University of BRISTOL

School of Physics

Name:	Wesley Richardson
Degree Course:	Physics with Scientific Computing BsC
Student Number:	U2164839

Assessed Exercise: Nearest neighbor algorithm optimization implementing MPI

Introduction

The code nearest_neighbors.cpp computes the answer to the question; for each atom in a set of atoms, how many atoms are within some distance, R, of that atom? There are many approaches to solve this problem, and this code implements 6 of these. The goal of this task is to compare the speed of different algorithm implementations.

1: Single-Threaded Brute force

This algorithm compares each atom, A, to each other atom, B if A is close to B, then B is a neighbor of A, and the neighbor list is increased at index A.

2: MPI Cyclic

A round-robin style implementation of MPI within the brute force algorithm. The logic is the same, but worker threads are only responsible for atoms with indices:

{ThreadID, ThreadID+NumThreads, ThreadID+2*NumThreads, ...}.

3: MPI Chunked

A sequential chunk MPI worker distribution implementing the brute force algorithm. Again, algorithm logic is unaltered, but work is distributed within the primary for loop. The worker threads are responsible for atoms with indices:

{ThreadID*ChunkLength, ThreadID*ChunLength+1, ..., (ThreadID+1) *ChunkLength –1}

4: Cell List

A more optimized method which skips unnecessary comparisons for atoms. The position domain is split into cubes of length R (where R is the maximum distance to be considered a neighbor). This means that you only need to perform position calculations on atoms if the other atom is within the same cell or a neighboring cell, elsewise it would be impossible for the other cell to be a neighbor.

5: MPI Cell List

A simple round robin implementation of MPI into the cell list algorithm. Worker threads are distributed over the atoms of the current cell. Worker threads are responsible for atoms with indices:

{ThreadID, ThreadID+NumThreads, ThreadID+2*NumThreads, ...}.

6: Single-Threaded Double Increment

Half the comparisons are skipped by changing the logic such that if atom A is a neighbor of B, then B is also a neighbor of A.

Solutions to the provided data files:

To validate all the methods, the neighbor results for all data files were compared across all methods and were found to agree.

Argon120.xyz; mean number of neighbors: 31.98, maximum number of neighbors: 63, minimum number of neighbors: 10.

Argon10549.xyz; mean number of neighbors: 28.06, maximum number of neighbors: 43, minimum number of neighbors: 4.

Argon147023.xyz: mean number of neighbors 52.31, maximum number of neighbors, maximum number of neighbors: 71, minimum number of neighbors: 12.

Performance Comparison/Discussion:

The 120-particle data set is unique, since the world space is small enough that the cell list model will only divide the world into 8 cells (2 cells per dimension, RC = 9, L=18). This means that the cell list model doesn't offer performance benefits here.

