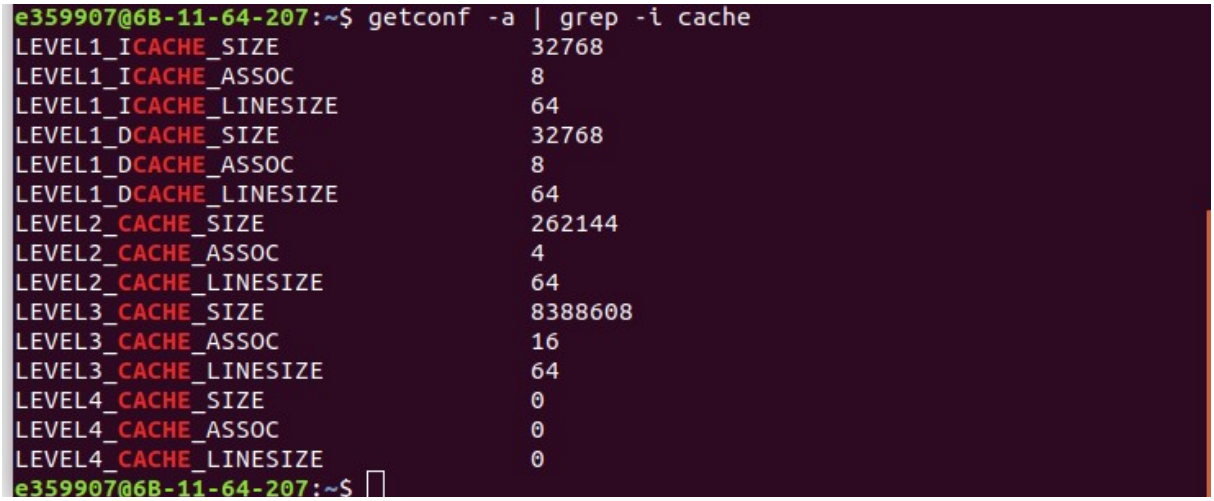
Práctica 3: Memoria caché y rendimiento

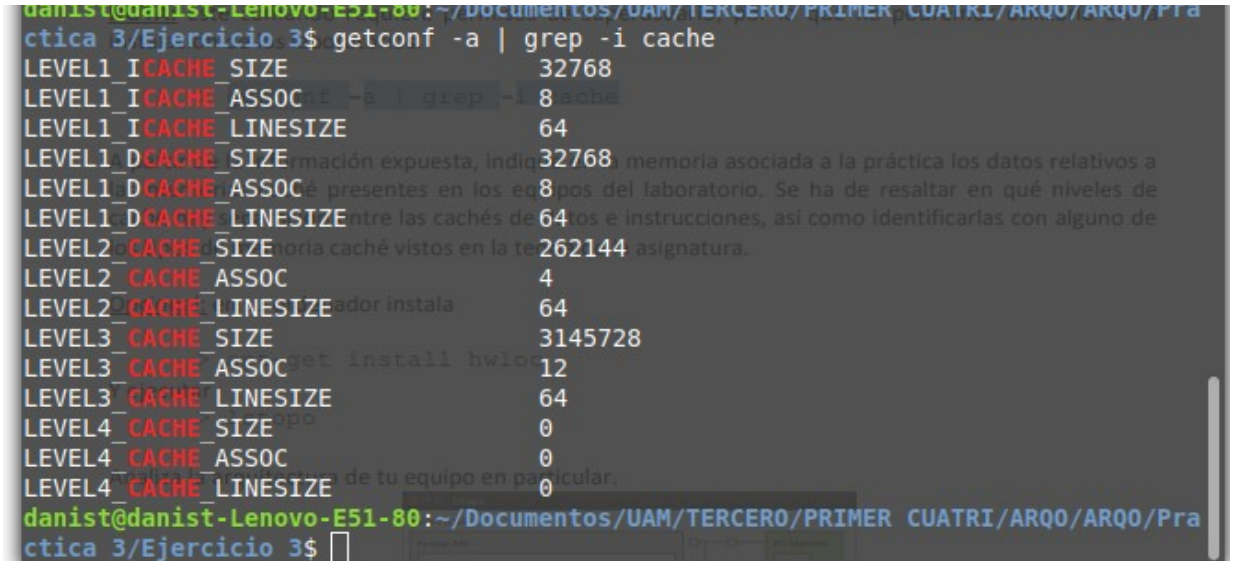
Ejercicio 0:

Ejecutamos en la terminal del ordenador del laboratorio el comando getconf -a | grep -i cache, y obtenemos el siguiente resultado en la terminal:



De esto se deduce que el equipo tiene dos cachés de nivel uno, una de datos y otra de instrucciones, y ambas son asociativas de 8 vías. Después hay una caché unificada de nivel dos, asociativa de 4 vías, y finalmente una caché unificada de nivel 3, asociativa de 16 vías.

