Report: **Project 1** (CSE 6230)

-- by Harsh Shrivastava (gtID: 903241811)

I have tried various techniques to parallelize the interaction function and also came up with a faster serial code for BD simulations.

In this report I'll show the results of 2 of my fastest parallel implementations. These are:

1. bd_fast.c (faster)
2. bd_cell.c

(NOTE: the makefile runs 'bd.c' code which is a copy of the 'bd_fast.c')

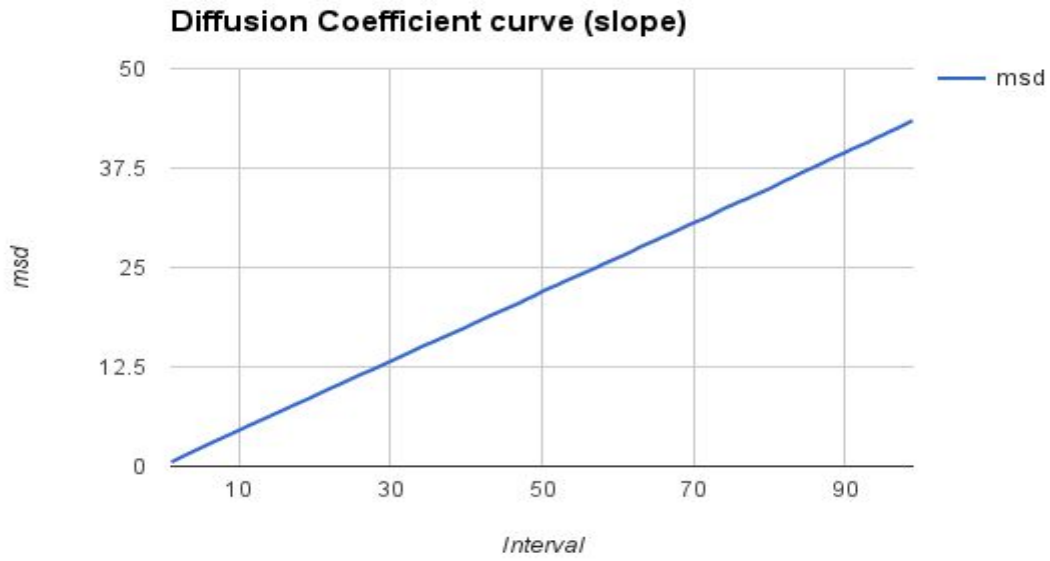
The report is divided in 3 parts, namely

A. Diffusion coefficient value verification
B. Speed up graph by increasing number of cores
C. Brief explanation of the technique

Part A: Diffusion coefficient value verification

NOTE: Results are shown for
N= 10,000 particles
a = 1 (radius of particle)
Phi = 20% vol fraction
The initial positions are initialised using uniform random function.
Input file: my_input_npos_10000.xyz
Each interval = 1000 Timesteps

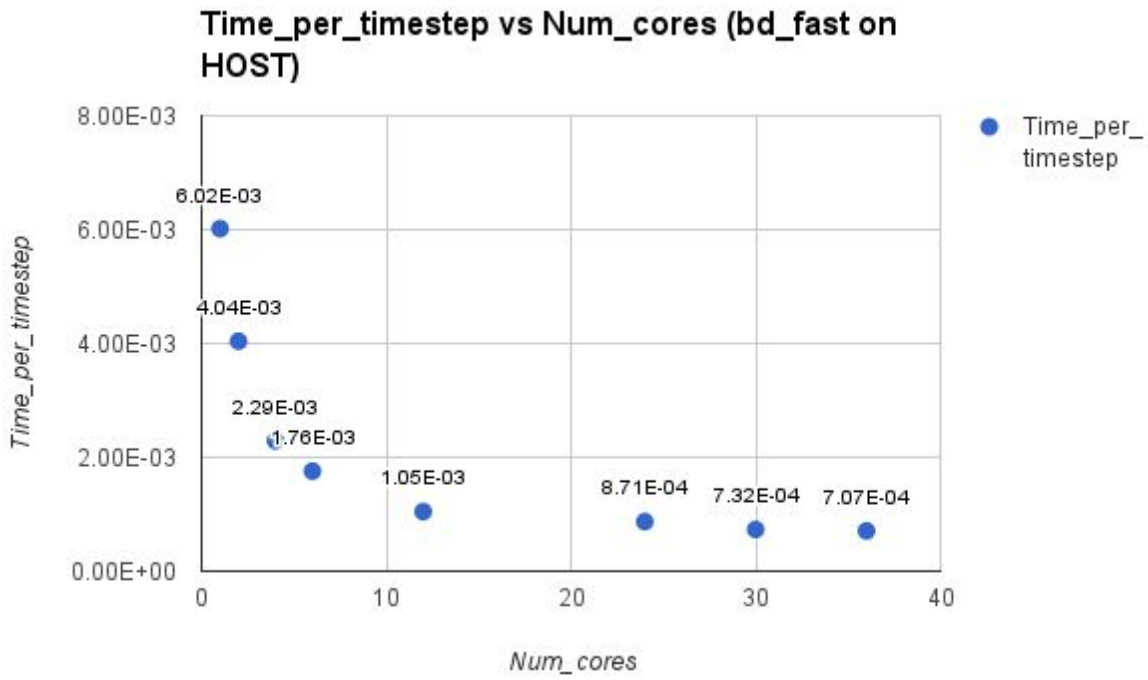
Algorithm	Diffusion Coefficient	Num_Intervals	Time per time step (in secs)	Num of cores
bd_fast	0.750536	10	0.000774542 **	30
	0.711228	100	0.000731972	30
bd_cell	0.750231	10	0.00108854	30
	0.715235	100	0.00109404	30



