Práctica3: Memoria caché y rendimiento

**Ejercicio 0. Información sobre la caché del sistema.**

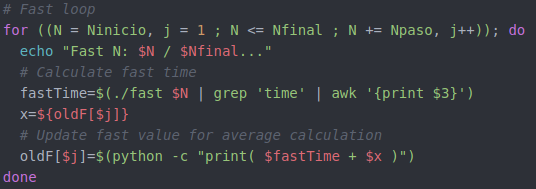


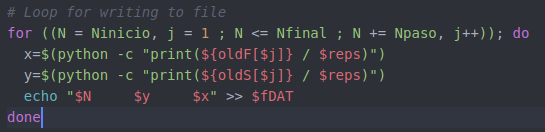
En esta primera imagen podemos ver que la cache de los equipos docente posee 3 niveles de cache (aunque aparezcan 4 se ve esa no existe, pues su tamaño y todo es 0). A su vez, el nivel 1 está dividido en 2, I y D. el atributo SIZE nos indica el tamaño de cada nivel, ASSOC nos indica que es asociativa, y LINESIZE nos muestra el tamaño de cada línea de la cache.

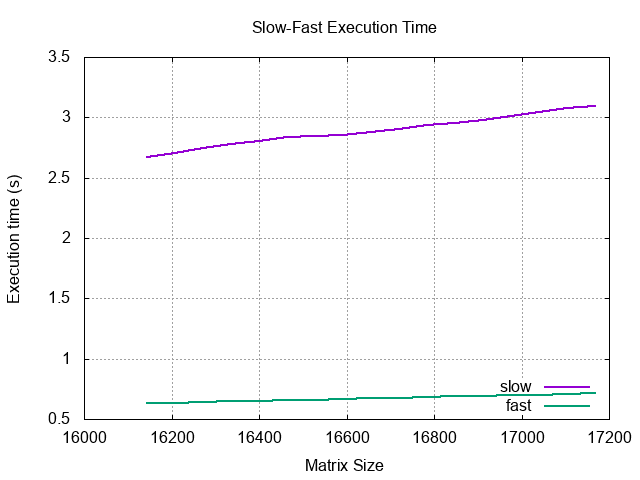
|  |  |
| --- | --- |
|  | Con este comando podemos observar también que hay 3 niveles de cache, pero de esta manera se ve que cada Core tiene cierta porción de cada nivel menos el nivel 3, que es común para todos los Cores. |

**Ejercicio 1: Memoria caché y rendimiento**

Para obtener los datos de este ejercicio, creamos un script en bash que primero reserva un array de 0s en memoria para los tiempos de fast y de slow, de tamaño 16 (para los 16 números entre 16144 y 17168 con pasos de 64). Luego creamos un bucle para iterar 20 veces sobre los bucles que calculan los tiempos fast y slow con la N correspondiente, y los suma al valor actual del array de tiempos en la posición correspondiente. Las sumas están hechas con una línea de código python. (Ejemplo abajo):

 Hacemos la media de cada uno de los N tiempos con otro comando python. (Ejemplo abajo):