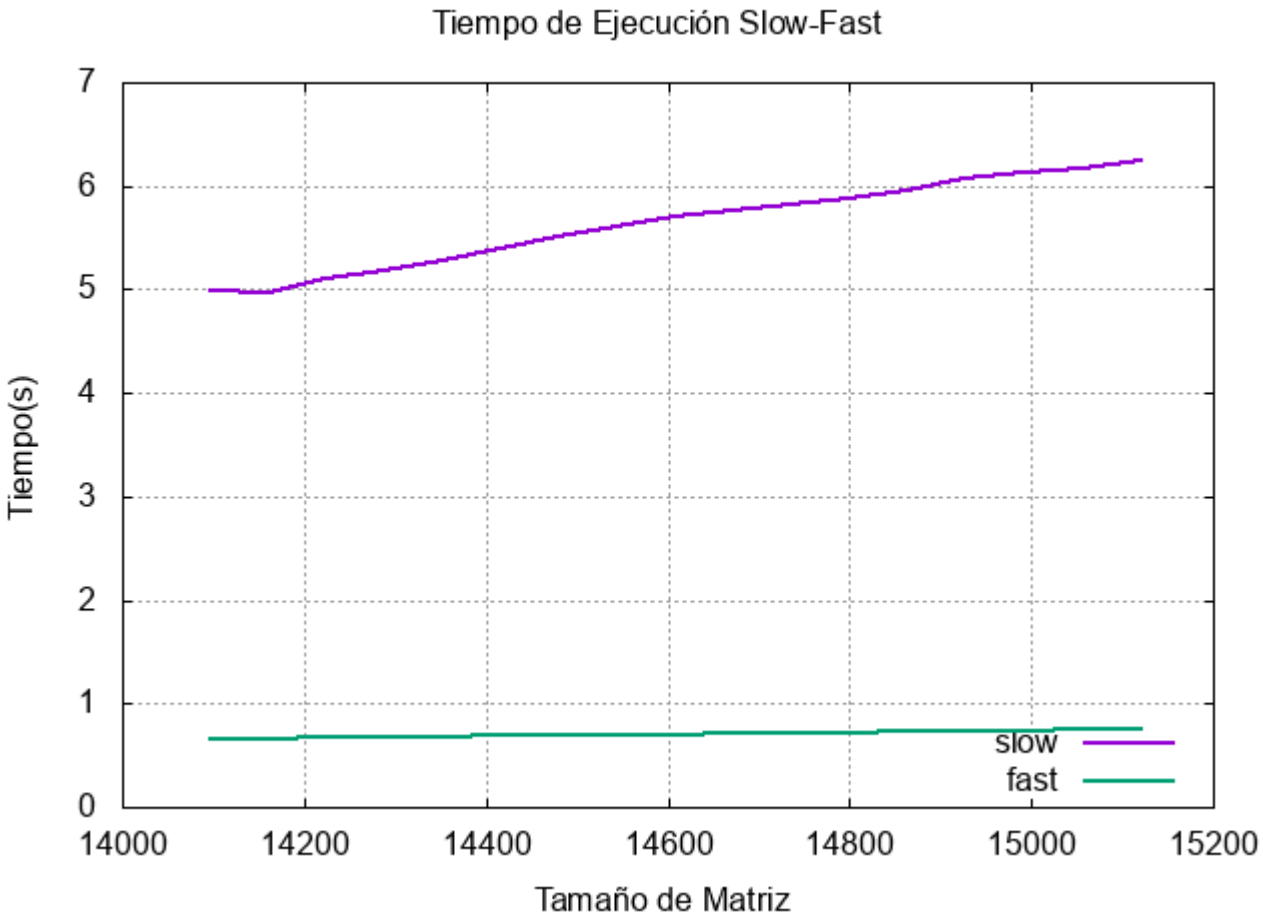
Las simulaciones de esta práctica se han realizado en un ordenador distinto a los equipos del laboratorio. Esta máquina presenta la siguiente configuración de la caché:



Este equipo tiene dos cachés de nivel uno, una de datos y otra de instrucciones, y ambas son asociativas de 8 vías. Después hay una caché unificada de nivel dos, asociativa de 4 vías, y finalmente una caché unificada de nivel 3, asociativa de 12 vías.

