PARALLEL IMPLEMENTATION OF GIBBS ENSEMBLE MONTE CARLO METHOD

Veerendra Harshal
ME17B124
veerendraharshal@gmail.com

Indian Institute of Technology, Madras

Abstract. *The Gibbs Ensemble Method in Molecular Dynamics belongs to a class of Monte Carlo Methods which rely on Random Walks which are inherently sequential in nature and do not allow for a straight forward divide and conquer parallelisation. Hence, a robust class of parallel programs not restricted by simultaneity is essential.The programs used in this implementation are designed to be generic and applicable outside of the scope of the Simulation.*
Keywords: Parallel Computing, Gibbs Ensemble, Lennard Jones, Monte Carlo, Molecular Dynamics

1 Theory

1.1 Monte Carlo Method

In molecular dynamics, the Monte Carlo method is used as a way to approximate thermodynamic ensemble averages involving integration over the ensemble with a weighted average over a sample generated by a random walk over the ensemble.[3]

1.2 Importance Sampling

The process of accepting new configurations is handled by the Metropolis Hastings Algorithm which imparts probabilities so that the final collection of accepted configurations are in accord to a desired probability distribution rendering a weighted average to be simpler.

1.3 Lennard Jones Potential

The algorithm generating new configurations needs a way to predict the probabilities of the new configuration which is a function of energy and Temperature. A good model to estimate the energy of a system is given by the Lennard Jones Potential Model.

1.4 Gibbs Ensemble

Useful to simulate vapour-liquid co-existence. Does so by simulating liquid and gas in separate boxes. Keeps T, Total Volume, Total Number of Particles constant. Allows particle displacement, volume exchange, particle exchange as trial moves. This helps to maintain the same T (Thermal Equilibrium), P(Mechanical Equilibrium) and μ(Chemical Equilibrium) across the two boxes.[5]

2 Algorithm

The Gibbs ensemble method was introduced by Panagiotopoulos and Panagiotopoulos et al. [4] for the direct simulation of gas-liquid and liquid-liquid equilibria. It combines canonical (NVT), isobaric-isothermal (NPT), and grand canonical (μVT) Monte Carlo techniques in a single simulation using two boxes at given initial densities at the desired temperature.When equilibrium is attained the gas is sampled in one of the boxes, the liquid in the other, in such a way that the chemical potentials and pressures in the two boxes become equal according to the conditions for phase coexistence.

Listing 1.1: Pseudo Code for Gibbs Ensemble Monte Carlo

```
#Program Begins
input data.in                          #Import Input Variables.
coord = Initialise_Lattice()           #Current coordinates
for step in range(1,Nsteps):
    mode = Choose_Trial_Move()         #Choose Trial Move
    if mode = 'Particle Displacement':
        coord = Displace(coord)
    elif mode = 'Volume Exchange':
        coord = Volume(coord)
    elif mode = 'Particle Exchange':
        coord = Swap(coord)
    data_summary.append(data_interpret(coord))
endfor
postprocess()                          #Perform Calculations
```

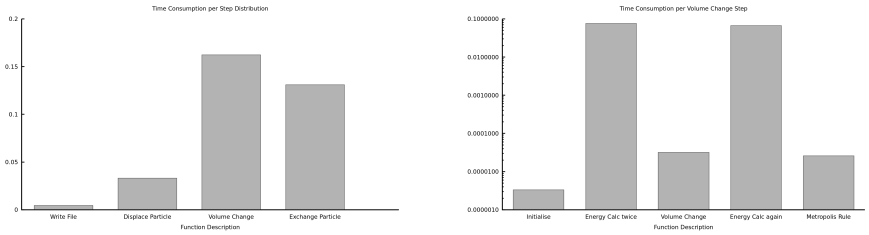
Listing 1.2: Pseudo Code for Gibbs Ensemble Monte Carlo

```
def Trial_Move(coord):
    c_o = coord
    c_n = trial_gen(coord)  #Propose a Trial Configuration
    acc_prob=calc_p(c_o,c_n)#Calculate P(Acceptance)
    if rand(0,1) < acc_prob:
        coord = c_n         #Accept
    return coord
```

3 Scope for Parallelisation

3.1 Requirements

The time-wise breakdown leads to a conclusion that the function call that calculates the Lennard Jones Energy is the most energy consuming function. But as it is not viable to store repetitive amounts of code for all processes, we need better functions to handle sequential programs.



