CS267 HW2.1 Report

Kahraman Demir
Nicholas Tomlin
Du Xiang

14 February 2023

Overview

- Serial: 162s for 1M particles | Parallel: 17s for 1M particles
- Optimizations used: binning, grid size tuning, removing redundant force computations, parallelization with OpenMP `for`/`atomic` and `omp_set_lock`

Implementation Details

- Data structures and serial implementation:
  - Data structure for binning:
    We represent the particles in a grid using a 2D array of vectors, in which each vector contains a list of particles associated with a certain region of the space. Each bin thus is accessed by bin[i][j], which corresponds to row i and column j of the grids.
  - Bin size tuning:
    Ideally, to capture the force interactions, we only need to consider each particle with +- 2 cutoff distances around it. Therefore, we tuned our bin size to be around 2 * cutoff size to maximize computational efficiency.
  - looping optimization
    During each step, we traverse through each cell in the grid, and then compute (a) inter-cell particle interactions and (b) interactions with neighboring particles in the 8 grids around it. Later, we optimized the `apply_force` to calculate bi-directional forces to speed up the computation. Specifically, we only need to calculate the forces in half of the nearby grid cells, allowing us to avoid redundant computation. This helps to improve our serial runtime efficiency from O(9n) to roughly O(5n).
- Parallelization & Openmp:

– For loop parallelization:

  All for loops including acceleration zeroing, particle moving, and particle binning were paralellized using `pragma omp for`. However, for force applicaiton function, when looping over the grid sub-regions, `pragma omp for collapse(2)` was used since the outer most loops that iterate over grid sub-regions are independent of the inner operations. Collapsing these loops allow for the greater division of loop tasks amongst threads, especially in the case were the number of sub-regions along the length of the grid is much smaller than the thread count.

– Combine steps & Shared memory synchronization:

  Instead of clearing all bins and doing rebinning every step, we combined particle binning with `move` to reduce the number of total steps for moving particles. In this case, each thread is only moving the particles if necessary. Doing so would require adding appropriate locks to prevent data races and ensure synchronization:

  A 2D lock array was used to ensure particle binning at each time-step was done with exclusive access to each bin by each thread. Atomic operations `pragma omp atomic` were used in the evaluation of the accelerations of the particles instead of using locks to avoid the further performance overhead of another lock system for particles.

Findings & Results

We saw improvements each time through the steps from the above implementation details. After creating bins and tuning for bin sizes, we reduced the overall runtime from $O(n^2)$ to $O(n)$. Notably, we see that looping optimization of calculating 4 neighbors with bidirectional forces instead of 8 dramatically sped up the runtime for both the serial and parallel algorithms. Other than the major improvements, we noted that using atomics to replace locks and looping trategies to shed off some constants also improved our runtime by some degrees.

We show roughly $O(n)$ scaling in Figure 1 for both the serial and parallel implementations. For very small numbers of particles (e.g., less than 10000), we find that the parallel code is actually slower, most likely due to the overhead of initialization. We also compare our parallel implementation to an idealized model which scales linearly with the number of cores in Figure 2. Although our model does become more efficient with additional cores, it does not quite approach the idealized model when run with 1M particles.

Parallel time analysis: since the total amount of work is the same between serial and parallel, then the computation time for parallel should be Runtime of Serial / P where P is the number of threads in parallel. And the synchronization/communication time can be calculated as $T_{syn} = T_{total} - T_{comp} =$

$T_{total} - T_{serial}/N_{threads}$. However, this is when we assume all the parts of serial work can be paralleled. In fact, this is generally not true, so we actually need to break down $T_{serial} = T_{parallizable} + T_{nonparallelizable}$.

The final parallel runtime can be broken down as:

$$T_{compute} = (T_{serial} - T_{nonparallizable})/N_{threads} + T_{nonparallizable}$$

$$T_{sync} = T_{openmp} - T_{compute}$$

Number of Particles	Serial Time	Parallel Time
1,000	0.0323972	0.134295
10,000	0.481293	0.550853
100,000	5.28225	2.04552
1,000,000	162.06	17.2663

Table 1: Results for plotting the serial vs. parallel particle simulation.

