Report - Nearest Neighbour Algorithm

1) **Functions :**

read_xyz_file	Read and convert the text file containing the 3d coordinates of Ar particles into a matrix.
flatten_matrix	Change matrix into 1d array instance.
average_neighbour	Calculate the average count of neighbours.
brute_force	Compute the distance between particles using coordinates and return the count of neighbours for each particle in a form of 1d array. Implementing grid distribution and MPI_Reduce.
write_summary	Print out the number of threads used, total number of particles, the running time, average number of neighbours, maximum neighbours and minimum neighbours in a sample.

2) **1D Block Distribution :** Tasks are divided according to blocks of matrix with number of rows equal to the total number of threads being used minus by 1. While the number of columns being the number of dimensions of particles which is 3. Reason for choosing to divide the tasks by the total number of threads being used minus by 1 is to allow for one root thread to not be involved in any computation. The root thread will only be used for the printing of results and collecting data from other threads. Below is an illustration to understand this implementation in detail. Say we have 4 threads and our data has 10 particles in 3d space. Firstly, we exclude one thread, say Thread 0 to be the root thread. Then, Thread 1 will compute nearest neighbour for particles in row 0 to row 2, Thread 2 will compute nearest neighbour for particles in row 3 to row 5 and Thread 3 will compute nearest neighbour for particles in row 6 to row 9 (Extra rows will be distributed for the last remaining thread). After completing the computation, we used MPI_Reduce to send the information to the root thread to be summarised and written into a text file.

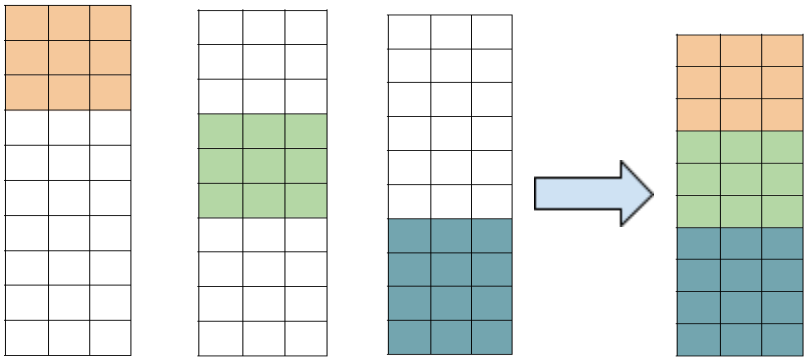


Diagram 1 : Task distribution for Thread 1 (orange), Thread 2 (green), Thread 3 (blue) and processed information collected by Thread 0 (right)

3) **1D Cyclic Distribution :** Tasks are divided according to rows cyclically to all threads excluding the root thread. While the number of columns being the number of dimensions of particles which is 3. Reason for excluding the root thread is to allow for one thread to not be involved in any computation. The root thread will only be used for the printing of results and collecting data from other threads. Below is an illustration to understand this implementation in detail. Say we have 4 threads and our data has 10 particles in 3d space. Firstly, we exclude one thread, say Thread 0 to be the root thread. Then, Thread 1 will compute the nearest neighbour for particles in row 0, row 3, row 6 and row 9. Thread 2 will compute the nearest neighbour for particles in row 1, row 4 and row 7. Thread 3 will compute the nearest neighbour for particles in row 2, row 5 and row 8. After completing the computation, we used MPI_Reduce to send the information to the root thread to be summarised and written into a text file.

