Assignment 2

Amin Alimoradi

November 30, 2022

1 Introduction

In this assignment, I consider bouncing particles in a box. This project illustrates a model that defines how I can find particles by sectors as well as the particles in their neighbors, simultaneously. I consider two regimes which are the Ballistic regime and the Diffusive regime. In this assignment, I develop my codes and solutions based on two theories: 1) Langevin theory, and 2) Lennard-jones theory. I explain briefly these theories. Based on Langevin's theory (Brownian motion) particles typically move by fluctuating randomly inside a fluid subdomain, then relocating to another subdomain. based on this theory, there exists no preferential direction of flow. On other hand, there is an intermolecular pair potential called the Lennard-Jones potential. It is considered as a model for simple yet realistic intermolecular interactions. Based on the Lennard-Jones theory, when particles become very close to each other, they will hit each other and goes randomly to other side of the box. Based on these two theories, if we want to find the location of the particles in the box, It will be difficult. Although what we can do is to divide the box into sectors to check them and find them in their neighbors if they will be there. This function determines the sector a particle is in based on its coordinates (x,y) and each sector will have a list of neighbors at the beginning of the array. Following that, we desire to parallel the code to speed the process up with parallelization over loops.

The hardware that we run our tests on is :

CPU Information
Model name: : Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz
CPU core is 4
Cache size is 6144 kb
CPU Max MHZ = 3500
CPU Min MHZ = 800
Threads per core is 1
RAM Information
Memtotal is 16263280 kb
Cached is 5104028 kb
Memfree 379620 kb
Memavailable 5475988 kb

2 Main Pseudo code

1- Loop over s1, Assume that s1 represents the sectors and $M^2 - 1$ represents the number of sectors in a row.
2- Call particle-list $< -s1$ p-list, When particle-list s1 matches current sector, then p-list holds particles in current sector
3- Now we have list of particles in S1
4- P1=loop so it means, We will loop over the particle list (starting from the first particle, and proceeding until we reach the counter-1)
For instance: [1,2,3,4,-1,-1,-1,4]
5- P=P-list(p1) In this case, p is the particle for which the interactions will be calculated
6- Loop over s2, Loop over sector s2 to determine the neighbouring sector s2 and the host sector s1.

7- Call particle-list $< -s2$, q-list In this case, s2 is the neighbour sector, so q-list now contains particles in s2
8- q1 = Loop over q-list ,From the first particle, go forward until we reach counter-1)
9- q=q-list(q1), Assuming that q represents the particle with which the interactions will be calculated at the moment
10- so if $p \neq q$:
get rx : The x-distance between particles p and q is given by rx
get ry : The y-distance between particles p and q is given by rx
get d : where d is the distance between p and q particles
If $d < cr$ where cr is critical radius
F is Lennard-Jones potential force
get ax where ax is acceleration in the x direction
get ay where ay is acceleration in the y direction

3 Pseudo code for Sector by position

Me and Abraham collaborated together to write this psuedo code :

```
Given Particle coordinates (x, y) find Sector :
1- let x = vector of horizontal position
2- let y = vector of vertical position
3- let M = number of divisions
4- let L = total length of grid
5- let WIDTH = L÷M
6- let SECTORi = array of sectors of size $M^2$ starting from 0 to $M^2 - 1$

loop over vector x
1-let (DISTANCE1i) = (Xi - (-L÷2) where DISTANCE1 is the distance from left side of grid
2- let columni = distance1 ÷ width (integer) (function is floor) where COLUMN is the column index

loop over vector y
1- let DISTANCE2i = (Yi - (-L÷2)) where DISTANCE2 is the distance from the top of the grid
2- let ROWi = DISTANCE2i ÷ WIDTH(integer)(Function is floor) where ROW is the row index

Loop over sector
let SECTORi = (ROWi) * M + COLUMNi
```

4 Pseudo code for Particles in given sector

```
X ∈ $R^n$ , Y ∈ $R^n$ secNumber ∈ Z p_list ∈$R^n$
Result: p_list ∈ $R^{n+1}$
Call sector_by_position(X,Y,S):
counter ← 1
While i ≤ n do
if S(i) = secNumber then
p_list (counter) ← i :
counter ← counter + 1:
else
p_list (i) ← -1:
end
end (while)
```

p_list (n+1) ← counter -1:
Algorithm 1: An algorithm that finds a list of the indices of the particles in a given sector. After finding a list of particles the algorithm tacks on a counter of the number of particels located within that sector for easier bookkeeping.

5 OPENMP

In this case, I want to use OPENMP to make the code parallel. If I want to use more particles it would be better to use OPENMP to decrease the processing time. Although, the OPENMP does not work efficiently for less number of particles. I have tested the code with t_max = 0.1d0. As it can be seen, when I increase the number of particles, The processing time for parallel programming is faster than non parallel programming. For example, the processing time for 4000 particles in non parallel programming is double than that in parallel programming. The result for parallel and non parallel codes as followed:

Number of Particles	Parallel	non Parallel
1000 Particles	6 sec	6.1 sec
2000 Paritlces	14 sec	23 sec
3000 Particles	26 sec	52 sec
4000 Particles	44 sec	95 sec
5000 Particles	65 sec	155 sec

Number of particles	Full Optimization(-O3) with OMP	Full Optimization(-O3) without OMP
1000 Particles	5.5 sec	6 sec
2000 Paritlces	14 sec	24 sec
3000 Particles	28 sec	54 sec
4000 partickes	47 sec	97 sec

As it can be seen,The processing time of full optimization is faster than without optimization for OPENMP code but the results is approximately same with OMP_NUM_THREADS=4. On the other hand, The processing time of full optimization is much faster than without Openmp

Number of particles	OMP_NUM_THREADS=1	OMP_NUM_THREADS=3
1000 Particles	5.7 sec	4.8 sec
2000 Paritlces	23 sec	14 sec
3000 Particles	52 sec	28 sec

Number of particles	OMP_NUM_THREADS=2	OMP_NUM_THREADS=4
1000 Particles	5 sec	4.7 sec
2000 Paritlces	15.3 sec	14 sec
3000 Particles	31 sec	27 sec

As it can be seen, I have used (export OMP_NUM_THREADS) for 1000-2000-3000 particles. When I used OMP_NUM_THREADS = 4, it is faster than other threads like 1,2 and 3.

