Table 1, we can observe that the running time is linear with respect to the number of particles, which means the optimized serial implementation achieves $O(n)$ runtime compared to the $O(n^2)$ runtime of the original naive implementation.

3.2 Parallel Implementation Speedup, Scalability and Efficiency

Setup

Implementation: parallel
Number of threads: 1, 2, 4, 8, 16
Number of particles: 10000 (per thread for weak scaling)

Result

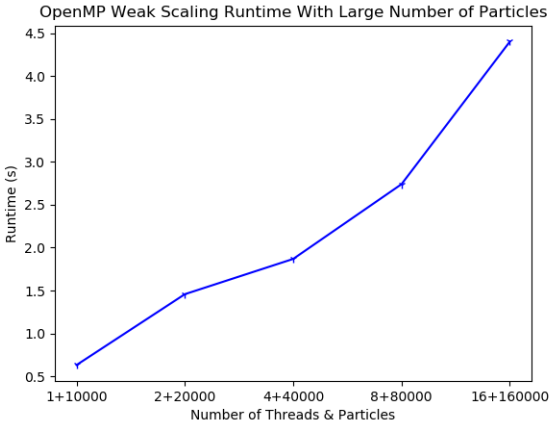


Figure 2: Weak Scaling Runtime

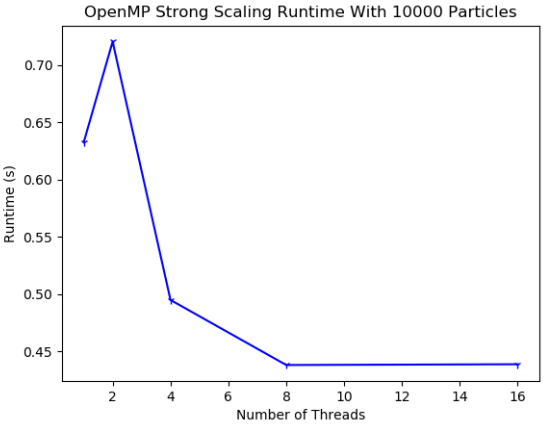


Figure 3: Strong Scaling Runtime

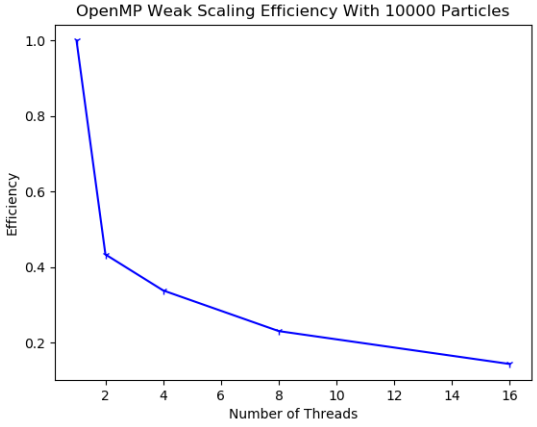


Figure 4: Weak Scaling Efficiency

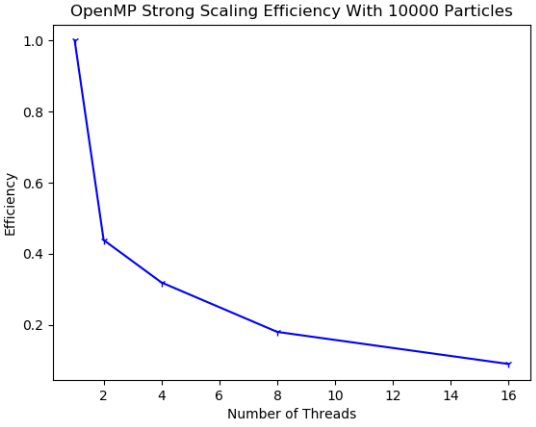


Figure 5: Strong Scaling Efficiency

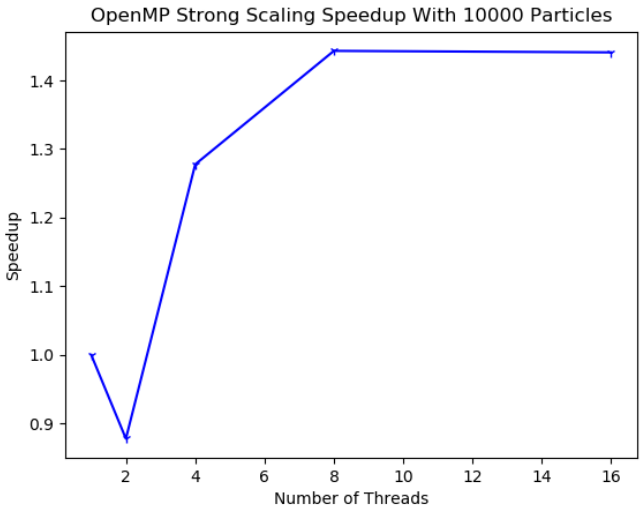


Figure 6: Strong Scaling Speedup

Strong Scaling	Weak Scaling
0.40577	0.4293

Table 2: Average Efficiency

Analysis

Our solution gets to 4.3 seconds for 160000 particles, which is quite impressive speed. We did several tricks to make the program efficient. In order to avoid the overhead of using locks, we make each thread independent by partitioning the grid, so that one square can only be updated by one thread. We also create one linked list node for each particle at the beginning, and keep reusing them to avoid repeating creation of linked list nodes. We do two-way interactions instead of one-way interactions to reduce the number of references.

The scalability, however, is not very ideal. We can see huge decrease in efficiency for both strong scaling and weak scaling efficiency, and the final speedup with 16 threads is about 1.5, which is far from the idealized p-times speedup. The re-computation of the grid in each iteration is part of the bottleneck. In order to save time and avoid the use of locks, each thread has to go over all the particles to update the grid, no matter how many threads there are. Another problem can be considered is load imbalance. Our strategy of calculating the interactions is by iterating through the small squares and checking the neighboring squares below and to the right. This means that squares in the last row will only need to check itself and the one on the right, which is much less work to do than squares in the middle.