CS3211: Parallel And Concurrent Programming
Project 2

1 Overview

2 Implementation

2.1 Definition of Data Structures

Each slave process will be associated with a `struct region`, where a master process has a `struct universe`. Each particle is modelled with `struct particle` that contains information about the particles. Some of the important fields are shown in the below listing.

Listing 1: struct particle

```
1  struct particle {
2      ...
3      bool is_large; /*  Size flag */
4      double vx; /*  X-axis velocity component */
5      double vy; /*  Y-axis velocity component */
6      double fx; /*  Force undergone on X-Axis */
7      double fy; /*  Force undergone on Y-Axis */
8  };
```

Listing 2: struct region

```
1  struct region {
2      ...
3      //Large Particles
4      struct particle* lgps; /*  Large particles in the region
           */
5      int num_large; /*  Number of large particles */
6      int cap_large; /*  Maximal capacity of the lgps array */
7
8      //Small Particles
9      struct particle* smps; /*  Small particles array */
10     ...
11
12     //Communication
13     struct particle** neigh_mov_ps; /*  Particles to be
           passed to neighbors */
14     int neigh_mov_cnt[8]; /*  Number of particles to each
           neighbor */
15     int neigh_mov_cap[8]; /*  Maximal size of the
           neigh_mov_ps[i] array */
16
17     //Canvas
18     unsigned char** canvas_blue; /*  Blue pixels canvas */
19     unsigned char** canvas_red; /*  Red pixels canvas */
20 };
```

2.2 Master-Slave Structure

The overall implementation follows a Master-Slave architecture, where a single slave is responsible for modeling one region, and the single master aggregates the results. A pseudocode for both the master and the slave is given below:

Master:

1. Read universe configuration
2. Initialize universe
3. Receive canvas from slaves
4. Draw the entire picture and save to file

Slave:

1. Read configuration
2. Initialize the region (Allocate memory for particles arrays)
3. Simulate
   (a) Compute interaction of particles locally
   (b) Communicate with neighbors in horizon and compute forces
   (c) Finalize location for each particle
   (d) Move out of bound particles to neighbors
4. Draw particles to canvas
5. Send canvas to master

Merging results from slave: There are at least two ways for the slaves to send back the data of computed regions to the master at the end of simulation. One is to send all the particles back, and then have the master convert those particles into a canvas (.ppm format files). The other is to have each slave convert the particles into a local canvas, send the canvas to the master, and then the master overlay these canvases.

I chose to do the later implementation as the size of the particle arrays are usually larger than the canvas. In addition, since the size of the canvas is fixed and known to both the master and the slave from the configuratin file, the buffer size passed to MPI functions could be pre-determined.

2.3 Communication routine

The communication routine for slaves is enclosed in the function `slave_converse`, where two rounds of MPI communication is done for each conversation. Processes will first communicate the number of particles pending to send. Then they will allocate buffer dynamically to receive the particles. Of course, an alternative which saves one round of communication will be to use a fixed size buffer and get the exact number of particles from the `MPI_Status`. Even though in the experiment conducted for this project, the number of particles to be sent is relatively small, such dynamic buffer management is preferred than fixed buffer to take into account large-scale simulation.

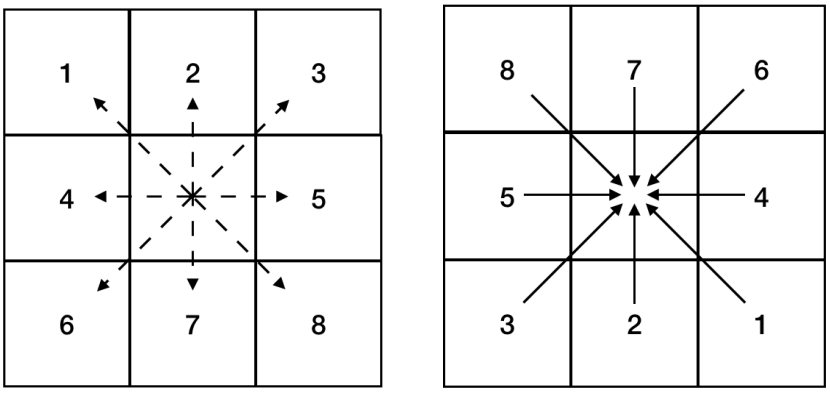
- **Round 1**: Exchange number of particles to be send.
- **Round 2**: Exchange particles

3 Analysis and Evaluation

3.1 Scaling with Communication

In this part of the experiment, each process will hand over particles that move out of its bound to the neighboring processes. Also, it will take into account interaction across-regions if the horizon configuration is set.