6 Order of complexity

The idea is that the program by itself is $O(n^2)$ but if we introduce domain decomposition then we can in theory chop down the complexity since each domain will have its own thread working on it. So the more domains we have the greater the speedup.In this section, I draw a graph for MSD equation by using Python 3. I show the results for 1000,2000,3000,4000 particles in T_MAX= 0.01d0.The axes in these figures are as follows: 1) x-axis represents the logarithmic time and 2) Y-axis represents logarithmic distance. We can see the difference between the behavior of the two regimes in the figures as the number of particles increases. For instance, the linear part is the ballistic regime and when it increases sharply, we have the diffusive regime.
The results as followed:
Figure1 = 1k particles
Figure2 = 2k particles

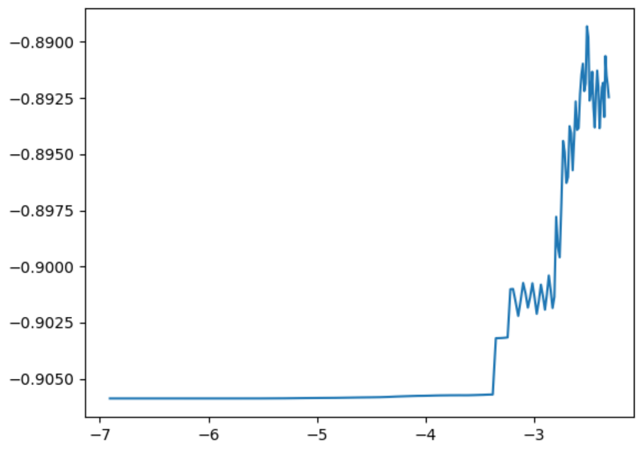

Figure 1:

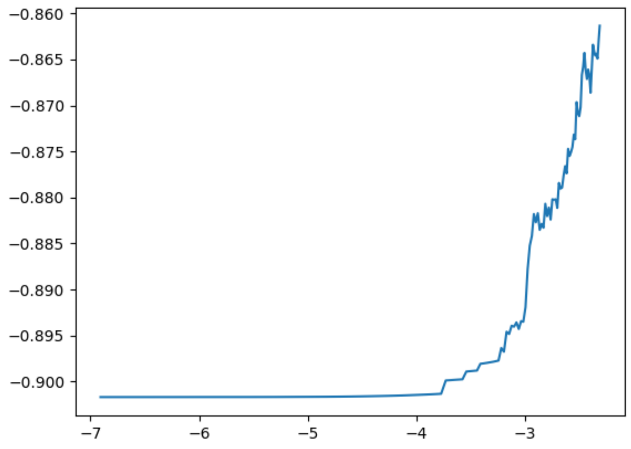

Figure 2:

Figure3 = 3k particles
Figure4 = 4k particles

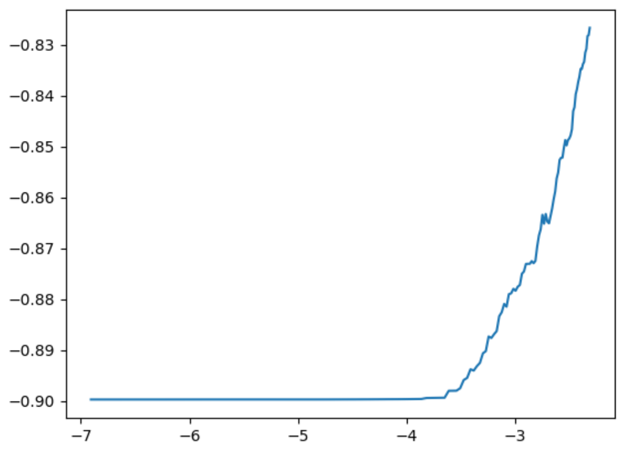

Figure 3:

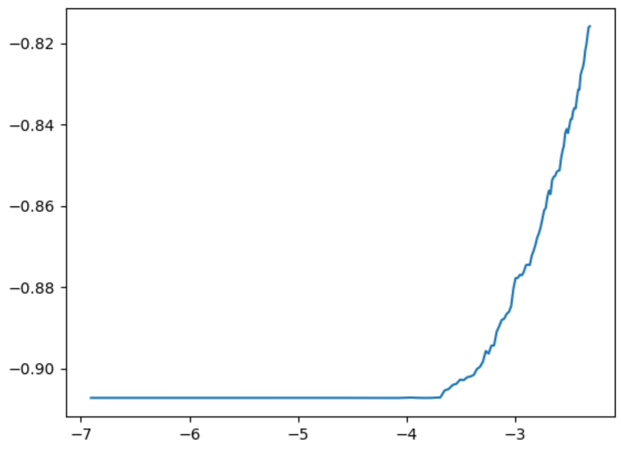

Figure 4:

7 Conclusion

It should be mentioned that based on the two Langevin and Lennard-Jones theories, all the members of the group contributed in writing a pseudo code for each separate part of the project . After that, I tested with different particles to illustrate processing time of code with OPENMP and without OPENMP. As a conclusion, I figured out that the parallel programming is much faster than non parallel programming based on the results provided.