(a) Time spent within a step (b) Time spent within a trial move

Fig. 1: Energy Calculation is the most expensive function

Parallel Function Receiver The idea is to keep the Slave Nodes awaiting instructions on the function call. The Parallel processing is limited only to the duration of the function call after which the slave nodes get inactive awaiting further instructions.

Background Worker The master node runs the program and leaves in the work space data that need further work. If the result of the work is not particularly pertinent to that of the master node,then the work can be delegated to the Slave Nodes by the master who moves on. The result has to be stored according to some protocol without bothering the master node.

Self-Motivated Worker A Dedicated set of processes can be left dedicated to accomplish minor activities within the major program. Printing Output to file, Animation, plotting etc can slowdown the progress of the master node and hence is best left to dedicated workers who can further parallelise amongst themselves.

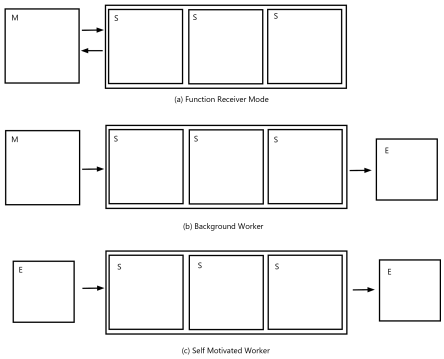


Fig. 2: Contrasting Data Flow among the three proposed styles.

3.2 Parallel Functionality

Function Calls The idea is to keep the slave nodes two inputs to perform a function. The first one is basically an On/Off switch, while the other is a function description which will lead the node to look for further inputs regarding the impending function call.

Data Flow With regards to the Master node, focus lies on how the inputs and outputs are structured. If the inputs of the slave processes are to be sent from the Master itself via an MPI message or if the input is self recognised by the slaves with respect to a variable or external stimuli. Similarly, the pertinent outputs of the Slave processes having to be sent to the master is also important. Often, tasks like printout, animation etc, do not return data back to the master. We hence need a better way of performing such functionality as well.

4 Parallel GEMC

The Function Call dedicated to calculating energy is $O(N^2)$ as it loops around all particle pairs.

The parallelisation lies at distributing the task of handling individual sets of particle pairs, calculating individual energy contributions and summing them up. This scales down the time taken by an order of M. If the Parallel function receiver is set up successfully, then the input is the array of coordinates, the indices specifying the particles dedicate to the process. Output shall be the energy contribution for each rank. As the energies are reduced by summing, the Total energy is gained at the master node.

Listing 1.3: Pseudo Code for Master Nodes - Parallel Function Receiver

```
def Master(comm, in, On_or_Off):
        switch = 1
        rank = comm.Get_rank()
        comm.barrier()
        comm.bcast(switch, root=0)
        out = []
        comm.barrier()
        comm.bcast(in, root=0)
        temp = rank_func(in, rank)
        comm.Reduce(temp, out, op, root=0)
        comm.barrier()
        comm.bcast(On_or_Off, root = 0)
        return b
```

Listing 1.4: Pseudo Code for Slave Nodes - Parallel Function Receiver

```
def Slave(comm):
        switch = 0
        rank = comm.Get_rank()
        in = None
        On = True
        while On:
                comm.barrier()
                switch = comm.bcast(switch, root = 0)
                if switch == 1:
                        out = []
                        comm.barrier()
                        in = comm.bcast(in, root=0)
                        temp = rank_func(in, rank)
                        comm.Reduce(temp, out, op, root=0)
                        switch = 0
                comm.barrier()
                On = comm.bcast(program, root = 0)
        return
```

For Other applications such as printing, web requests etc. that belong to the Background worker category, the switch can be set off thereby prompting the slaves to find their inputs from an external source independent of the master process.

