CSE 6730 Project Report: Parallel Molecular Dynamics Simulation

Mary Benage, Andrew Champion,
Zhejiang Dong, Mohan Rajendran

Georgia Institute of Technology

April 27, 2012

Introduction

Molecular dynamics simulations allow researchers to investigate phenomena emergent in systems at the molecular scale, such as protein assembly and formation of hydrodynamic flows. The goal of this project is to implement a cluster-parallelized molecular dynamics simulation (PMDS) for periodic systems on the order of thousands of entities. To meet this goal, this project's objectives are to implement a simulation of Lennard-Jones potentials [3] and bonds between atoms and parallelize this simulation with MPI by distributing atoms and decomposed blocks of the force matrix to nodes in the system. We used only the short-range force models and ignore the Coulomb forces to limit the computational effort and to meet the criteria of [4] to parallelize the algorithm. To validate this implementation, a model of lipid-like entities in a solution was simulated and the results compared with theoretical expectations and simulation outcomes from the LAMMPS Molecular Dynamics Simulator [4, 1].

Background

Molecular dynamics simulations are used to compute equilibrium and transport properties of interacting molecules that obey the laws of classical mechanics. A typical simulation has a timestep around femptoseconds and therefore tens, hundreds or thousands of timesteps are needed [4]. The first molecular dynamic paper was in 1956 by Alder and Wainwright at Livermore National Laboratory. Molecular dynamics simulations have since been used essentially as real experiments aimed at better understanding properties of materials such as liquids and solids. The approach for these numerical experiments are as follows: prepare the sample with the proper structure, equilibrate the system by solving Newton's equations of motion for the N atoms until there is no change, then perform the experiment [2].

The individual force equations for each atom are derived from the potential energy functions, such as the Lennard-Jones potential used for calculating van der Waals forces. MD simulations extensively only solve the short-range forces – bonded forces and van der Waals – and ignore the long-range Coulombic forces [4, 1]. MD simulations typically have periodic boundary conditions and a cutoff distance for the distance between each pair of atoms, i and j. The cutoff distance is used to limit the number of calculations, thus if the distance between two atoms is greater than the cutoff distance, their force is not calculated. The distance between two atoms is used to solve the Lennard-Jones potential (or other potentials) and update the forces for each atom [2]. The most computationally intensive part of MD simulations is the computation of the forces. There are two common methods to reduce computation by limiting the number of times the algorithm has to check that atoms are within the cutoff distance: the neighbor list and link-cell method or a combination of the two [4, 1]. Once the forces are determined, individual atoms' motion is found by integrating Newton's equation of motion, commonly using the Verlet algorithm [2].

Molecular dynamics are considered inherently parallel since the force and position updates can be done simultaneously. We will follow the methods proposed by [4] to parallelize. The two basic assumptions for this method are that the forces are limited in range or are only short-range forces, and the atoms can undergo large displacements. The three basic methods proposed for parallelization are either atom decomposition, force decomposition, or spatial decomposition. Atom decomposition assigns a subset of atoms to each processor and that processor is responsible for calculating the total forces on each atom and updating the velocity and positions of it's atoms. Force decomposition assigns each processor with a fixed set of atom pairs that it is responsible for calculating the force. There is then a fold operation and each processor is then responsible for updating the position and velocity of a

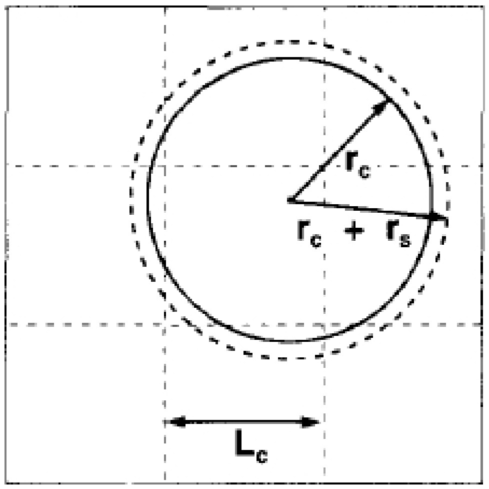


Figure 1: Illustration of relationship between cut-off radius r_c, skin distance r_s, and cell length L_c.

subset of atoms (it is the same as in the atom decomposition). Lastly, spatial decomposition assigns a fixed space to each processor. At each timestep, the processor calculates force and updates position and velocity for each atom in its associated space. In this decomposition, the atoms will move among processors. In summary, the atom and force decompositions are analogous to a Lagrangian frame of reference and spatial decomposition is analogous to an Eulerian frame of reference [4, 5].

Neighbour List

Due to the fact that the intermolecular forces considered are short-range, we do not need to consider all the pairs of atoms for the purpose of force calculation. A naïve method which goes through all the pairs of particles assigned to a processor would have a time complexity $O(N^2)$. Thus, the concept of Verlet neighbour list presented in [6] is used. This concept uses an array for each particle in each processor to maintain the list of local particles within the cut-off radius r_c plus a small distance r_s called skin distance.

When particle interactions are calculated, only the particles in the neighbour list are considered. This saves computation time by removing the need to look for all the particles in the system. Further, the skin radius gives leeway by tracking particles slightly beyond of the cut-off radius. This means that the neighbour list need not be updated every timestep.

Further, the solution domain may be divided into cells of length $L_c \leqslant r_c + r_s$ and each cell can have its own linked list of atoms lying within that cell. When a neighbour list is built, the program need not check all the cells but only 9 cells forming the square around the atom in consideration. This reduces the computation time further. However, if the number of particles tracked by each processor is small, there is no need to create the cell list.

Serial Simulation

The following simulation procedure has been adapted from [1] and gives a general procedure and formulas involved in the solution procedure. Please refer to Figures 3-5 for a flow chart of the algorithms used.

