MCSC 6030G : High Performance Computing
Assignment 3: Langevin Dynamics of Interacting Particles

Parikshit Bajpai
100693928

1 Introduction

In 1908, Paul Langevin successfully applied Newtonian dynamics to a Brownian particle to come up with what is now called as the 'Langevin equation', the stochastic physics equivalent to the second law of motion ($F = ma$). This stochastic differential equation is applicable to continous Markov processes and shows that the root mean square dispalcement of a Brownian particle is proportional to the square root of time. The Langevin equation contains both viscous and random forces with the two forces related by the fluctuation-dissipation theorem. In the modern notation, Langevin equation takes the following form:

$$m\,dv = -\gamma\,v\,dt + \sqrt{dt}\,c\,\eta(t) \tag{1}$$

where the velocity of the particle, $v \in \mathbb{R}^2$ in a two-dimensional space, γ denotes the viscosity contribution of the force, and the random force variable η is drawn from a standard normal distribution. The random force coefficient equals, from the fluctuation-dissipation theorem, $c = \sqrt{2\gamma k_B T}$, where k_B is the Boltzmann constant and T is the temperature.

In Langevin dynamics, at short time scales, i.e. $t \ll \tau_p$, where $\tau_p = m/\gamma$ is the momentum relaxation time of particles, the dynamics of a Brownian particle is dominated by its inertia, and, the motion is this region is termed as *ballistic Brownian motion*. At larger time scales, i.e. $t \gg \tau_p$, *diffusive Brownian motion* is observed.

2 Methodology

2.1 Objective

In the present study, Langevin dynamics has been further analysed by including interaction forces between the particles and implementing domain decomposition and parallelisation. The objective of this study is to analyze the Brownian motion of interacting particles with domain decomposition and multi-threading, and, to study the impact of model parameters on the efficiency of parallelization.

2.2 Machine Configuration

The initial computations for the present study were performed on the local machine followed by detailed study on HPC systems accessible through *Compute Canada*.

2.2.1 Local Machine

Manufacturer & Model: Lenovo ThinkPad Yoga 370
Processor: Intel Core i5 -7200U (2 physical cores, 2 hyperthreads)
Clock Rate: 2.50 GHz
RAM: 16 GB
Operating System: Ubuntu 18.04

2.2.2 GRAHAM Cluster

Manufacturer: Huawei
Processor: Intel E5-2683 v4 (Broadwell) (Total 36160 cores, maximum 32 cores used in the study)
Clock Rate: 2.10 GHz
RAM: Node specific
Operating System: CentOS 7

2.3 Implementation

The Langevin equation was discretised using the Velocity Verlet algorithm, which is similar to the Leapfrog method, except that the velocity and position are calculated at the same value of the time variable. The mathematical formulation of the algorithm is shown below.

$$\begin{aligned} \mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2}\mathbf{a}(t)\Delta t^2 \\ \mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + \frac{\mathbf{a}(t) + \mathbf{a}(t+\Delta t)}{2}\Delta t \end{aligned} \tag{2}$$

While an implementation of the above algorithm essentially involves computing forces (which manifest themselves in the code in form of accelerations) and updating velocities and positions at each time step, additional book-keeping tasks must be performed in order to include domain decomposition. The principle behind domain decomposition is reducing the computation time by avoiding operations which whose results are a-priori known. It has been well established that the interaction forces between particles at a distance larger than a critical distance, r_c, are effectively zero. By selecting a suitable sector size, Domain Width $\geq r_c$, the number of interactions to be computed can be minimized to interactions between particles in the same and surrounding sectors, therby resulting in a decrease in wallclock times. Furthermore, each sector can then, at least in theory, be alloted to a separated thread in order to implement multi-threading. The overheads in domain decomposition arise from the requirement to maintain neighbour lists and to account for the particles and updating their sectors at each time step.

In the present work, periodic boundary conditions have been implemented along with domain decomposition. The following pseudo-codes exhibit the algorithms required to develop a Langevin dynamics code for interacting particles with periodic boundary conditions and domain decomposition.