3.1.1 Communication Model



(a) Asynchronous Sending Sequence (b) Synchronous Receiving Sequence

Figure 1: Communication model for slaves

Communication - Particles Out of Bound For a process i, it will send the particles to neighbors in the sequence of North-West, North, North-East, West, East, South-West, South, South-East. (Shown in Figure 1a). For processes at the boundary, their neighbors will wrap over across the boundary to the other side.

Also, each send is followed by a receive from the neighbors in the exact opposition sequence as send (Shown in Figure 1b). Since each process will undergo a send and receive in the same iteration, deadlock could happen with synchronous sending and receiving. Therefore, to avoid deadlock, the send is made asynchronous.

This model of communication should achieve high throughput since at each stage, every process is communicating with some other processes. All information (particles) will be transmitted after 8 stages.

Communication - Interaction with Neighbours Another sequence of communication happens when the $conf.Horizon$ variable is set to non-zero. The communication happens at the beginning of each timestep, where a process will pass all its particles to the neighbors within the horizon, and receive the particles from the neighbors as well. The communication pattern follows the Figure 8 exacly when the Horizon is 1. For cases when Horizon is larger than 1, similar approaches are used in which a process will send to the furtherst neighbor in the North-West direction, and first receive from neighbors in the South-East direction. After get the particles from the neighbor, it will then calculate the force of neighbors' acting on itself.

3.1.2 Scaling Results

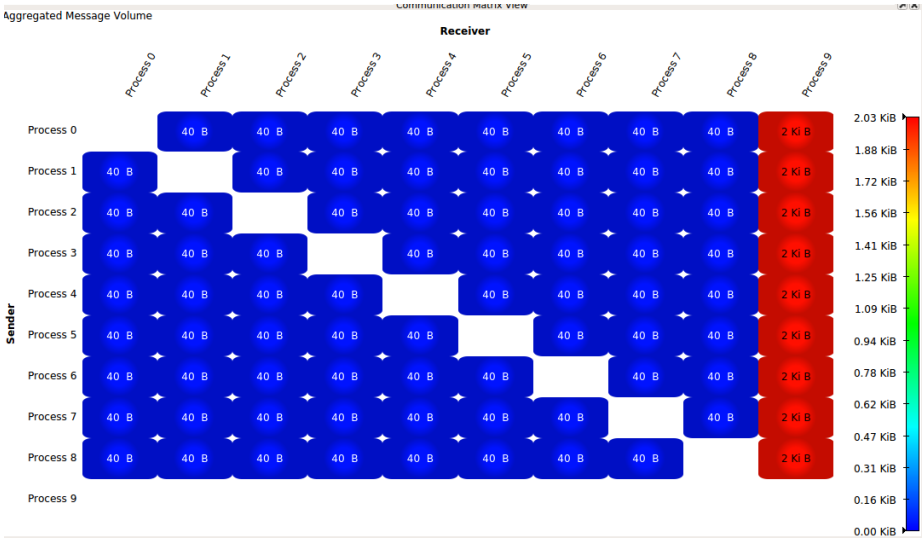
In this part of the project, I focus on the effect of communication between multiple processes when the number of processes scale up. Some of the constrains that I work with in order to generate results in reasonable time (A snapshot of the initialspecs.txt file is shown in Listing).

