CS267 Homework 2 (Part 3)

Parallelizing a Particle Simulation

Team members: Tzu-Chuan Lin, Chin-An Chen, Byron Hsu

Abstract

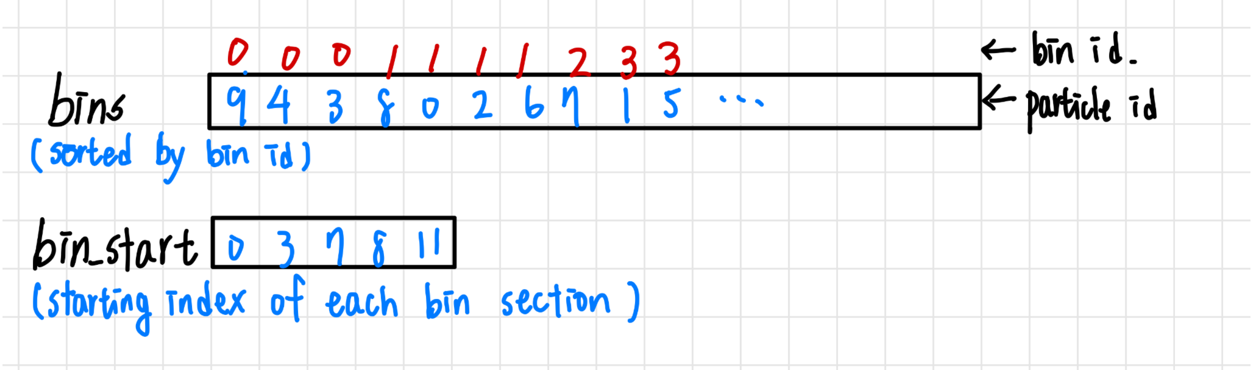
In this assignment, we will be parallelizing a toy particle simulation (similar simulations are used in mechanics, biology, and astronomy) with GPU CUDA. In our simulation, particles interact by repelling one another but only when closer than a cutoff distance.

Suppose we have a code that runs in time T = O(n) on a single processor. Then we'd hope to run close to time T/p when using p processors. You will attempt to reach this speed up with a GPU. As the result, we got the best performance on GPU in around **1.6 seconds for 1M particles.**

Implementation

As the methods recommended by the GSI, instead of using std::vector, which would cost lots of copies from host to GPU per step, we used arrays.

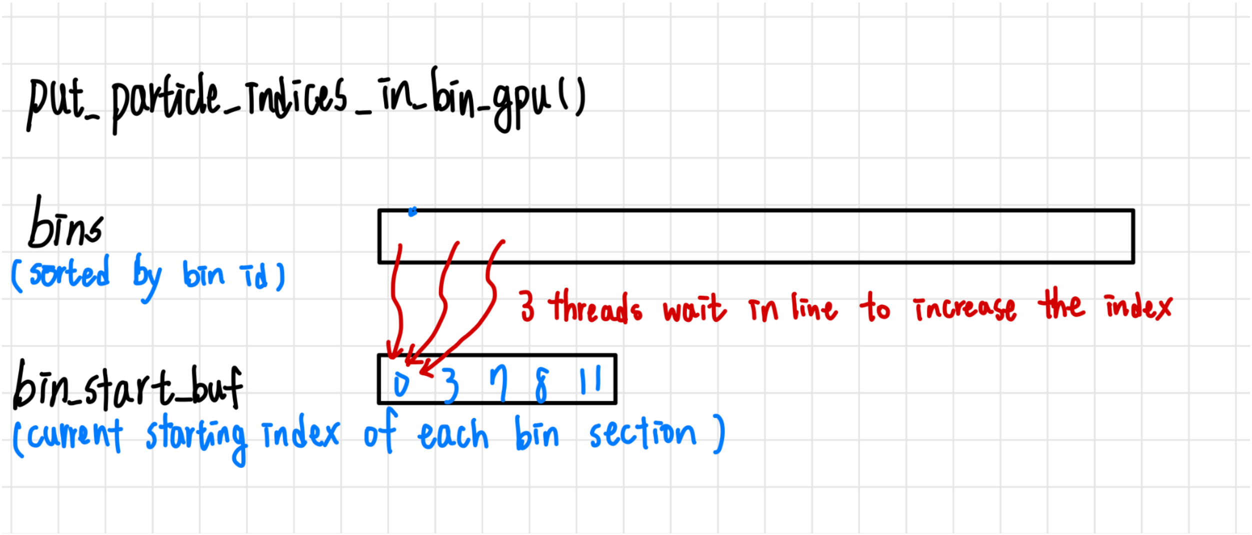
There are arrays as below we used in this assignment.