Ejercicio 1:

Para este ejercicio, hemos desarrollado un script de bash(slow_fast_time.sh). Este script necesita un parámetro de entrada que indica el número de veces a repetir el experimento. Dicho experimento consiste en ejecutar el programa “slow”, almacenando el tiempo de ejecución en un array, y seguidamente hacer lo mismo con el programa “fast”,pasándole como argumento a ambos programas un valor de N que varia entre 14096 y 15120, con saltos de tamaño 64 . Una vez repetido esto tantas veces como indique el parámetro de entrada, se realiza la media del tiempo transcurrido para cada N y se almacenan los resultados en un fichero llamado slow_fast_time.dat. Finalmente, graficamos los resultados obteniendo lo siguiente:



Se puede observar una diferencia sustancial entre el tiempo de ejecución de “fast” y el de “slow”, tardando el primero menos tiempo en ejecutarse (en media) que el segundo. A raíz del código y de esta gráfica ,se puede deducir como se guardan las matrices en memoria.“Slow” hace la suma de los elementos de una matriz yendo columna por columna, mientras que “fast” va por filas, y que uno

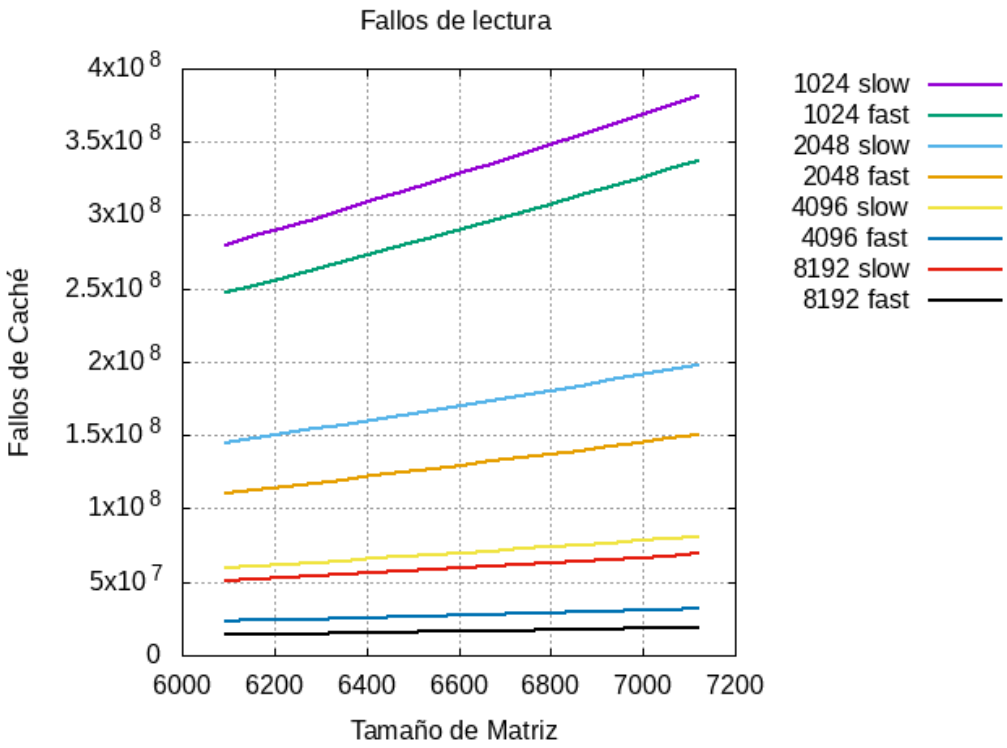
tarde menos que el otro solo se explica si las matrices se almacenan de la forma en la que “fast” lo lea más rápido, es decir, por filas.

