Report

on

MPI-backed Parallel Particle Simulation

Course No. **: CSCI-522**

Submitted
to
Dr. Laurence T. Yang

Submitted
by
Md Mokarrom Hossain
Student ID # 20164460
X2013idf@stfx.ca

Winter 2014

Problem Description

In this report we will present a MPI-backed parallel implementation of particle simulation and verify the ideal gas law PV = nRT by simulating particles movement and interaction. Also, show something called Brownian motion as bonus project. Here, we assume that particles are hard with radius 1 and all collisions are elastic. The box is a 2-dimensional rectangle without any friction. In order to show the Brownian motion a particle with greater mass and radius has been inserted in the box. The trajectory of this big particle has been monitored. Since *CSSun cluster* does not support any graphics environment (e.g. MPE extension, graphis.h) I could not plot the trajectory of big particle.

The verification of gas law (PV = nRT) can be done by calculating $R = \frac{PV}{nT}$ for different combinations of P, V, T and n and verify that it is constant. Here,

P is the pressure; this is calculated by counting total momentum absorbed by the box wall during t seconds and dividing this by the perimeter of the box and t.

V is the volume of gas; here this is the area of 2-dimensional rectangular box.

T is the temperature of gas; this is proportional to the average energy of the particles (here proportionality constant is defined as 1 and particle energy = $\frac{v^2}{2}$, *v* is the velocity).

n is the number of moles (here number of particles).

R is the magic constant; need to verify that it is constant.

In this project, we will look at typical physics simulation codes to parallelize and optimize them with MPI. The goal of the parallelization is to make the code highly efficient for a parallel system and to make it as scalable as possible to do large-scale simulations as well as using this parallelized application as a benchmark for our MPI tests.

Distribution of particles

There are several ways to distribute particles over the processor. In this report we will explore the trade-offs between block partitioning and column base strip partitioning.

Column base strip partitioning:

The box region can be divided column wise. Column base strips partitioning is shown in figure-1. Suppose we have ***P*** number of processor and an ***m x n*** 2-dimensional box i.e. width of box is m and height of box is n. In column base strip partitioning, the box is divided into P equal columns (sub-boxes). One processor is responsible for one sub-box and each sub-box has $\frac{m}{P}$ width and n height.

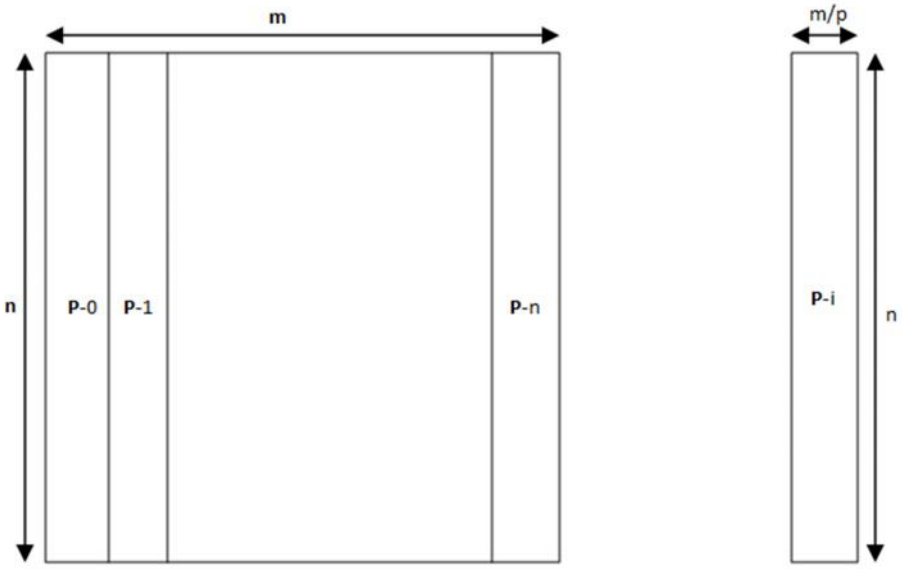


Fig-1: Column wise strip partitioning; dividing the box region into **P** equal columns