1. Set all the atoms in their locations by reading from an input file from LAMMPS.
   See `parser.f90`, `in.micelle`, and `data.micelle`.
   (a) 1200 atoms - 750 water(1), 150 head(2), 150 tail1(3), 150 tail2(4)
   (b) 300 bonds
   (c) Bounding box (0,0) to (35.85686,35.85686)
2. Let system attain thermal equilibrium
   See `parser.f90`, `initial.f90`, `pmds.f90`, `force_soft.f90` (or `force_soft_neighbor.f90`, `neighbor.f90` for use of neighbor list), `integrate.f90`, `normalize_vel.f90`, and `in.micelle`.
   (Note: parameters are defined below in equations used)
   (a) Set special bonds to fene which sets LJ interaction with parameters 0, 1.0 and 1.0
   (b) Set pair style between all atoms as soft interactions with parameters 0, 1.2246
   (c) Set bond style between atoms to be harmonic with parameters 50.0, 0.75
   (d) Set initial velocity with uniform distribution for temperature 0.45
   (e) Set integration conditions(fix)

        i. Set NVE(Number, Volume, Energy) conservation integration
        ii. Update temperature to 0.45 every 100 time steps by rescaling velocity
        iii. Set soft interaction constant A to ramp from 1 to 20 as time proceeds

    (f) Run for 1000 timesteps

3. Actual development of micelle layer

    (a) Remove soft interactions
    Set interaction style to Lennard Jones with cut-off

atom1	atom2	ϵ	σ	r_c
water	water	1.0	1.0	2.5
head	head	1.0	1.0	2.5
water	head	1.0	1.0	2.5
tail 1	tail 1	1.0	0.75	2.5
tail 2	tail 2	1.0	0.50	2.5
tail 1	tail 2	1.0	0.67	2.5
water	tail 1	1.0	1.0	1.12246
water	tail 2	1.0	1.0	1.12246
head	tail 1	1.0	0.88	1.12246
head	tail 2	1.0	0.75	1.12246

    (b) Run for 60000 time steps

The serial algorithm for the force computation and integration is below. For the serial version we took advantage of Newtons Third Law, thus decreasing the force computation cost in half.

Force

See `force.f90` and `soft_force.f90` (or `force_neighbor.f90`, `force_soft_neighbor.f90` and `neighbour.f90`)

- Loop through each atom to calculate the distance between two atoms (or loop through neighbor list)
    - 1st loop through particle $i = 1$ to $N_{atom} - 1$
    - 2nd loop through particle $j = i + 1$ to N_{atom}

        - Calculate distance (X and Y) between two particles based on their positions.
        - e.g. Distance $= X(i) - X(j)$
    - apply periodic boundary conditions to both D_X & D_Y
        - $D_X = D_X - \texttt{box} * \texttt{nint}(D_X/\texttt{box})$
    - Calculate $R^2 = D_X^2 + D_Y^2$

- Determine the atom type for i & j using the array atom type (called AT in code)

- Use the atom types to query matrix of parameters needed to calculate force
    - Lennard Jones: cut off radius, sigma, epsilon (See 3.b. above)
    - Soft: cut off radius, energy (See 2.b. above)
    - Harmonic Bonds: energy/distance^2 , equilibrium bond distance (See 2.c. above)

- Calculate the Nonbonded Forces using either Soft for initialization or Lennard-Jones for the Simulation Run.
    - Lennard Jones: $F = -\frac{48\epsilon r}{r^2\sigma}\left(\frac{\sigma}{r}\right)^6\left[\left(\frac{\sigma}{r}\right)^6 - 0.5\right] dx$ (or dy)
    - Soft: $F = -A\left[\frac{\pi}{r_c}\sin\left(\frac{\pi r}{r_c}\right)\right] dx$ (or dy)
    - Update total forces on atoms using Newtons Third Law:

$$F_x(i) = F_x(i) + FD_X$$
$$F_y(j) = F_y(j) - FD_Y$$

- Loop through the atom bonded list to calculate the bond forces
    - Harmonic Bonds: $F = -2k(r - r_o)dx$ (or dy)
    - Update total forces on atoms using Newtons Third Law:

$$F_x(i) = F_x(i) + FD_X$$
$$F_y(j) = F_y(j) - FD_Y$$

- End when have looped through all the atoms.

Integration

See `integrate.f90` and `normalize_vel.f90`

- Loop through each particle
  - Calculate predicted position
    $x'(t + \Delta t) = 2x(t) - x(t - t\Delta) + \frac{F(t)}{\Delta t^2 2}$
  - Calculate predicted velocity
    $v'(t) = \frac{x'(t+\Delta t) - x(t-\Delta t)}{2\Delta t}$
  - Sum up kinetic energy
    $U_{kin} = U_{kin} + \frac{v_x^2 + v_y^2}{2}$
- At every Estep, rescale to ensure desired temperature
  - Calculate velocity scaling
    $scale = \sqrt{\frac{(2N-2) * T_{desired}}{U_{kin}}}$
- Loop through each particle
  - Find actual velocity
    $v(t) = scale * v'(t)$
  - Find actual position
    $x(t + \Delta t) = x(t - \Delta t) + v(t) * 2\Delta t$
  - Increment timestep
    $x(t - \Delta t) = x(t)$ $x(t) = x(t + \Delta t)$
  - Make sure all particles end up inside box

Neighbor List

See `neighbour.f90`

- Set up cells
  - Set the cell width to be equal to the maximum of cut-off radius + skin radius
  - Increase the cell width slightly so that the cell size divides the box size evenly

- Calculate number of cell in each direction(no. of cells = numcell^2)