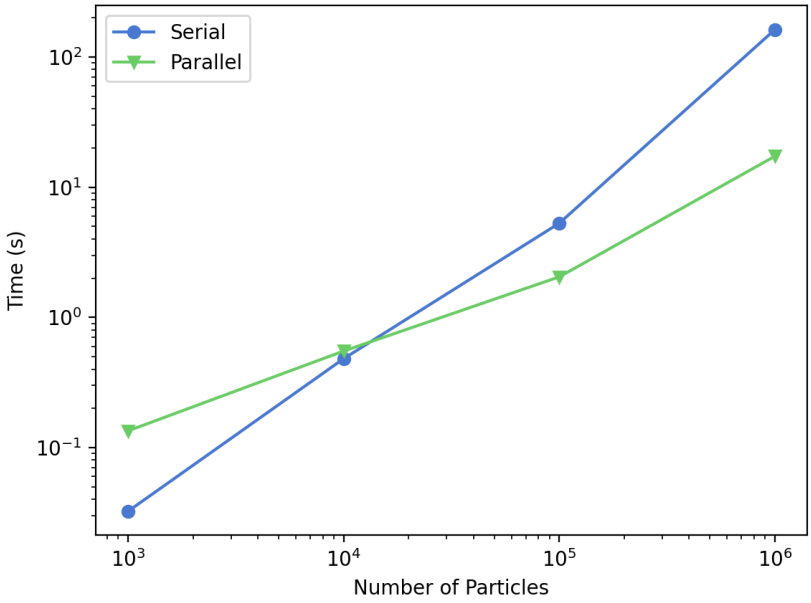

Figure 1: Time to run serial and parallel particle simulation (in log-log scale).

Number of Threads	Parallel Time	Ideal Time
1	148.079	148.079
2	99.8918	148.079 / 2
4	63.2817	148.079 / 4
8	33.5096	148.079 / 8
16	21.0827	148.079 / 16
32	18.9846	148.079 / 32
64	16.2584	148.079 / 64

Table 2: Results for plotting the ideal parallel runtime vs. actual runtime for 1M particles.

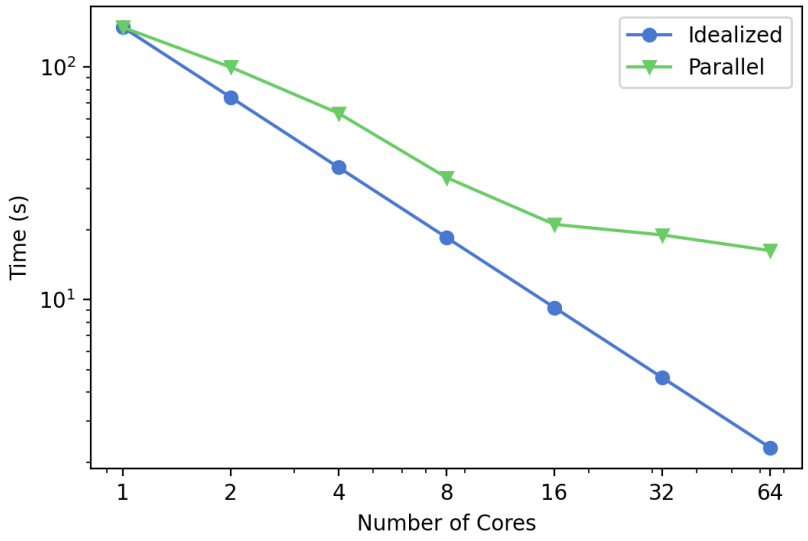

Figure 2: Improvement in parallel simulation based on number of cores compared to an idealized model which scales linearly with the number of cores. All experiments run with 1M particles.

Future Improvements

Although our serial implementation was optimized greatly, our parallel/openmp implementation didn't scale as well as expected (shown in graphs). This is mainly due to some parts of our code were not paralleled or memory-optimized. These includes possible improvements over initial binning and better memory allocation for our bins. For example, we could have used a flat structure to store our bins or memory reservation, thereby resulting in better memory locality and

usages. In addition, shared memory allocation could be better by using more fined-grained locks that applies specifically to each thread. Right now, we are using locks generically for each bin where some threads are acquiring locks that not needed. Lastly, we could use more customized parallel sturcture such as static/manual allocation instead of `pragma parallel for`, possibly allowing for less need for memory synchronization.

Collaboration Statement

- Kahraman Demir: Worked on tuning grid size, tinkering with OpenMP implementation, and identifying further code optimizations. Efforts did not end up in submitted code. Contributed to report.
- Nicholas Tomlin: built initial $O(n)$ serial implementation with particle binning, ran scaling and idealized parallel model experiments and generated figures for report, and also wrote initial draft of the report.
- Du Xiang: Improve upon Nicholas's serial implementation: Tuning grid sizes, data structures, looping optimization to achieve 162s/1M. Implemented openmp to achieve 17s for 1M particles. Collaborated with teammates on writing the report.