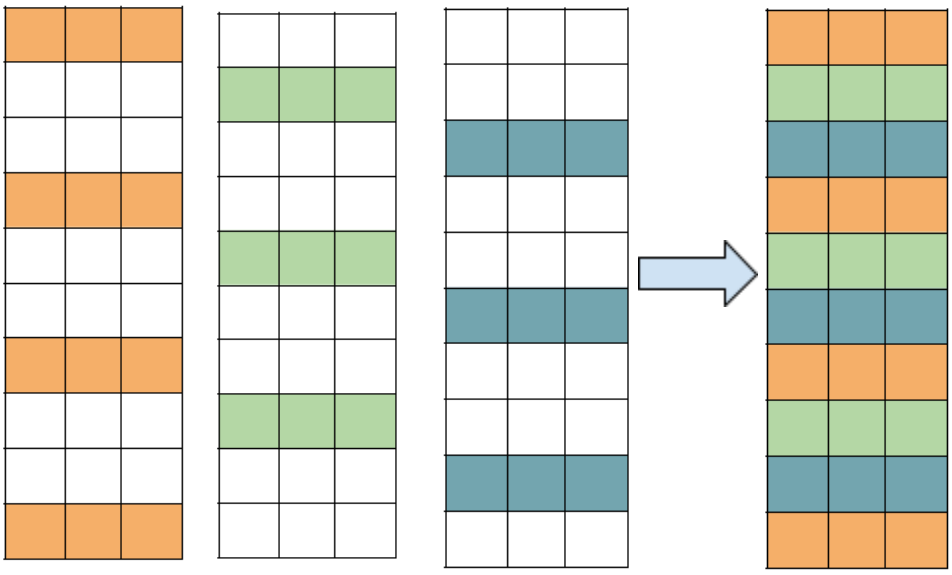


Diagram 1 : Task distribution for Thread 1 (orange), Thread 2 (green), Thread 3 (blue) and processed information collected by Thread 0 (right)

4) **Investigation :** To study the reading of average neighbours, maximum neighbours and minimum neighbours from the collected results using nearest neighbour algorithm. The algorithm will be tested on three different particle systems containing 120 particles with cube length of 18 A, 10549 particles with cube length of 100 A and 147023 particles with cube length of 200 A. Also, in determining the ideal approach for MPI implementation, we will compare the running time for using both 1d Block approach and 1d Cyclic approach.

5) Results

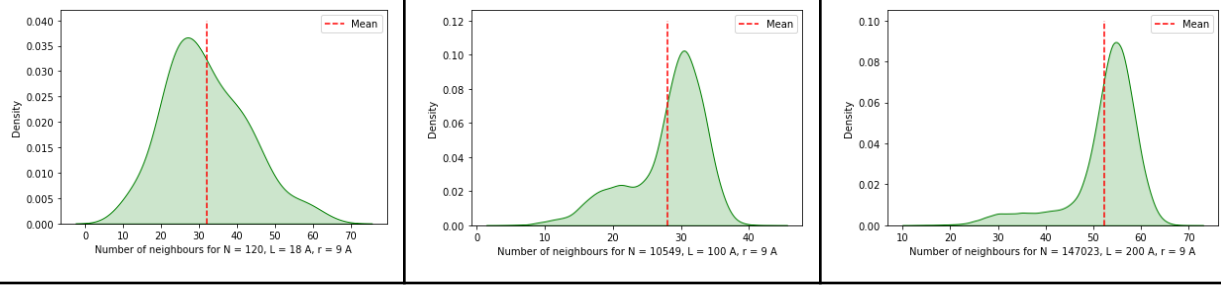


Diagram 2 : Distribution diagram for the count of neighbours in each samples

Number of Particles	120	10549	147023
Size of Radius (A)	9.0	9.0	9.0
Average neighbour	31.9833	28.0626	52.3069
Maximum neighbour	63	43	71
Minimum neighbour	10	4	12

Table 1

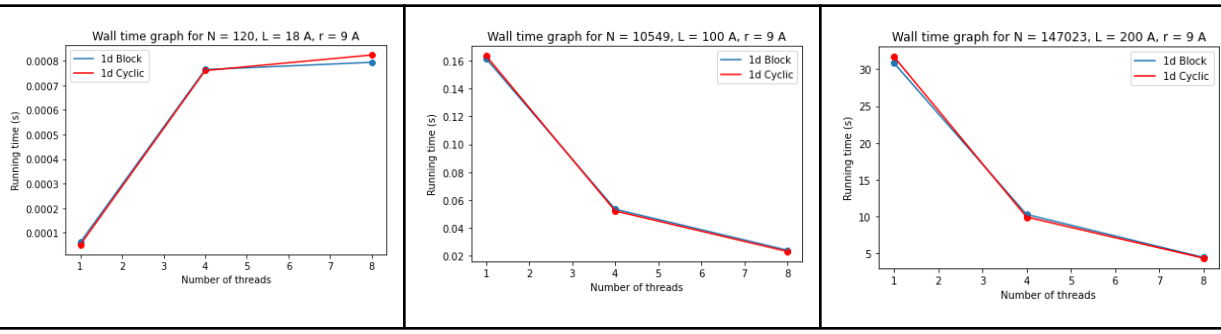


Diagram 3 : Wall time graph to show the trend of runtime against number of threads

