CS3210 Parallel Computing
Assignment 2 Report

Discrete Particle Simulation with MPI

Keven Loo Yuquan (A0183383Y) & Lee Yong Jie, Richard (A0170235N)

Contents

1 MPI Program Design

The discrete particle simulation is implemented fully in MPI. The overall architecture of the simulator is summarised in Figure 1 below.

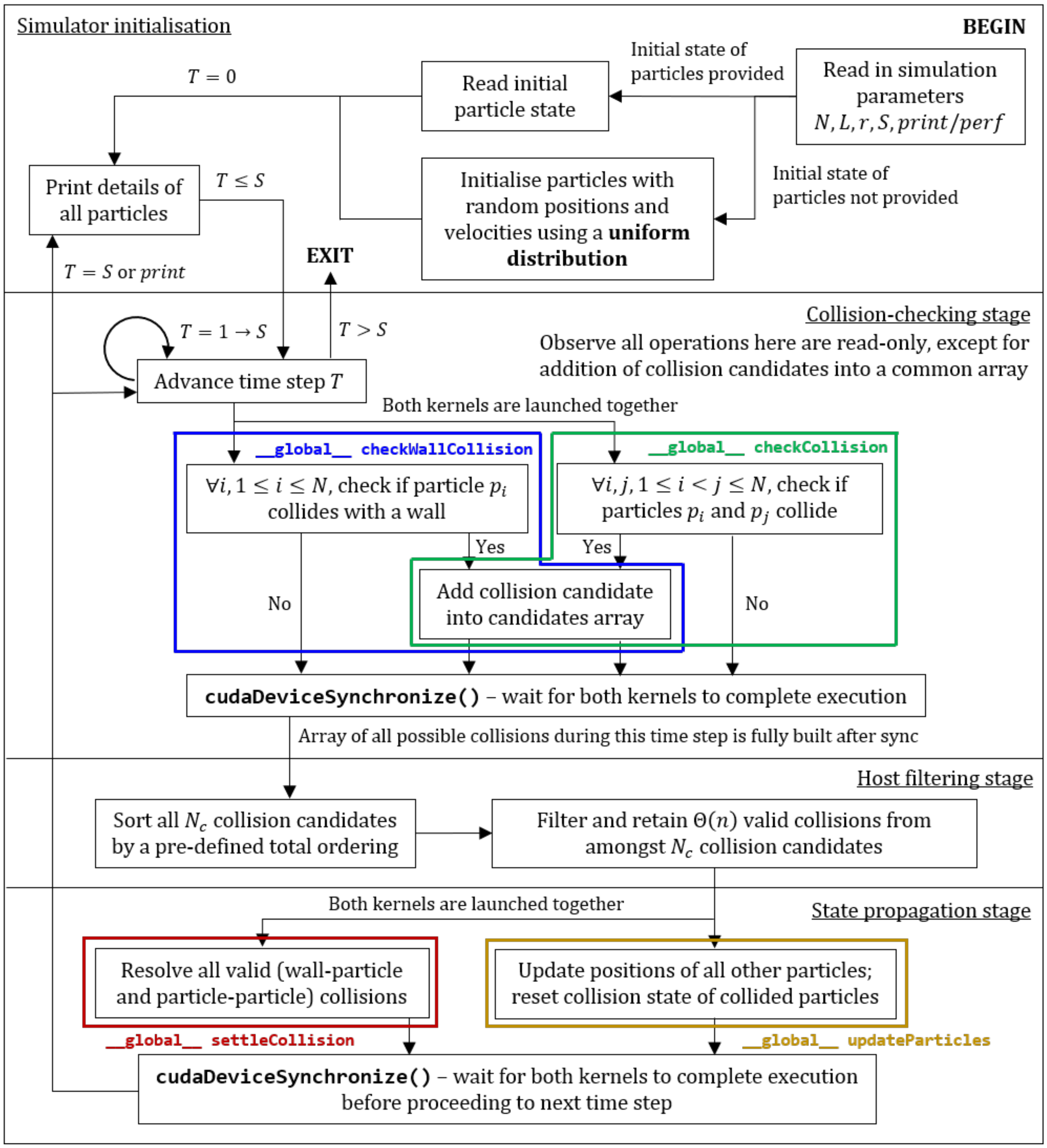
Simulator initialisation
BEGIN
Initial state of particles provided
Read in simulation parameters $N, L, r, S, print/perf$
Read initial particle state
$T = 0$
Initial state of particles not provided
Initialise particles with random positions and velocities using a **uniform distribution**
Print details of all particles
$T \leq S$
EXIT
$T = S$ or $print$
Collision-checking stage
Observe all operations here are read-only, except for addition of collision candidates into a common array
$T = 1 \rightarrow S$
$T > S$
Advance time step T
Both kernels are launched together
__global__ checkWallCollision
__global__ checkCollision
$\forall i, 1 \leq i \leq N$, check if particle p_i collides with a wall
$\forall i, j, 1 \leq i < j \leq N$, check if particles p_i and p_j collide
Yes
Yes
No
No
Add collision candidate into candidates array
cudaDeviceSynchronize() – wait for both kernels to complete execution
Array of all possible collisions during this time step is fully built after sync
Host filtering stage
Sort all N_c collision candidates by a pre-defined total ordering
Filter and retain $\Theta(n)$ valid collisions from amongst N_c collision candidates
Both kernels are launched together
State propagation stage
Resolve all valid (wall-particle and particle-particle) collisions
Update positions of all other particles; reset collision state of collided particles
__global__ settleCollision
__global__ updateParticles
cudaDeviceSynchronize() – wait for both kernels to complete execution before proceeding to next time step

