Task 3
Computer Architecture

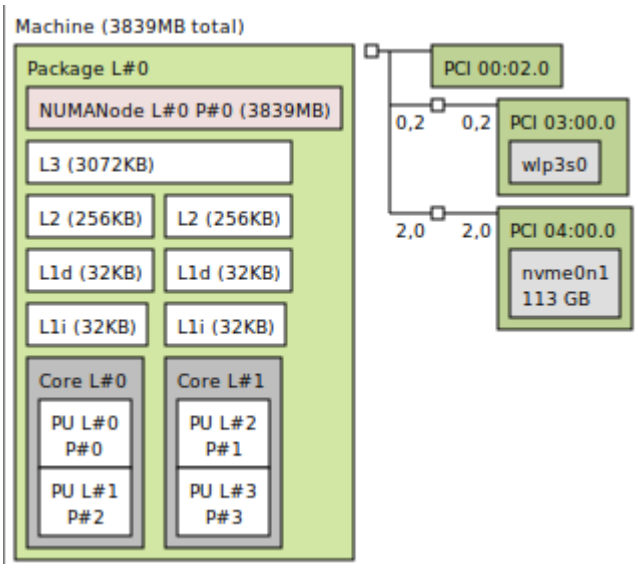
3rd year Computer Science and Engineering
Escuela Politéctina Superior
Universidad Autónoma de Madrid

Pablo Ernesto Soëtard García

Exercise 0:

This is the information retrieved form the computer:

```
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                 262144
LEVEL2_CACHE_ASSOC                8
LEVEL2_CACHE_LINESIZE             64
LEVEL3_CACHE_SIZE                 3145728
LEVEL3_CACHE_ASSOC                12
LEVEL3_CACHE_LINESIZE             64
LEVEL4_CACHE_SIZE                 0
LEVEL4_CACHE_ASSOC                0
LEVEL4_CACHE_LINESIZE             0
```



As we can see, it has 3 levels of cache:

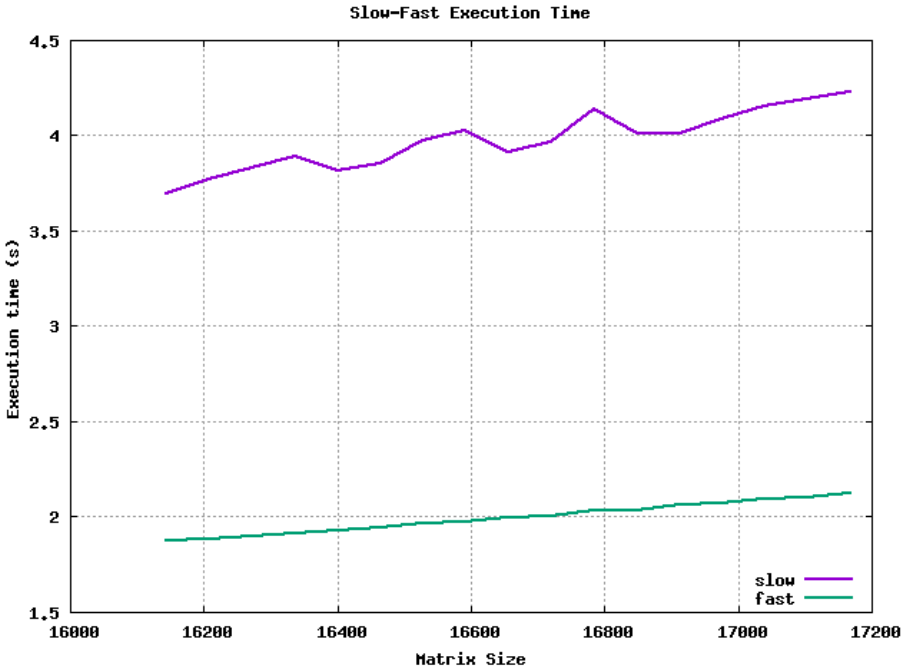
- Each core has two 32KB, 8 way associative, 64 bytes blocks L1 caches, one for instructions and the other one for data, in total 4 L1 caches.

- Each core has one 256KB, 8 way associative, 64 bytes blocks L2 caches, in total 2 L2 caches.