Algorithm 1: Main

Data: Number of particles (N), Number of rows and columns (d)
Result: Compute RMS displacement and trajectories
Allocate and initialize all variables
Create neighbour list
Initialize particles and perform ordering
while $t < t_{max}$ **do**
  Update half-step velocities $\vec{v}_h(t + \Delta t/2) \leftarrow \vec{v}(t) + 0.5\vec{a}\Delta t$
  Update particle positions $\vec{x}(t + dt) \leftarrow \vec{x}(t) + \vec{v}_h(t + \Delta t/2)\Delta t$
  Impose reflecting boundary conditions
  Reorder particles according to sector
  Compute random forces acting on particle and update particle acceleration $\vec{a}$
  for *Sector* $S1 = 0, 1, \cdots, d^2 - 1$ **do**
    for *Each particle* i *in sector* $S1$ **do**
      for *All neighbours* $S2$ *of sector* $S1$ **do**
        for *Each particle* j *in sector* $S2$ **do**
          Compute distance between particles i and j
          if *Distance between* i *and* $j <$ *Critical distance* r_C **then**
            Compute interaction force $\vec{F}_i$
            Update particle acceleration $\vec{a}_i \leftarrow \vec{a}_i + \vec{F}_i/m$
  Update velocities $\vec{v}(t + dt) \leftarrow \vec{v}_h(t + \Delta t/2) + 0.5\vec{a}\Delta t$
  Update time $t = t + \Delta t$
  Write particle position $\vec{x}$ to 'Trajectories'
  Write RMS displacement $\sqrt{\frac{\sum_{i=1}^{N}(\vec{x}-\vec{x}_0)^2}{N}}$ to 'Means'

Algorithm 2: Initialization

Data: Particle position ($\vec{x} = \langle x, y \rangle$), Particle velocity ($\vec{u} = \langle u_x, u_y \rangle$), Particle acceleration ($\vec{a} = \langle a_x, a_y \rangle$)

Result: Set physical parameters and random initial position and velocity of particles

Initialize physical parameters

for *Particle* $i = 1, 2, \cdots, N$ **do**

- $x_i \leftarrow$ Uniform random number $\in (-L/2, L/2)$
- $y_i \leftarrow$ Uniform random number $\in (-L/2, L/2)$
- $u_{x_i} \leftarrow$ Normal random number $\in \mathcal{N}(0, \frac{K_B T}{m})$
- $u_{y_i} \leftarrow$ Normal random number $\in \mathcal{N}(0, \frac{K_B T}{m})$
- $a_{x_i} \leftarrow 0$
- $a_{y_i} \leftarrow 0$

Algorithm 3: Reflecting Boundary Conditions

Data: Particle position ($\vec{x} = \langle x, y \rangle$), Particle velocity ($\vec{u} = \langle u_x, u_y \rangle$), Box length ($L$)

Result: Updates particle position and velocity if outside the box

for $i = 1, 2, \cdots, N$ **do**

- **if** $x_i < -L/2$ **then**
  - $x_i \leftarrow -L - x_i$;
  - $u_{x_i} \leftarrow -u_{x_i}$;
- **if** $x_i > L/2$ **then**
  - $x_i \leftarrow L - x_i$;
  - $u_{x_i} \leftarrow -u_{x_i}$;
- **if** $y_i < -L/2$ **then**
  - $y_i \leftarrow -L - y_i$;
  - $u_{y_i} \leftarrow -u_{y_i}$;
- **if** $y_i > L/2$ **then**
  - $y_i \leftarrow L - y_i$;
  - $u_{y_i} \leftarrow -u_{y_i}$;

Algorithm 4: Assign particle sectors

Data: Particle position ($\vec{x} = \langle x, y \rangle$)