Listing 3: initialspecs.txt

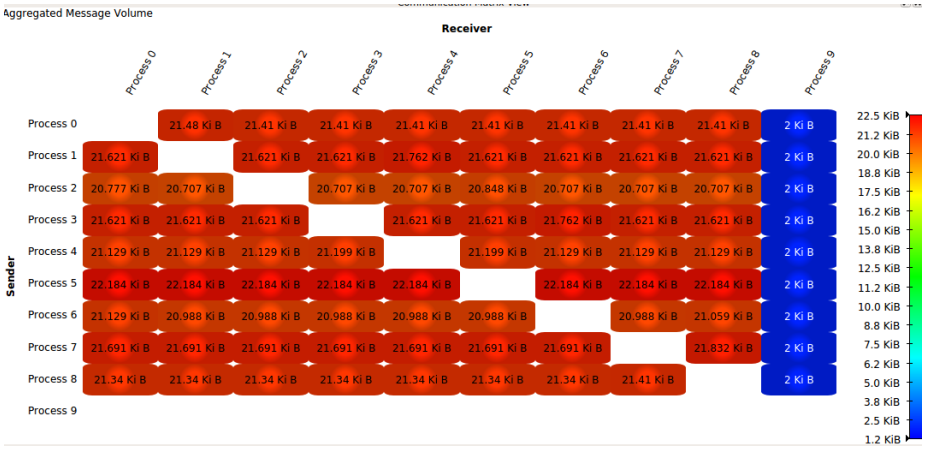
```
1  TimeSlots: 100
2  TimeStep: 20
3  Horizon: 0
4  GridSize: 32
5  NumberOfSmallParticles: 100
6  SmallParticleMass: 0.0001
7  SmallParticleRadius: 0.0001
8  NumberOfLargeParticles: 3
9  5 20000 6.5 9.11
10 2 200 25 20
11 3 10 10.0 17.0
```

- The number of small particles are kept low (100), as the computation time increases exponentially with the number of particles (num_small^2).

- The timeslots is set to a low number so that the experiments could be done.
- The timestep in each timeslot is purposely amplified so that the particles could have enough time to accelerate to a speed.



(a) Message Volume with Horizon = 0



(b) Message Volume with Horizon = 1

Figure 2: Message Volume For Communication from Vampir

As shown from the Figure 2, different horizon model significantly changes the communication pattern and cost.

When the horizon is set to 0, the communication cost is dominated by the slaves sending the canvas back to the master(Process 9). While intra-slaves

communication is minimal with synchronizing on particles moving between regions.

When the horizon is set to 1, the communication cost between slaves dominate the overall message volumes.

Due to limited license of the above profiling program, I explored the scaling effect over horizon and number of processes in the below section.

3.1.3 Tabulation of Results

Finally, I conducted multiple experiments with different number of processes and the horizon value. For each slave, the time taken for communication is accumulated before and after the communication.

- **Slave Execution Time:** Total time a slave runs. (Approximately the summation of computation time, and communication time.)
- **Slave Communication Time:** Time spent in communication. This includes the time taken to transmit particles for force update, movement of particles, as well as the time to send the canvas back to the master. The timer is resumed and paused at the beginning and end of the communication routine.

When the number of processes within the horizon is greater than the total size of the processes, the case is not considered since a process would be in its own horizon in that case.

	Number of Processes								
	1	4	9	16	25	36	49	64	81
Horizon	Slave Execution Time (seconds)								
0	0.269	0.286	0.290	0.350	0.400	0.360	0.576	0.419	0.520
1	-	-	4.059	4.586	4.731	4.578	4.670	4.832	4.925
2	-	-	-	-	12.978	13.043	13.749	13.318	14.826
3	-	-	-	-	-	-	26.810	26.510	27.865
4	-	-	-	-	-	-		-	50.112
	Slave Communication Time (seconds)								
0	0	0.02	0.01	0.07	0.125	0.078	0.134	0.117	0.197
1	-	-	0.120	0.533	0.706	0.618	0.666	0.721	0.908
2	-	-	-	-	2.186	2.020	2.458	2.266	3.965
3	-	-	-	-	-	-	5.410	4.501	6.570
4	-	-	-	-	-	-	-	-	12.210
	Communication / Total Execution Time								
0	0.00%	6.99%	3.45%	20.00%	31.25%	21.67%	23.26%	27.92%	37.88%
1	-	-	2.96%	11.62%	14.92%	13.50%	14.26%	14.92%	18.44%
2	-	-	-	-	16.84%	15.49%	17.88%	17.01%	26.74%
3	-	-	-	-	-	-	20.18%	16.98%	23.58%
4	-	-	-	-	-	-	-	-	24.37%

