Parallelizing Molecular Dynamics (MD) Simulation

Afonso Franco
Universidade do Minho
Braga, Portugal
pg53595@alunos.uminho.pt

Alice Teixeira
Universidade do Minho
Braga, Portugal
pg52670@alunos.uminho.pt

***Abstract*— The virtual simulation of physics can often times be a time-consuming, resource-heavy procedure. In this paper, we utilize the API OpenMP in order to parallelize a molecular dynamics simulation, evaluating the quality of the results obtained.**

***Index terms*—optimization, programming, strenght-reduction, loop-invariant code motion, molecular dynamics**

I. Introduction

In some scientific fields, the simulation of entities' behaviours is useful since it provides researchers a visual resource which they may control, facilitating their task of correlating input data and effects produced. Moreover, the more complex such simulations progressively become, the more resource-costly and time-intensive they are potentially rendered. This way, it is crucial that their development is over viewed under a resource-aware lens. Parellilizing the code makes use of available cores in order to produce a more time-efficient computation of the instructions.

A. *Paper overview*

In this paper we propose the parallelilization of a c program provided within the course of Computação Paralela, taught at University of Minho, which has been modified with optimization techniques. This simulation represents a molecular dynamic's simulation of Argon atoms.

The original code is part of FoleyLab/MolecularDynamics: Simple Molecular Dynamics, which introduces a simulation of particle movements over time by calculating the force, accelaration and position of the atoms. The calculations are based on the classical understanding of mechanics and Newton's laws of movemement, in order to compute thermodynamic properties of materials.

The parallelized version of the code we present aims to minimize the time consumed in the computing of this aforementioned simulation. Furthermore, the results obtained will be analysed in the following sections of this paper.

II. Performance before parellelizing

Our simulation worked with a number of 5000 particles rendered, and the best sequential time we could achieve was that of 28.9 seconds. This will be the time utilized when calculating the speedup between sequential and parallel code.

As previously done in phase 1, we used gprof, perf and valgrind to analyse where the hotspots in our code were. Since, structurally, the code remained the same as the end results of phase 1, the hotspots had already been contemplated previously.

III. OpenMP

We enhanced the efficiency of the "PotentialAndAcceleration" function by parallelizing its main loop, responsible for computing potential energy and acceleration for 5000 particles. We employed a `#pragma omp parallel for reduction(+:Pot, a)` to introduce a multi-threaded execution within the for loop.

The reduction clause ensures the creation of private copies for the float variable Pot and the accelerations matrix (a) for each thread. Ultimately, it aggregates the values of these private variables into the original ones. This strategy eliminates data races among threads, as each thread operates on its distinct set of private variables.

This parallel approach surpasses the sequential version in terms of efficiency by concurrently processing loop iterations, significantly reducing computational time.

IV. Theoretical Results

Utilizing Amdahl's law, we computed the theoretical speedup, taking into consideration that the Search cluster, where our simulation was executed, had a capacity of 40 threads. Given that the code is nearly entirely parallelizable, the ideal speedup is directly proportional to the number of threads employed, reaching its maximum potential at 40 threads.
Illustratively:
2 threads result in a 2x speedup
8 threads yield an 8x speedup
40 threads achieve the maximum 40x speedup

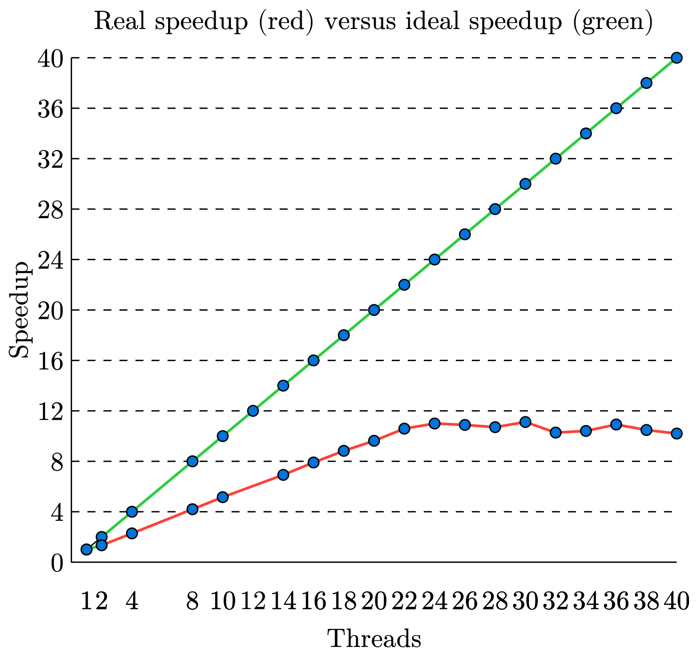
V. Practical Results

To test our code, we ran it multiple times in the Search cluster given by our university, with an increasing number of threads on each iteration. We noted down the real time in seconds necessary for the computing of the code (consultable at Table 1).

Number of threads	Real time (s)
1	28.637
2	21.558
4	12.652
8	6.881
10	5.604
12	4.746
14	4.174
16	3.655
18	3.273
20	3.002
22	2.729
24	2.628
26	2.655
28	2.699
30	2.6
32	2.812
34	2.777
36	2.647
38	2.757
40	2.833

Table 1: Timing results

We then calculated the speedup of each iteration, and compared it to the ideal Speedup curve (as can be seen in the graph bellow).



VI. Result Analysis

When comparing the ideal speedup with the real results we obtained, we can note a difference between the two, as the realistic results seem to stabilize after 24 threads with results between 10 to 11 times faster than the serialized code. This stabilization comes from the exaustion of the memory bandwidth. After 24 threads it seems that we have saturated the memory channel, making any threads beyond that have to wait in a queue longer to receive their data, basically eliminating any performance gain.

VII. Conclusion

In conclusion, our endeavor to parallelize a molecular dynamics simulation using OpenMP has proven to be successful in significantly reducing computational time. The performance improvement achieved demonstrates the effectiveness of parallelization in optimizing the computing of the simulation.

The theoretical analysis, based on Amdahl's law, provided insight into the potential speedup achievable with the parallel approach. However, practical results obtained through multiple runs on the university's Search cluster revealed that the real-world speedup deviated from the ideal projections. Notably, the performance gains stabilized after employing 24 threads, indicating a saturation point in memory bandwidth.

This observation suggests that beyond a certain threshold, additional threads face increased wait times in the data queue, offsetting the performance benefits. As a result, our realistic speedup plateaued at approximately 10 to 11 times faster than the serialized code after 24 threads.

In essence, while the parallelization of the molecular dynamics simulation effectively enhanced efficiency, it also unveiled limitations related to memory bandwidth constraints. This finding underscores the importance of considering hardware constraints when implementing parallel computing solutions. Despite the observed plateau, the achieved speedup remains a substantial improvement, showcasing the potential of parallelization techniques in optimizing resource-intensive scientific simulations.