Assignment 1
Particles Movement Simulator using Parallel Programming

CS3210 – 2019/20 Semester 1

30 Sep 2019: Clarification and corrections shown in red.

Learning Outcomes

This assignment is designed to enhance your understanding of parallel programming with shared-memory (OpenMP) and GPU programming using CUDA. You will apply parallelization models you learned in class to solve a real world problem.

Problem Scenario

Simulating the movement of particles is one of the large-scale problems of current interest in climatology, plasma physics, fluid dynamics and celestial mechanics. You are tasked to implement and parallelize a simulator of particles in a 2D space.

Figure 1 shows a 2D space that contains particles. The particles are located on a square surface of a given size. They are all assumed to have the same mass and size (radius r), and they move with given velocities. The position is defined using (x, y) coordinates of the center of the particle, expressed from the left lower corner of the square. The particles have collisions with other particles and with the walls of the surface. The collisions are elastic.

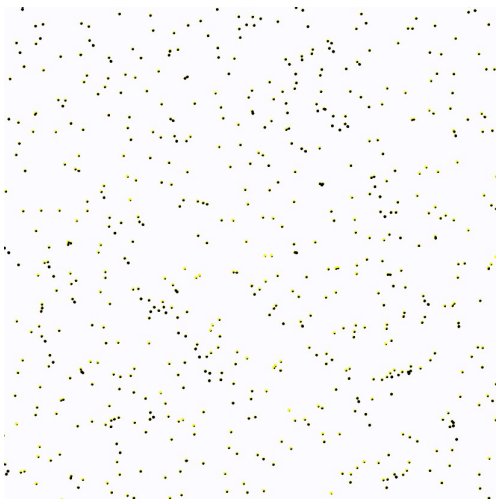

Figure 1: Particles on a square surface

Your task is to write parallel programs in (i) OpenMP and (ii) CUDA to simulate this movement of particles in the square. The simulation runs for given time, in steps. You will need to provide two modes of running this simulation:

- Correctness mode: at every step, you need to output the the position of every particle.
- Speed mode: you need to output the position of every particle before the first step and after the last step.

Given an initial position of the particles, the final position should be the same for both modes.

The collisions with the walls and other particles are modeled as elastic collisions where the momentum and kinetic energy are conserved (details to be found in the next sections).

One collision per step for each particle is considered. The collisions with the walls and other particles may happen during every step (time unit) of the simulation. As a simplification, each particle is involved in at most one collision during each step (all following collisions can be ignored). There can be multiple collisions happening during the same step, but no one particle is involved in more than one collision. You need to use the velocities from before and after the collision to compute the final position of the particle after each time step. In general, the step is fine enough (particles are slow enough and not too many) such that there would be less than one collision for each particle during each step. However, the collisions might happen at any time during a step.

Only the earliest collisions during a step is considered. If there are multiple possible collisions with the wall or other particles during the same step, only the earliest collision will be considered (and others ignored). When collisions between two particles are ignored it means that they move beyond each other without their velocity and direction changing. They might overlap at the end of the step. In that case, the collision is computed in the next step from the overlapped positions. If a particle needs to collide with a wall as its second collision in a step, the particle stops just next to the wall (it will collide in the next step).

Multiple simultaneous collisions during a step. In the unlikely case that a particle is involved in two collisions exactly at the same time (in the step), the particles with the lowest index collide, and the other collisions are ignored. If a particle is involved in a collision with the wall and with another particle exactly at the same time (in the step), the particle collides with the wall, and the other collisions are ignored.

This assignment is divided into two parts:
(i) OpenMP Implementation is Assignment 1 Part 1 due on 30 Sep, 11am
(ii) CUDA Implementation is Assignment 1 Part 2 due on 21 Oct, 11am

Inputs and Outputs

Input file strictly follows the structure shown below:

1. N – Number of particles on the square surface
2. L – Size of the square (in μm)
3. r – Radius of the particle (in μm)
4. S – Number of steps ($time_{units}$) to run the simulation for
5. $print$ or $perf$– If word $print$ appears, the position of each particle needs to be printed at each step. Otherwise $perf$ should appear.
6. Optional: For each particle, show on one line the following information, separated by space:
   - i – the index of the particle from 0 to $N-1$
   - x – intial position of particle index i on x axis (in μm)
   - y – intial position of particle index i on y axis (in μm)
   - v_x – initial velocity on the x axis of particle i (in $\mu m/time_{unit}$)
   - v_y – initial velocity on the y axis of particle i (in $\mu m/time_{unit}$)

If the initial location of the particles is not provided, you need to generate random positions and velocities for all particles. The positions should be values within the 2D surface $L \times L$, while velocities should be in

the interval $\frac{L}{4}$ and $\frac{L}{8r}$. To show the direction of movement, velocities have positive and negative values. The particle i, the velocity v is $\sqrt{v_x^2 + v_y^2}$ with and angle α with x-axis with $\tan\alpha = \frac{v_y}{v_x}$. The velocity vector is $\vec{v} = \langle v_x, v_y \rangle$.