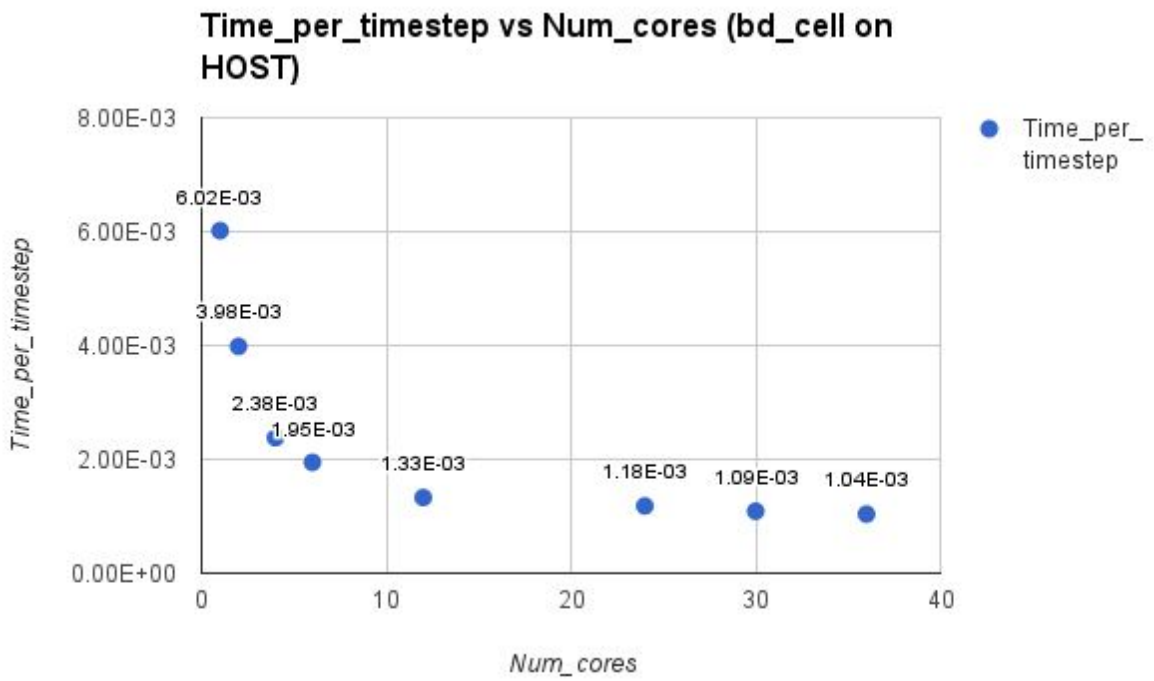
** - Have attached the sample output at the end of report (in Appendix)
The MSD vs Intervals graph (for 'bd_fast.c' code), slope of which is used to calculate the diffusion coefficient. (generated from a_msd3.m file)

Part B: Speed up graph by increasing number of cores
NOTE: Graphs are shown for num_INTERVALS = 10

1. **Results on HOST**
   (Logarithmic scale was not helping much in visualisation, so plotted on normal scale)
   1a. 'bd_fast.c