Number of particles	**Running time with respect to number of threads (s)**					
	1		4		8	
	1d Block	**1d Cyclic**	**1d Block**	**1d Cyclic**	**1d Block**	**1d Cyclic**
120	0.000050	0.000061	0.000760	0.000764	0.000824	0.000794

10549	0.163507	0.161549	0.052144	0.053394	0.023083	0.023966
147023	31.68530	30.8350	9.91694	10.2571	4.34013	4.4353

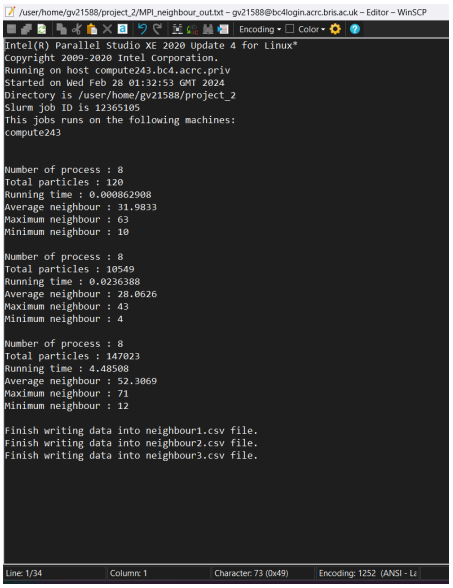
Table 2

6) Summary

a) Based on Table 1, we can observe that for a system that is occupied by more particles, the frequency for the count of neighbours will be skewed to the left. This implies that a system with more particles has higher reading of mode and median compared to the sample mean number of neighbours. However, more particles does not mean that there will be a significant increase of neighbouring particles. This is because there is no significant increasing or decreasing trend of neighbouring particles count in relation to increasing number of particles. For example, the average neighbouring particles count for a system with 10549 particles is the smallest compared to any others.

b) Based on Table 2, we can observe that for both systems with 10549 and 147023 particles, there will be a significant improvement in the runtime of the computations as the number of threads increases. Also, we can observe how 1d Block approach is slightly better compared to 1d Cyclic approach when we work on larger particle systems. However, since the measure of performance between the two approaches do not differ much we cannot tell whether 1d Block is better than 1d Cyclic in general for this task. This scenario however does not translate well for smaller particle systems. Since, we are expected to get an increase in overall runtime as we use more threads. This can be due to the loads of sharing small information from different threads which can outweigh the computation of the nearest neighbour itself.

7) Output example


/user/home/gv21588/project_2/MPI_neighbour_out.txt – gv21588@bc4login.acrc.bris.ac.uk – Editor – WinSCP
Encoding
Color
Intel(R) Parallel Studio XE 2020 Update 4 for Linux*
Copyright 2009-2020 Intel Corporation.
Running on host compute243.bc4.acrc.priv
Started on Wed Feb 28 01:32:53 GMT 2024
Directory is /user/home/gv21588/project_2
Slurm job ID is 12365105
This jobs runs on the following machines:
compute243
Number of process : 8
Total particles : 120
Running time : 0.000862908
Average neighbour : 31.9833
Maximum neighbour : 63
Minimum neighbour : 10
Number of process : 8
Total particles : 10549
Running time : 0.0236388
Average neighbour : 28.0626
Maximum neighbour : 43
Minimum neighbour : 4
Number of process : 8
Total particles : 147023
Running time : 4.48508
Average neighbour : 52.3069
Maximum neighbour : 71
Minimum neighbour : 12
Finish writing data into neighbour1.csv file.
Finish writing data into neighbour2.csv file.
Finish writing data into neighbour3.csv file.
Line: 1/34
Column: 1
Character: 73 (0x49)
Encoding: 1252 (ANSI - La