- Then we have a shared 3072KB, 12 way associative, 64 bytes blocks L3 cache.

Exercise 1:

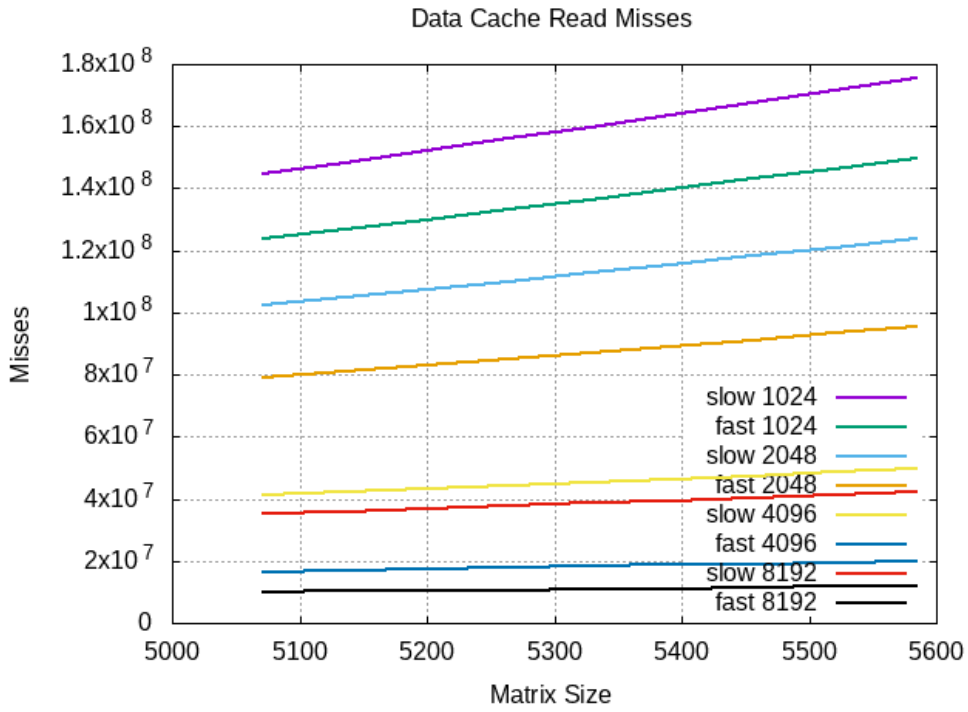
The reason why each measurement must be taken several times is due to the non "ideality" of the setup, as we are running our program in a computer that is executing other programs at the same time than ours. This affects the measurements "randomly" as in certain moment the processor could stall the execution of our program and replace it momentarily by the execution of another one that was waiting for the processor, thus affecting the execution time of our program. That is why we need to take several measurements of the program time and average them to get a better approximation to the real execution time of our program.
That likeliness of the execution of our program being interrupted by the processor increases as the program takes more time to execute, thus bigger matrix processing programs must have a longer execution time also due to that interruption time.