- Bin atoms via a linked list
  - Loop through all cells
    - Set head of linked list(cellnum) = 0
  - Loop through atoms
    - Calculate the i and j index of the cell where the current atom belongs to
    - Using i and j calculate the actual cellID (j*numcell+i)
    - Set LL(atomID) = head(cellID)
    - head(cellID) = atomID
      - This method stores numbers in an array of size equal to the number of atoms. To retrieve the atoms in a given cell, we just go to LL(head(cellID)) and loop backwards to LL(LL(head(cellID))) and so on till LL(atom) = 0
- Set up neighbour list
  - Loop through all atoms
    - Set number of entries in current atoms neighbour list to zero
    - Calculate the cell that the current atom belongs to
    - Offset the cell by -1,0 and 1 in each direction
      - At each of these cells, loop through the atoms in the cell If distance between the two atoms is less than $r_{skin} + r_{cutoff}$, add to the neighbour list of whichever atom has the higher atomID

Equations Used

The following equations are used in the simulation, where E is potential energy and F is force.

- Lennard-Jones interaction

$$E = 4\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \text{ for } r < r_c$$

$$F = -\frac{48\epsilon r}{r^2\sigma}\left(\frac{\sigma}{r}\right)^{6}\left[\left(\frac{\sigma}{r}\right)^{6} - 0.5\right] dx \text{ (or } dy)$$

Parameters: ϵ, σ, r_c

- Soft interaction

$$E = A\left[1+\cos\left(\frac{\pi r}{r_c}\right)\right] \text{ for } r < r_c$$
$$F = -A\left[\frac{\pi}{r_c}\sin\left(\frac{\pi r}{r_c}\right)\right] dx \text{ (or } dy)$$

Parameters: A, r_c

- Harmonic bonds

$$E = k(r-r_o)^2$$
$$F = -2k(r-r_o)$$

Parameters: k, r_o

- Temperature-velocity formula for 2D cases

$$\frac{1}{2}m < v^2 >= kT$$

- Verlet velocity integration (NVE)

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{F(t)}{2m}\Delta t^2$$
$$v(t+\Delta t) = v(t) + \frac{F(t)+F(t+\Delta t)}{2m}\Delta t$$

- Kinetic energy

$$U_{kin} = \sum \frac{1}{2}m(v_x^2+v_y^2)$$

- Temperature

$$T = \frac{U_{kin}}{2N-2}$$

- Pressure

$$P = \frac{NT}{V} + \frac{1}{6V}\sum r_{ij}\cdot f_{ij}$$

Parallelization Process

The parallelization process that was used is adopted from [4]. The method involves the use of a distributed memory system to run the simulation procedure. For simplicity, atom and processor numbers are chosen so N/P atoms can be assigned to each processor. The parallelization was implemented using a procedure called atom decomposition, where the atoms are assigned to processors at the start of the simulation and the processor is responsible for updating the position of its assigned atom through the time integration of Newtons equations of motion. We implemented two forms of the atom decomposition, called A1 or A2. A1 calculates the total force on each atom and does not take advantage of Newton's Third Law. The A2 implementation takes advantage of Newton's Third Law and decreases the computation cost of the forces in half as compared to A1. There are added communication costs for A2, since while A1 requires only a single collective operation per timestep to exchange atom positions, A2 must exchange forces and pressures.

The algorithm for atom decomposition 1 per timestep and per processor is as follows:

1. Send and receive the particle positions to and from all the processors (fold operation).
2. Determine which forces to be computed using the neighbor lists.
3. Compute the total force for each atom
4. Do time integration of Newton's equations of motion to update assigned atom velocity and position.

Atom Decomposition 2

This approach takes advantage of Newtons 3rd Law, $F_{ij} = -F_{ji}$, where it computes the force between a pair of atoms once and not twice. A processor is responsible for calculating the force that N/P atoms have between them and the rest of atoms. And, it calculates the force between a pair of atom when their ID satisfies: $i < j$ and $i+j$ is even, or $i > j$ and $i+j$ is odd. Figure 7 shows an 8 atoms 2 processors example, processor 1 is responsible for the force calculations for lattice marked in red and processor 2 is responsible for those in blue. They will compute the force only for the lattice marked with a cross sign. And, as shown in the right matrix of Figure 7, if we fold the matrix along the red line, the lattice will cover lower triangle of the matrix without overlapping, which indicates the property that the force between all the pair have been covered once and only once. After finishing the force calculation, each processor does a `MPI_ALLREDUCE` operation to compute the total force acting on each atom, and then they update the position for N/P atoms with performing an `MPI_ALLGATHER` operation to collect the updated position for all atoms.

Performance Evaluation

The performance of our program is evaluated over time consumption, which was obtained from 1200 atoms, 10,000 time steps simulation runs. As shown in Figure 6, with neighbor list optimization we obtained over 6 times the speedup compared to the case without any optimization. The best case atom decomposition can lead to another 30% speed up on top of neighbor list implementation due to utilizing Newtons third law and the elimination of redundant calculations. Atom decomposition 2 has slightly better performance than atom decomposition 1. An interesting observation is that the introduction of more processors increased the time consumption. One possible reason for this phenomenon is when the problem scale is relatively small, the time consumed during communication can overwhelm the speedup gain by introducing more computational power.

To evaluate the weak scaling performance of the serial, A1 and A2 implementations, a generator was written to create micelle datasets for arbitrary numbers of atoms. In these generated datasets, the atoms are initialized on a grid of the same scale as the original dataset, only the bounds of the simulation and extent of the grid are varied to fit the requested number of atoms. Molecules are randomly initialized on atoms in the grid as in the original dataset, maintaining the same ratio of molecules to atoms as the original dataset. As shown in Figure 11, with these generated datasets, the runtime of each of the implementations was averaged across 5 trials for process counts of 2^i for $i \in [0, 4]$ (or 1 process for the serial implementation). Since weak scaling considers performance as the problem size per process is held constant, this corresponds to problem sizes between 1200 and 19,200 atoms. As expected, the serial implementation has the longest runtime. Though it requires somewhat less communication than A2, A1 is slower than A2 since it requires twice the amount of computation.

