https://www.overleaf.com/project/5bd484c8e62c55235b9f4455

Universitat Autònoma de Barcelona

Parallel Programming

Sergio Oyaga & Carlos Mougan

C Programming and Performance Engineering

Submitted To:
Professor
Parallel Programming
Computer Science

Submitted By :
Sergio Oyaga
Carlos Mougan
Msc in Modelling
CRM

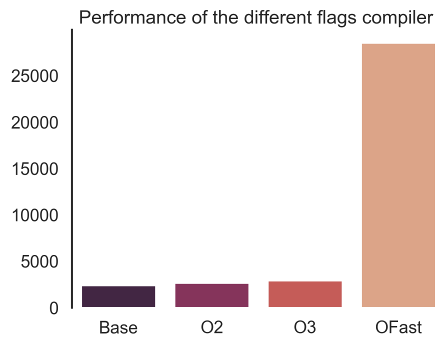
1 Introduction

When we talk about mathematical simulations we have to talk about the problem of size. Size is about the dimensionality of the simulation we are carrying on. In this assignment we are dealing with ”Newton’s Force Law” the dimensionality has to do with the size of the cube where the simulation is running on. In this program the cube is expressed as a 3-dimensional matrix: A triple for loop.

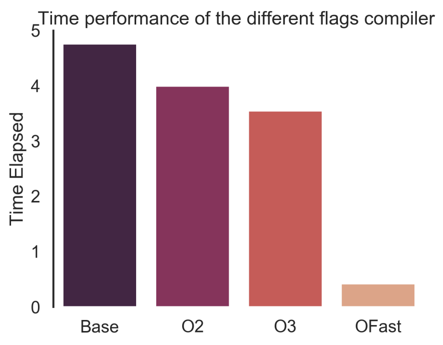
For this assignment we considered three different compilation flag options base,02, 03 & 0Fast. We will mainly use Ofast due to:

- **Ofast** is the Highest ”standard” optimization level. The included -fast-math did not cause any problems in our calculations, so we decided to go for it, despite of the non standard-compliance. ’Ofast’ gives permissions to the CPU to alter the execution of the different math orders.
- In the Figure 1:Compilation Flag metrics we can see that the best relation of Instructions and time is using the Ofast compiler. We can also appreciate that the time elapsed is faster to OFast than any other flag used

In conclusion we easily see that the OFast flag is way better than any other that we have used. The main reason not to use OFast is in cases ofs strict standards compliance and our required precision doesn’t allow us to use it.



(a) (Inst/Time)*10.000



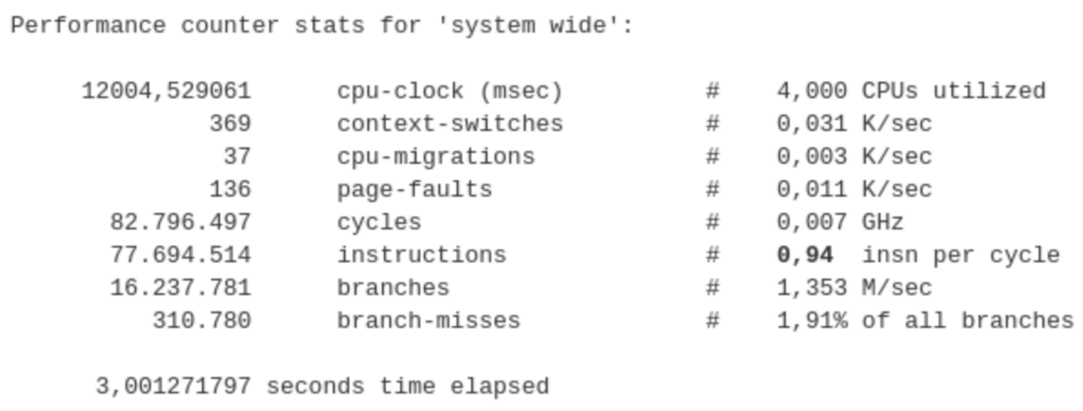
(b) Time Elapsed

Figure 1: Compilation Flag Metrics

2 Gather Data

For the different compilations of our code we have used the 'perf stat' command to obtain information of the different performance results. This command has as output raw datatable with time data Instructions, IPC and Clock Frequency of the execution of the code. From this results and with Equation (1) we can define perfomance.