Sample Input

```
1000
20000
1
1000
print
```

Output file strictly follows the structure shown below:

1. Print the positions and velocities of each particle in the beginning of the simulation. For each particle, show on one line the following information, separated by space:
   - 0 – step 0 in the simulation
   - i – the index of the particle
   - x – initial position of particle index i on x axis (in μm)
   - y – initial position of particle index i on y axis (in μm)
   - v_x – initial velocity on the x axis of particle i (in $\mu m/time_{unit}$)
   - v_y – initial velocity on the y axis of particle i (in $\mu m/time_{unit}$)
2. If $print$ is used in the input file, at each step (time unit) t_u, your program should output the positions and velocities of each particle. The print should be done after the particle movement is computed for that step. For each particle, show on one line the following information, separated by space:
   - t_u – the step in the simulation
   - i – the index of the particle
   - x – position of particle index i on x axis (in μm) at time t_u
   - y – position of particle index i on y axis (in μm) at time t_u
   - v_x – velocity on the x axis of particle i (in $\mu m/time_{unit}$) at time t_u
   - v_y – velocity on the y axis of particle i (in $\mu m/time_{unit}$) at time t_u
3. At the end of the simulation, for each particle show on one line the following information, separated by space:
   - S – last step in the simulation
   - i – the index of the particle
   - x – final position of particle index i on x axis (in μm)
   - y – final position of particle index i on y axis (in μm)
   - v_x – final velocity on the x axis of particle i (in $\mu m/time_{unit}$)
   - v_y – final velocity on the y axis of particle i (in $\mu m/time_{unit}$)
   - $p_{collisions}$ – total number of collisions with other particles
   - $w_{collisions}$ – total number of collisions with the wall

To avoid the errors associated with the floating pointing computations, use double precision floating point operations (double) for x, y, v_x, v_y. However, when you print the values to the file show only the first 8 digits after the decimal point (for example, use $10.8f$ for printf).

The Physics Engine

The collisions with the walls and other particles are modeled as elastic collisions where the momentum and kinetic energy are conserved.

When the particles collide with the walls of the square they reflect with the same velocity, in a different direction. When a particle collides with the horizontal wall, the velocity after the impact are modeled using the following vector: $\vec{v\prime} = \langle v_x, -v_y \rangle$. When a particle collides with the vertical wall, the velocity after the impact are modeled using the following vector: $\vec{v\prime} = \langle -v_x, v_y \rangle$. When particle collides with the corner of the square, both velicities on the x and y-axis are reversed: $\vec{v\prime} = \langle -v_x, -v_y \rangle$

For a collison with another particle, you need to compute the new directions and velocities for particles according the laws of physiscs for 2D elastic collision for 2 particles with **equal masses**. For example, Figure 2 shows a collision between two particles with velocities v_1 and v_2. You can see the normal unit (connecting the centers of the two particles) and the tangent unit (perpendicular on the normal) vectors: $\vec{un}$ and $\vec{ut}$. The unit vectors are computed as follows: $\vec{un} = \langle x_2 - x_1, y_2 - y_1 \rangle$ and $\vec{un} = \frac{\vec{n}}{\sqrt{n_x^2 + n_y^2}}$ and $\vec{ut} = \langle -un_x, un_y \rangle$. The velocities can be decomposed along normal and tangential vectors, as shown in orange: $\vec{v} = \langle v_n, v_t \rangle$, where $v_n = \vec{un} \cdot \vec{v}$ and $v_t = \vec{ut} \cdot \vec{v}$