Figure 3: Tabulation of Execution/Communication Time For Slaves

There are a few results that worth highlighting:

1. Computation time (Total execution time - Communication time) remains constant when the horizon is not changed as number of processes increase. This is not surprising since the number of particles to simulate at each process remains constant.

2. When horizon is 0, the communication time has high percentage of the total execution time when number of processes increase. In this case, processes do not communicate with others in computation, but only synchronize on particles out of bound. Therefore, the computation time increases significantly as the number of particles increase, since calculating forces on small particles between regions will run in $O(n^2)$, n being the number of small particles in a region. On the other hand, the process still has to synchronize with neighbors on particles movement. Total of at least 800 ($8 \times timeslots$) MPI Communication calls will be the overheads.
   Moreover, there is an increase in the communication time as horizon remains constant. This could be the overheads as more processes are allocated to the same machine, where more sharing of resources happens. This is further explored in the later section on machine topology.

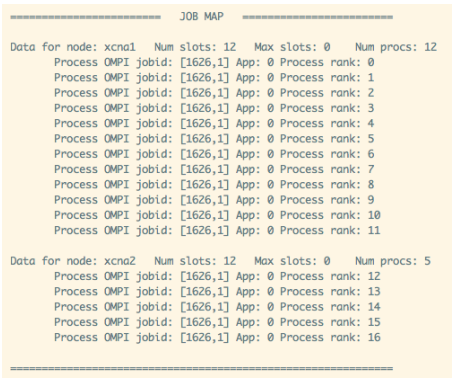
3. The communication time is proportional to the number of MPI calls made, which is proportional to the horizon value. For $horizon = 0$, at least 801 calls (800 for neighbor movement checking, 1 for sending canvas) are made. When $horizon = 1$, each time slot will have additional 16 calls to send/receive particles (8 calls to inform the number of particles, another 8 for particles transmitting, which is further elaborated in section 2.3), which results in 1600 additional calls with timeslot(100). However, the communication time increases more than 5x for all number of processes. It turns out that the transmitting of around 100 particles will take around 4 times longer than just transmitting an integer.
   When $horizon = 2$, there are additional 32 MPI calls (16 additional neighbors in horizon, and 2 calls each for communication) for a process every timeslot, and thus $[200\%, 300\%]$ increase in communication time.

3.2 Virtual Topology vs. Physical Topology

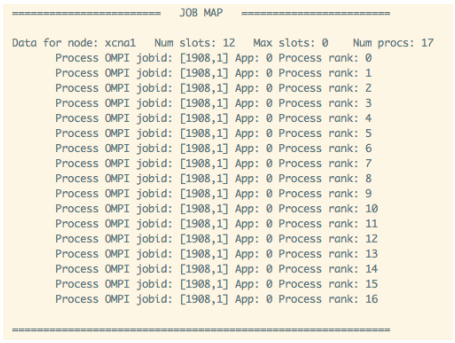
As mentioned in the previous section, there is this 'mysterious' trend of increasing communication time when the number of processes increase with fixed horizon. My first guess was that it might have something to do with how processes are mapped to the hardwares.