The self-Motivated worker can also work from a similar framework except that the Master process need not initiate the transaction. The worker can himself look for external triggers like existence of output files, time crossing midnight etc and get to work.

Here we use parallel function receiver to distribute energy calculation across processes. The Background worker approach can be used to prompt printing messages following repetitive completion of a particular number of samples.

Finally, the Self-Motivated worker can read coordinate export by himself and converts them to image files.

Exports, printing etc, has been found out to be quite less expensive against the energy function call. Which would mean taking out processes from energy call and assigning to them inexpensive tasks will only lead to higher time taken. Hence, in the interest of performance, the other functionality has not been included and is mentioned only in a capacity of discussing parallel functionality for inherently sequential programs but has no relevance with the Gibbs Ensemble Simulation.

5 Results

5.1 Simulation Results

Parallelisation is done using the mpi4py library[1] [2] and is done in Linux Virtual Machine in Windows 10 OS.
Please find the files used at
https://github.com/VeerendraH/GibbsEnsembleMonteCarlo
Current Simulation performed using 4 processors. Actual results may vary.

```
Gibbs Ensemble Monte Carlo Simulation of
the Lennard - Jones Fluid
Temperature of the System = 0.8
Volume of the Simulation = 432
Number of Particles = 250
Net Density = 0.5787037037037037
Step Number: 2000
Density 1 = 0.45517336468375685
Density 2 = 0.7997939736392827
Standard Deviation 1= 0.01921138833633007
Standard Deviation 2= 0.05203479644687997
Time 14m2.203s
```

State	$\bar{\rho}$	σ_{rho}	$\|E\|$	σ_E
Liquid	0.805598	0.030788	469.061	21.28593
Gas	0.453827	0.009155	285.323	5.875716

Table 1: Results pertinent to stabilized regions.

The Simulation clearly demarcates the separation of a uniform substance to two materials one dense and the other light. We also notice that the two densities suggest a liquid vapour co-existence. The Energies also seem to settle in the two boxes thereby indicating stability or equilibrium.