Validation

LAMMPS [1] is a widely used and well tested molecular simulation tool. We validated the following data in our simulation against that obtained by running LAMMPS. We ran 10 runs for our code and 10 for LAMMPS and each was started with a different random seed for initial velocity. We then took the average of the 10 runs and compared the following values:

1. Total Energy over time
2. Final velocity distribution
3. Pressure over time

As shown in Figures 8-10, with same trend and a acceptable deviation, our simulation result agree with that of LAMMPS simulation.

References

[1] Lammps molecular dynamics simulator. http://lammps.sandia.gov.

[2] D. Frenkel and B. Smit. *Understanding molecular simulation: from algorithms to applications*, volume 1. Academic Pr, 2002.

[3] JE Jones. On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738):463–477, 1924.

[4] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995.

[5] D.E. Shaw. A fast, scalable, method for the parallel evaluation of distance-limited pairwise particles interactions. *Journal of Computational Chemistry*, 26(13):1318–1328, 2005.

[6] Loup Verlet. Computer "experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.

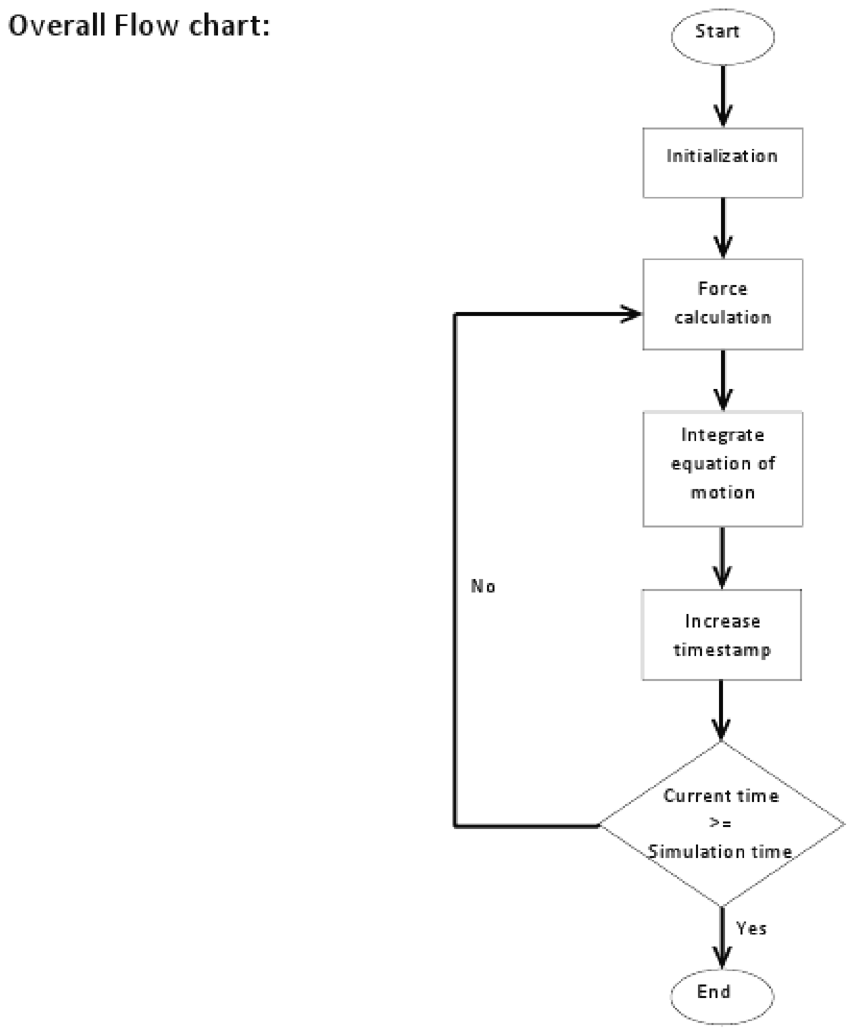


Figure 2: Flow chart of the main simulation loop.

Initialization Flow chart - sequential:

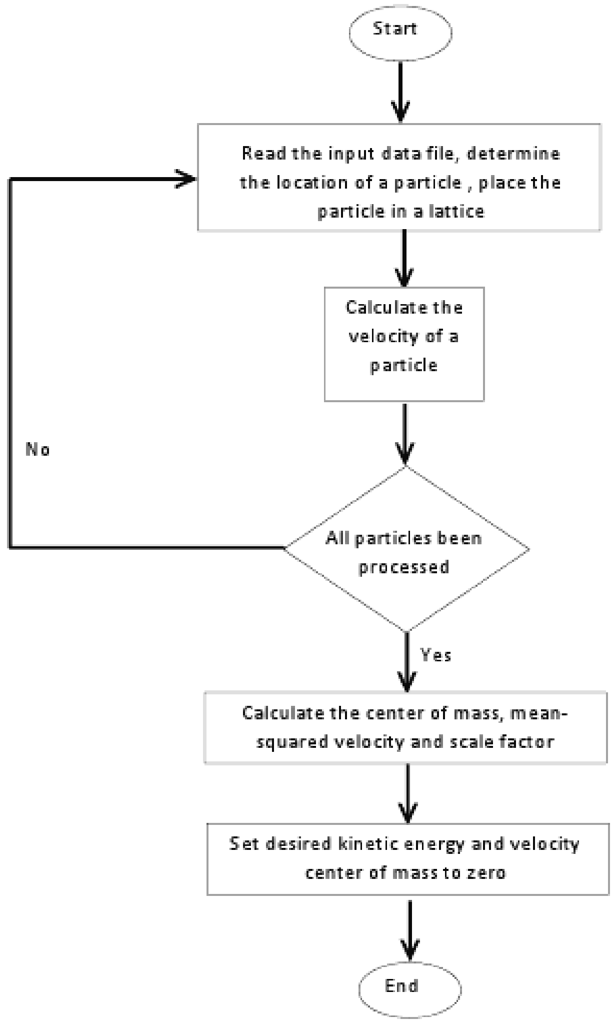

Start
Read the input data file, determine the location of a particle , place the particle in a lattice
Calculate the velocity of a particle
No
All particles been processed
Yes
Calculate the center of mass, mean-squared velocity and scale factor
Set desired kinetic energy and velocity center of mass to zero
End

Figure 3: Flow chart of the model initialization.

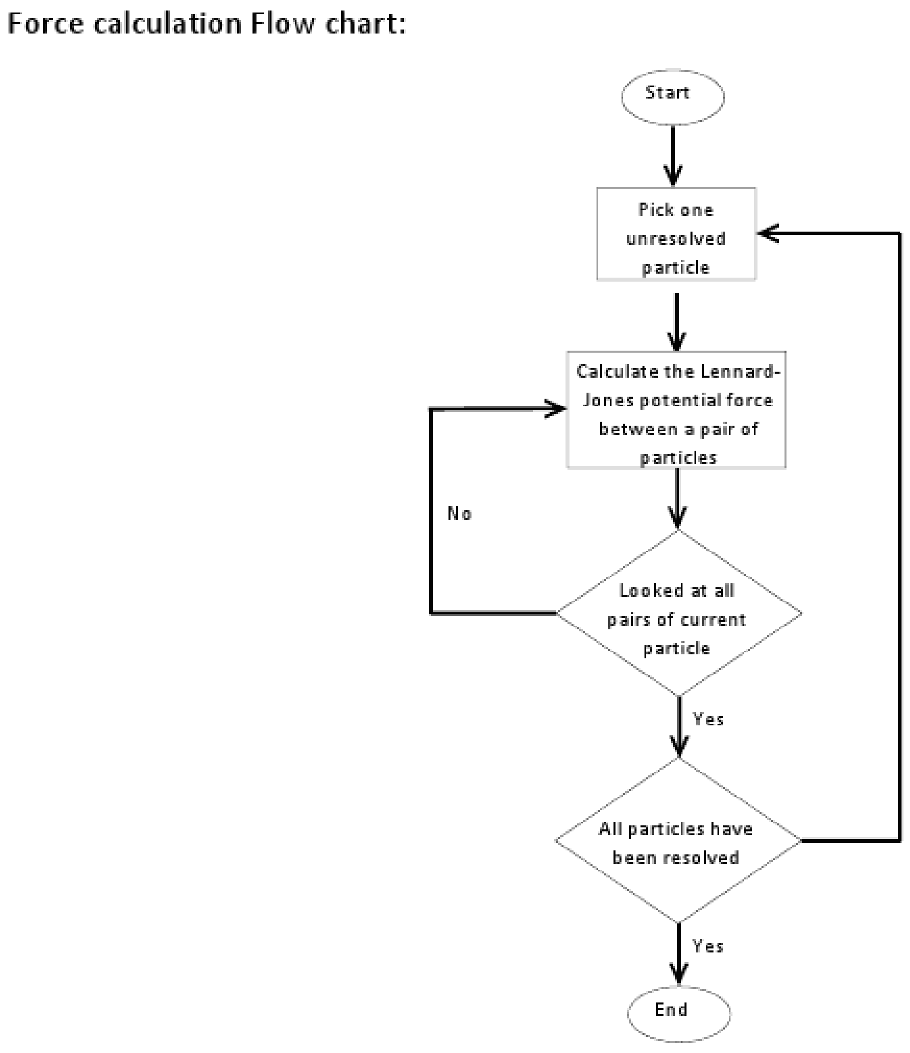

Force calculation Flow chart:
Start
Pick one unresolved particle
Calculate the Lennard-Jones potential force between a pair of particles
No
Looked at all pairs of current particle
Yes
All particles have been resolved
Yes
End

Figure 4: Flow chart of sequential force calculation.