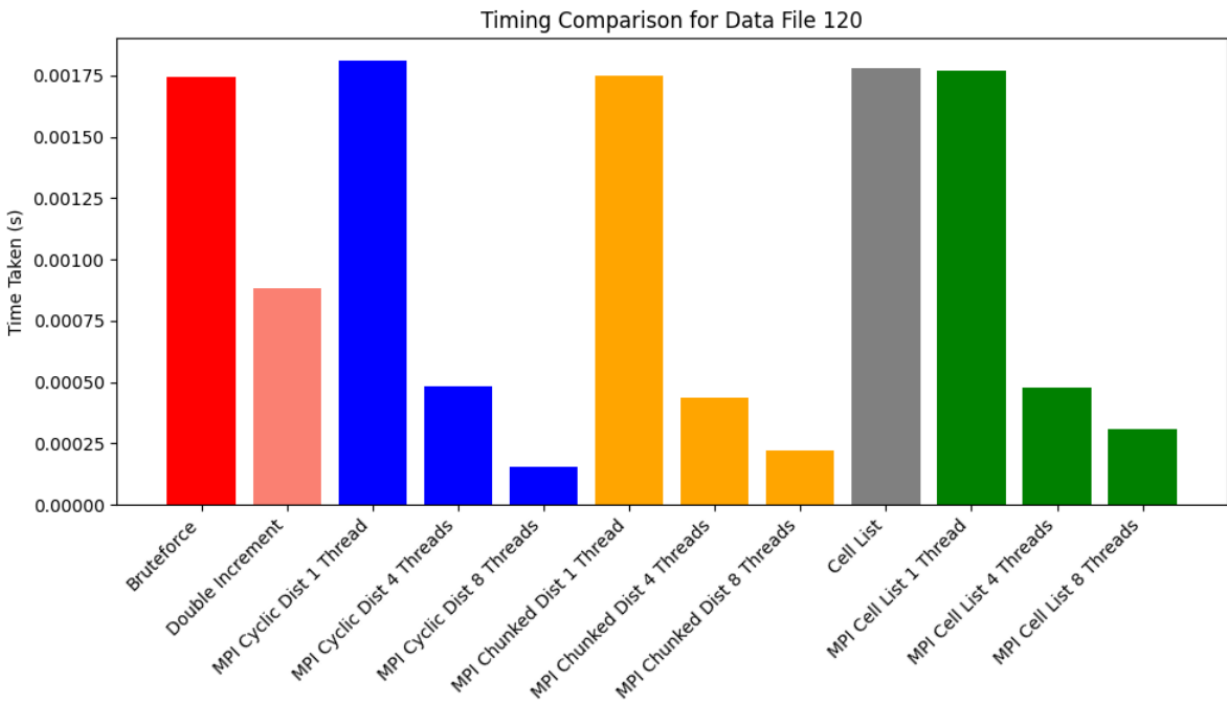

Fig 1 Algorithm comparison for 120 atom data set.

Here we can see that double increment is roughly twice as fast as Brute force, which is as expected. It is possible to implement double increment into all other methods making them roughly twice as fast.

The MPI overhead is negligible, considering this is the smallest data set, where the overhead would be most significant.

We can see that the cyclic thread distribution benefitted more from adding additional threads because the workload is distributed more evenly. In the chunked distribution, the program will only be as fast as the slowest thread, which may have been given more difficult computations to perform. The cell list received the least benefit from additional threads in this

Cell list offers significantly improved in larger world sizes where it is possible to skip more computations. Larger world sizes also offer greater performance returns when running the algorithm with additional threads.

Even on small world sizes, the overhead of the cell list model is negligible. Making it the suitable algorithm for most cases. The logic would still work for world sizes even smaller than 2*Rc.