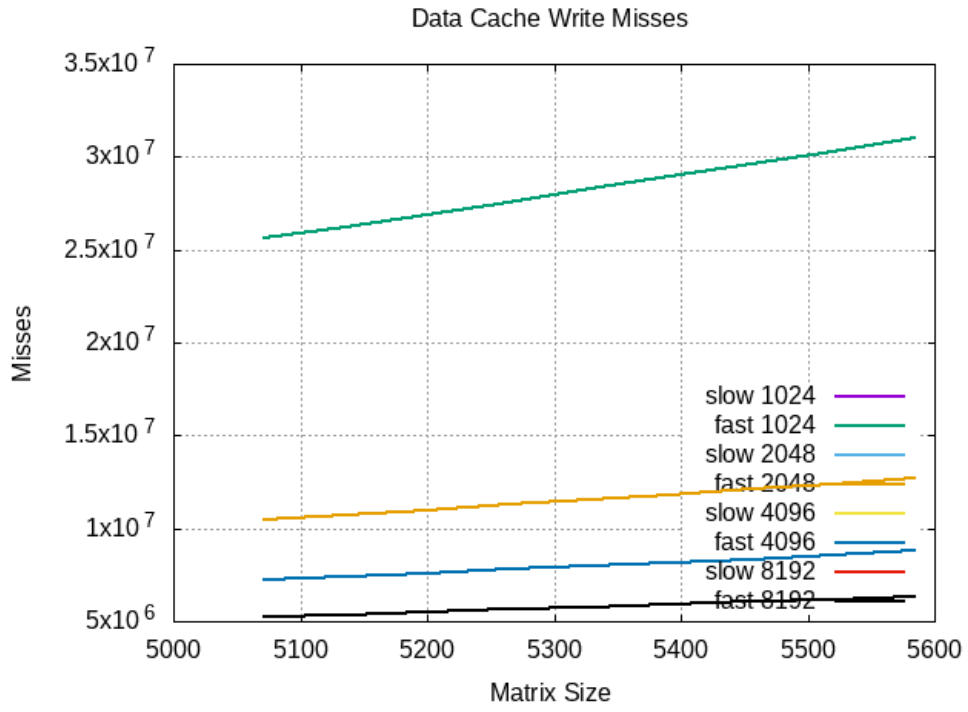


To generate the plot we have taken as a starting point the slow_fast_time.sh script given, and we have modified it in such a way that it creates a temporary file 'raw_data.dat' that stores, as the name suggest, the raw data obtained in all the executions, then we average each measurement with an awk script and store that result in 'time_slow_fast.dat', then that same script generates the plot. The reason why the execution time for small matrices is similar is due to its lower size that allows the cache to hold almost all the matrix and thus, the way accessing the matrix does not affect that much to the execution time. The plot above has been obtained with 7 iterations of data samples (maximum number of iterations that the cluster allowed to run in 1 hour), as we can see, there is a significant difference between slow and fast execution times. This is due to the way each program traverse the matrix to perform their calculations, that in one case is favorable to the locality principle, and therefore for the number of cache misses being lower and the execution time too, and in the other no.

Exercise 2:

The plots obtained by executing the 'slow_fast_cache.sh' script that we have created for this exercise in the cluster are the following ones:





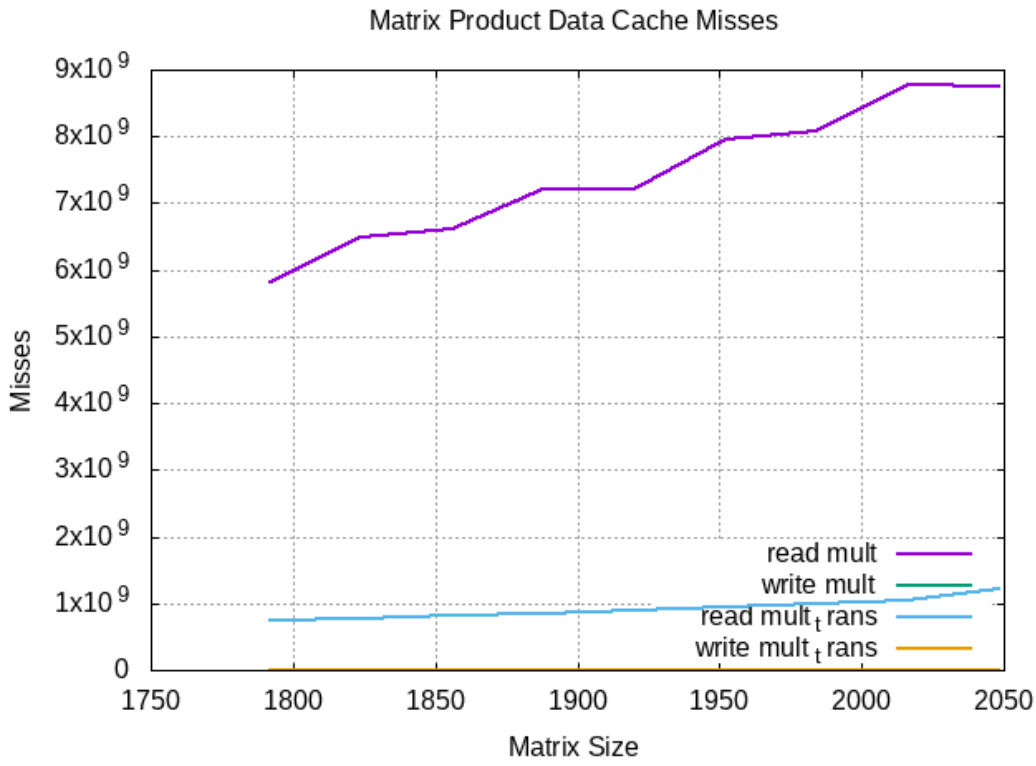
As we can see in the plots above, there is a visible trend when varying the cache size, as we increase the cache size, the misses decrease, this is true for both, fast and slow versions, that is because more blocks fits in one cache and thus, they are less often replaced due to the logicality principle.
If we look at an specific cache size and we compare the slow and fast programs, the results vary between the Read and Write misses:

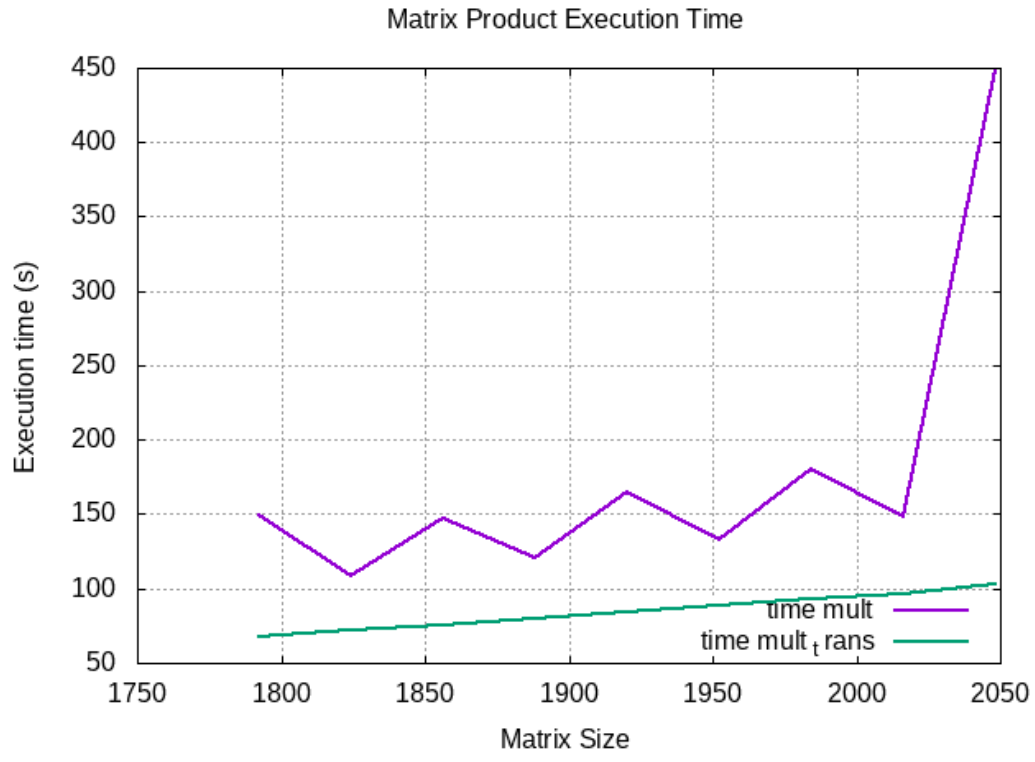
- In the read misses plot, the slow program always has more misses than the fast, this is due to the way each program traverse the matrix. The slow program traverses the matrix by columns, causing more misses due to the locality principle than the fast program (that traverses the matrix by rows), as explained in previous exercises.

- In the write plot, we can see that both slow and fast programs have the same write cache misses given a specific cache size. This is because both programs perform the same writes, one for each element of the matrix.

Exercise 3:

The plots obtained by executing the 'multiplication.sh' script that we have created for this exercise in the cluster are the following ones:





As we can see in the plots above, we obtain similar results to the ones from the previous exercise, there is a visible trend when varying the matrix size, as we increase the matrix size, the misses and execution time increases too, this is true for both, multiplication and multiplication_transpose versions, but with different rates.

