Computação Paralela

1st Inês Ferreira
Informatics Department
Master in Informatics Engineering
University of Minho
Braga, Portugal
PG53879

2nd Marta Sá
Informatics Department
Master in Informatics Engineering
University of Minho
Braga, Portugal
PG54084

***Abstract*—This document is a model and instructions for LaTeX. This and the IEEEtran.cls file define the components of your paper [title, text, heads, etc.]. *CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

The aim of this project is to explore optimization techniques applied to a single thread program. The program in question that we have to analyse and optimise is part of a simple molecular dynamics' simulation code applied to atoms of argon gas. Throughout this project we tested different optimization methods and kept in the simulation code those that led to an improvement on several metrics, namely the execution time without changing the simulation output significantly. In section 2 we talk about these methods and explain how they contribute to an improvement of the execution time. Finally in section 3 we close this report with a brief conclusion.

II. Methods used to increase the program's performance

We started this work using *gprof* which is a performance analysis tool used to determine where time is spent during program execution. And with the help of this tool we concluded that most of the execution time was spent on two functions: *Potential()* and *computeAccelerations()*. Given this information, our focus became to reduce the execution time spent on these functions as much as possible.

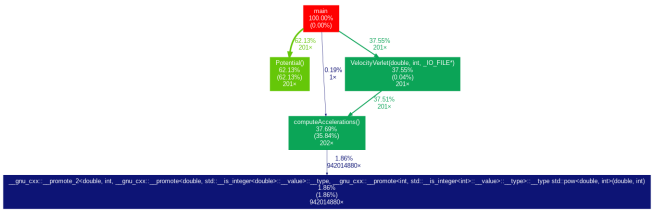

Fig. 1. Command *gprof ./MD.exe | gprof2dot | dot -Tpng -o original.png*.

A. Removal of Pow() and Sqrt()

Our first aproach, after experimenting with the optimisations flags, was to replace the functions *Pow()* and *Sqrt()* with other mathematically equivalent operations given that they are two functions that consume a lot of execution time. In the equations below we present the reasoning we followed to do so:

$$quot^2 = (sigma/rnorm)^2 \wedge rnorm = sqrt(r2) \equiv \quad (1)$$

$$quot^2 = sigma * sigma/rnorm^2 \wedge rnorm = sqrt(r2) \equiv \quad (2)$$

$$quot^2 = sigma * sigma/(sqrt(r2))^2 \equiv \quad (3)$$

$$quot^2 = sigma * sigma/r2 \quad (4)$$

And:

$$term2 = quot^6 \equiv \quad (5)$$

$$term2 = quot * quot * quot * quot * quot * quot \equiv \quad (6)$$

$$term2 = quot^2 * quot^2 * quot^2 \quad (7)$$

So to avoid executing unnecessary multiplications to calculate term1 and term2 and using the sqrt function unnecessarily we can simply consider:

$$quot = sigma * sigma/r2 \quad (8)$$

$$term2 = quot * quot * quot \quad (9)$$

$$term1 = quot * quot * quot * quot * quot * quot \wedge \quad (10)$$

$$term2 = quot * quot * quot \equiv \quad (11)$$

$$term1 = term2 * term2 \quad (12)$$

B. Removal of constants

When analyzing the functions, we noticed that some contained operations within cycles where, for example, two constants multiplied. And in cases where such a scenario occurred, we chose to create variables outside the cycle to contain the results of these operations so that it could later be used within the cycles without always repeating calculations leading to an increase of the execution time.

C. Transposing matrices and changing the order of cycles

As we know, data in computers is stored following a memory hierarchy. When the CPU fetches data from memory, it stores in the cache along with the data that was allocated in the nearby positions. So if it is then necessary to access nearby positions it will take less time, as it is faster to access the cache than the main memory. In the case of matrices, data is stored in rows, so it is faster to access positions in the same row and different columns than the other way around, due to spatial locality. It was taking this information into account that we decided to transpose the matrices *v*, *a* and *r* leading to a decrease of the number of cache misses.