$$Performance = \frac{Operations}{Instructions}\frac{Instructions}{Clockcycle}\frac{Clockcycle}{seconds} \tag{1}$$



```
 Performance counter stats for 'system wide':

      12004,529061      cpu-clock (msec)          #    4,000 CPUs utilized
               369      context-switches          #    0,031 K/sec
                37      cpu-migrations            #    0,003 K/sec
               136      page-faults               #    0,011 K/sec
        82.796.497      cycles                    #    0,007 GHz
        77.694.514      instructions              #    0,94  insn per cycle
        16.237.781      branches                  #    1,353 M/sec
           310.780      branch-misses             #    1,91% of all branches

       3,001271797 seconds time elapsed
```

Figure 2: Data table made by 'perf stat' on C

3 Temporal & Memory Complexity

In this first section we are going to discuss about the Temporal & Memory Complexity from N bodies problem.

In Figure:1 we can see the mean execution times versus the N bodies complexity.

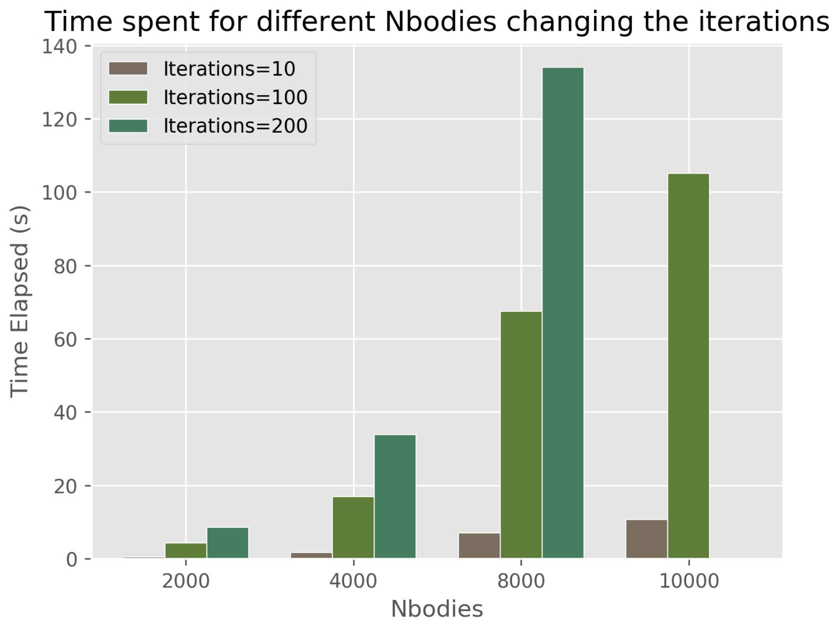


Figure 3: Bar plot of the Nbodies vs Time Elapsed with different iterations

In the bar graph is distributed by the different Nbodies that we have run (2000, 4000, 8000, 10000) in the x-axis, the vertical axis is the mean time elapsed. We have run at least three times each Nbodies with different iterations settings in order to have some statistical information.

We have calculated and plotted the standard deviation, but we can not appreciate it due to the small error that we have. This encourages us to just do one measure given the size of this deviation. From here we will only do one measure in the optimization part.

The iteration=10 and Nbodies =2000 can't be seen because it is in relation much smaller than the others. For the same reason we have decided not to plot the biggest one Iterations=200 and Nbodies= 10000.

To solve this visualization problem we have decided to plot Performance and Nbodies. In figure 4 we can see that the performance slightly increases with Nbodies. This graph is done only for 10 iterations loop.