- particle_t* parts_gpu: the array of given particles (size == #of particles)
- int* bins: array of particles sorted by bin id (size == #of particles)
- int* bin_start: Prefix sum of the bin counts (size == #of bins)
- int* bin_start_buf: instead of literally sorting algorithm to update the array bins, we used bin_start_buf to keep track of the indices on array bins we are going to modify.

Dataflow:

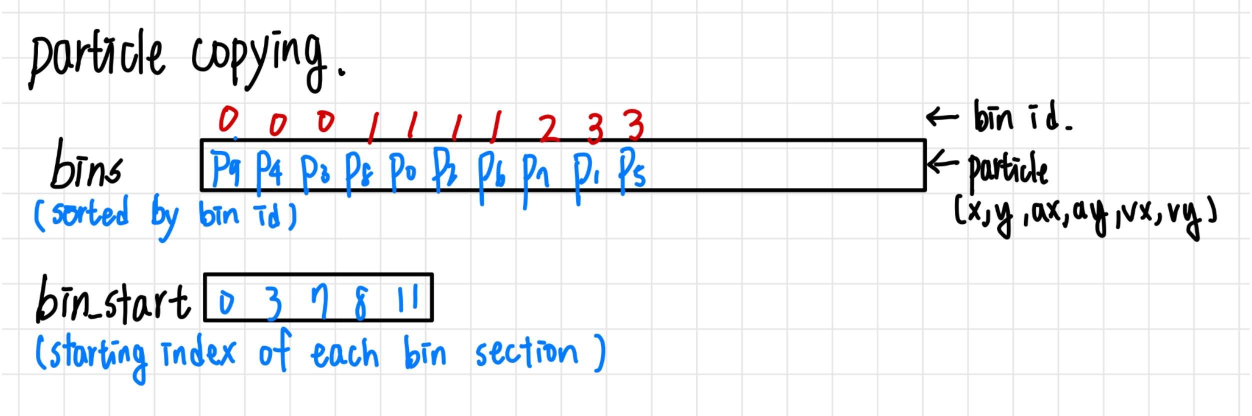
- init_simulation(): memory allocation on GPU for arrays bins, bin_start, and bin_start_buf
- simulation_one_step():
  - count_each_bin_gpu() and inclusive_scan(). Calculate the prefix sum for each bin, and record the result in array bin_start.


put_particle_indices_in_bin_gpu()
bins
(sorted by bin id)
3 threads wait in line to increase the index
bin_start_buf
0 3 7 8 11
(current starting index of each bin section)