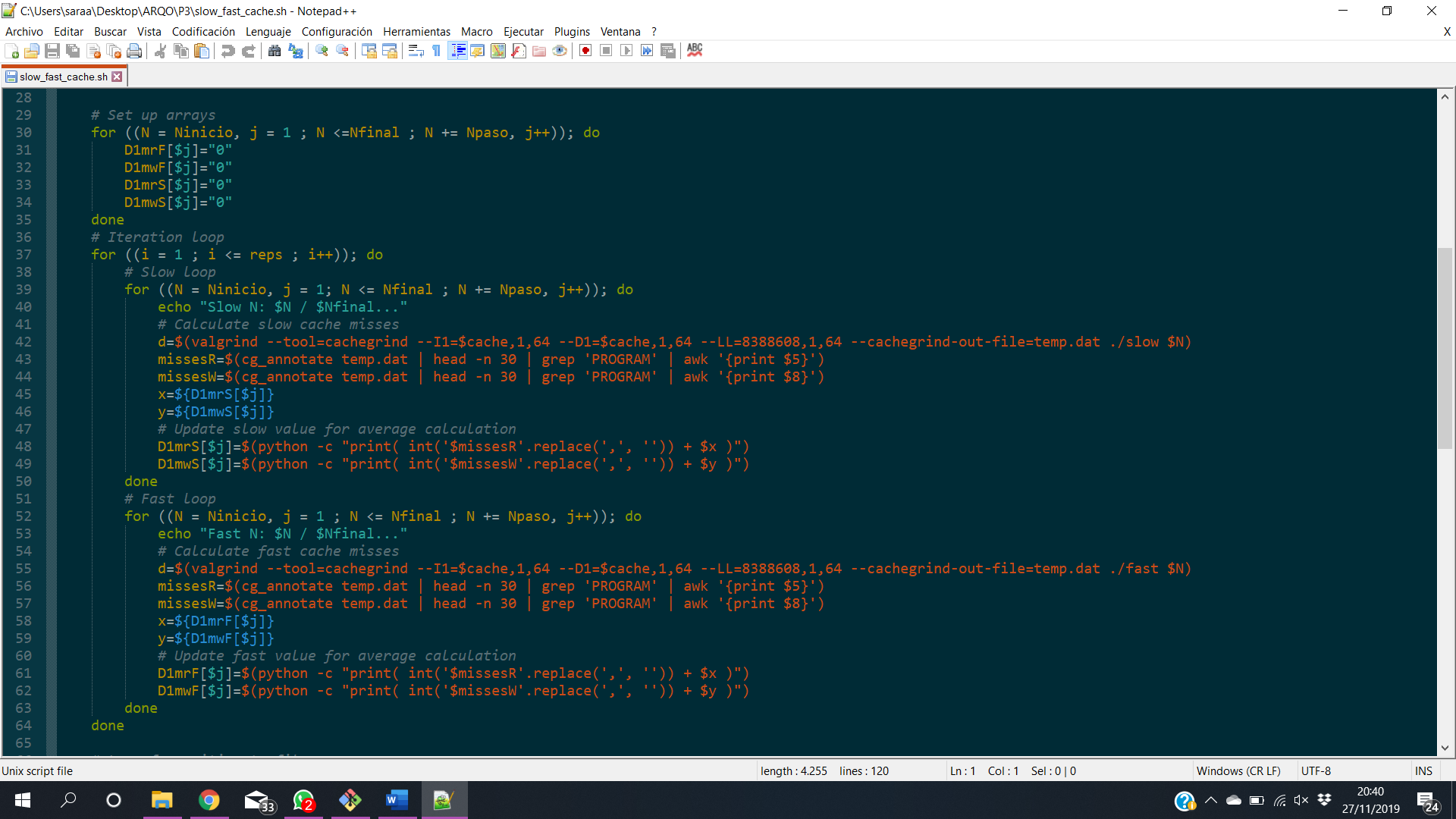
 Por último, mandamos los datos a GNUPlot para sacar las graficas de los tiempos (Mirar slow\_fast\_time.png)



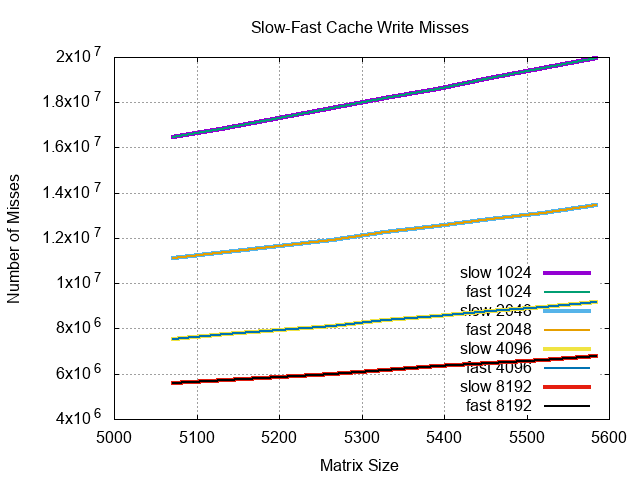
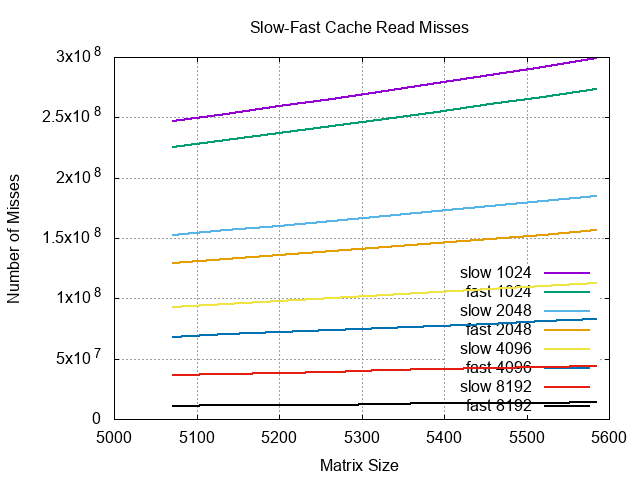
La grafica (slow\_fast\_time.png) tiene ese aspecto porque la manera de acceder a las matrices de slow es a través de columnas que es mucho mas lento que acceder a través de filas (como hace fast) ya que las caches lo que devuelven son filas, no columnas. Los datos que se guardan/extraen en las filas de las matrices se hace de manera secuencial, es decir, de uno en uno sin saltarse ninguna posición, mientras que para las columnas debe ir saltando los datos de las filas, de ahí el aumento de tiempo.