Result: Sector indices and number of particles in each sector

for *Particle* $i = 1, 2, \cdots, N$ **do**

- **if** *Particle within domain* ($-L/2 \leq x \leq L/2$ ***and*** $-L/2 \leq y \leq L/2$) **then**
  - Assign sector to the particle;
  - Number of particles in sector $\leftarrow$ Number of particles in sector + 1;
- **else**
  - Put the particle in garbage bin;
  - Number of particles in garbage $\leftarrow$ Number of particles in garbage + 1;

Algorithm 5: Ordered list of particles

Data: Particle position ($\vec{x} = \langle x, y \rangle$)), Particle velocity ($\vec{u} = \langle u_x, u_y \rangle$)
Result: Particles ordered according to the sector they are in.
for *Particle* $i = 1, 2, \cdots, N$ **do**
    Sort the particles according to their sector.

Algorithm 6: Neighbour list

Data: Neighbour list
Result: List of neighbours of each sector
for *Sector* $k = 1, 2, \cdots, d^2 - 1$ **do**
    for *All surrounding sectors* **do**
        if *Valid sector* **then**
            Add to neighbour list

3 Results and Discussion

(1) The mass, space and time scales can be defined for the Langevin equation in terms of the system parameters as follows:

$$Massscale\,[\mathrm{kg}] = m$$

$$Timescale\,[\mathrm{s}] = \frac{m}{\gamma}$$

$$Spacescale\,[\mathrm{m}] = \frac{\sqrt{m}\cdot c}{\sqrt{\gamma^3}}$$

When the domain is bounded within a closed box of dimension $L \times L$, the length scale gets scaled and we obtain an additional non-dimensionless term, $\frac{\sqrt{m}\cdot c}{L\cdot\sqrt{\gamma^3}}$. Physically, this results in the saturation of the root mean square displacement and a plateau is obtained beyond the diffusive region.

For Langevin dynamics, the mean square displacement can be given by the following expression:

$$\langle x^2 \rangle = \frac{2k_BT^2}{\gamma}t + \frac{2k_BT}{\gamma^2/m}(1 - e^{-\gamma t/m}) \tag{3}$$

For long time scales, we obtain the following behaviour:

$$\langle x^2 \rangle = \frac{2k_BT}{\gamma}t$$

and, for the short time scales, the general expression reduces to the following:

$$\langle x^2 \rangle = \frac{2k_BT}{m}t$$

Therefore, since m and k_BT play a role only in scaling and do not affect the nature of the stochastic equation, we can set them both to unity as this would not affect the behaviour that we want to observe which relates to the time scales.

(2) Parallelisation was implemented using OpenMP directives and we observe a change in the structure of the code in order to reap the maximum benefits of parallelisation. In the present case, two threads were used with one thread handling the computation and the other thread handling the writing to disk. This was done since writing is the most time consuming process in the code and with the vectorized code implementation, the computational time is no more a stumbling block in the performance. In order to implement the parallelisation, we must ensure that the race condition must not arise. With this aim, the time variable, t, was listed using the *lastprivate* & *lastprivate* clauses and temporary variables *x_temp* & *y_temp* were defined. Moreover, since writing and computing are now handled by separate threads, the writing task does not necessarily need to follow the computation. In fact, we see from the code that the OpenMP section responsible for writing data to disk precedes the OpenMP section looping over the particles to run the velocity verlet algorithm.

A direct comparison of the two codes can be performed in terms of the speed-up. The obtained results have been presented in table **??**

(3) (a) One of the possible tests to verify the particle distribution is to divide the box into sectors and count the number of particles in each sector. Since the particles have been distributed uniformally, all the boxes should contain almost the same number of particles. This test has been performed using a

(b) Another possible test to check the proper implementation of the boundary conditions is to

Table 1: Speed-up and efficiency of parallelisation.

Optimization	Wall Time [s]		Parameter	
	Serial	Parallel	Speed-up	Efficiency
O0	1.158	1.015	1.141	0.779
O3	0.893	0.5730	1.559	0.570

count the total number of particles in the box at t_0 and T_{max}. Provided that the particles have reasonable velocities and accelarations, the number of particles lost during the simulation must be reasonably small.
(c) The root mean square displacement of the particles should show two distinct phase - a ballistic phase with root mean square dispacement proportional to t and a Brownian motion phase with root mean square displacement proportional to $t^{1/2}$. Furthermore, when a box of sufficiently large dimensions is used, the root mean square displacement should reach a constant plateau after the ballistic and Brownian phases.