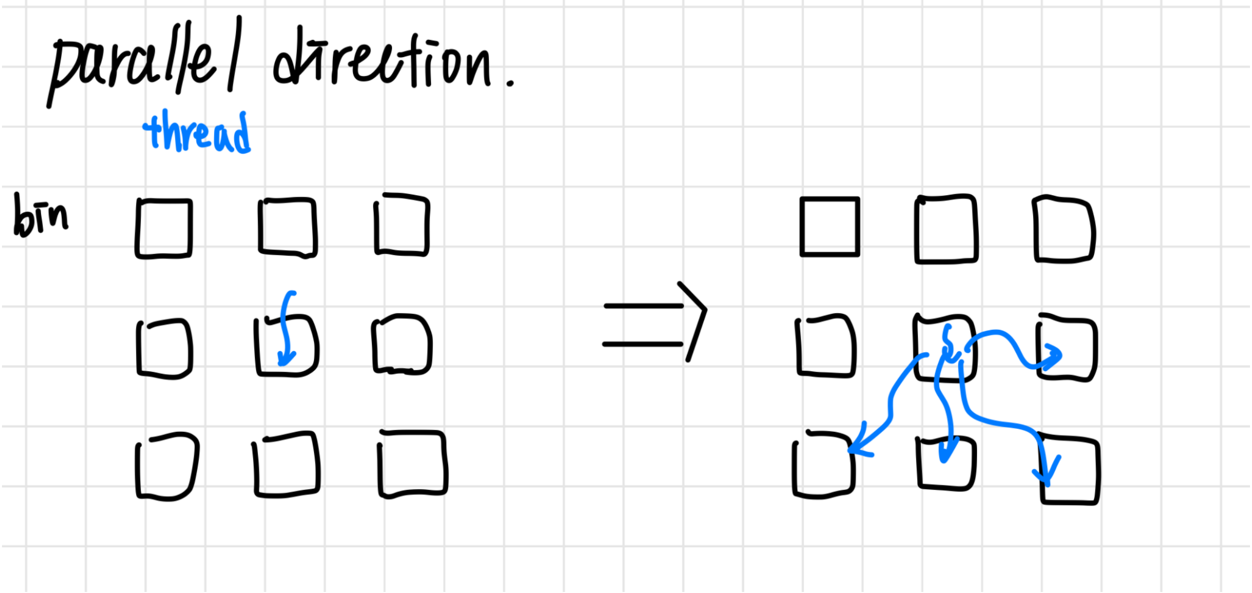
  - put_particle_indices_in_bin_gpu(): Sort the particles with their bin indices and store them in an array bin.
  - clear_ax_ay(): Initialization of the acceleration of each particle
  - compute_forces_gpu(): Calculation of the force between particles and their acceleration.
  - move_gpu(): Move the particles based on the acceleration we got in compute_forces_gpu.

Experiments

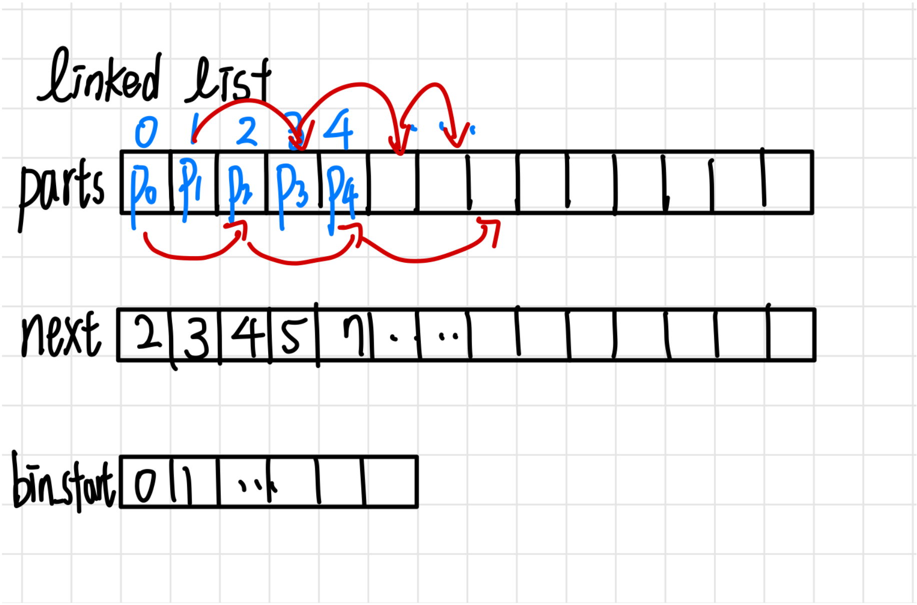
1. One direction force:
   Calculate the force for the current particle only; hence, we **do not need atomic add** when updating the acceleration.
2. Bi-direction force:
   Calculate the forces for the current particle and the particle paired with it; hence, we need **atomic add** when updating the acceleration
3. Bi-direction force calculation with particle copying:
   Not only storing the particle indices in the array bin, but we also stored the position, velocity, and acceleration of the particle in the array bin.