The default way of allocating processes to machine is done by socket, meaning that MPI will try to allocate processes to sockets of hosts in the machine file.Therefore, for $num\_slave = 16$, with 8 hosts in the machine file: xcna1 to xcna8, MPI will allocate the first 12 processes on xcna1, and the remaining on xcna2. (Figure 4a). This would allow processes to be run on the same host but different socket. Such mapping is better than spreading processes to machines that are far away since inter-node communication could be more costly. Figure 4 shows three different ways to map processes to the hardware, figure 4a is the

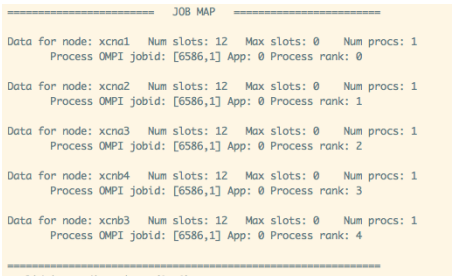
default by socket, figure 4b runs with `--oversubscribe` option, which maps all processes into the only available node in the machinefile. Figure 4c(NP = 5) maps processes to different nodes across two clusters (xcna and xcnb) by including hosts from both clusters in the machinfile and run with `--map-by node` option. Figure 4d maps processes to nodes in the same cluster.



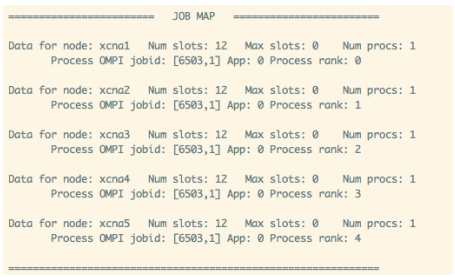
(a) Normal Mode



(b) Oversubscribe



(c) Map to node across clusters(xcna, xcnb)



(d) Map to node in same cluster

Figure 4: Different Config for allocating processes

3.2.1 Tabulation of Results

Results of experiments which explore the relationship of different mapping with horizon are tabulated as below. The experiment is ran with 81 slaves and 1 master, using the same configuration (Listing 3.1.2)

Apparently, config 1 and config 4 where processes are mapped to socket or node in the same cluster have the best performance, while oversubscribing and mapping across clusters will incur significant communication cost. This has shown that the physical topology on which the processes are run would have a huge influence on the overall scaling and performance of the system.

	Config 1	Config 2	Config 3	Config 4
Horizon	**Average Communication Time(seconds)**			
1	0.83	10.88	52.47	1.09
2	2.38	29.87	101.5	2.29
3	5.17	57.76	227.57	4.39
4	8.72	95.8	331.3	9.35
	Maximal Communication Time (seconds)			
1	1.01	11.19	58.14	1.35
2	2.975	30.66	108.9	2.83
3	6.04	59.97	243.3	6.13
4	10.21	98.4	352.84	12.09
	Minimal Communication Time(seconds)			
1	0.58	10.48	34.52	0.76
2	1.948	28.79	36.509	1.47
3	4.17	55.602	77.7	2.94
4	6.04	92.1	132.66	6.09

Figure 5: Tabulation of Communication Time For Slaves in Different Config

3.2.2 Creating Cartesian Topology

Another feature that I explored for the topology was the use of `MPI_Cart_create` to create a virtual topology of 2D torus from global communicator. However, in contrary to what I expected, the new communicator using virtual topology has no effect on the communication efficiency. But it would have resulted in much simpler neighbor discovery by using the coordinates in the 2D torus than my own hardcoded one based on the rank. Also, the power of creating new communicator is great, as in the bounce-off implementation, the created communicator which includes only the slaves will be used to facility all-to-all communication.

3.3 Accuracy

The simulation deals with much floating point calculation, and the specific design of the project makes such calculation more prone to rounding errors.

- The large particle and small particles have great difference in size and mass. Therefore, when small particle has force acting on the large particle, consider the force on the large particle, $f_{large} = f_{large} + \delta f$, δf is usually small comparing to f. Many precision digits will be lost when such calculation is performed.
- Rounding errors that come from such floating point calculation will be ac-

cumulated and amplified as timeslots increase. Since the error in rounding is random, and the error will thus be different on different machines. With a substantial number of timeslots, the location of particles will be different on different machines: Figure **??**.