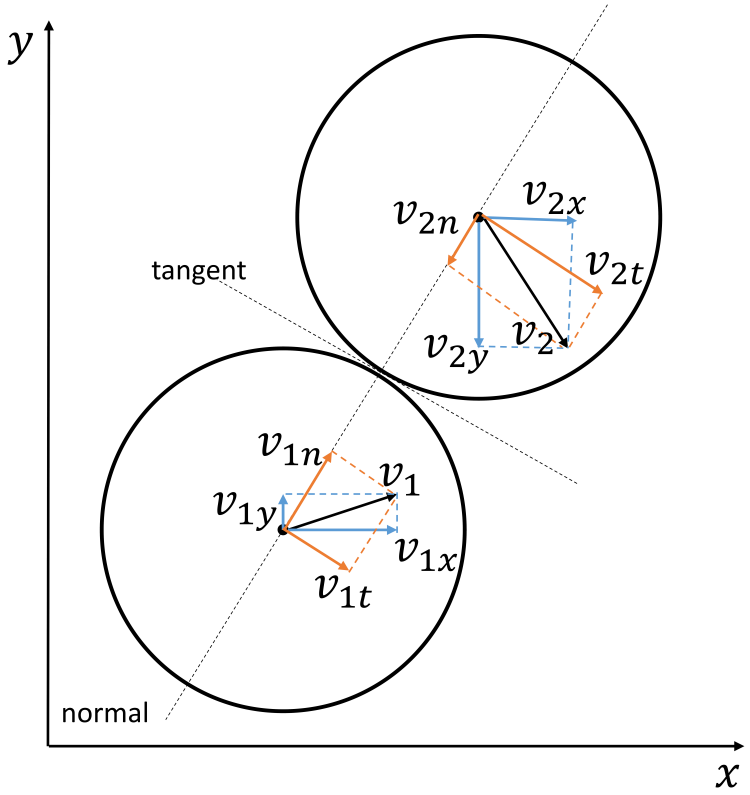

Figure 2: Before 2D collision of two particles

Intuitively, at collision, the new velocities change as if the particles would hit against an imaginary wall located along the tangential vector. Figure 3 shows the velocity for both particles, with their components along the normal and tangential axis, and x and y-axis.

The tangential components of the velocities do not change. Hence the velocities on the tangential vector after the collision are: $v_{1t}\prime = v_{1t}$ and $v_{2t}\prime = v_{2t}$
The normal components of the velocities after the collision are as follows (particles have equal masses): $v_{1n}\prime = -v_{2n}$ and $v_{2n}\prime = -v_{1n}$.

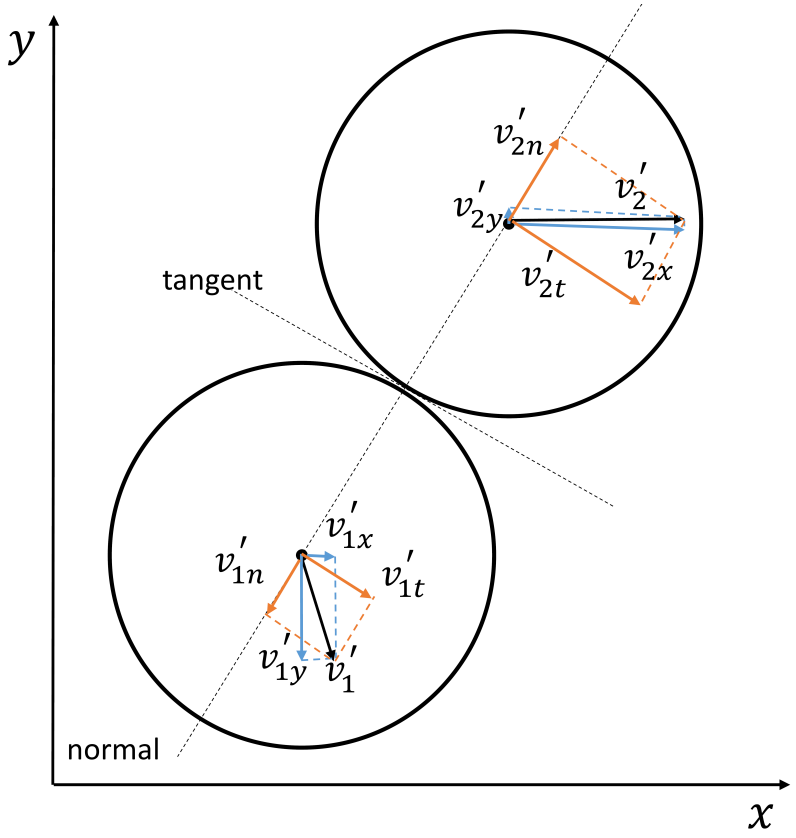

Figure 3: After 2D collision of two particle

For further details on the physics engine, see the file titled **2dcollisions.pdf**.