Ejercicio 2:

El ejercicio 2 tiene como script el fichero ejercicio2.sh. En primer lugar, podemos observar la gráfica cache_lectura.png donde se indican los fallos de caché respecto al tamaño de la matriz en lectura, en ambos programas, “slow” y “fast”. Se puede apreciar que “slow” varía mucho dependiendo del tamaño de caché, teniendo más fallos con el tamaño 1024B y menos con 8192B, a menor tamaño de caché, las funciones se van separando más, es decir, la distancia entre “8192 slow” - “4096 slow” es mucho menor que la distancia entre “2048 slow” - “1024 slow”; a mayor caché, más distancia entre las funciones.

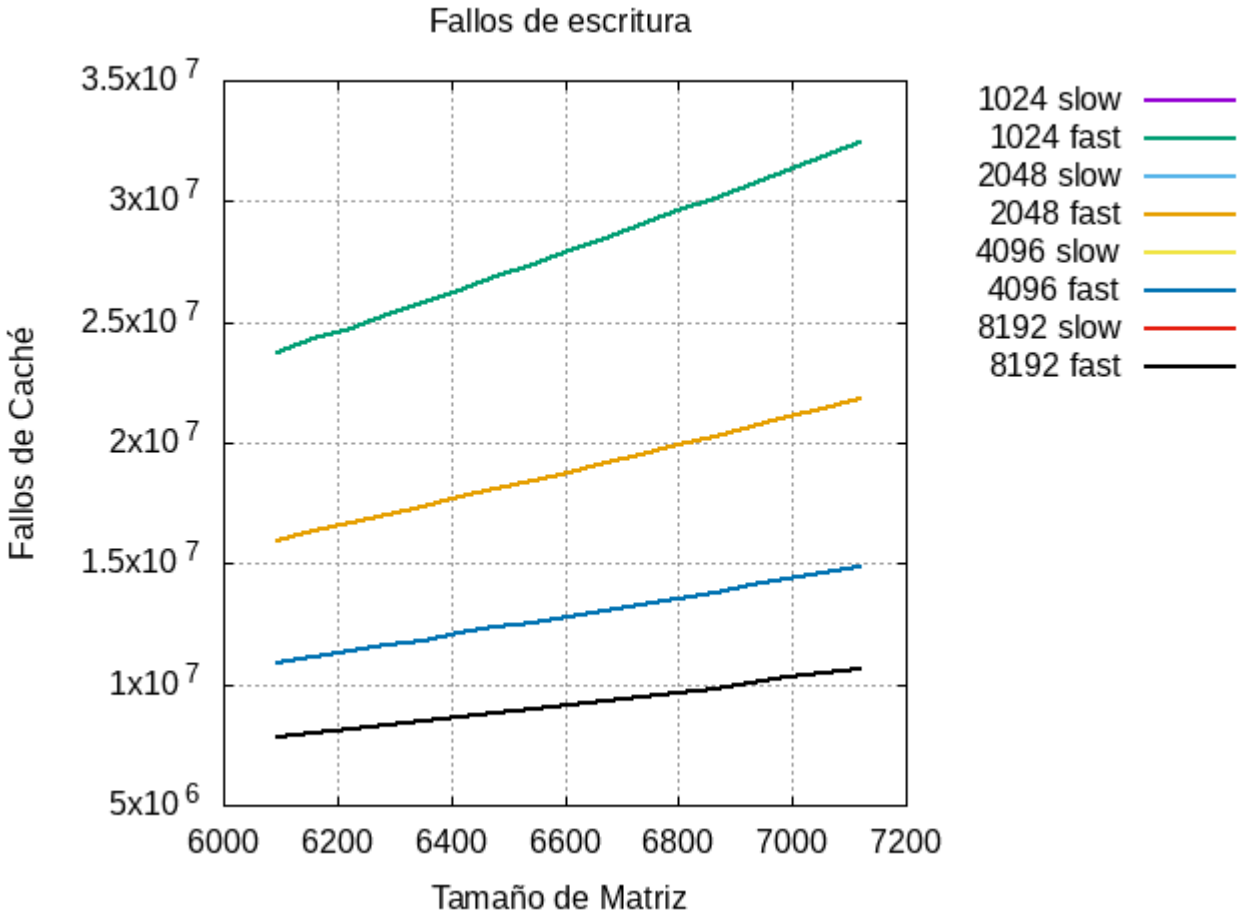
Por otro lado, también varía en cantidad para “fast”, la diferencia entre 1024B y 2048B es muy grande, sin embargo, al diferencia entre 1024B y 2048B muy pequeña. Cuanto más pequeña es la caché, más fallos de memoria hay pues no hay suficiente espacio para guardar tantos datos, y cuanto más grande sea, menos fallos habrá, siendo mejor “fast” que “slow”. En el código se puede apreciar que “slow” accede a las matrices por columnas, mientras que “fast” las va recorriendo por filas. Como la memoria se guarda por filas, “fast” será mucho mejor a la hora de leer.