4. Bi-direction force and parallel directions:
   Because when we calculate the forces bi-directionally, we have to go through five bins (current, right, bottom left, bottom, bottom right), we then modified the algorithm to calculate the forces **all by once in parallell.**



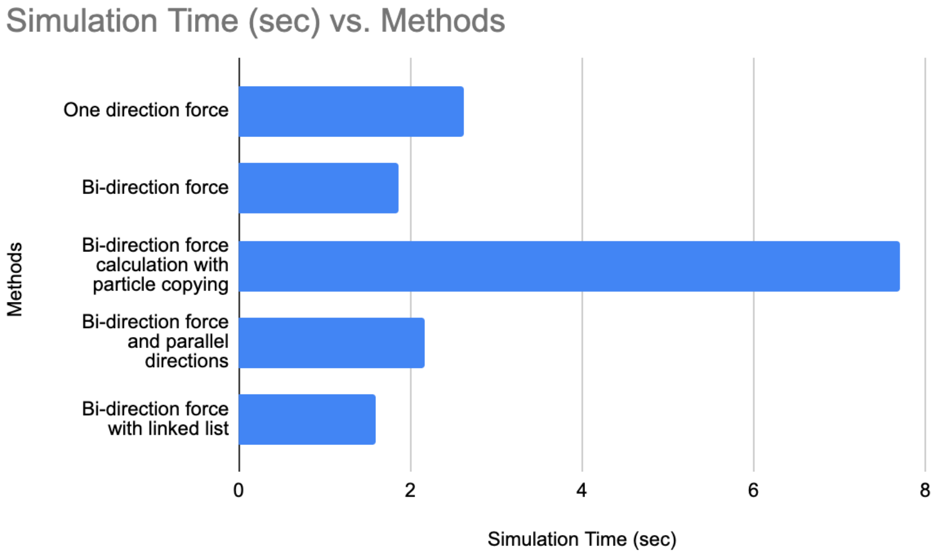
5. Bi-direction force with **a linked list for each bin**:
   We used the array **next** to record the next particle of the corresponding particle in the array gpu_parts. When constructing the next array, we traversed the gpu_parts and logically added the particle at the end of the list (bin).



