Study of Algorithms to find the Nearest Neighbours under Periodic Boundary Conditions

Pritipriya Dasbehera
Roll no. 2111086

August 17, 2022

under the supervision of *Dr. Nishikanta Khandai* and Project Report submitted to the School of Physical Sciences, **National Institute of Science Education and Research**, Bhubaneswar

Abstract

Simulation of a large number of bodies in cosmology, hydrodynamics and aerodynamics involves considering a large number of interactions between them which almost always requires information of the nearest neighbours. The Periodic Boundary Conditions are used to simulate an infinite-size system by studying the dynamics of only a smaller part of it. This study is based on the implementation of two algorithms to find nearest neighbours: brute-force method and grid/sub-cell method, whose time-complexity is shown to be $O(n^2)$ and $O(n)$ respectively.

1 Introduction

In the case of hydrodynamics and aerodynamics, where short-range interaction is involved, only nearest neighbours can exert forces on a body. In the case of cosmology, where long-range gravitational interaction is involved, ideally, the interaction of each body with every other body should be considered. But that needs computation time of the order $O(n^2)$, which is not feasible. Thus, the widely used approximation is to find the nearest neighbours of each body and consider the interaction between them. As for the bodies farther away, they can be clumped into groups and approximated as single bodies at the centre of mass of the groups [figure 1]. This reduces the calculation cost considerably depending on the algorithms used.

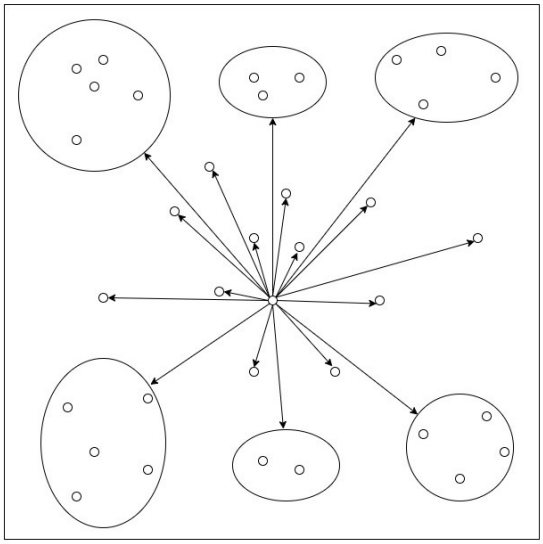

Figure 1: Approximating a long range interaction using nearest neighbours and grouping, in 2-D space

To simulate an infinitely large system, **Periodic Boundary Conditions** (PBC) are often used. Only a small unit of the system needs to be accounted for under PBC. The system is assumed to consist of infinitely many copies of

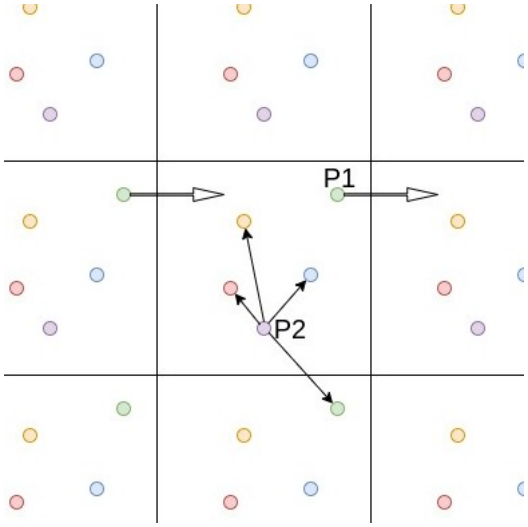

Figure 2: Periodic Boundary Conditions - When P1 moves out of the cell from the right, it enters from the left. Also the closest image of P1 for P2 is not in the original unit but in the sub-cell below it.