Optimizing your Simulation

Note that it is easy to implement a simulation using a sequential algorithm. However, the challenging part is to come up with a parallel implementation that scales when increasing input size, number of threads and hardware capabilities. As such, you might need to try several approaches to parallelize the algorithm and several parallel implementations. You are advised to retain your alternative implementations and explain the incremental improvements you have done.
To analyze the improvements in performance, for a carefully chosen input, you should measure the execution time for increasing number of threads. You are advised to focus more in Part 2 on optimizing your implementation, and comparison between OpenMP and CUDA. You are allowed to change/improve your implementation for Part 1 for your Part 2 submission.

Running your Simulation

For OpenMP, run your experiments varying the input size (number of particles, surface, etc) and the number of threads used. For performance measurements, run each simulation at least 3 times and take the shortest execution time. You should use the machines in the lab for your measurements:

- Dell Optiplex 7050 (Intel Core i7-7700)
- Dell Precision 7820 (intel Xeon Silver 4114)

For CUDA, the following resources should be used to complete the assignment:

- **Jetson TX2 Boards** – located in the Lab. You should run, test and make measurements to your implementations here. Take note which board you ran your program on.
- **GPU Nodes in SoC Compute Cluster** – SSH into one nodes reserved for you: xgpe0, xgpe1, xgpe2, xgpe3, xgpe4, xgpf0, xgpf1, xgpf2, xgpf3, xgpf4. To use these nodes, you must enable SoC Compute Cluster from your MySoC Account page. You must use your SoC account to access these nodes. If you are connecting from outside NUS, SSH first into the SoC network (sunfire.comp.nus.edu.sg). As with the Jetson, these nodes should be used for testing and measurement purposes.

FAQ

Frequently asked questions (FAQ) received from students for this assignment will be answered in this file. The most recent questions will be added at the beginning of the file, preceded by the date label. Check this file before asking your questions.

If there are any questions regarding the assignment, please post on the LumiNUS forum or email Julius (julius@u.nus.edu).

Useful resources for Assignment 1 - Part 2

- CUDA Programming Guide
- CUDA `nvprof` Guide

FAQ

- FAQ file for Assignment 1

Assignment Submission Instructions

You are allowed to work in groups of maximum two students for this assignment. You can discuss the assignment with others as necessary but in the case of plagiarism both parties will be severely penalized. Assignment submission will be done in in two rounds.

1. **Mon, 30 Sep, 11am – Assignment 1 – part 1 [12 marks]**: OpenMP implementation and test cases with report about your implementation design and assumptions. Analyze the execution time of the OpenMP program and briefly discuss the advantages of using simulation (simulation time-units versus execution time).
2. **Mon, 21 Oct, 11am – Assignment 1 – part 2 [13 marks]**: OpenMP and CUDA implementations and test cases with report including implementation details, assumptions, and performance comparison between OpenMP and CUDA implementations.

Your report should include:

- A brief description of your programs design (how they work) and implementation assumptions.
- A brief explanation of the parallel strategy you used in your OpenMP and CUDA implementations.
- Any special consideration or implementation detail that you consider non-trivial.
- Explain the modifications that you have made to your code and their impact on performance.
- Details on how to reproduce your results, e.g. inputs, execution time measurement etc.
- Execution time measurements of your sequential and parallel implementations. For these measurements, run your implementations on node 1 and node 2 in our lab (Dell Precision 7820 and Dell Optiplex 7050).
- Analyze your measurement results and explain your observations. Performance comparison between OpenMP and CUDA implementations should be included in your final report (part 2). State your assumptions.

There is no minimum or maximum page length for the report. Be **comprehensive**, yet **concise**.

Submit your assignment before the deadline under LumiNUS Files. Each student must submit one zip archive named with your student number(s) (A0123456Z.zip - if you worked by yourself, or A0123456Z_A0173456T.zip - if you worked with another student) containing:

1. Your C/C++ code along with three sample input files, and the output produced by running your code on these files.
2. `README` file, minimally with instructions on how to execute the code.
3. Report in PDF format (a1_report.pdf)

Note that for submissions made as a group, only one most recent submission (from any of the students) will be graded, and both students receive that grade.
Penalty of 5 marks per day for late submissions will be applied.