Finalmente, se puede observar que la diferencia para cada tamaño entre “slow” y “fast” es prácticamente fija, apenas varía la distancia entre ellos. Entre “1024 slow” – “1024 fast” hay prácticamente la misma distancia que entre “2048 slow” – “2048 fast” y el resto. Asimismo, a mayor tamaño de caché aumenta el número de fallos pues hay que leer más datos.



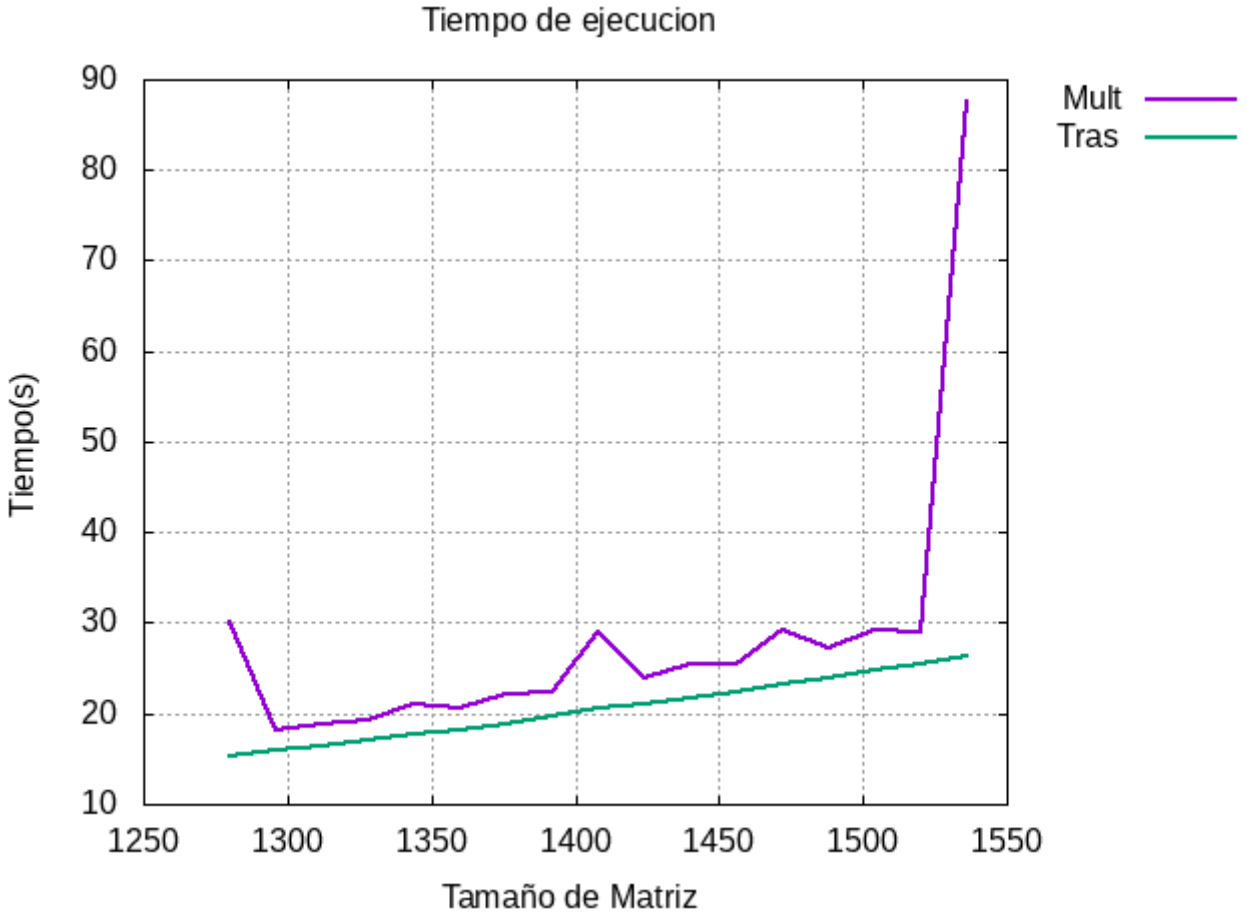
En segundo lugar, los fallos de escritura son los mismos tanto para “slow” como para “fast”. En la gráfica se puede apreciar mejor, para cada tamaño de caché coinciden los fallos y las funciones de “fast” se superponen respecto de las de “slow”. Esto se puede apreciar mejor en los .dat correspondientes, y se debe a que la escritura se realiza en una sola matriz y solo una vez, siendo siempre la misma escritura ya que solamente se escribe la suma de dos matrices, mientras que la lectura se hace de dos matrices distintas, teniendo que reemplazar los datos que lee a cada matriz, siendo fast más eficaz para leerlas.