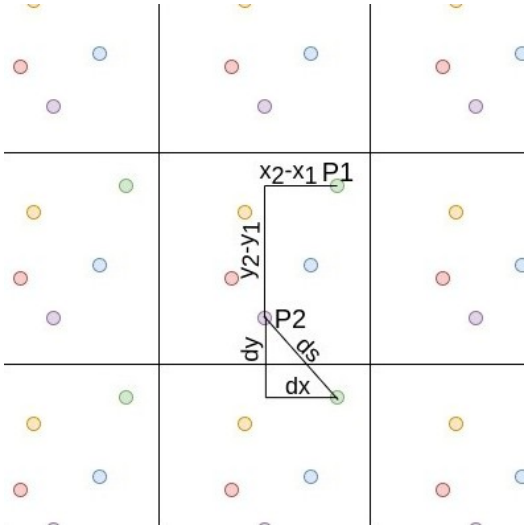

Figure 3: Example of distance calculation in 2D where $ds = \sqrt[2]{dx^2 + dy^2}$ .Here, $dx = |x_2 - x_1| < l/2$ but $dy = l - |y_2 - y_1|$ as $|y_2 - y_1| > l/2$

these small units. When a body leaves the original unit, it reappears with the same velocity from the opposite direction. And each particle interacts only with the closest images of the remaining particles [Figure 2].

2 Methodology

2.1 *Distance calculation*

If the two coordinates are (x_1, y_1, z_1) and (x_2, y_2, z_2) and length of cell is l then the effective distance is

$$ds = \sqrt[2]{dx^2 + dy^2 + dz^2} \tag{1}$$

where

$$dx = \begin{cases} |x_2 - x_1| & \text{if } |x_2 - x_1| < l/2 \\ l - |x_2 - x_1| & \text{if } |x_2 - x_1| \geq l/2 \end{cases} \tag{2}$$

similarly for dy and dz.

But using if-else conditions in a program breaks the flow which makes parallelization difficult and may increase time, to avoid it the following equation is used

$$dx = a_x l + (1 - 2a_x)|x_2 - x_1| \tag{3}$$

where[1]

$$a_x = \left\lfloor \frac{2|x_2 - x_1|}{l} \right\rfloor = \begin{cases} 0 & \text{if } |x_2 - x_1| < l/2 \\ 1 & \text{if } |x_2 - x_1| \geq l/2 \end{cases} \tag{4}$$

similarly for dy and dz. From here on distance refers to effective distance unless stated otherwise.

2.2 *Brute-force method*

This method involves calculating the distances of each body to every other body and keeping track of the closest 'k' bodies to it. This involves n^2 operations [2] where n is the total number of bodies. Thus this method should have time complexity $O(n^2)$.

2.3 *Grid method*

This method involves dividing the cell into a grid of smaller sub-cells and keeping track of the bodies in each sub-cell. If there are enough bodies in each sub-cell, then the nearest 'k' bodies are present in the same sub-cell or sub-cells immediately surrounding it. If not, then they are in the

[1] $\lfloor \cdot \rfloor$ is the Greatest Integer Function

[2] This is independent of 'k' the number of nearest neighbours required

2^{nd} immediate surrounding sub-cells or the 3^{rd} immediate surrounding and so on [Figure 4].

For each body, about a constant number of sub-cells are checked for nearest neighbours. Thus, its time complexity should be $O(n)$.

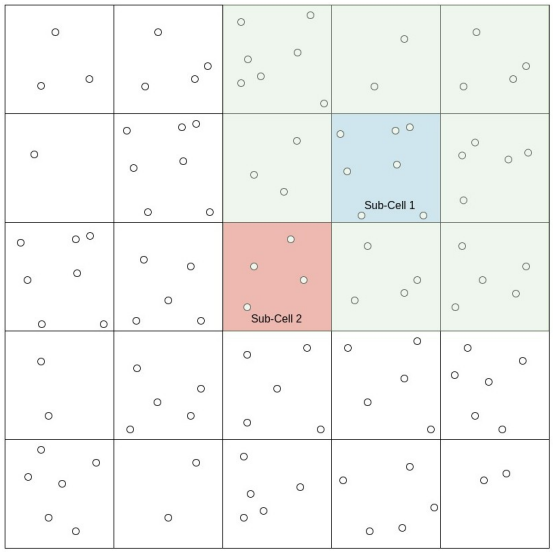

Figure 4: If a point in sub-cell 1 is considered with k=10, then the 3x3 subgrid(shaded), immediate surrounding, containing it will for sure have the nearest neighbours.
For a point in sub-cell 2 and k=50, the 5x5 grid(whole grid), 2^{nd} immediate surrounding, will have the nearest neighbours.