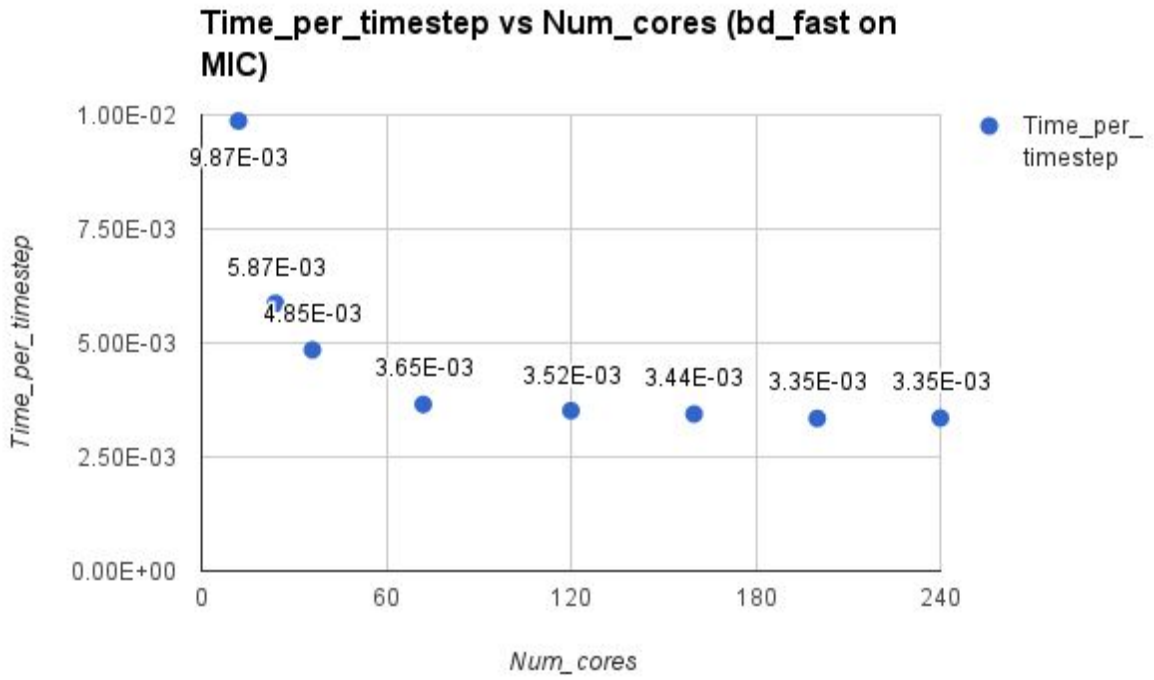
1b. ‘bd_cell.c’



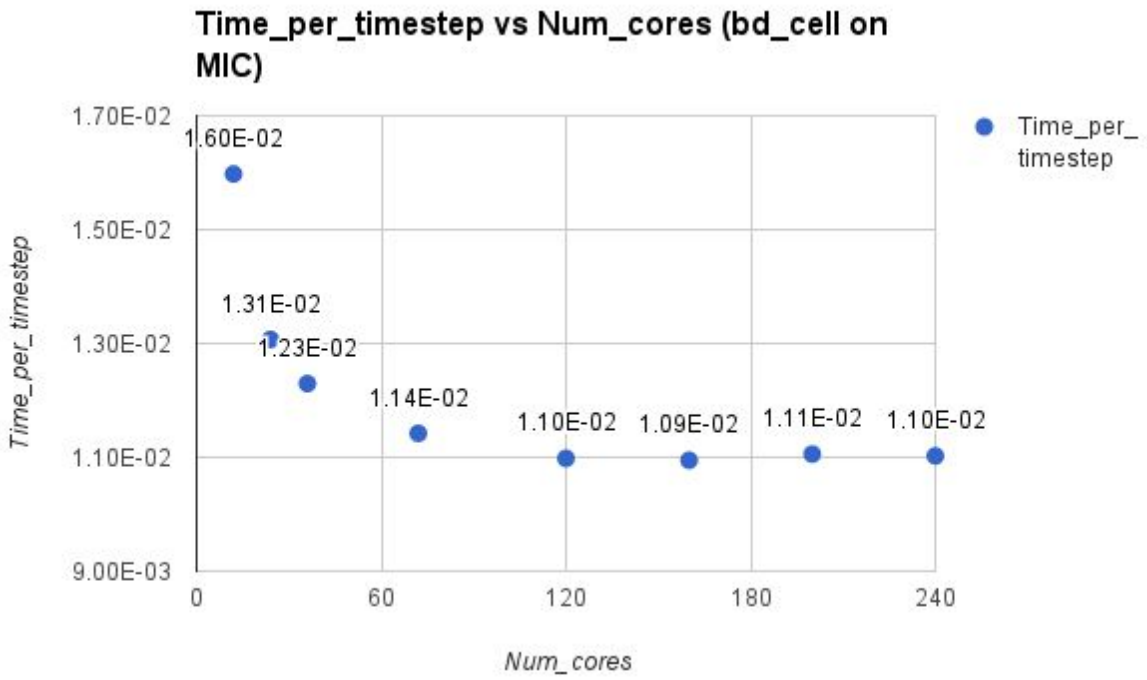
2. Results on MIC

NOTE: On mic, if more than 60 threads are used, then the position update part becomes the most time consuming one compared to cell list updation and forces calculations
SImulations were run using KMP_AFFINITY=verbose,granularity=fine, scatter/compact/balanced

