PACS LABORATORY 2

by

Daniel Ariza Antón (775545) and Marcos Ilarraza Sarto (797720)

In this lab the objective is to learn how to use profiling tools and write down our results for the four exercises in which this lab is divided.

EXERCISE 1

For this exercise we have modified both our implementation and the Eigen version of the multiplication of matrices in order to make the process of creating and multiplying the matrices. We have also tested the resulting code with the *time* command in order to compare the times.
To avoid outliers we have executed each test five times and taken the mean of the times (except for our implementation with the 5000x5000 matrix, which takes around five minutes to finish its execution. For this one we have used two executions).

Time \ alg_dim	**ours_1200**	**ours_2500**	**ours_5000**	**eigen_1200**	**eigen_2500**	**eigen_5000**
User time mean	3,8618s	36,0584s	5min 26,9375s	0,222s	2,3096s	17,9638s
Sys time mean	0,0046s	0,0102s	0,0555s	0,0048s	0,0126s	0,1152s
Clock time mean	3,86474	36,06636s	5min 27,0587s	0,27366s	2,32048s	17,98862s
User time St. Dev.	0,100427586 s	0,03068876s	0,0516188s	0,109057324s	0,012856905s	0,070037133s
System time St. Dev.	0,001516575 s	0,003271085 s	0,03323402s	0,001643168s	0,002302173s	0,181726993s
Clock time St. Dev.	0,099743987 s	0,033372264 s	0,01074802s	0,003525337s	0,013653095s	0,064773004s

The main point of *clock* is that it measures the time only in a section of code. While time measures all the time from the beginning to the end of the execution of the program. This is important in the case of having some sections that you don't want to include (in our case it would be the filtering of the id of the test to know which one to execute).
The reason why it is bigger than the sum of the user and system times (or close to it) is because clock measures the equivalent of what the *time* command names *real time*, which includes the time in which the program isn't working.

EXERCISE 2

For this exercise we have modified the code from the previous exercise in order to change the *clock* calls by the *gettimeofday* calls. We have also tested the resulting code with the *time* command in order to compare the times.
To avoid outliers we have executed each test five times and taken the mean of the times (except for our implementation with the 5000x5000 matrix, for the same reason given in the previous exercise).

Time \ alg_dim	**ours_1200**	**ours_2500**	**ours_5000**	**eigen_1200**	**eigen_2500**	**eigen_5000**
User time mean	3,8062s	36,2124s	5min 28,862s	0,2684s	2,2946s	17,862s
Sys time mean	0,0042s	0,0108s	0,06s	0,0054s	0,0104s	0,0276s
gettimeofday time mean	3,8171818s	36,369825s	5min 30,353096s	0,2724956s	2,3073228s	17,9179662s
User time St. Dev.	0,023636836 s	0,293820183 s	0,01414214s	0,001140175s	0,013427584s	0,062405929s
System time St. Dev.	0,00130384s	0,008729261 s	0,00707107s	0,00181659s	0,003781534s	0,00838451s
gettimeofday time St. Dev.	0,024660528 s	0,305438953 s	0,00996526s	0,001937489s	0,010445686s	0,064334992s

The main point of *gettimeofday* is exactly the same as *clock*'s: to measure only a section of the code. As in the case of *clock*, *gettimeofday* measures *real time* instead of *user* and *system* times. But if we take a look at the *ours_5000* test we can see that the difference is about two seconds between the sum of *user* and *system* times and the *gettimeofday* time, whereas with *clock* the difference is about 0.6 seconds. In the other tests the difference is not that big because of the small lapse of time they cover.

EXERCISE 3

For this exercise we executed once each of the programs with the *strace* command. We have considered not necessary to execute them more than once since every call should be used the same amount of time for this kind of program. The dimensions of the matrices used are 2500x2500 (6.250.000 elements).

The calls we have found interesting are:

- *clock_gettime*: This is the system call that is called when executing *clock*. It is executed twice because of we calling it that exact amount of times. But what intrigued us is the lack of a counterpart for *gettimeofday*, for which we have checked that the system calls with the same version of the multiplication share the same types of system calls, but lacking the calls corresponding to *clock*. This suggests that *gettimeofday* doesn't make use of system calls.
- *mmap*: This system call is called when mapping files (data) or devices into memory. The instances with same method of multiplication share the amount of calls, being our implementation the one who calls it 17 times and the Eigen version which calls it 19 times. Therefore, the difference must be due to the method of multiplication.
- *munmap*: This system call is called when unmapping files (data) or devices from memory. As happened with *mmap*, the amount of calls is the same for the calls with the same multiplication algorithm. Our implementation calls it once and the Eigen version calls it 6 times.
- *write*: This system call is called when writing to a file descriptor. It is a system call whose amount of calls is different for each one of the four instances we tested. Ours+*clock* calls it 3 times, Ours+*gettimeofday* calls it 4 times, Eigen+*clock* calls it 2 times and Eigen+*gettimeofday* calls it 3 times. Both of the *clock* versions call it one time less than their *gettimeofday* counterparts, so it could be related to this last method. Also, Eigen versions call it one time less than their respectives counterparts, so it is also related to the implementation of the multiplication.

Algorithm\SysCall	**clock_gettime**	**mmap**	**munmap**	**write**
Ours + clock	2	17	1	3
Ours + gettimeofday	0	17	1	4
Eigen + clock	2	19	6	2
Eigen + gettimeofdaay	0	19	6	3

EXERCISE 4

For this exercise we executed once each one of the four versions of the code with the *perf* command. For each call we used the options *stat*, to get only the most common performance statistics, and *-r* with 5 as its argument, to get each one executed 5 five times and obtain directly the means and the standard deviations. We considered it was important to repeat the calls more than once since almost all these metrics can be related to time.

The following tables include the means of the metrics we have considered the most important:

Alg\Metric	Instructions	branch-misses	branches	task-clock	time elapsed
Ours + clock	174.664.232.382	8.773.757	16.376.841.306	36,12187s	36,2035s
Ours + gettimeofday	174.664.000.986	8.777.448	16.376.776.576	36,23350s	36,614s
Eigen + clock	27.610.770.932	483.427	367.842.420	2,30985s	2,31507s
Eigen + gettimeofday	27.610.770.962	534.814	367.842.432	2,29404s	2,295563s

The *task-clock* metric specifies the time that the program has been executing, whereas *time elapsed* corresponds to the real time it has taken to finish its execution. The last one is much closer to the actual mean of the times measured by the *clock* and *gettimeofday* methods.
With both metrics we get the percentage of the CPU used, resulting in our implementations having a 99'8% and a 99% of usage with *clock* and *gettimeofday* respectively and Eigen versions having a 99'8% and a 99'9% respectively. Since this relation depends on how the task manager takes care of the execution, we can say that the CPU is used efficiently by the programs (being better with Eigen versions).
The *branches* and *branch-misses* can be used to calculate the percentage of wrong branch predictions that the program has made. It is true that both metrics are way higher in our implementation than in Eigens, making Eigens performance better, however the *percentage of branch-misses* is higher in Eigens versions. This means that although it has less branch instructions executed, it has more branch cycles, since the most common miss is to predict a positive jump when it actually isn't.
The *instructions* metric specifies the amount of instructions that have been executed in the CPU, so at first we thought that for both versions algorithm+clock would be higher than algorithm+gettimeofday and vice versa. But if we check the values on the table *Ours+clock* number of instructions is bigger than *Ours+gettimeofday*. While in Eigens versions it is the opposite. We have checked that the standard deviation is zero for all cases, so it means that the number of instructions of each method to measure time must be different, not regarding the actual call, but the arrangement to get to it.