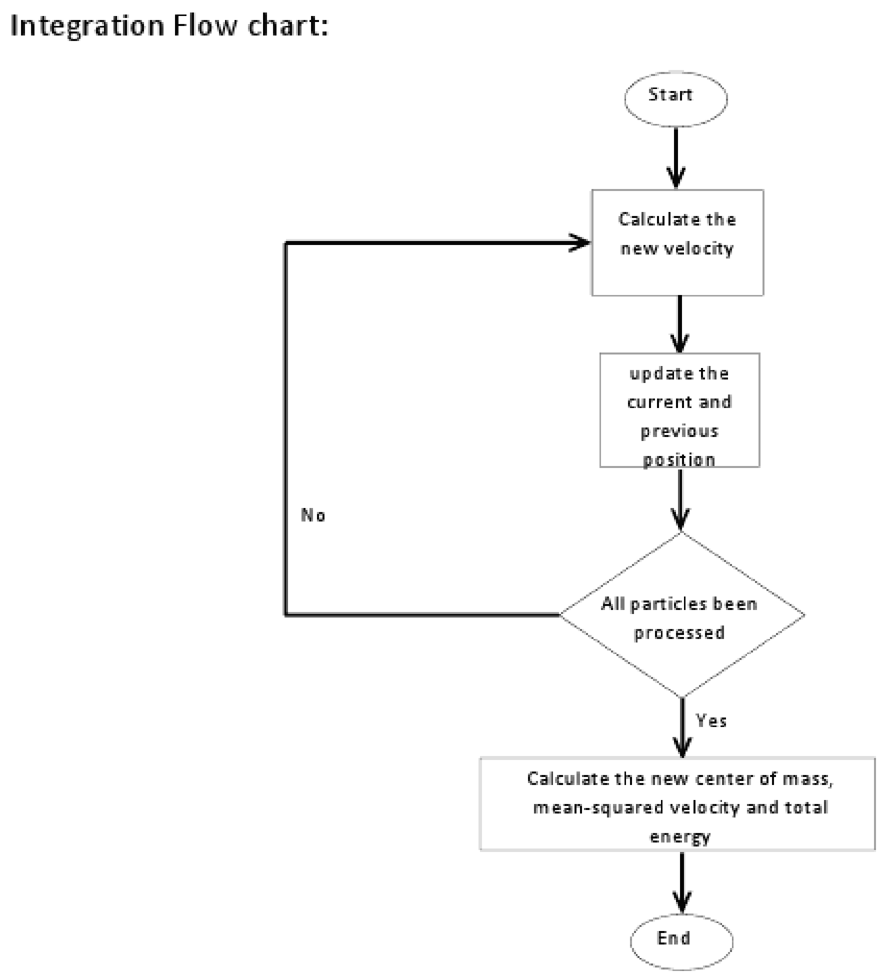

Integration Flow chart:
Start
Calculate the new velocity
update the current and previous position
No
All particles been processed
Yes
Calculate the new center of mass, mean-squared velocity and total energy
End

Figure 5: Flow chart of velocity integration.

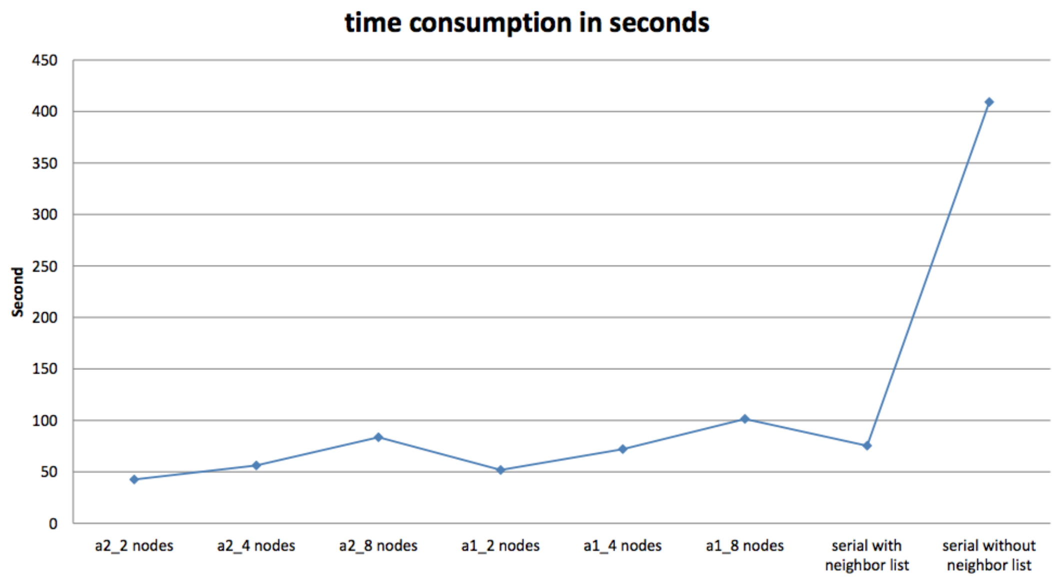
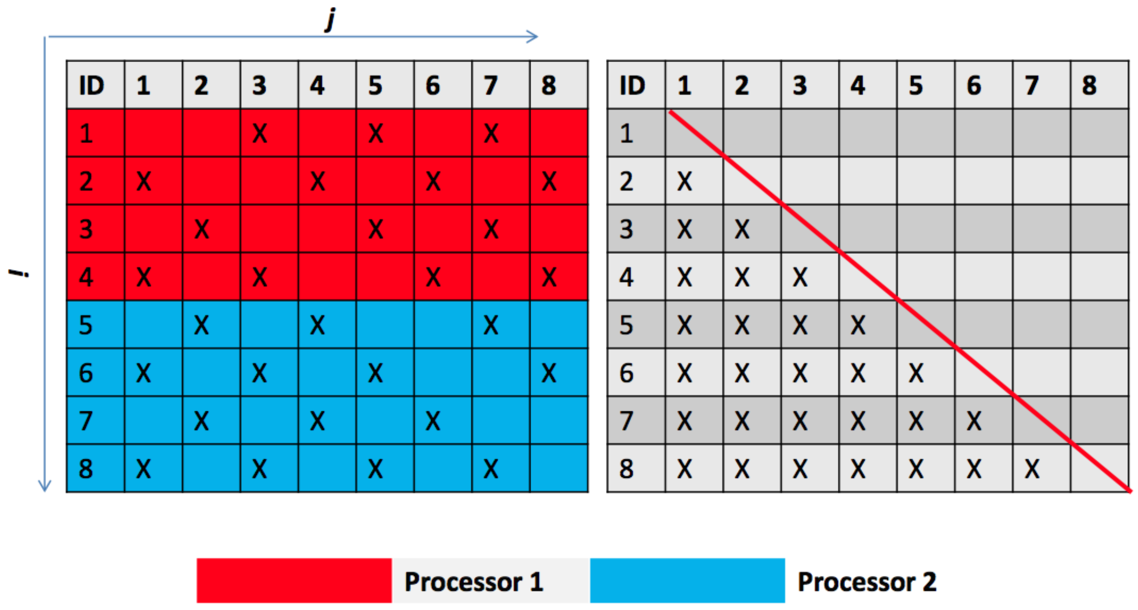