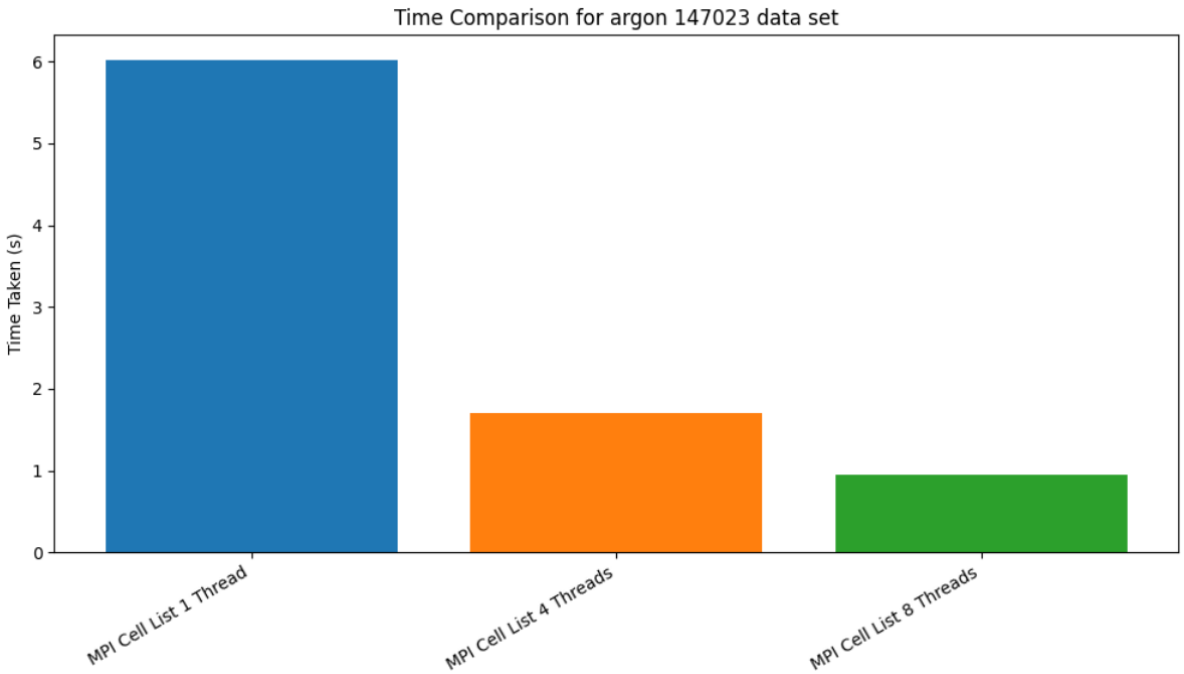

Fig 2: Multithreaded cell list implementation comparison for a larger dataset.

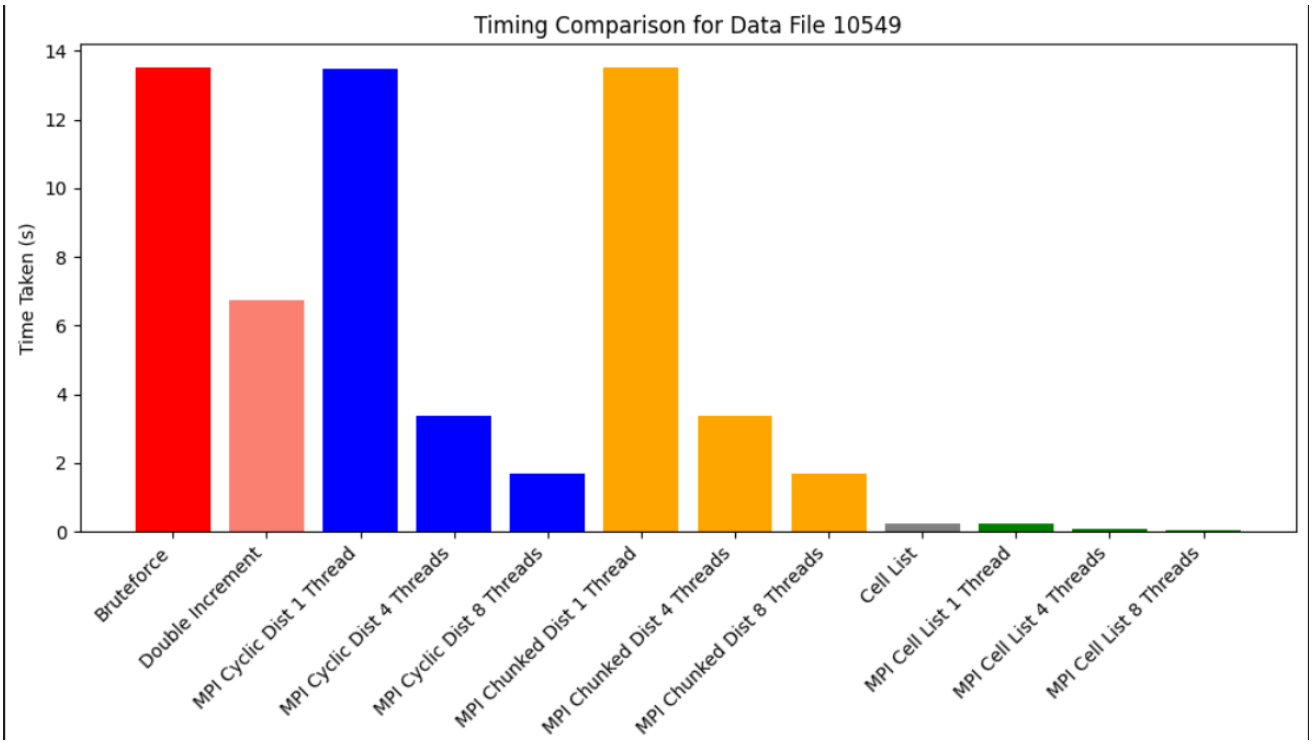

Fig 3: Algorithm Comparison for a larger data set containing 10549 atoms.