There are multiple ways to improve the accuracy of the system, for example, using a compensation summation algorithm to keep track of the error. What I did to improve the system is to use the GNU MP library for the calculation. I added the below fields for the `struct particle` which will be used to calculate particle's interaction.

```
[Node 1]Particle: 20000.000000kg, 5.000000m @(17.130420, 15.006816), v=(0.000042,0.000047)
[Node 2]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006809), v=(0.000042,0.000046)
[Node 8]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006830), v=(0.000042,0.000047)
[Node 0]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006816), v=(0.000042,0.000047)
[Node 9]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006816), v=(0.000042,0.000047)
[Node 4]Particle: 20000.000000kg, 5.000000m @(17.130420, 15.006816), v=(0.000042,0.000047)
[Node 6]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006816), v=(0.000042,0.000046)
[Node 5]Particle: 20000.000000kg, 5.000000m @(17.130418, 15.006830), v=(0.000041,0.000047)
[Node 7]Particle: 20000.000000kg, 5.000000m @(17.130419, 15.006816), v=(0.000042,0.000047)
```

Figure 6: Different Particle Position and Velocity on Different Machines

Listing 4: struct particle

```
1   struct particle{
2       ...
3       #ifdef ACCU_TEST
4
5       mpf_t m_r;
6       mpf_t m_m;
7       mpf_t m_locx;
8       mpf_t m_locy;
9       mpf_t m_vx;
10      mpf_t m_vy;
11      mpf_t m_fx;
12      mpf_t m_fy;
13
14      #endif
15  }
```

3.3.1 Results

While the performance is almost times slower for the GNU MP version, all the processes produce the same results across different hosts.

Number of Small Particles	50	150	300	500
Non- GNU MP	2.9s	15.8s	71.3s	213.2s
GNU MP	7.2s	87.1s	320.1s	657.3s

Figure 7: Tabulation of Execution Time with Accuracy Improvement

4 Bounce-off Frobisher

4.1 Overview

As the last part of the project, I implemented the bounce-off simulation. An overview of the implementation is given as below:

1. Each slave first computes interactive forces of each particle.
2. Each slave then determines if there exits a pair of particles that collide or a particle that crosses boarder in the timestep.
3. All salves communicate with each other using `MPI_Allgather` to determine the shortest timestep that could be ran without any collision or broader crossing event. Set its timestep to be this global minimal timestep.
4. Each slave runs the simulation until that global minimal timestep.
5. The slave which has a collision event or broader crossing event now handles the event by calculating the collision effect, or passing the particle to the correct neighbor.
6. Repeat from step 1.

4.2 Implementation Details

There are a few key routines in the implementation that will be discussed in details here:

4.2.1 How to adopt collision mode?

A static timestep is no longer appropriate for collision mode since a collision could happen within a discrete and fixed timestep. Therefore, a dynamic timestep is needed. Effectively, we want to apply a total ordering of all the collision events so that change in velocity of the collided particles could be re-calculated at each collision.

So a dynamic timestep will be the nearest timestep (smaller than the configured timestep) from now where no collision will happen in all regions. Each process can safely run its simulation until that timestep.

Each process will first calculate its local safe timestep, and all processes will synchronize on a global safe timestep.

One caveat and simplification made in the system is that a broader crossing event is also considered 'unsafe'. There might be a possibility that one particle go beyond the boundary and run into another particle. Without stopping at broader crossing event, the program has to be able to calculate collisions with broader crossing, which adds too much complexity.

4.2.2 How to detect collision?

After calculating forces being applied on each particle, there is enough information to predict the movement of a particle, and thus capture detection.

Two particles collide whenever their distance (between centers) are smaller or equal to the sum of their radius. With such relationship, I can construct a polynomial equation for the distance between two particles with respect to t, and find the value of t when the equation is smaller or equal to r^2.

Each particle's position can be modeled as $\vec{x}$, and its velocity as $\vec{v}$. Their distance d is:

$$d = \left\| (\vec{x_1} + t\vec{v_1}) - (\vec{x_2} + t\vec{v_2}) \right\|$$

Solving the above equation such that $d = r^2$, where r is the distance between centers of any two particles. For all particle pairs, the smallest t, t_{min} that is positive and smaller than timestep will be used as the local timestep value. This routine is implemented in `p_collide_time()` in `particle.c`.

