Programming and Architecture of Computing Systems
University of Zaragoza

Laboratory 2: Instrumented and Event-based Techniques

Alberto Zafra Navarro & Daniel Sanz Valtueña

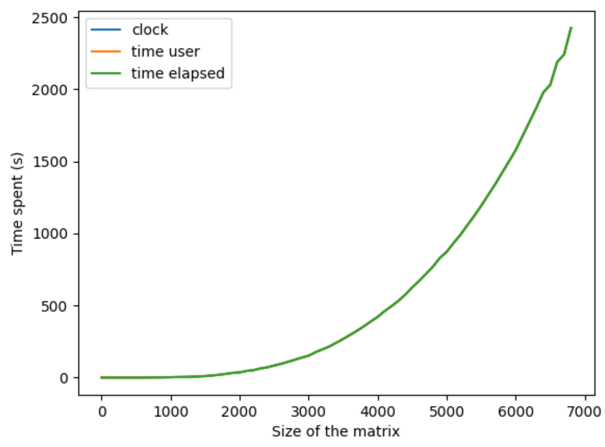
During the second laboratory session, some profiling techniques (Instrumentation and Event-based) have been performed on the programs coded in the first laboratory session.

In the first part, the library functions called `clock()` and `gettimeofday()` will be implemented manually within the code to obtain the execution time of the programs or even specific parts of the program, such as matrix declarations or initializations. The system calls of these programs will be also monitorized with the `strace` command, whereas in the second part, the behavior of the hardware and its associated events will be measured by some metrics provided by the command-line tool `perf`.

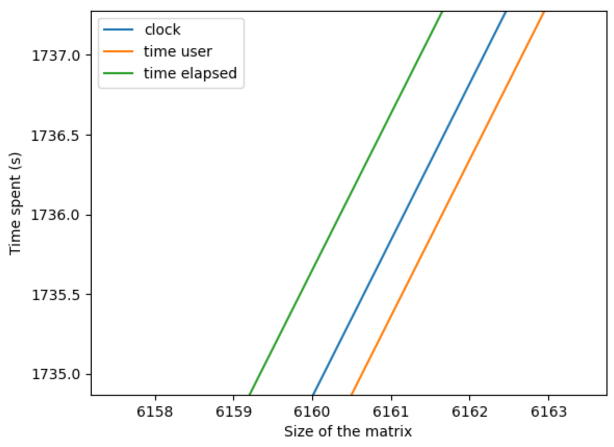
1 Instrumentation profiling

1.1 clock()

The function `clock()` is used to measure the amount of CPU time consumed by a program or a specific part of the program. It returns the number of clock ticks (processor time units) that have passed since the program started or since a specific point in time, afterwards it is possible to convert this number of ticks to seconds by dividing the measured cycles by the constant `CLOCKS_PER_SEC`. On the other hand, `time` command used in session 1 also did so, but splitting up the execution time in 3 different metrics and returning them directly in seconds. In this section, we will present the calculated execution times with both `clock()` function and `time` command in order to compare them.