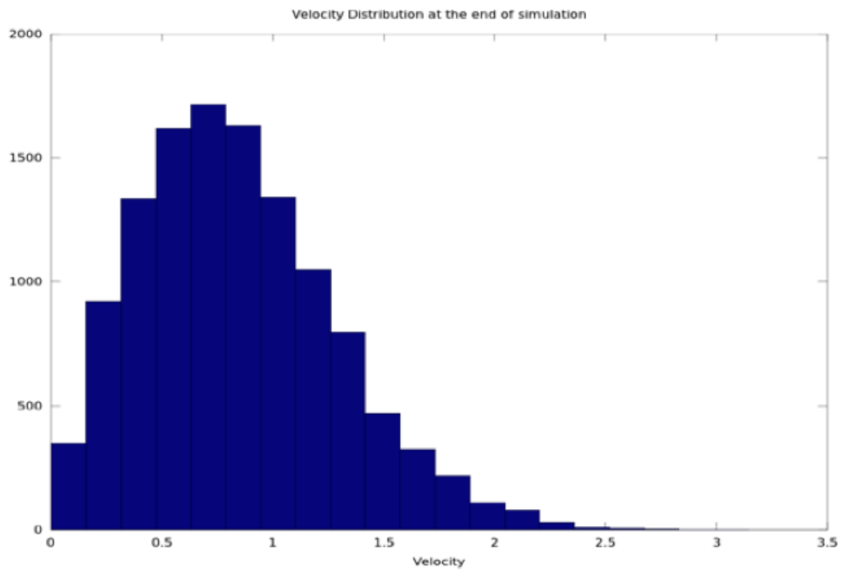
Figure 6: Strong scaling performance for small problem size of the three implementations.



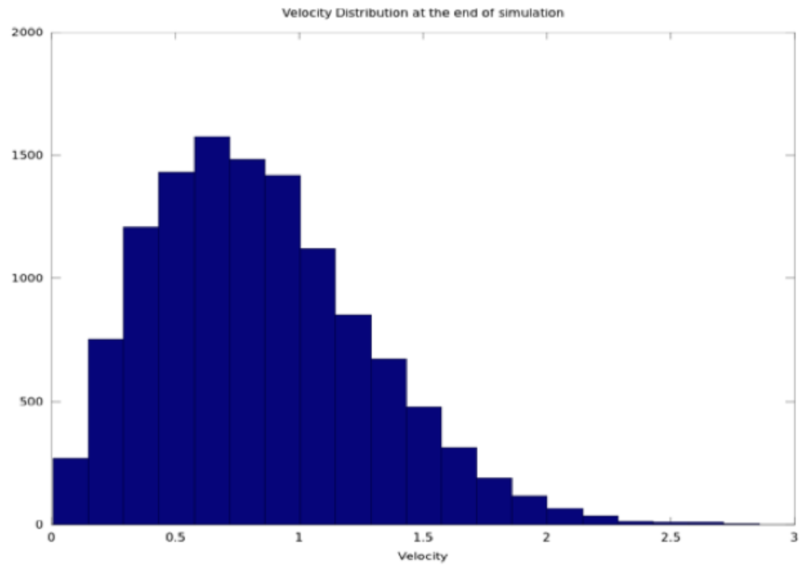
ID	1	2	3	4	5	6	7	8
1			X		X		X	
2	X			X		X		X
3		X			X		X	
4	X		X			X		X
5		X		X			X	
6	X		X		X			X
7		X		X		X		
8	X		X		X		X	

ID	1	2	3	4	5	6	7	8
1								
2	X							
3	X	X						
4	X	X	X					
5	X	X	X	X				
6	X	X	X	X	X			
7	X	X	X	X	X	X		
8	X	X	X	X	X	X	X	

Figure 7: Atom decomposition 2 of force matrix calculation. Entries marked with 'X' are calculated, while others are skipped. The right matrix shows that if folded along the diagonal, the left matrix is triangular.



(a) Our Implementation



(b) LAMMPS

Figure 8: Distribution of velocity at the end of simulation averaged over 10 trials.