For simplicity in later comparison, we assume that our 2d box is a square (i.e. ***n x n***). Therefore, area and perimeter of each sub region are-

$$\text{Area} = n \times \frac{n}{P} = \frac{n^2}{P}$$

$$\text{Perimeter} = 2\left(n + \frac{n}{P}\right) = 2\left(\frac{nP+n}{P}\right) = \frac{2n(P+1)}{P}$$

$$\frac{Area}{Perimeter} = \frac{n}{2(P+1)}$$

Block partitioning:

The box region is divided into equal size small rectangles. If we have ***P*** number of processor and an ***m x n*** 2-d box then each sub region's width is $\frac{m}{\sqrt{P}}$ and height is $\frac{n}{\sqrt{P}}$ (i.e. each processor is responsible for $\frac{m}{\sqrt{P}}$ ***x*** $\frac{n}{\sqrt{P}}$ 2-dimensional small rectangle). In figure-2 blocks partitioning has been depicted.

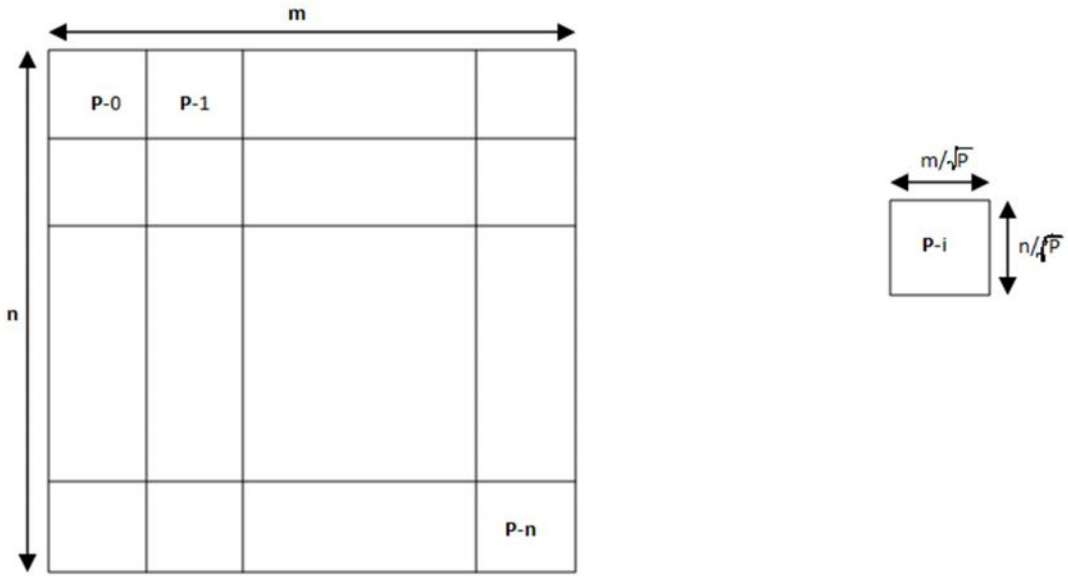


Fig-2: Block partitioning; didviding the box region into equally big rectangles.

For simplicity in later comparison, we assume that our 2d box is a square (i.e. ***n x n***). So each sub-box is also a square. Thus, the area and perimeter of each sub region is given by-

$$\text{Area} = \frac{n}{\sqrt{P}} \times \frac{n}{\sqrt{P}} = \frac{n^2}{P}$$

$$\text{Perimeter} = 2\left(\frac{n}{\sqrt{P}} + \frac{n}{\sqrt{P}}\right) = 2\left(\frac{n+n}{\sqrt{P}}\right) = \frac{4n}{\sqrt{P}}$$

$$\frac{Area}{Perimeter} = \frac{n}{4\sqrt{P}}$$

Block vs Strip (column) partitioning:

In both type of partitioning, entire box is divided into a number of small regions and each processor is responsible for one region. So at the beginning of each time-step, particles that are in the same region only need to be checked for collisions. Since there are no interactions between particles in different region, some collusion may be ignored. But if the time step and velocities are small this will not affect the final result much.