Figure 1: Overall simulator design for CUDA implementation

2 Implementation Assumptions and Details

2.1 Assumptions

The first assumption from the earlier report has been removed due to changes in the testing conditions.

1. A particle is involved in no more than one collision per time step.
   - Implications
     (a) If a particle collides with a wall after any collision, it is placed at the wall at the end of that time step, and will immediately collide with the wall at the beginning of the next time step
     (b) If particle P collides with a particle Q after any collision, it will phase through the particle Q for the remainder of this time step, or will ignore Q the next time step if they happen to overlap
2. All collisions between particles and with the wall are elastic (kinetic energy and momentum are conserved).
3. It is possible to fit all N particles of radius r into the square of length L without overlapping.
   - Implication
     (a) The simulation exits if one of these two conditions fail: $L < 2r$ (not possible to fit a single particle) or $Nr^2 > L^2$ (not possible to pack N particles in a grid arrangement in a square with area L^2)
4. The set of possible collisions C satisfies a **total ordering**.
   - Implication
     (a) For each time step T, it is possible to sort all collision candidates by this total ordering to determine which collisions should be prioritised over others.

2.2 Implementation Details

Changes in general implementation details from the previous report are in blue.

1. If the initial state of particles are not provided, particles are generated with initial random positions and velocities using a **uniform distribution**.
   (a) We use the pseudo-random number generator function `rand` in the C library seeded with the number 3210.
   (b) Particles are placed randomly in the square without overlapping.
2. Each particle keeps track of its own state: $\text{ID}, x, y, v_x, v_y, w_c, p_c$.
3. Our simulation has an additional parameter `SLOW_FACTOR` in `simulator.cu` that increases the granularity of the simulation for greater accuracy.
   (a) Setting `SLOW_FACTOR` to an integer > 1 slows the initial velocities of all particles by that factor. `SLOW_FACTOR` should be set to a power of two to avoid introducing additional floating-point errors when dividing the particles velocities.
   (b) The number of steps of the simulation is multiplied by `SLOW_FACTOR` to compensate, i.e. each original step now corresponds to `SLOW_FACTOR` "micro-steps".
4. Collisions are of two types: particle-wall collisions or particle-particle collisions. We describe a particle-wall collision as (P, `null`) and a particle-particle collision as (P, Q).
   (a) As particle-particle collisions are symmetric (i.e. P collides with $Q \iff Q$ collides with P), we generate these collisions such that P is the particle with lower integer ID to avoid duplicates.
5. When sorting collision candidates with the C library function `qsort`, we enforce this total ordering in the function `cmpCollision`. For two collision candidates C_1 and C_2,
   - $C_1 < C_2$ if C_1 occurs before C_2 (priority by time)
   - If C_1 and C_2 occur at the same time
     - $C_1 < C_2$ if ID of P in $C_1 <$ ID of P in C_2 (priority by ID)
     - If C_1 and C_2 both involve the same particle P
       - $C_1 < C_2$ if C_1 is a wall collision (priority by type)