3 Results

3.1 *Brute-force method*

The plot of the program run-time(T) against the number of points (n) [Figure 5] is not linear. To find the order of time complexity log plot is used. The log graph [figure 7] is straight line thus the dependence is of polynomial form (n^b) and not exponential (e^n).

If the general form is considered to be

$$T = an^b \tag{5}$$

then

$$log(T) = log(a) + blog(n) \tag{6}$$

which is a straight line for log(T) vs log(n) plot and has slope b, which is the required order.

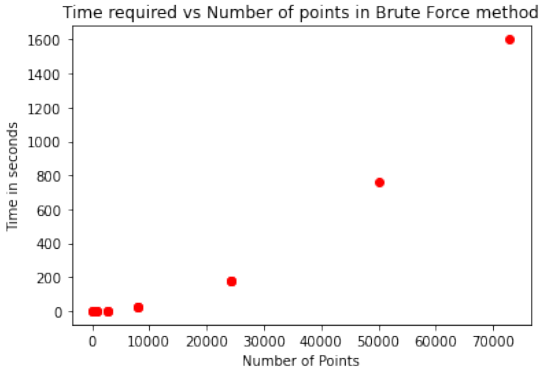

Figure 5: Run-time against number of points for Brute-force method

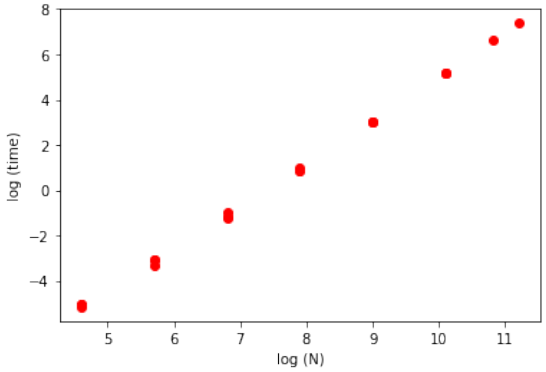

Figure 6: log of run-time vs log of number of points for Brute-force method

The slope of best fit line is calculated (by iterative linear regression) to be 1.86, nearly equal to 2. *This confirms time complexity of Brute-force method to be* $O(n^2)$.