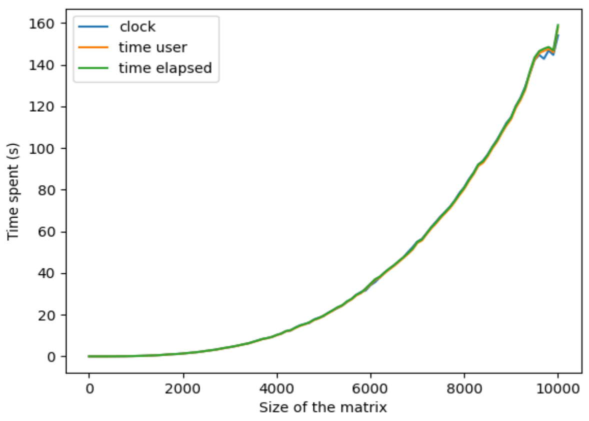


(a) General Plot

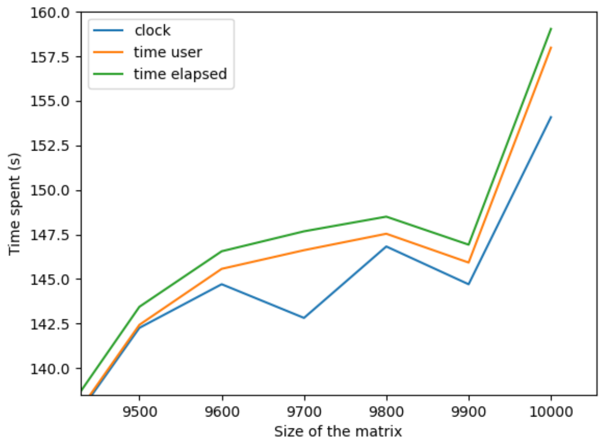


(b) Zoomed Plot

Figure 1: Time measured by the clock() and time methods using the standard matrix multiplication.



(a) General Plot



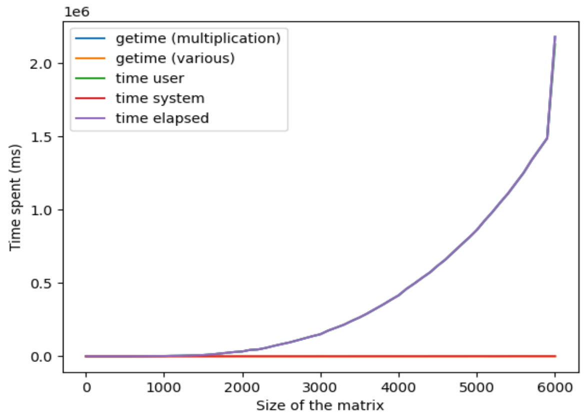
(b) Zoomed Plot

Figure 2: Time measured by the clock() and time methods using the Eigen multiplication.

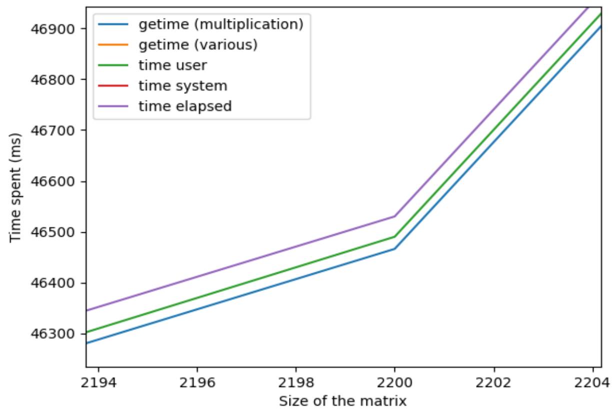
Figures 1 and 2 show the comparison between the different measurements provided by `clock()` function and `time` command. They also show a zoomed comparison of the three measurements where it is possible to see that there is a slight difference in the results provided by each measurement tool when the size of the matrices increases. The `clock()` function is suposed to provide a more precise measurement, as it is implemented within the code but, when examining the data from each metric separately, it becomes apparent that the `time` command offers as well a precise measurement of program execution time, although it is executed externally to the program (from the terminal). This is evident from the average standard deviation between both methods of 0.195 seconds for standard matrix multiplication and 0.213 seconds for Eigen matrix multiplication.

1.2 gettimeofday()

This second function also permits the programmer to measure the program's execution time. In this case, the code is divided into two distinct parts: specifically, the matrix multiplication, and the remaining aspects of the program such as matrix declarations and memory allocations, among others. Consequently, two separate execution times will be determined.