- ■ If C_1 and C_2 are both particle-particle collisions
  - □ $C_1 < C_2$ if ID of Q in C_1 < ID of Q in C_2 (priority by ID)

6. For each particle in each time step, we check once if the particle collides with any of the four walls.

   (a) A particle is treated to have collided with a wall if it would come within a distance of $\epsilon =$ `1E-8` to the wall within that time step.

7. All possible particle-particle collision pairs are checked for each time step. The total number of collision checks performed is thus

$$\begin{aligned} N_{potential\ collisions} &= N_{wall-particle} + N_{particle-particle} \\ &= N + \frac{N(N-1)}{2} \\ &= \Theta(N^2) \end{aligned}$$

8. Particle-wall collisions are checked by solving the trajectory equation of a particle and the position equations of the wall. The equations of the walls are $x = 0, x = L, y = 0, y = L$.

9. Particle-particle collisions are checked by solving trajectory equations of two particles during the given time step.

   Consider two particles P, Q during a given time step $0 \le \Delta t \le 1$. From Pythagoras theorem, the distance between them is

$$d = \sqrt{((x_Q + v_{xQ}\Delta t) - (x_p + v_{xP}\Delta t))^2 + ((y_Q + v_{yQ}\Delta t) - (y_p + v_{yP}\Delta t))^2}$$

   or re-written in terms of deltas (differences in state)

$$d = \sqrt{(\Delta x + \Delta v_x \Delta t)^2 + (\Delta y + \Delta v_y \Delta t)^2}$$

   The particles intersect when $d = 2r$, i.e. the particles touch at their circumference, hence by expanding and collecting terms we get the quadratic equation of form $A(\Delta t)^2 + B\Delta t + C = 0$,

$$\begin{aligned} &\Delta x^2 + 2\Delta x \Delta v_x \Delta t + \Delta v_x^2 (\Delta t)^2 + \Delta y^2 + 2\Delta y \Delta v_y \Delta t + \Delta v_y^2 (\Delta t)^2 = (2r)^2 \\ \implies &(\Delta v_x^2 + \Delta v_y^2)(\Delta t)^2 + (2\Delta x \Delta v_x + 2\Delta y \Delta v_y)\Delta t + (\Delta x^2 + \Delta y^2 - 4r^2) = 0 \end{aligned}$$

We observe that $A = (\Delta v_x^2 + \Delta v_y^2) > 0$ and thus the curve $y = d(\Delta t)$ is concave up. The discriminant for this quadratic equation, $B^2 - 4AC$, is

$$\text{discriminant} = (2\Delta x \Delta v_x + 2\Delta y \Delta v_y)^2 - 4(\Delta v_x^2 + \Delta v_y^2)(\Delta x^2 + \Delta y^2 - 4r^2)$$

If this discriminant is ≥ 0, then the particles collide for some value of Δt, and we solve for this Δt. There are two possible roots,

$$\Delta t = \frac{-B \pm \sqrt{\text{discriminant}}}{2A}$$

Since the quadratic curve is concave up, we only compute and examine the first root (when the particles are approaching each other). This is

$$\Delta t = \frac{-B - \sqrt{\text{discriminant}}}{2A}$$

If $0 \leq \Delta t \leq 1$, then particles P, Q collide during this time step.

Note that, it is possible for $\Delta t < 0$ - this corresponds to cases where particles are overlapping from a previous step. We do not consider these as collisions in the current step and let these two particles phase through each other.

10. To ensure the collision candidates array can accommodate $\Theta(N^2)$ potential collisions (in the limit of large N when particle-particle collisions dominate), we dynamically allocate an array in managed memory with sufficient space to store $N^2/2$ collision structs.

    - Unfortunately, `cudaMallocManaged()` fails to allocate the required space when $N = 64000$.