If we look at the data cache misses plot, the results vary between the Read and Write misses:

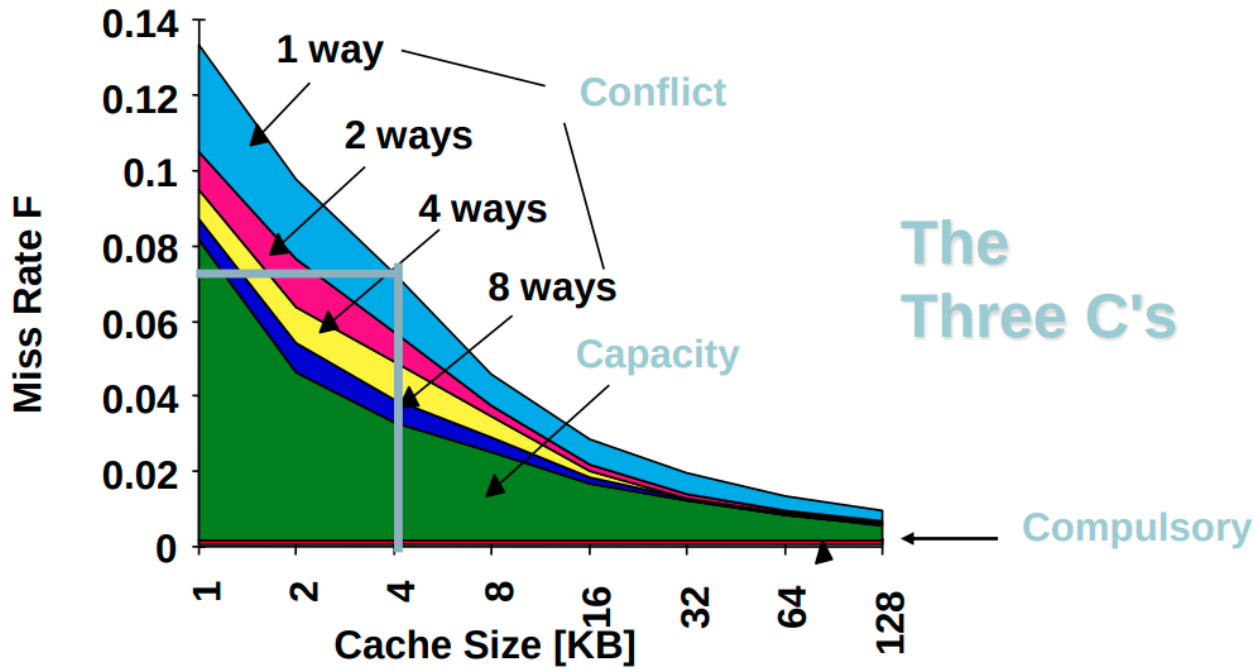
- The read misses for the multiplication program are greater than the ones for the multiplication_transpose, this is due to the way each program traverse the matrix to perform the multiplication. The multiplication program traverses the matrix by columns, causing more misses due to the locality principle than the multiplication_transpose program (that, after transposing the matrix, it traverses it by rows), as explained in previous exercises.

- The write misses for both multiplication and multiplication_transpose are much lower than those of reading, this is normal, as you only have to write once each 2N reads when you multiply two matrices, being N the size of the matrix.

We can observe the stalled time taken by the processor from our process, that we mentioned in exercise 1, is much more significant in the execution time plot. We would expect a polynomial expression for the time mult functions, but instead we get that jittering sort of exponential, that jitter is caused by the "random" time that the processor stops our process in order to execute others.

Exercise 4:

As the cluster has been too busy while we where doing this exercise we have not been able to carry out the experiments needed for this part of the assignment, but we are going to explain what we would expect based on the knowledge gathered during this practice and the theory classes.



As we have seen in theory class and in the experiments carried out in past exercises, if we increase the size or the ways of our cache the misses and execution time of our programs will be reduced. But, as we can see in the above chart, there is a limit where no matter how much you increase your cache size or number of ways, you will always get a miss rate penalty. So in order to minimize resources cost and time we think that the sweet spot of the chart of 8 ways, 4KB caches, would be perfect for our needs.

The results would be proportional to the ones from exercise 3 but with lower execution times and cache misses. The write cache misses would be the same for both programs, as they perform the same write operations.