However, there is a geometrical relation between the number of missed collisions and the ratio of area and perimeter of the sub region. In order to minimize the number of missed collisions, the area of each region divided by its circumference should be maximized. The minimum number of missed collisions should lead to fewer communications between processors.

We observed that the area divided by perimeter of each sub region in block partition is greater than strip (column) partition provided that the dimension of sub region is sufficiently larger than number of processors.

Block partition: $\frac{Area}{Perimeter} = \frac{n}{4\sqrt{P}}$ (when the box is ***n x n*****)**

Strip (column) partition: $\frac{Area}{Perimeter} = \frac{n}{2(P+1)}$ (when the box is ***n x n*****)**

If P << n, obviously $\left(\frac{n}{4\sqrt{P}}\right) > \left(\frac{n}{2(P+1)}\right)$. So block partition is better than strip partition. Indeed, block partition is a reasonable good way to minimize the number of missed collisions in this type of scenario.

Implementation

In our implementation, for each sub-box one processor is assigned. Each processor creates and distributes uniformly $\frac{n}{P}$ particles in its region. Same time-step is running in all processors. When a time-step is finished particles that cross the local box need to send appropriate neighbors. Therefore, after each time-step a list of particles is sent to all neighboring processors. Particles are only sent to neighbors that are adjacent horizontally and vertically. So processors are only need to communicate horizontal and vertical neighbors. Therefore we use *MPI_Dims_create* for creating a 2-dimensional Cartesian grid. Since we are sending only those particles that move between sub-regions, inter-process communication is fairly low. Internally each processor uses linked lists to store particles.

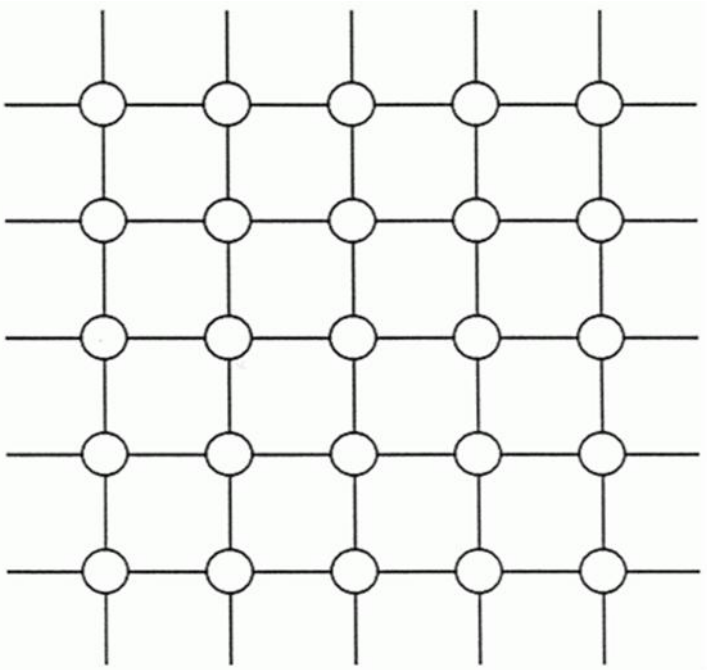


Fig-3: Processors are connected horizontally and vertically like a 2-d mesh topology.

In a nutshell, simulation is performed by following steps-

- Processor 0 reads the simulation data broadcast it to all processors.
- Processor 0 calculates Volume.
- Partitioning: Divide the box into P equal sub-boxes. Each processor computes the dimension of its own sub-box.
- For each processor,
    - Initiate particles: Distribute $\frac{n}{P}$ particles and their velocities uniformly in the local sub box.
    - Communicate to all processors and calculate temperature (master will have the final temperature).
    - Main loop: for each time-step do
        - for all particles do
            - Check for collisions with big particle, for every collision calculate new velocities and positions for the particles.
            - Check for collisions between particles, for every collision calculate new velocities and positions for the particles.
            - Move particles that have not collided with another.
            - Move big particle.
            - Check for wall interaction and add the momentum.
        - Communicate with four neighbors and update the particle list (Particles are sent between processes if they out of local sub box).
    - Communicate to all processors and calculate pressure (processor 0 will have the final pressure).
- Processor 0 calculates R.