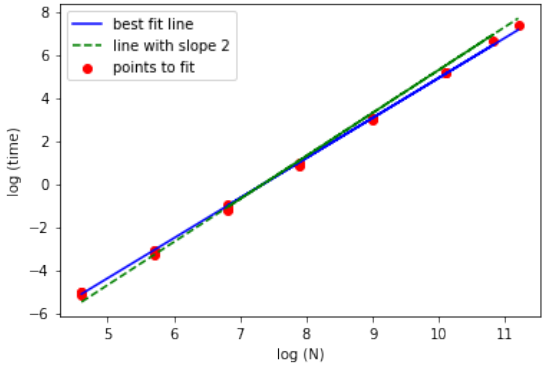

Figure 7: Best fit line for log(T) vs log(n) graph

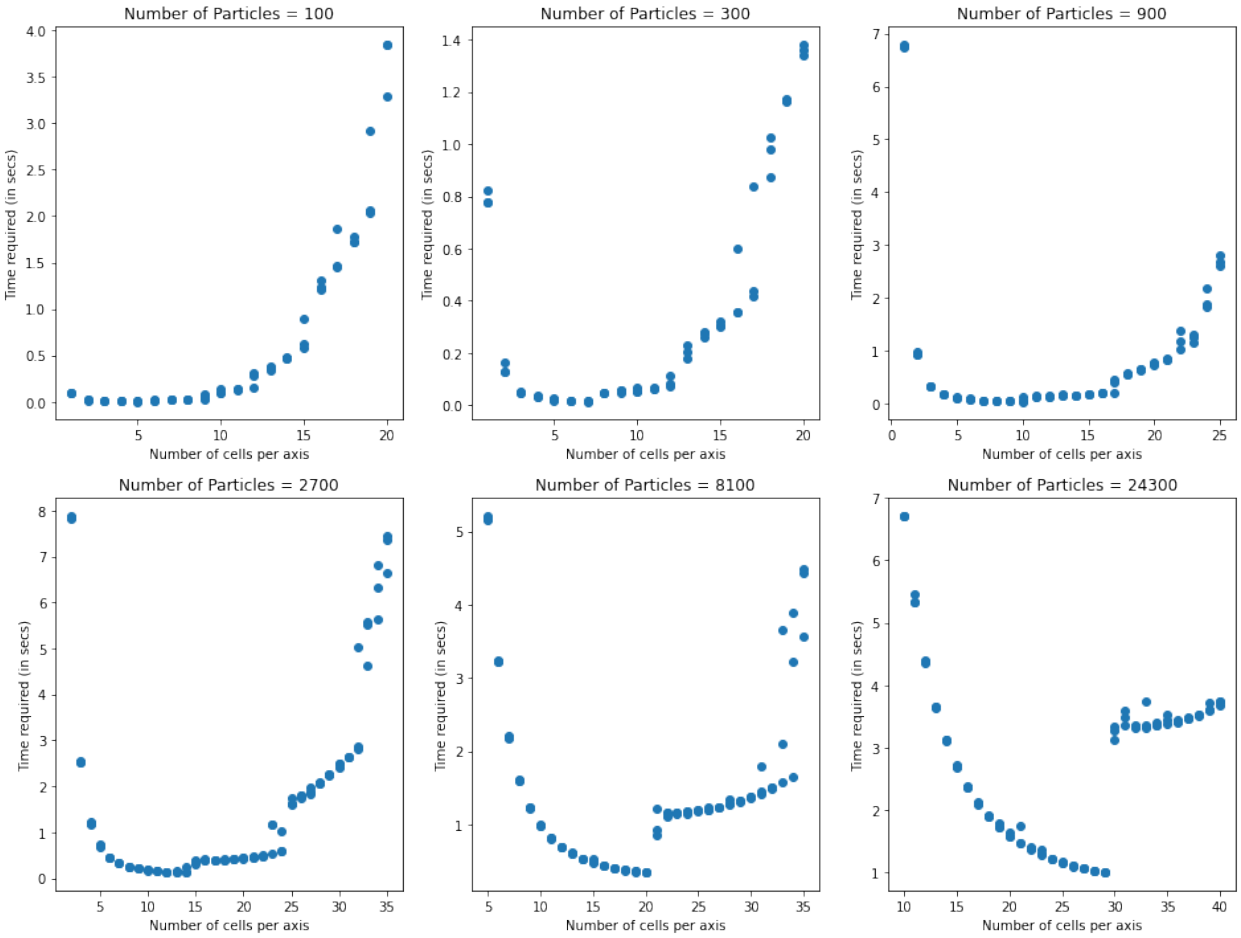

Figure 8: Run-time vs number of sub-cells per axis for different number of points

3.2 *Ideal number of sub-cells per axis for Grid method*

The grid method requires dividing the cell into sub-cells [Figure 9]. If the number of sub-cells is too many, then each sub-cell contains less number of points/bodies. A larger sub-grid will have to be searched to find the nearest neighbours. If the number of cells is too few, then each sub-cell has many points, and a larger number of calculations need to be done. So there exists an ideal number of sub-cells which requires the least runtime. This is evident from the figure 8.

By taking the ideal number of sub-cells (the number of sub-cells per axis with the least runtime) from figure 8, a plot can be made to find the relation between the number of sub-cells per axis and the total number of points/bodies.