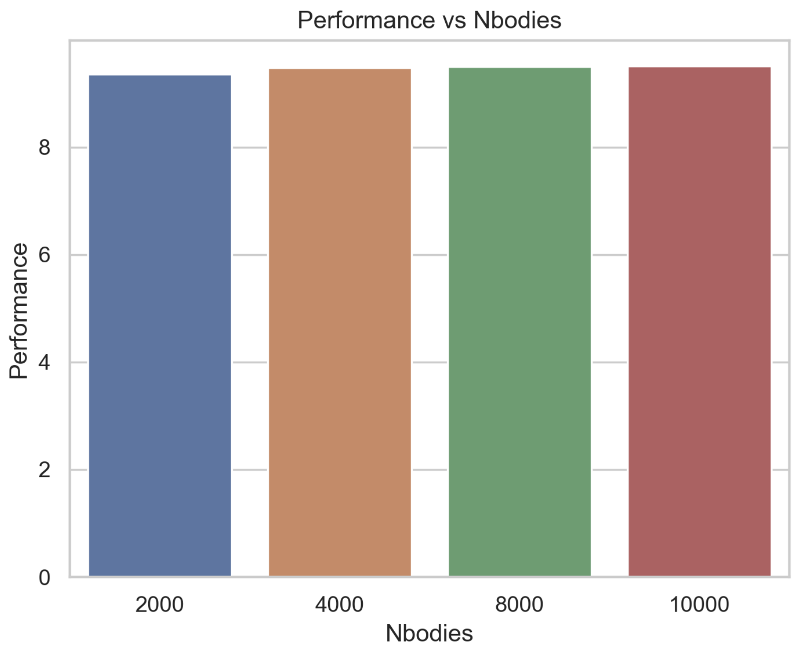


Figure 4: Performance of the different Nbodies size

4 Optimization

We have been researching which code optimization we can make. From the initial 'Nbodies' script we have decided to do the following optimizations:

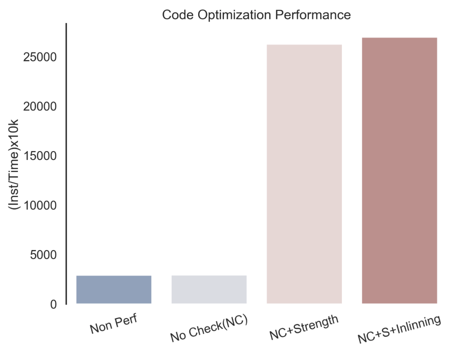
Non Perf Raw given code.

No Check This optimization avoids checking the computations with itself. Reduce the number of checks made in the main code.

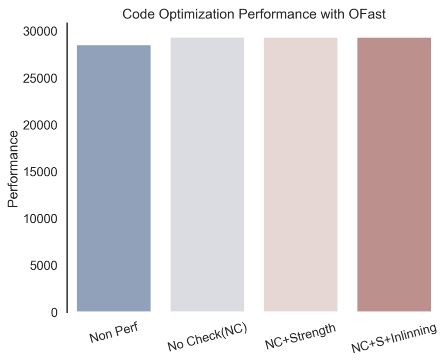
NC +Strength Here we use the No Check optimization and the reduction of the Strenght in the mathematical calculations.

NC+S+Inlinning In this last optimization we have used all others and introduce all the function inside the main code.

We have applied this optimizations to the code and measured its performance that can be seen in Figure 5. We have all the performance with different flags O3 and Ofast in order to see if there are same differente between them.



(a) Optimizations with O3



(b) Optimization with Ofast

Figure 5: Performance optimizations with different flags metrics

5 Dicussion

In the *temporal and memory complexity* we can detect that the relation between the x and y axis in Figure 3 is fitted **quadratic(x2)** for constant iterations and it behaves linear for constant Nbodies, as we have thought theoretically. To show a more representative value we decided to plot the performance instead the elapsed time.

For the *Ofast flag compiler* (Figure 5:b) we see that the optimizations we made dont improve significantly. After seeing this results we decided to do use the O3 flag compiler. In this case we see that the Strength code optimization improves the performance noteworthy. So after seeing this results we can see that Ofast reduces the strength of mathematical calculations by itself.

6 Conclusion

The performance of a program is always something we always have to take into consideration when developing a code, thought the Ofast compiler already takes some optimizations into account.

Before we start to write any code we have to think how are we going to implement our solution, taking some decissions with time can lead to an optimal performance.