(4) The velocity autocorrelation function can be defined as

$$A(\tau) = \frac{\langle v(t) \cdot v(t-\tau) \rangle}{\langle v(t) \cdot v(t) \rangle}$$

From the requirement of equipartition of energy at equilibrium, it can be shown that the autocorrelation function, $A(\tau)$ decays as $\exp -\gamma t/m$. To observe this behaviour, the following implementaion of the autocorrelation function was implemented:

$$A(t) = \frac{\langle v(t) \cdot v(0) \rangle}{\langle v(t) \cdot v(t) \rangle}$$

This implementation of the velocity correlation function was implemented using the following additional lines of code:

```
! Declare variables to save initial velocity for correlation
double precision, allocatable, dimension(:) :: vx0,vy0
.
.
! In the subroutine initialize\_particles, save the initial velocities in vx0 and vy0
vx0(i)=vx(i)
vy0(i)=vy(i)
.
.
! Open additional file to save the velocity correlation function
open(13,file='velocity_correlation')
.
.
.
do while(t.lt.t_max)
   .
   .
   write(13,*) t,(sum(vx*vx0 + vy*vy0)/real(n,8))/(sum(vx*vx + vy*vy)/real(n,8))
end do
```

The additional line in the code writes the velocity correlation at each time step to the file *velocity_correlation* and the values were plotted against time. As shown in figure **??**, the velocity correlation function follows the desired exponentially decaying trend but the exact values shows a slight departure from the exact expected curve. This fluctutaion is a result of the statistical nature of the function and the trend confirms the expected behaviour. As shown in figure **??**, the same

Figure 1: Computed velocity correlation function and expected behaviour.

velocity correlation function has been plotted on a semilog scale for time range from 0 s to 2.5 s and we can observe that the function follows the expected behaviour, i.e. $e^{-\gamma t/m}$, where γ and m are equal to zero in the present study.

4 Conclusion

In the present exercise, the impact of parallelisation on the code was analysed using speed-up and efficiency and a number of test cases were developed. It was observed that the box size dictates the behaviour observed in the simulations and therefore the box must be sufficiently large and the time sufficiently long to observe the expected behaviour. A number of test cases were developed to analyse the performance and the velocity correlation function was computed and the expected behaviour was observed.

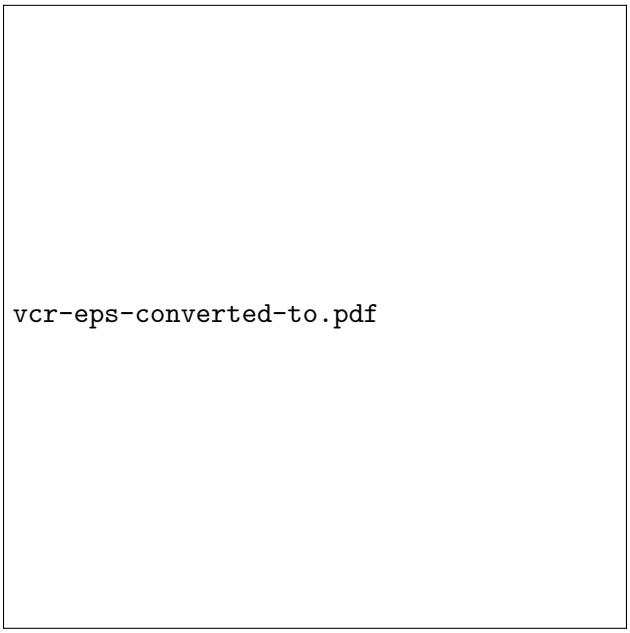

Figure 2: Semilog behaviour of computed velocity correlation function.