2a. ‘bd_fast’



2b. ‘bd_cell’



Part C: Explanation of code

Changes done to ‘interactions.c’ function: (done in both ‘bd_cell’ and ‘bd_fast’)

1. Including {0, 0, 0} in box_beighbors and removing the while loop for calculation of within box interactions.
2. Adding a my_EPS to prevent divide by zero condition
3. Including the forces calculations in inside the interaction function itself thereby getting rid of pairs and distances arrays
4. Fixing Boxdim = (L/4)*a (where a is the radius of the molecule), so that we divide the cells into as small as possible to just fit minimum number of particles
5. Replace the idx, idy & idz by index, so that we can parallelize the loop to calculate forces for particles in different boxes simultaneously

Difference between ‘bd_cell.c’ and ‘bd_fast.c’:
The cell updation (assigning particles to cells) part in ‘bd_cell’ is serial while in ‘bd_fast.c’ it is parallelized. We can put a ‘critical’ pragma around the part of updating the ‘next’ list and ‘b->head’ to prevent the race condition, but this becomes a bottleneck to parallelize the cell updation part as ‘critical’ pragma is very slow.

One very interesting observation is, let’s calculate the probability that the above race condition occurs while updating cell lists.

Say, we have a BD simulation problem with
P = number of particles, Nt = number of threads, L = box width

The probability of 2 or more threads writing to the same box at the same time?
While parallelizing the for loop using #pragma omp for, if we take static scheduling for Nt threads, we get P/Nt different sections of the 'for' loop.

Now, calculating the probability of particles from one box occurring in such 2 or more sections simultaneously is:

(Snippet written in Latex)

Probability for race condition $= ({}^{N_t}C_2) * ((\frac{4}{L})^3 * N_t)^2 * (\frac{N_t}{P})^2 + \ldots + ({}^{N_t}C_k) * ((\frac{4}{L})^3 * N_t)^k * (\frac{N_t}{P})^k + \ldots + ({}^{N_t}C_{N_t}) * ((\frac{4}{L})^3 * N_t)^{N_t} * (\frac{N_t}{P})^{N_t}$

In the case of $P = 10000$, say running of $N_t = 100$ threads with box width $L = 60$, the probability for getting a race condition is (Note: the 1^{st} term is significant compared to other terms), so just calculating the first term of the above equation, we get

Probability for race condition in one time step $= ({}^{100}C_2) * ((\frac{4}{60})^3 * 100)^2 * (\frac{100}{10000})^2$ which approximately is $4e-05$ and can be neglected.

Hence, 'bd-fast.c' can be safely used for simulations where particles P are very large in number and also the density is less. (as vol. fraction ϕ is inversely proportional to L). The empirical results demonstrate the same.

Appendix
A1: **** Sample output for 10 intervals**

```
[Ahshrivastava3@joker proj1]$ make runhost
rm -f output.xyz
./bd_host my_input_npos_10000.xyz output.xyz 10
Number of particles: 10000
Number of intervals to simulate: 10
Simulation box width: 59.386291
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001673 for initiating the cell head
Time: 0.100814 for assigning particles to cells
Time: 0.470113 for force calculations
Time: 0.246114 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001648 for initiating the cell head
Time: 0.080570 for assigning particles to cells
Time: 0.442928 for force calculations
Time: 0.253629 for pos update
```

```
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001623 for initiating the cell head
Time: 0.080568 for assigning particles to cells
Time: 0.433748 for force calculations
Time: 0.232719 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001629 for initiating the cell head
Time: 0.080545 for assigning particles to cells
Time: 0.430805 for force calculations
Time: 0.232663 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001623 for initiating the cell head
Time: 0.080754 for assigning particles to cells
Time: 0.423492 for force calculations
Time: 0.232675 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001638 for initiating the cell head
Time: 0.081054 for assigning particles to cells
Time: 0.431695 for force calculations
Time: 0.245372 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001629 for initiating the cell head
Time: 0.080758 for assigning particles to cells
Time: 0.425799 for force calculations
Time: 0.232603 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001641 for initiating the cell head
Time: 0.080895 for assigning particles to cells
Time: 0.424097 for force calculations
Time: 0.232619 for pos update
-----------------------------------------------------
L = 59.386291 cutoff2 = 4.000000 boxdim = 14
-----------------------------------------------------
Time: 0.001632 for initiating the cell head
```

Time: 0.080841 for assigning particles to cells
Time: 0.422882 for force calculations
Time: 0.232763 for pos update

L = 59.386291 cutoff2 = 4.000000 boxdim = 14

Time: 0.001642 for initiating the cell head
Time: 0.080961 for assigning particles to cells
Time: 0.424634 for force calculations
Time: 0.232806 for pos update

Time: 7.745418 for 10 intervals
Time per time step: 0.000774542