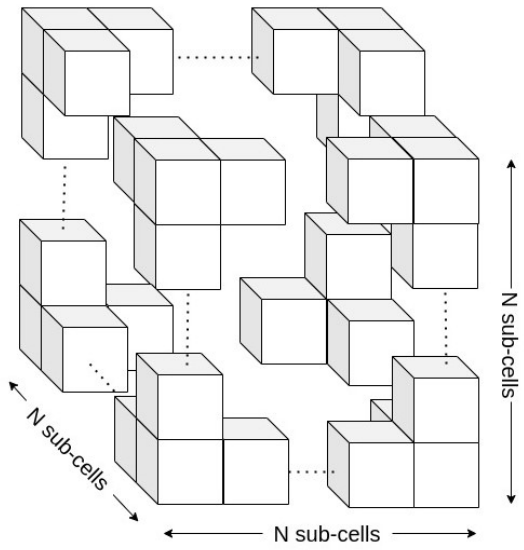

Figure 9: N sub-cells per axis means total N^3 sub-cells

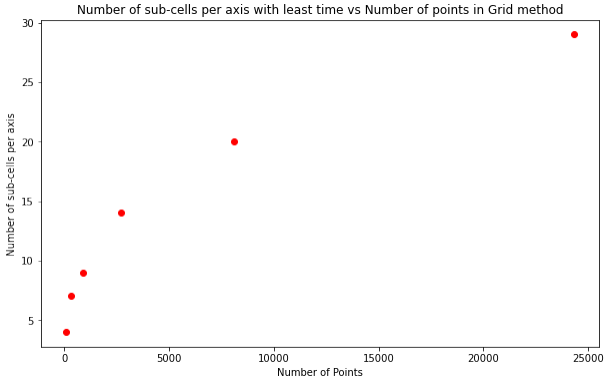

Figure 10: Number of sub-cells per axis vs Total number of points/bodies

The plot [figure 10] of sub-cells per axis vs the number of points is not linear, but the log plot [figure 11] is linear. Thus the relation is of polynomial form.

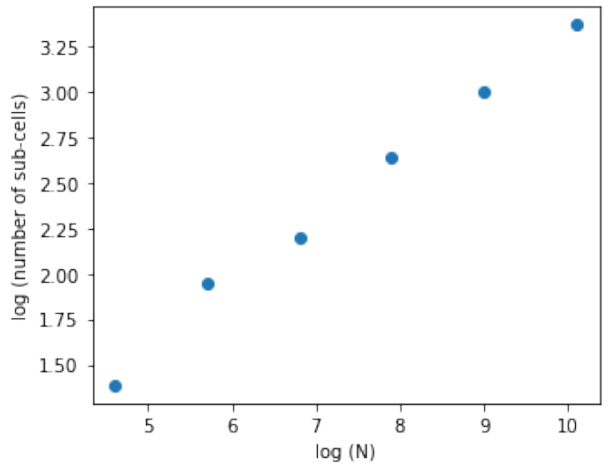

Figure 11: log of number of sub-cells per axis vs log of number of points

Considering the dependence to be

$$N = an^b \tag{7}$$

where N = Number of sub-cells per axis and n = Total number of points/bodies.

$$log(N) = log(a) + blog(n) \tag{8}$$

From the slope and intercept of best fit line [Figure 12], $b = 0.33$ and $a = 1$ i.e.

$$N = n^{0.33} \tag{9}$$

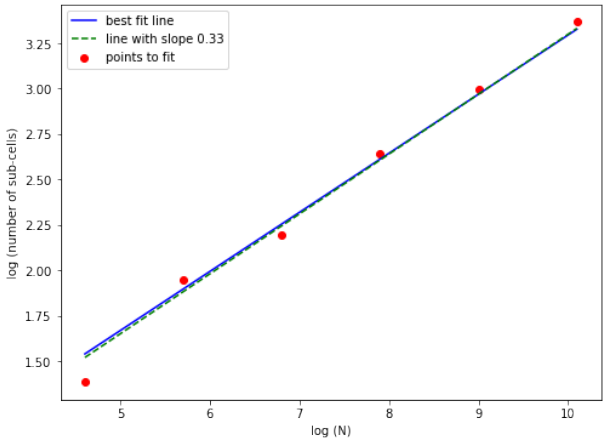

Figure 12: best fit line for the log(sub-cells) vs log(n) plot has slope nearly 0.33 and intercept 0