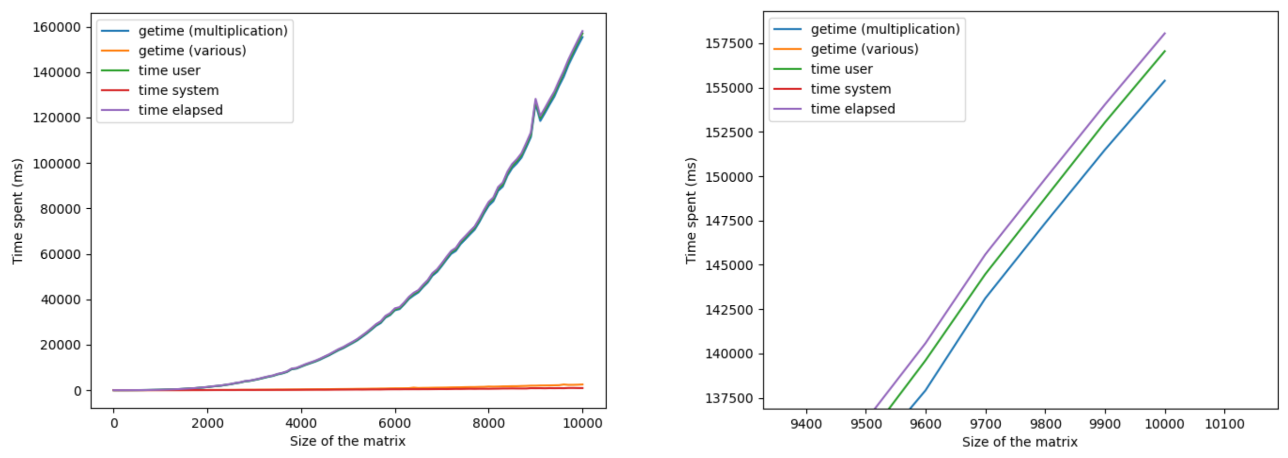


(a) General Plot



(b) Zoomed Plot

Figure 3: Time measured by gettimeofday() and time methods using the standard matrix multiplication.



(a) General Plot (b) Zoomed Plot

Figure 4: Time measured by the gettimeofday() and time methods using the Eigen library.

Figures 3 and 4 show the comparison between the different measurements provided by the `gettimeofday()` function and the `time` command; and a zoomed comparison of the three measurements. In the previously mentioned graphics it is easily distinguishable the purpose of each of the metrics provided by the `time` command. Similar to the `clock()` function, the *user* and *elapsed* metrics measured by the `time` command provide an approximate measurement of the time spent during the whole execution. Furthermore, it is obvious that this measurement is directly related to the time spent during the matrix multiplication, as it is the core of the program. Therefore it is understandable that the time measured with the `gettimeofday()` function on that part of the program is similar to these metrics.

On the other hand, the time spent during the memory reservation, assignation and deallocation measured by the `gettimeofday()` function (measurements that have been grouped under the label of "various") is almost negligible compared to the matrix multiplication metrics. In spite of being proportional to the matrices' size, its maximum (about 5.3s for the standard matrix multiplication and about 2.5 s for the Eigen multiplication) is not comparable with the multiplication execution time, as the overhead of the initialization with respect to the entire execution time is around 1.588 % when calculating the multiplication of a matrix of size N=10 000; having an average overhead of 7.6% for the standard multiplication and an average overhead of 8% for the Eigen multiplication. It is worth mentioning that the overhead of the initialization time is inversely proportional to the size of the matrices.

Finally, it is possible to state that the *system* metric provided by the `time` command is somehow related to the "various" time measured by the `gettimeofday()` function as the system metric represents the time the CPU spent running system-level code on behalf of the command or program. This includes time spent in the kernel handling system calls, file I/O, and other system-related tasks. While it's not an exact measure of initialization time, it can include some overhead related to the setting up of the matrices.