In this implementation each particle can only collide with one other particle during each time-step. This is a simplification, but it is reasonable if the number of collisions is low. This means that the number of particles has to be low compared to the area of the box and the length of each time-step.

In parallel implementation, we try to use most suitable MPI routine and always inclined to minimize the communication cost. For example, in case of sending block of memory we use MPI derived data type instead of MPI primitive data type. We used mpiPartType derived data type for exchanging particles with the neighbors.

```
MPI_Type_struct (2, blockcounts, offsets, oldTypes, &mpiPartType);
MPI_Type_commit (&mpiPartType);
```

In order to avoid deadlock during communication to neighbor processors we used non-blocking send (*i.e. MPI_Isend*) and blocking receive (*i.e. MPI_Recv*) function. After the non-blocking send we use an *MPI_Waitall* routine for waiting until send operation complete.

```
MPI_Isend (sendBuffer[j], sendCounts[j], mpiPartType, neighbours[j], 0, gridComm, &(request[j]));
MPI_Recv (recvBuffer[j], COMM_BUFFER_SIZE, mpiPartType, MPI_ANY_SOURCE, 0, gridComm,
&(status[j]));
MPI_Waitall (4, request, MPI_STATUS_IGNORE); //Wait for non-blocking send completion.
```

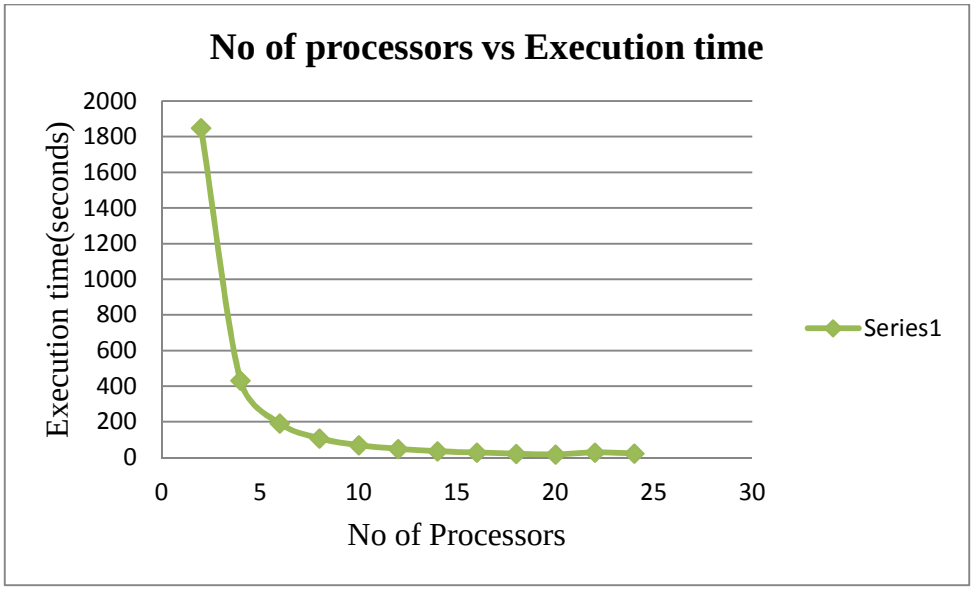
Experiments and Results

In order to ease the experiment we have built a test script in *basch script*. We have executed our test script several times and noted the execution time. Here is the one observed data keeping time-step (50) and number of particles (24,000) constant.

