

# Arquitectura de ordenadores. Practice 3.

Pablo Cuesta Sierra, Álvaro Zamanillo Sáez

November 10, 2021

## Contents

<a href="#">Exercise 0</a>	1
<a href="#">Exercise 1</a>	3
<a href="#">Exercise 1</a>	3

## Exercise 0

We have been submitting our tasks to the *MV* queue for all exercises of the practice because it returned the best (smoothest) results.

After running the commands shown in the statement, we got these results.

As can be seen in figure 1, the first level is the only one where there are two different caches: one for data and another for instructions. Both are of the same kind: size of 32768 B (32 KB), with blocks of 64 B and 8 ways. In the second level, there is a unique cache which is larger (2097152 B = 2 MB) but has the same degree of associativity. For the third level, the size increases (16777216 B = 16 MB) as well as the associativity (16 ways) whereas the block size remains the same.

```
LEVEL1_ICACHE_SIZE      32768
LEVEL1_ICACHE_ASSOC      8
LEVEL1_ICACHE_LINESIZE  64
LEVEL1_DCACHE_SIZE      32768
LEVEL1_DCACHE_ASSOC      8
LEVEL1_DCACHE_LINESIZE  64
LEVEL2_CACHE_SIZE      2097152
LEVEL2_CACHE_ASSOC      8
LEVEL2_CACHE_LINESIZE  64
LEVEL3_CACHE_SIZE      16777216
LEVEL3_CACHE_ASSOC     16
LEVEL3_CACHE_LINESIZE  64
LEVEL4_CACHE_SIZE       0
LEVEL4_CACHE_ASSOC      0
LEVEL4_CACHE_LINESIZE  0
```

Figure 1: Output of `getconf -a | grep -i cache`

Here is the info of the each of the processors (this is processor 0, but they are all the same):

```
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 26
model name : Intel Core i7 9xx (Nehalem Class Core i7)
stepping : 3
cpu MHz : 2399.996
cache size : 16384 KB
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
flags : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush mmx
      fxsr sse sse2 syscall nx lm constant_tsc rep_good xtopology unfair_spinlock pni ssse3
      cx16 sse4_1 sse4_2 x2apic popcnt hypervisor lahf_lm
bogomips : 4799.99
clflush size : 64
cache_alignment : 64
address sizes : 46 bits physical, 48 bits virtual
power management:
```

Figure 2: First lines of `/proc/cpuinfo`

The following diagram shows the architecture of the different caches in each of the 8 cores.

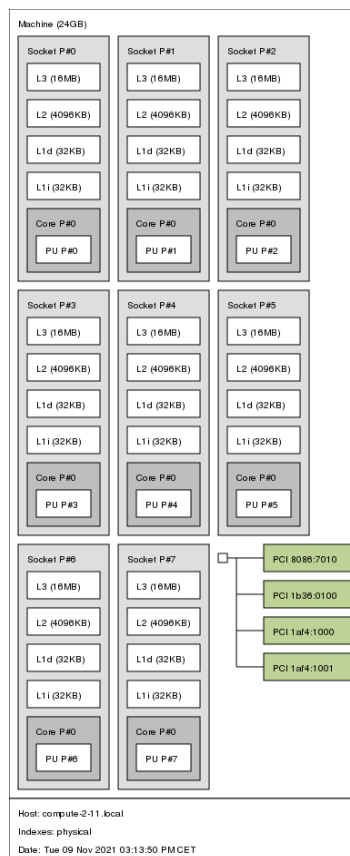


Figure 3: Topology of the used CPU.

## Exercise 1

The following image shows a graph with the execution time of both programs for different sizes of the matrix.

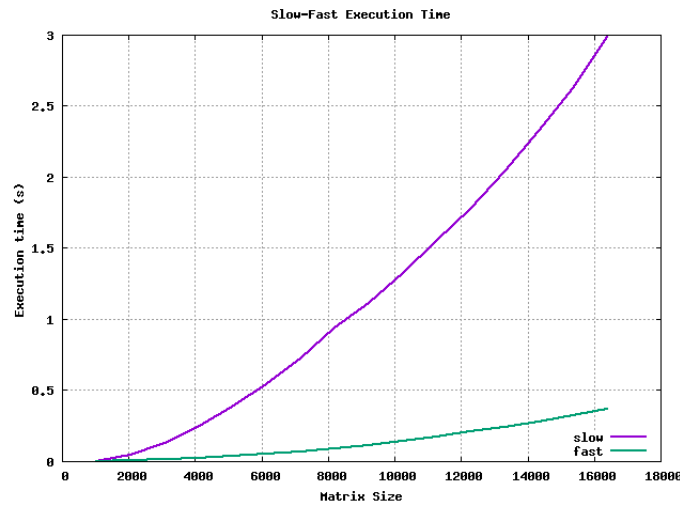


Figure 4: Slow vs Fast execution times.

As pointed by the graph the execution time for small values of  $N$  is similar for both programs (most of the matrix can be stored completely in the cache). However as  $N$  grows the execution time for the slow program shows a quadratic behaviour while the fast version barely grows.

This is due to the fact that the matrix is stored by rows in main memory, and therefore stored in the cache by rows too. The slow version access the matrix by columns which leads to a higher number of misses and consequently, a higher number of accesses to main memory.

For obtaining the results, it is necessary to run the simulation several times in order to calculate a mean and que no !!!!!!!dependa de un momento concreto!!!!!!!. Furthermore, the chosen execution pattern alternates between the slow and fast version to avoid the program to reuse the cache from the previous execution (with the same size).

## Exercise 1

First we will analyze the write misses:

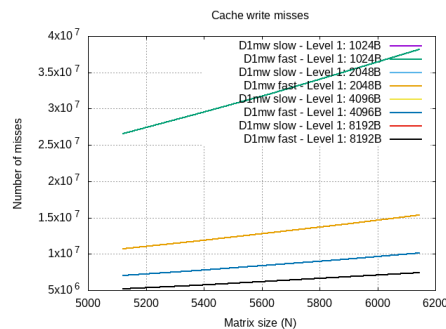


Figure 5: Cache read misses.

Looking at the graph it seems that only four lines have been plotted, however this is not the case. What is happening is that the slow and fast version have basically the same number of write misses (for the same execution values of  $N$  and size of the cache). The reason behind is that the result matrix is written in the

same order for both programs: by rows. If the data files are analyzed, the fast version has a few more misses which could be due to the fact that this program also has to calculate the transpose.

On the other hand,