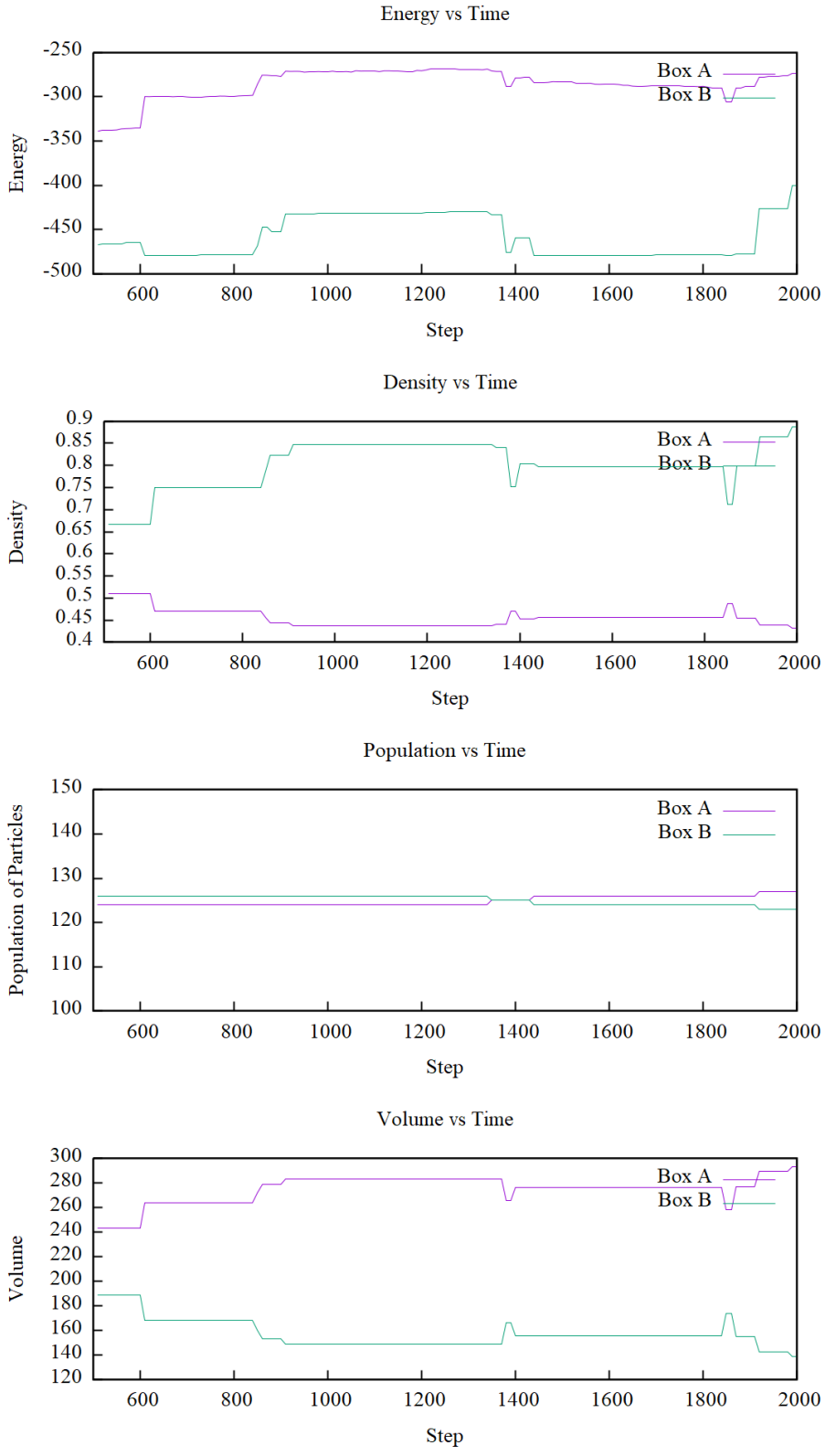
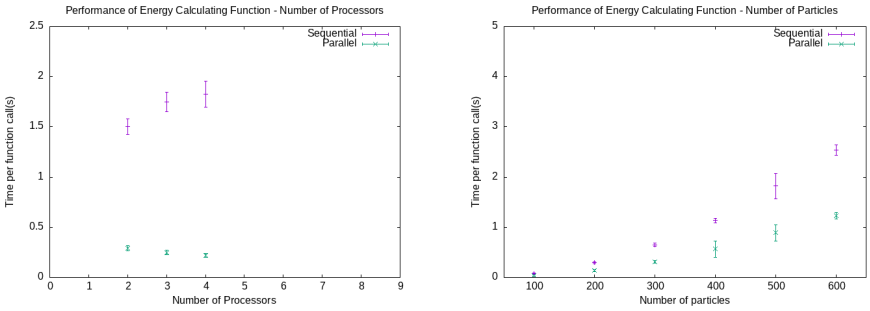


Fig. 3: Evolution of Box Properties

5.2 Parallel Performance Review



(a) Done with 200 particles (b) Done using 4 processors.

Fig. 4: Performance improvement due to Parallelising.

6 Acknowledgements

I acknowledge the course AM5080 and it's instructor Prof.Sarith sir. The TA's were also also very helpful throughout the course.

References

1. Dalcin, L.: Tutorial mpi for python 3.0.3 documentation (2021), bluehttps://mpi4py.readthedocs.io/en/stable/tutorial.html
2. Dalcin, L.D., Paz, R.R., Kler, P.A., Cosimo, A.: Parallel distributed computing using python. Advances in Water Resources **34**(9), 1124–1139 (2011). https://doi.org/10.1016/j.advwatres.2011.04.013
3. Frenkel, D., Smit, B.: Understanding molecular simulation. Academic Press (2002)
4. Panagiotopoulos, A.Z.: Direct determination of phase coexistence properties of fluids by monte carlo simulation in a new ensemble. Molecular Physics **61**(4), 813–826 (1987). https://doi.org/10.1080/00268978700101491, bluehttps://doi.org/10.1080/00268978700101491
5. Vlugt, T.J.H., Eerden, J.P.J.M.v.d., Dijkstra, M., Smit, B., Frenkel, D.: Introduction to molecular simulation and statistical thermodynamics (2008)