3.3 *Grid method*

Now that the optimum number of sub-cells is known, the run-time of the grid method for different numbers of points/bodies can be plotted[Figure 13], which turns out to be linear. *This confirms the time complexity of the grid method to be* $O(n)$.

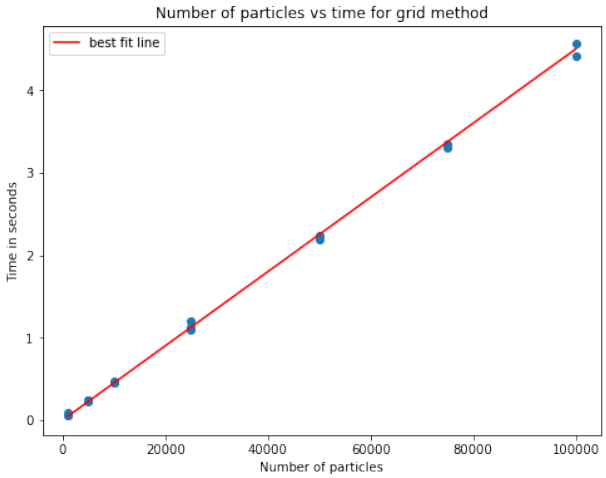

Figure 13: best fit line for run-time vs number of points

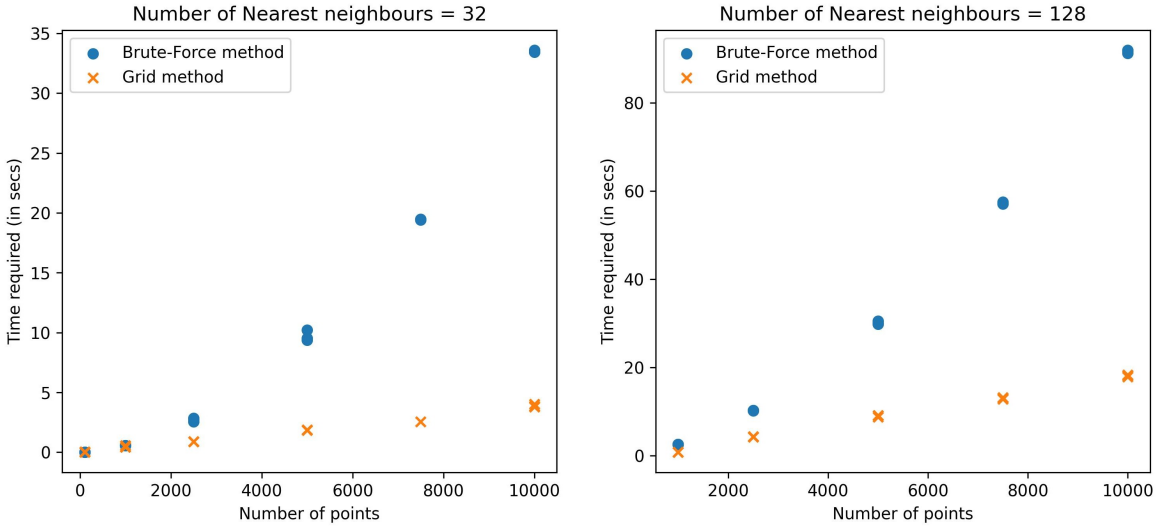

Figure 14: Comparing the two methods for different number of points/bodies

4 Conclusion

Comparing the two methods[Figure 14] we can observe that both methods perform equally well when the number of points/bodies is less. As the number of points/bodies increases, the difference in time becomes more significant and the superiority of the grid method becomes evident.