D. Merging of functions

Afterwards, we noticed that both functions *Potential()* and *computeAccelerations()* perform the same calculation to determine the value of the variable *rSqd*. Therefore, in order to reduce the number of instructions, we choosed to merge both functions in the *computeAccelarations()* function and to make the variable *PE* global. As a consequence, we had to adapt the content of the cycles as a way to obtain the same final result.

```
for (i = 0; i < N-1; i++) {   // loop over all distinct pairs i,j
    for (j = i+1; j < N; j++) {
        // initialize r^2 to zero
        rSqd = 0.;

        //  component-by-componenent position of i relative to j
        rij[0] = r[0][i] - r[0][j];
        rij[1] = r[1][i] - r[1][j];
        rij[2] = r[2][i] - r[2][j];
        //  sum of squares of the components
        rSqd = rij[0] * rij[0] + rij[1] * rij[1] + rij[2] * rij[2];

        // Calculate the potential energy of the system
        quot = sigma2 * (1./rSqd);
        term2 = quot * quot * quot;
        term1 = term2 * term2;
        Pot += term1 - term2;

        //  From derivative of Lennard-Jones with sigma and epsilon set equal to 1 in natural units!
        rSqd_3 = rSqd*rSqd*rSqd;
        rSqd_7 = rSqd_3*rSqd_3*rSqd;
        f = 24. * (2./rSqd_7 - 1./(rSqd_3*rSqd));

        //  from F = ma, where m = 1 in natural units!
        a[0][i] += rij[0] * f;
        a[0][j] -= rij[0] * f;

        a[1][i] += rij[1] * f;
        a[1][j] -= rij[1] * f;

        a[2][i] += rij[2] * f;
        a[2][j] -= rij[2] * f;
    }
}

PE = 8. * epsilon * Pot;
```

Fig. 2. Part of the function *computeAccelarations()* code.

As we can see in the previous figure, we choosed to interate over the matrices using the original *computeAccelerations()* cycles, avoiding the use of *if* statements and making the code more vectorizable. However, in order to obtain the same final result, we had to calculate the *Pot* variable differently. Besides this we also noticed that due to the distributive property it is possible not to multiply by *epsilon* inside the cycle to obtain the *Pot* result but to multiply later outside the cycle.

E. Loop Unrolling

Finally, we decided to use loop unrolling in the cycle where we initialize the matrix *a* to allow two instructions to be executed at the same time, thus reducing execution time and enhancing vectorization. In addition to this, inside the function *computeAccelerations()*, we removed completely the cycles iterated by the variable *k* as it only iterated the same operation 3 times with different values of *j* and *i* and replaced it with the 3 operations and variables that were previously required in the cycle.

TABLE I
RESULTS OBTAINED

Metrics (Average of 10 runs)	Execution Time (s)	Number of Instructions	Clock Cycles	CPI	Cache Misses
-O0 (Default)	93,87+-38,20	622 946 629 283	302 136 452 682	0.48	221:u
-O3	46,70 +- 17,74	299 678 928 707	149 679 834 080	0.50	68:u
-Ofast	10,034 +- 0,390	52 233 737 783	32 970 353 734	0.63	78,497
Removal of Pow and Sqrt	8,206 +- 0,205	50 822 365 131	26 943 332 320	0.53	69 276
Removal of constants	8,022 +- 0,167	50 821 441 903	26 291 488 016	0.51	66 707
Transpose and Exchange of Cycles	8,580 +- 0,161	50 366 475 392	28 106 748 244	0.56	71 663
Merge	5,397 +- 0,140	30 214 432 538	17 454 459 118	0.57	46 608
Loop unrolling	5,340 +- 0,132	30 214 154 137	17 444 238 735	0.57	45 541

III. CONCLUSION

Observing the results obtained, we conclude that the best approach is to use the flag -Ofast along with the implementation of loop unrolling, vectorization and techniques that allow the reduction of the number of instructions.

Besides this, with the accomplishment of this practical work, the group was able to integrate concepts acquired during the classes about optimizations and analyze their impacts on performance. Additionally, it provided us a way to explore the *perf* tool for performance analysing. We believe that we have accomplished with objective of project, however there is room for improvement in the future, such as implementing a more parallelized algorithm and doing more testing with different metrics.