COMP3320/COMP6464

High Performance Scientific Computing

Molecular Dynamics Project

Abu Abraham

u6325322

Answers:

1- Approximation errors are prone to occur on the computation of floating point operations. The molecular dynamics simulation has several floating-point computations, thereby, resulting in several approximation errors. In the assignment, we have 3 factors that affect the magnitude of errors, increasing number of atoms, increasing number of iteration and increasing timestep.

Increasing number of atoms: Increasing no of atoms have a direct effect when the potential energy of the system and the forces acting on atoms are computed. There is an O(NlogN) computation for finding the potential energy and an O(N^2) for finding force coordinates. The increase in N would hence result in an increase in the magnitude of errors.

[Increasing no of atoms to over 4, started to throw errors in ctypes module. The values of arrays passed from python were not correctly received in C. Verified by printing the passed arrays in 3 computers including Raijin.]

Increasing number of iteration: Increasing number of timestep increases the errors on the computation of potential errors and forces. Both potential energies will have an increase in error by the magnitude of the increase in the number of iteration.

Increasing time-step: Increasing time-step increases the forces computed and velocity. However, it has no effect on the computation of potential energy. As we consider time-step as a factor while computing force and velocity, it is directly proportional to the magnitude of energy.

2- The per-iteration execution time of code for various time steps is mentioned below.

No of atoms	Length/ R	Execution time (micro-seconds)
8	2	674
27	3	1624
64	4	1984

3- The problem scales as a function of N^2. Ie, the complexity is O(N^2) as we find the forces based on the interaction of particles with all the other particles.

4- The most computationally intensive component according to time is the module which computes the forces to update the coordinates. As stated in the above answer, computing forces is an O(N^2) operation and involve computationally intensive division and square-root operations.

5- During each iteration, the proportion of time spent on calculating coordinates, forces, and velocities when N=3 roughly remains **5:170:1.** For various values of iteration and time-steps. However, a leap in the proportion of time taken by forces is observed when N is increased to 4. When N=4, the proportion is roughly 4:350:1.

[*Results are printed and computed using manual profiling as its neater and easier to read than gprof]*

6- Yes, the most computationally intensive components are executed the most. The code blocks for computing the distance is executed the most. Computing distances involve sqrt operations which in most cases stalls the pipeline. Its followed by the code block calculating forces, which is also a very computationally costly operation.

7- I choose, total cycles, total instruction, L1 cache miss, L2 cache miss, L1 store miss, L2 store miss and branch miss predictions.

A) Total cycles: It indicate the amount of clock cycles required to complete the application. It is an important attribute as it showcases the total resources used. The lesser the total cycles, the better it is.

B) Total Instructions: Total instructions indicate the total number of instructions executed by the program. It is an important factor affecting the running time of the program. The lesser number of instructions, the faster the execution time.

C) L1 cache miss: L1 cache, is the fastest accessible memory unit. Caches are used to improve the execution speed of program as its much faster to load data from cache than other forms of memory. If all data and instructions could be accessed from L1 cache, the execution speed would be very high. Each L1 miss forces processor to access data from lower lever caches, main-memory or even from the disk, stalling the pipeline.

The lower the value of L1 cache miss, the better it is.

D) L1 store miss: L1 store miss indicate the number of times a write operation had a cache miss. As store operations are slower than load, store misses should be reduced as much as possible.

E) L2 Cache miss: L2 cache is the second level of cache after L1. If data is not found in L1 cache, processor checks L2 cache. The significance of L2 cache is similar to that of L1 cache. The lower the misses, the better it is.

L2 cache miss occurs only if there is an L1 miss.

F) L2 Store miss: It is a similar metric to L1 store metric.

G) Brach miss prediction: Each time a conditional statement is encountered by the compiler. An internal branch prediction is used to predict which condition will be executed. An incorrect prediction would force the compiler to flush the instructions loaded into the pipeline. Hence branch miss prediction is an important metric in determining the execution efficiency.

Lower the value of branch miss prediction, the better it is.

Results in raijin, when Number of atoms is 27 and iterating 1000 times.

Total cycles: 829869725
Total Instructions: 1785859236
L1 cache miss: 145865
L1 store miss: 8574
L2 cache miss: 21523
L2 store miss: 1856
Brach miss prediction: 2854

IMPORTANT: My coding logic involved starting the counter in python and calling C using ***ctypes*** after the specified interval. Starting and stopping PAPI counters gives occasional errors. Hence, I

wrote a condition to start it only once, and not execute again if an error occurred while stopping/reading values.

In results displayed, you could see the number of iterations considered while displaying the counter values.

The supported events in each machine can be found in:
http://icl.cs.utk.edu/projects/papi/presets.html

8- Changing the code layout to incorporate techniques such as loop-unrolling and loop-fusion did not seemingly improve the results. As C used row-major approach to store multi-dimensional arrays, changing the array layout increased the number of cache misses and thereby, increased the execution speed.

IMPORTANT

Along with the profiling an option to manually profile is added as that enables easy and quick view of the time spent in various modules. The results were as accurate as the python profiler cProfile and C profiler gprof.

Answers to COMP6464 specific questions:

9- LINPACK Benchmark is a measure of the systems floating point computing power. It measures the computing power of a system by performing linear algebraic computations.

On compiling results for TOP500, the results generated, Rmax, on N and N (1/2) along with the peak performance achieved is considered. The algorithm used for LINPACK Benchmark conform with LU factorization and must be double precision floating point operations [1]. Benchmark excludes operations which compute solution in precision lower than 64-bit floating point. [1]

The ordering is respect to the Rmax scores and RPeak is considered on equal scores of Rmax.

[The top500 list of supercomputers in order of the results in LINPACK can be found here: https://www.top500.org/lists/2017/11/]

[1] - https://www.top500.org/project/linpack/

10- As mentioned in Q9, various Linear algebra computations are performed in LINPACK Benchmark operations. Similar to LINPACK Benchmarks, we can use the number of floating point operations as a metric.

The float count of a program can be understood as a factor of the complexity. In our program,

Say, we execute in **m** steps.

Potential energy is computed m times, each with **O(NlogN)**

For updating coordinates, m times, **O(N2)**

And several other less intensive operations like incrementing and updating values

In effect, our operation takes => **m* O(N^2)+ m*O(NlogN)**

Hence, FLOP count could be thought of as O(N^2), however, m*O(N^2) would be more appropriate.

11- Rmax score describes its maximal achieved performance and Nmax gives the value in achieving Rmax.

In LINPACK, N=100 and N=1000 are used for benchmarking. I am using N = 2 and N =3 (Because of the ctypes error mentioned in **Issues Encountered**)

In Raijin, on running my code as benchmark:

Rmax: 1418 and Nmax: 27

I**n Lab computer**:

Rmax: 321 and Nmax: 8

In raijin, higher dimensional values will result in better Rmax as a result of having multiple processors. Whereas in lab/personal computers performance will degrade with increasing value of N.

13- My benchmark just asses the execution time. It does not consider the peak_performace achieved, neither does it consider the memory bandwidth.

Issues Encountered:

The connection between python and C was established using the ctypes module in python. Ctypes, lets us call a C module exposed as a sharable object to be called from python. It also facilitates passing parameters by reference.

However, such an approach resulted in creation of several key errors in C.

1- On declaring and initializing values, it randomly gets set to a very high value.

Example: double a,b,c=0.0;

At times, a or b gets initialized to very high value like 134413123131314143242

2- On passing the array reference from python to C, when total number of atom is greater than 64, the values are not being received correctly in C.

Also, when connecting from python, PAPI_Module starting error is thrown when more than 5 items are added to the Events array.