3 Assignment 1B (CUDA) addendum: Float performance on the NVIDIA Titan V GPU

Some words here.

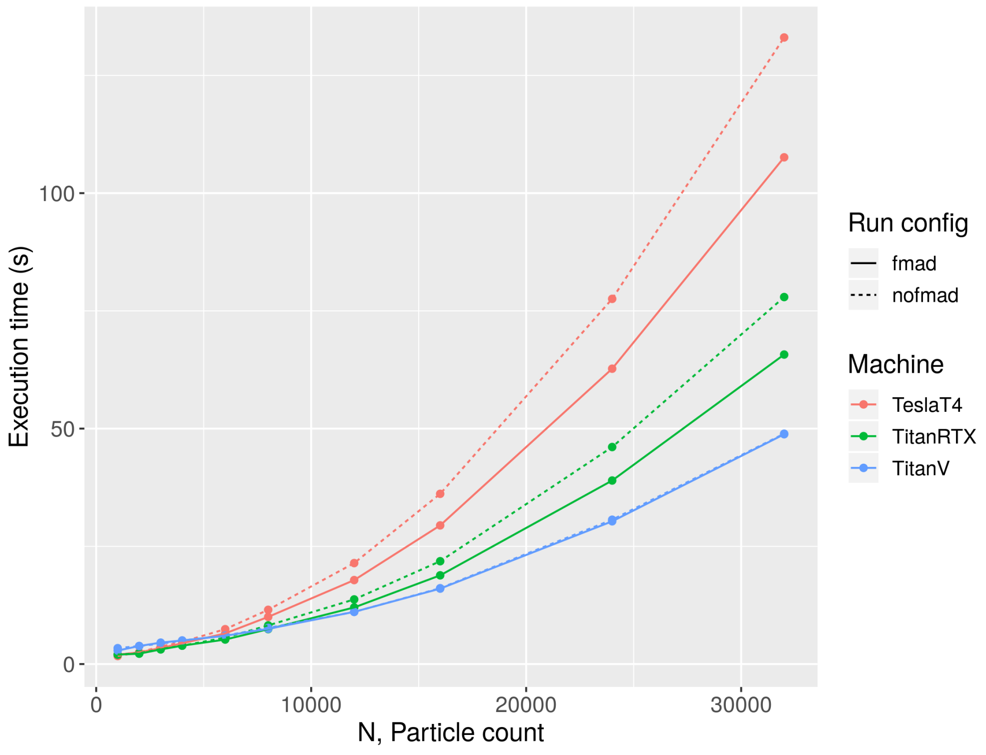

Figure 2: Plot of execution time against particle count, N

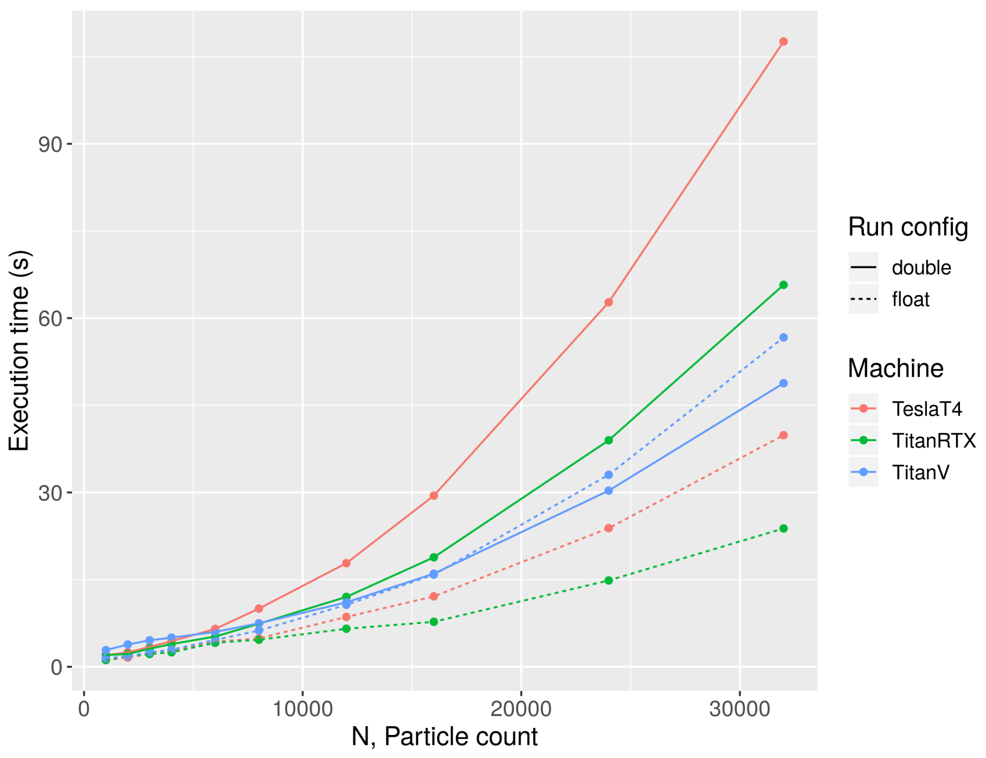

Figure 3: Plot of runtime (s) against N for `float` and `double` programs (`--fmad=true`)

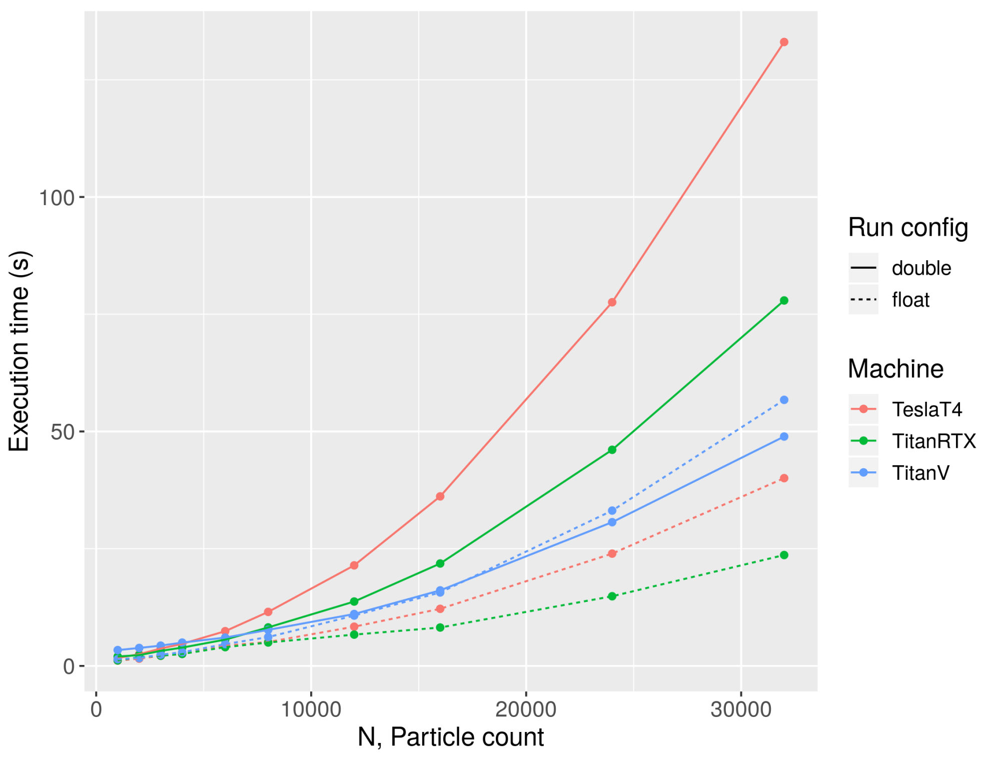

Figure 4: Plot of runtime (s) against N for `float` and `double` programs (`--fmad=false`)

4 Filler section

We cite our previous assignments, 1A [1] and 1B [2] for no apparent reason. Who knows, we might reference them soon.

5 References

[1] K. Loo and R. Lee. CS3210 Assignment 1 - Particle Movement Simulator (Part 1), 2019.

[2] K. Loo and R. Lee. CS3210 Assignment 1 (Part 2), Discrete Particle Simulation with CUDA, 2019.