1.3 strace

The `strace` command is used to monitor the system calls made by a program. It shows the consumed time by each system call and which and how many times, the system calls are called during the execution of the program. Comparisons between standard matrix multiplication and Eigen library have been carried out, both in cases using `clock()` and `gettimeofday()` functions. The chosen system calls for analysis are "brk" (used for memory allocation), "stat" (used to retrieve information about files), "mmap" (used for memory mapping) and "munmap" (used to unmap memory). Tables 1, 2, 3 and 4 show the information obtained by the `strace` command during the program execution for each of the aforementioned system calls with a size of 6000 elements per row.

syscall	% time	sec	usecs/calls	calls	errors
brk	95.28	0.069	11	6006	
stat	1.40	0.001	1	752	658
mmap	0.17	0.000	5	24	
munmap	0.02	0.000	13	1	

Table 1: Standard matrix multiplication using clock().

syscall	% time	sec	usecs/calls	calls	errors
brk	91.84	0.064	10	6006	
stat	2.87	0.002	2	752	658
mmap	0.18	0.000	5	24	
munmap	0.02	0.000	14	1	

Table 2: Standard multiplication using gettimeofday().

syscall	% time	sec	usecs/calls	calls	errors
brk	0.01	0.000	1	3	
stat	2.93	0.002	2	752	658
mmap	0.24	0.000	4	29	
munmap	91.11	0.049	8178	6	

Table 3: Eigen matrix multiplication using clock().

syscall	% time	sec	usecs/calls	calls	errors
brk	0.01	0.000	2	3	
stat	3.85	0.002	3	752	658
mmap	0.20	0.000	4	29	
munmap	89.64	0.055	9165	6	

Table 4: Eigen multiplication using gettimeofday().

In general it can be seen that the usage of the ”brk” system call is specially high when the standard matrix multiplication is executed, due to the enormous quantity of memory required for evaluating the multiplication of each cell individually. In contrast, this particular system call sees minimal usage during the execution of the program utilizing the Eigen library. This is because the library optimizes memory allocation within its procedures to the bare minimum. Opposed to this, the matrix multiplication using the Eigen library contains an excessive quantity of memory deallocations which may be due to the use of vectorization, as it can perform computations efficiently on a large set of data with fewer memory allocations, reducing the need for dynamic memory allocation. Regarding to the ”stat” and the ”mmap” system calls, there is not a big variation, as the memory mapping is simillar in each version and the only difference is shown in the ”stat” system call, because the number of files opened by the Eigen matrix multiplication is bigger due to the usage of the library itself.

2 Event-based profiling

2.1 perf

The command-line tool `perf` is used to profile and trace the performance of programs using hardware performance event counters. As in the previous section, comparisons between different matrix multiplication and execution time measuring methods will be performed. The chosen metrics for evaluating the usage of the `perf` command are ”cpu-clock” (represents the CPU time used by the program), ”cycles” (represents the computational work performed by the CPU), ”instructions” and ”branches” (represents the control flow changes that are executed, such as if statements and loops). Table 2.1 shows the information obtained by the `perf` command during the program execution for each of the aforementioned metrics with a size of 3000 elements per row.

	standard + clock	standard + gettimeofday	Eigen + clock	Eigen + gettimeofday
cpu-clock [s]	1224.766	1453.04	39.689	38.762
cycles [e+09]	842.457	1360.786	26.286	24.370
instructions [e+09]	456.580	717.729	58.584	58.250
branches [e+9]	57.779	128.560	1.836	1.768

Table 5: Perf metrics evaluation.

As it has been demonstrated and justified in previous sections, the number of CPU cycles and the total amount of CPU time used by the different implementations is significantly bigger in the standard matrix multiplication, this is because the standard multiplication method requires to execute more instructions than the one using Eigen. Logical result, as the Eigen vectorization implicitly performs several operations just with one instruction. For the same reason, more branches are called for standard than Eigen method but is interesting to highlight the proportion of branches for the total instructions. Between 12.6% and 17.9% of the instructions for standard multiplication are branches, while for Eigen multiplication, just a 3% of the instructions are branches. The main reason for that difference is that standard multiplication is performed through 3 nested loops (loops are branches) and the Eigen library utilizes other different methods for matrix multiplication, apart from the already mentioned vectorization.