Number Of Processor	Execution Time (seconds)	Number Of Processor	Execution Time (seconds)
2	1849.08	14	35.11
4	431.82	16	27.64
6	191.95	18	21.50
8	107.93	20	17.68
10	69.09	22	28.77
12	48.48	24	23.45

Table-1: Reduction of execution time.

Following line graph shows the relationship between execution time and number of processor more clearly.



Speed up

The speedup of a parallel code is how much faster it runs in parallel. In conventional speed up calculation, problem size is assumed to be fixed. If the time it takes to run a code on single processors is T_s and the time it takes to run the same code on P processors is T_P, then the speedup is given by

$$S_p = \frac{Execution\ time\ using\ one\ processor}{Execution\ time\ using\ a\ multiprocessor\ with\ p\ processors} = \frac{T_s}{T_p}$$

No of particles	Sequential Execution Time (T_S) (seconds)	No of Processors	Parallel Execution Time (T_P) (seconds)	Speed Up ($S_p = \frac{T_s}{T_p}$)
24,000	3619.94	2	1849.08	1.96
		4	431.82	8.38
		8	107.93	33.54
		12	48.48	74.67
		16	27.64	130.97
		20	17.68	204.75
		24	23.45	154.37

Table-2: Speed up for 2, 4, 8, 12, 16, 20, 24 processors [time step =50].

Scaled Speed up

In scaled speed up, parallel execution time is assumed to be fixed, but problem size is scaled up. If the problem size (here number of particles) is scaled up from N to N_S then scaled speed up is defined by

$$S_s = \frac{Single\ processor\ execution\ time\ for\ N_S\ particles}{Multiprocess\ execution\ time\ for\ N_S\ particles}$$

Experimentally, scaled speed up is calculated by keeping the ratio $\frac{N}{P}$ as constant between single processor case and multiprocessor case. Here we assume that $\frac{N}{P} = 500$.

No of processors (P)	No of particles (N)	Sequential Execution Time T_S (seconds)	Parallel Execution Time T_P (seconds)	Scaled speed up [$S_p = \frac{T_s(N)}{T_p(N)}$]
2	1,000	11.34	2.85	3.98
4	2,000	47.04	2.89	16.28
8	4,000	189.57	3.04	62.36
16	8,000	772.06	3.07	251.49
24	12,000	1809.97	3.18	569.17

Table-3: Scaled speed up for processors 2, 4, 8, 16, 24 [time step =50].

PV = nRT Verification

In order to verify the gas law (PV = nRT) we need to verify $R = \frac{PV}{nT}$ is constant for different set up of P, V, T and n. For all set ups time step =100.

Number of particles (N)	Volume (V) w *x* h	Pressure (P) $\frac{\sum_{i=1}^{t} 2mv}{2(w + h)}$	Temperature (T)	Magic constant (R)
20,000	100,000,000	0.082	413.608	0.995
	400,000,000	0.021	414.575	0.999
	2,500,000,000	0.003	414.248	1.027
40,000	100,000,000	0.165	419.036	0.986
	400,000,000	0.043	416.789	1.021
	2,500,000,000	0.007	413.622	0.992
80,000	100,000,000	0.333	414.351	1.005
	400,000,000	0.084	416.466	1.003
	2,500,000,000	0.013	416.357	0.993
100,000	100,000,000	0.408	415.049	0.983
	400,000,000	0.103	416.840	0.990
	2,500,000,000	0.017	418.762	1.020

Table-4: PV=nRT verification for different set ups [time step =100].

Discussion

Our observed results of particle simulation seem to agree with the gas law. In different observation we calculate R by changing the number of particles n and size of the box. Though calculated R in table-4 varies slightly, but almost in all runs it is nearly 1. Thereby, R seems to be constant. If we assume that time-step is a constant, the asymptotic complexity of sequential program is $O\ N^2$, where N is the number of particles. Here, each particle needs to be checked against every other particle for collisions. In parallel implementation, if the distribution of particles is assumed uniform the asymptotic complexity is $O\ \frac{N^2}{P}$, where P is the number of processor.

Attachment

Annotated source code of the program has been enclosed.