Design choice and the effects

Tested on 1M particles

Method	1	2	3	4	5
Simulation time (sec)	2.62339	1.86601	7.71509	2.17554	1.59818

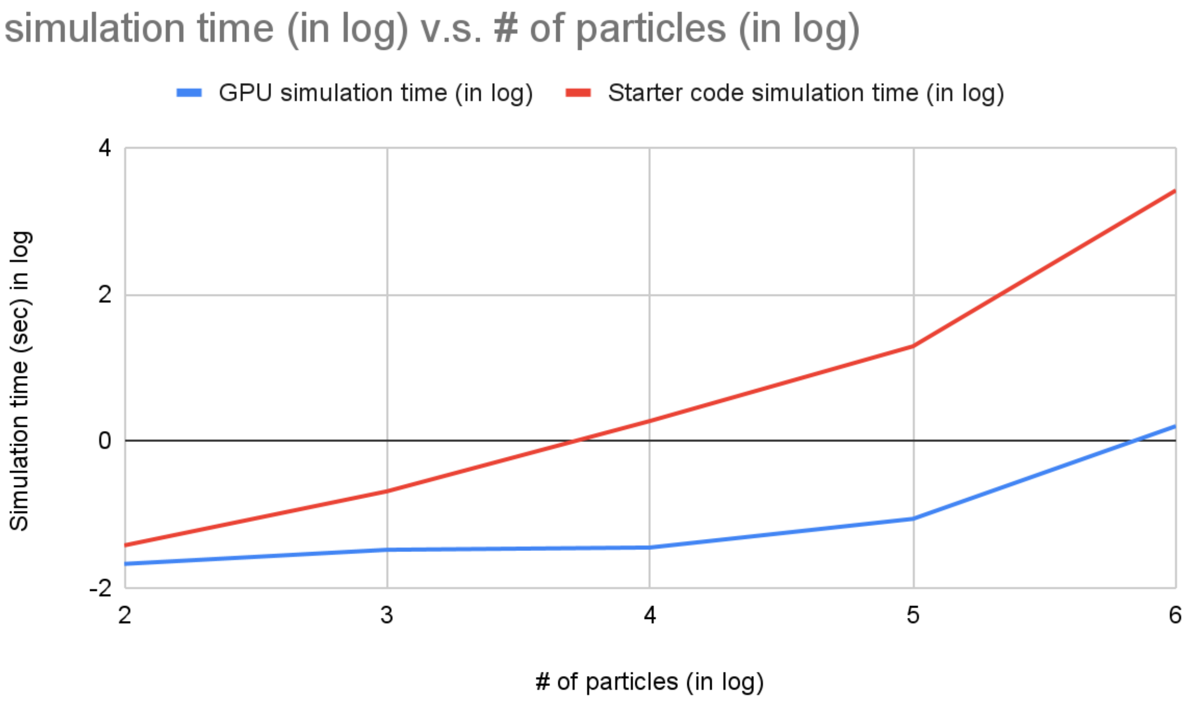


Performances

Log-log (Time versus particles)

Tested with method 5 (bi-direction and linked list)

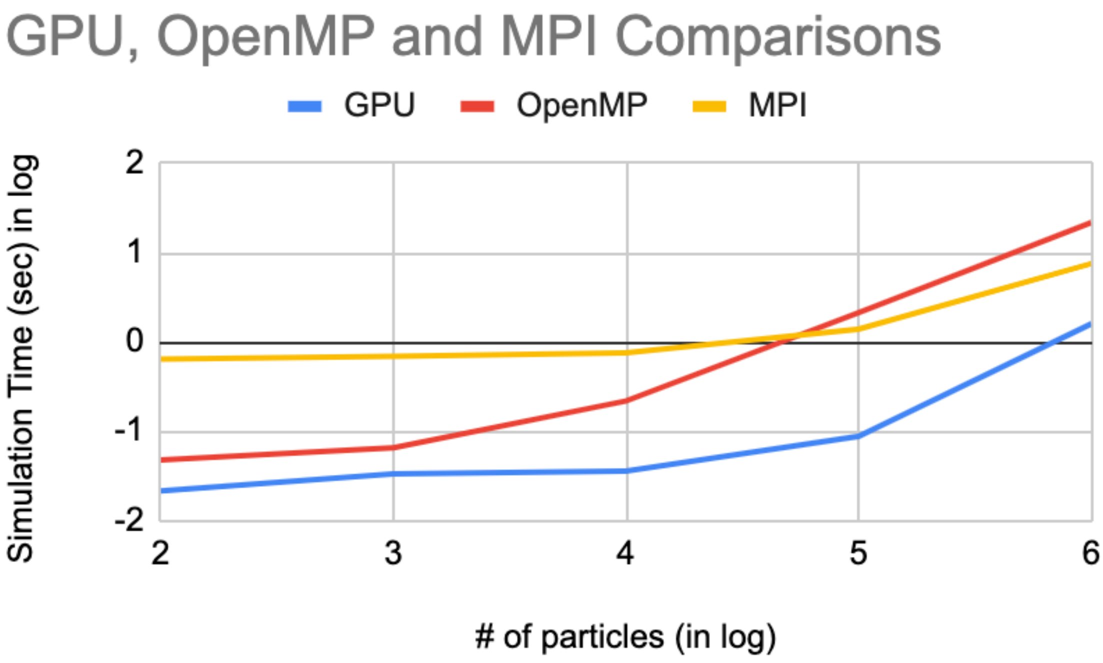
# of particles	100	1000	10000	100000	1000000
Starter-code on GPU (O(n^2))	0.0379855	0.207346	1.87929	19.531	2593.46
GPU	0.0212772	0.033097	0.0355539	0.0872949	1.59818



Parallizing using GPU versus OpenMP and MPI

MPI: 2 node 68 tasks per node
OpenMP: 1 node 68 cores 1 thread per core

	100	1000	10000	100000	1000000
GPU	0.0212772	0.033097	0.0355539	0.0872949	1.59818
OpenMP	0.0475355	0.0648152	0.217363	2.12787	21.8643
MPI	0.640243	0.692009	0.755319	1.39015	7.5557



Breaking down the runtime

- Computation time: slightly in proportion to the size of particles
- Synchronization time: atomic / cudaDeviceSynchronize
- Communication time: cudaMemcpy

Contribution among the team

Tzu-Chuan: Implementation
Chin-An: Text of the report
Byron: Illustration of the report