4.2.3 How to detect boarder crossing?

The coordinates of a particle could be modelled with the equations:

$$\vec{x_{final}} = \vec{v}t + \frac{1}{2}at^2 + \vec{x_{now}}$$

Solving for the equations when any of the coordinates that goes beyond the boundary will lead to a local minimal safe timestep similar with the above situation.

4.2.4 How to synchronize on the timestep?

The collective communication routine `MPI_Allgather` is called every time before finalizing the locations of the particles. Each process will send the local safe minimal timestep to all other slave processes using the communicator `slave_comm`.

4.2.5 How to compute collision?

Once we know which particles collide, and information such as velocity, mass, locations, size, the rest is simply physics.

According to conservation of kinetic energy and momentum in elastic collisions, I can calculate the final velocities of two particles:

$$\vec{v_1'} = \vec{v_1} - \frac{2m_2}{m_1 + m_2} \frac{\langle \vec{v_1} - \vec{v_2}, \vec{x_1} - \vec{x_2} \rangle}{\|\vec{x_1} - \vec{x_2}\|^2} (\vec{x_1} - \vec{x_2})$$

$$\vec{v_2'} = \vec{v_2} - \frac{2m_1}{m_1 + m_2} \frac{\langle \vec{v_2} - \vec{v_1}, \vec{x_2} - \vec{x_1} \rangle}{\|\vec{x_2} - \vec{x_1}\|^2} (\vec{x_2} - \vec{x_1})$$

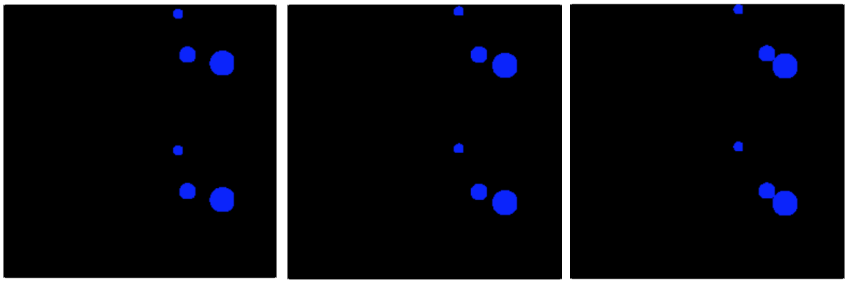
The routine is implemented in the `p_collide()` at `particle.c`.

One more caveat for the collision case is that simply revert the velocity at collision turns out to be not enough for the actual simulation to work. This is due to rounding errors in the calculation, that even after the velocity is reverted, the particles will soon be declared as colliding with a short value of t. This chain of collision will make particles stick together since the timestep at each timeslot is extremely small.

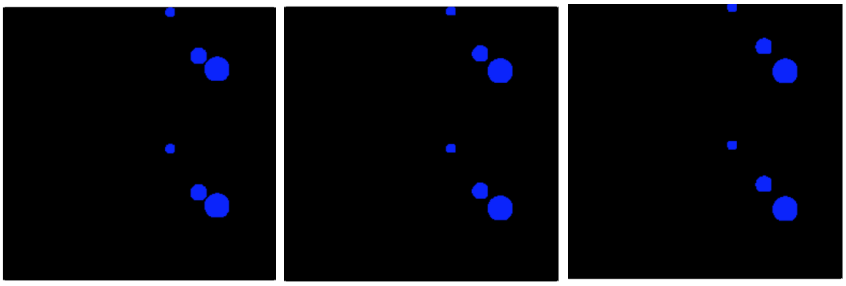
To counter this issue, at each collision, the particles are allowed to move with its new velocity for a short period of time. While this does bring in errors for the system, but it allows the particles to get out of collisions black-hole.

4.3 Implementation Result

It is difficult to tune the parameters so that a nice collision pattern is actually seen. The below figures are generated with no small particles (to reduce time), and 0 horizon.



(a) Timestep: 40000 (b) Timestep: 43000 (c) Timestep: 43500



(d) Timestep: 45300 (e) Timestep: 46000 (f) Timestep: 47000

Figure 8: Communication model for slaves