Además, a mayor tamaño de caché, la diferencia entre las funciones es mayor, igual que a mayor tamaño de caché aumentan los fallos de caché pues hay que escribir más datos.

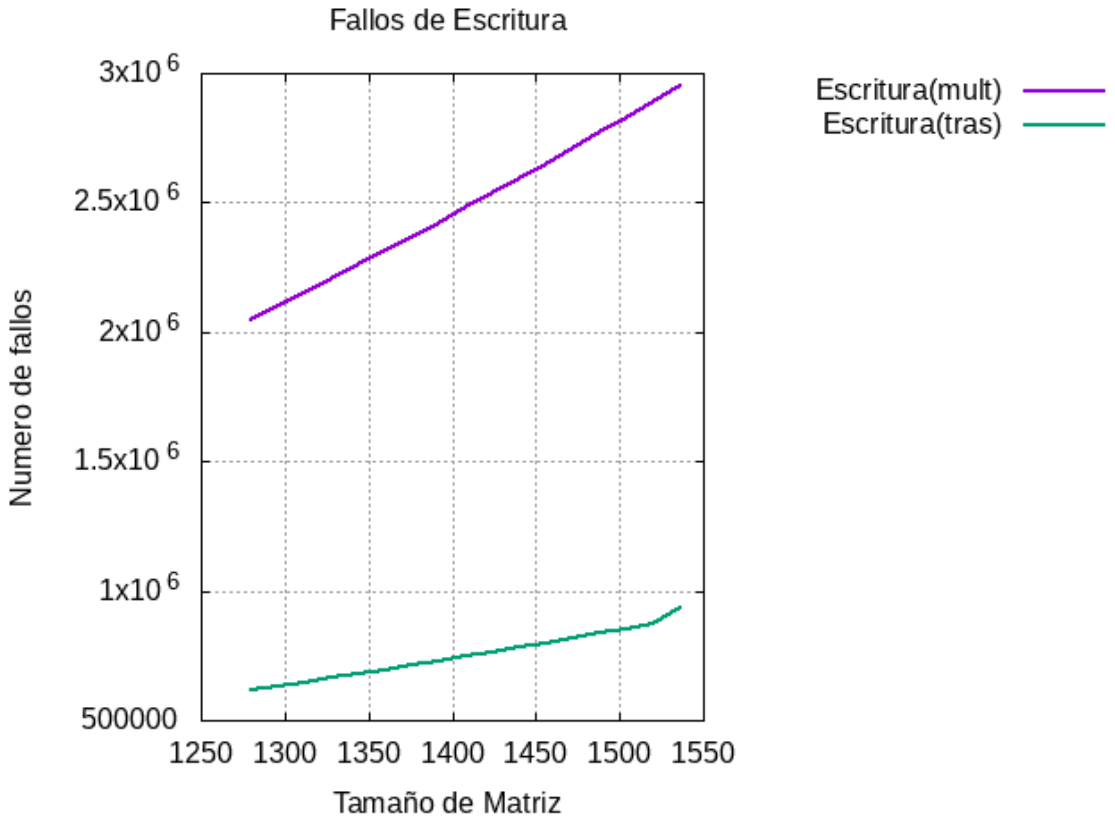


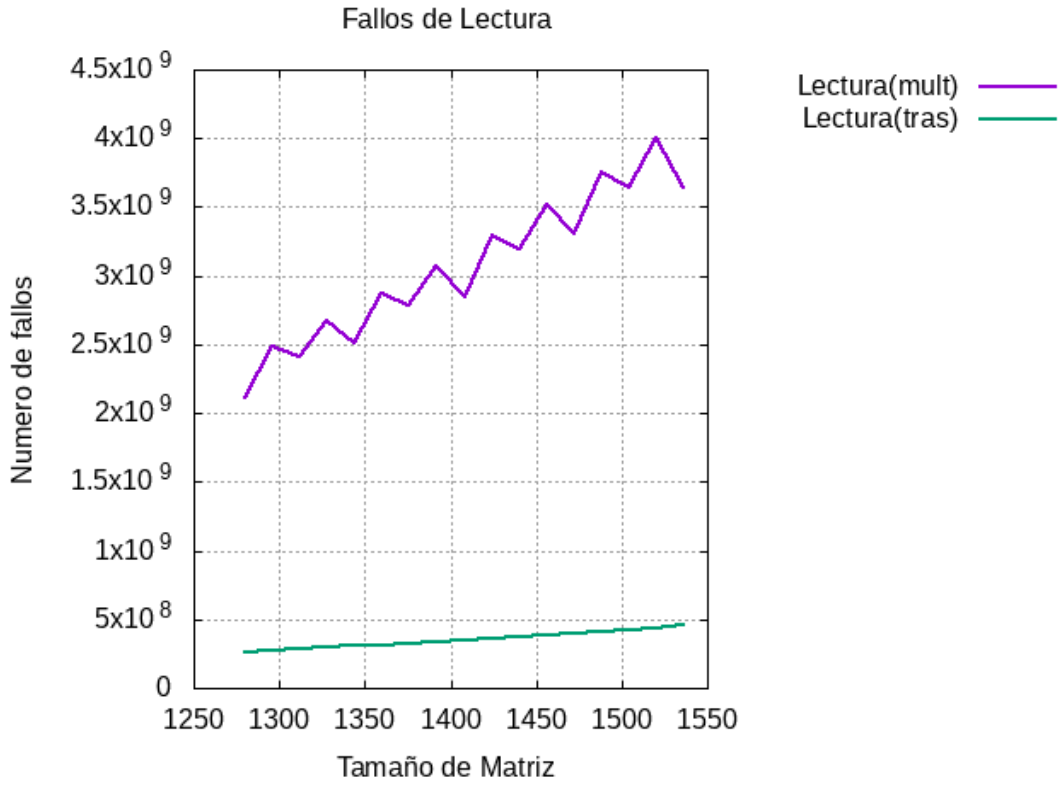
Ejercicio 3:

En este ejercicio, hemos creado dos programas , “ej3_multiplicación” uy “ej3_transpuesta”. El primero genera dos matrices y las multiplica(matriz generada al multiplicar filas por columnas) , mientras que el segundo genera tambien dos matrices ,pero transpone una de las dos, para a continuación multiplicarlas de tal manera que el resultado sea igual al de “ej3_multiplicación”(multiplicando esta vez filas por filas).
Usando un script (ejercicio3.sh), ejecutamos ambos programas para tamaños de N entre 1280 y 1536 , con saltos de tamaño 16, almacenando los tiempos de ejecución y los fallos de caché. Obtenemos los resultados reflejados en las siguientes gráficas:



De la gráfica de tiempos se observa que la multiplicación después de transponer tarda menos que la multiplicación normal. Eso se debe a que las matrices se almacenan en memoria por filas, y el programa “ej3_transpuesta” hace la multiplicación leyendo las matrices por filas, mientras que “ej3_multiplicación” lee una matriz por filas y otra por columnas, tardando así más en obtener los datos. La misma explicación se aplica al porque hay más fallos de cache en “ej3_multiplicación” que en “ej3_transpuesta”, como se observa en las siguientes gráficas


Fallos de Escritura
Numero de fallos
3x10 6
2.5x10 6
2x10 6
1.5x10 6
1x10 6
500000
1250 1300 1350 1400 1450 1500 1550
Tamaño de Matriz
Escritura(mult)
Escritura(tras)


Fallos de Lectura
Numero de fallos
4.5x10 9
4x10 9
3.5x10 9
3x10 9
2.5x10 9
2x10 9
1.5x10 9
1x10 9
5x10 8
0
1250 1300 1350 1400 1450 1500 1550
Tamaño de Matriz
Lectura(mult)
Lectura(tras)