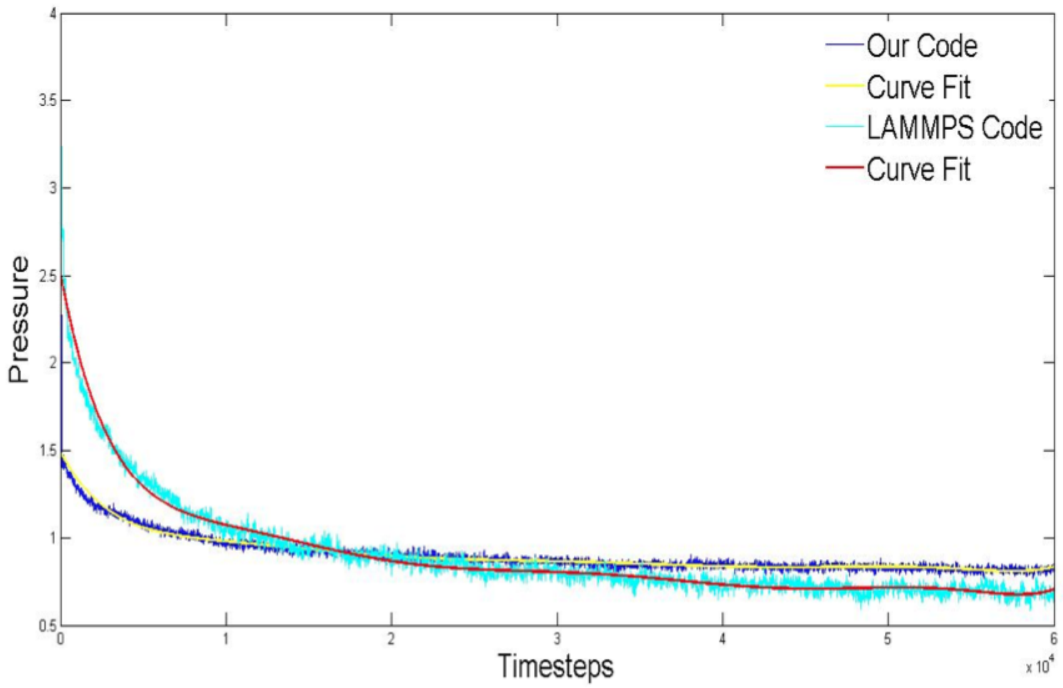


Figure 9: Pressure over simulation run averaged across 10 trials.

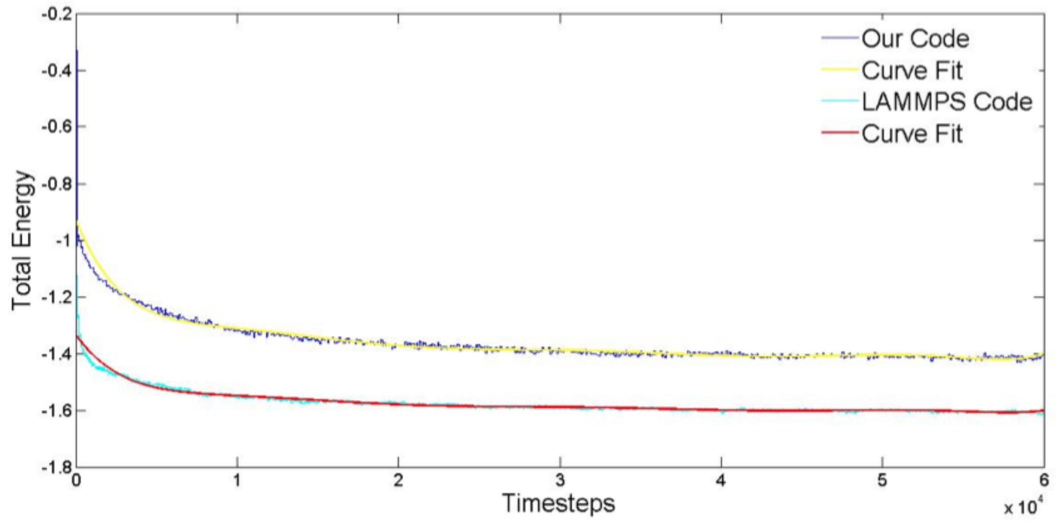


Figure 10: Total energy over simulation run averaged across 10 trials.

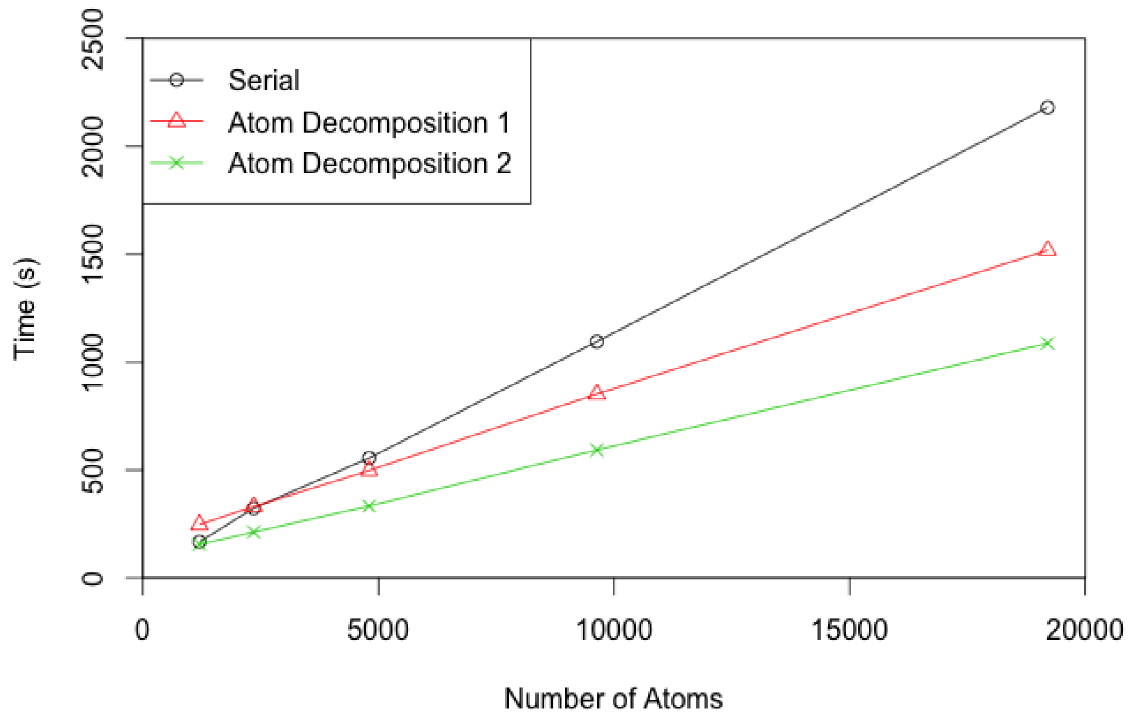


Figure 11: Weak scaling performance of the three implementations.