Para tamaños pequeños de matriz, la diferencia entre la búsqueda por filas o columnas es despreciable por ser pequeño.

**Ejericio2. Tamaño de la cache y rendimiento.**

Para obtener todos estos datos hemos realizado un script de bash en el que ejecutamos tanto slow como fast con la herramienta cachegrind de valgrind, y mediante comandos de python anotamos en un ficherolos datos sobre los errores en cache. Parte del codigo utilizado:

Una vez tenemos los archivos .dat donde hemos guardado toda la informacion, con GNUplot generamos las graficas (Mirar *cache\_escritura.png* y *cache\_lectura.png)*

****

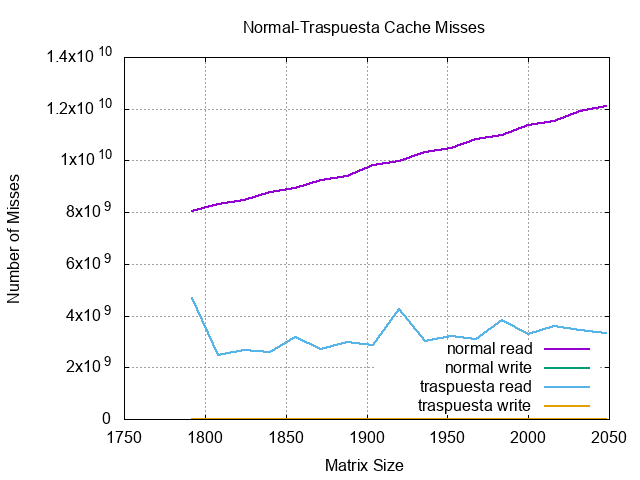
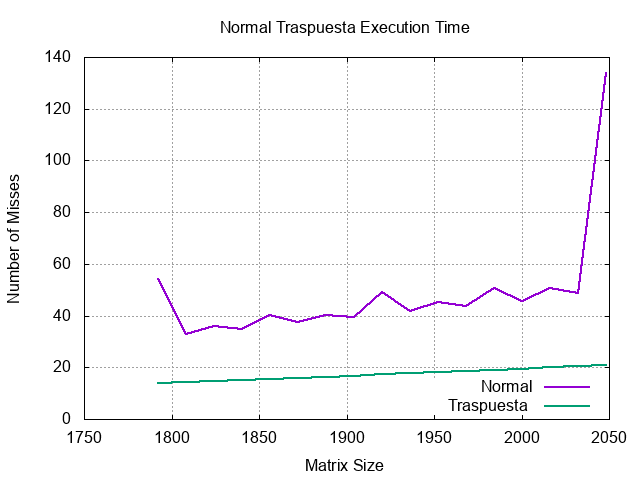
Tanto como para lectura como para escritura, cuanto mas pequeña es la cache más errores se producen. Al ser la cache más pequeña, los datos guardados son más limitados, de ahí que se produzcan muchos más errores al buscar porque la probabilidad de que ya estén en cache más pequeña que si la cache tuviese más capacidad.

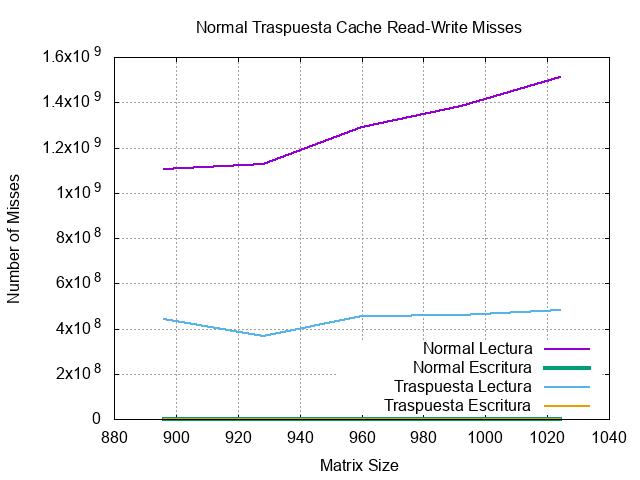
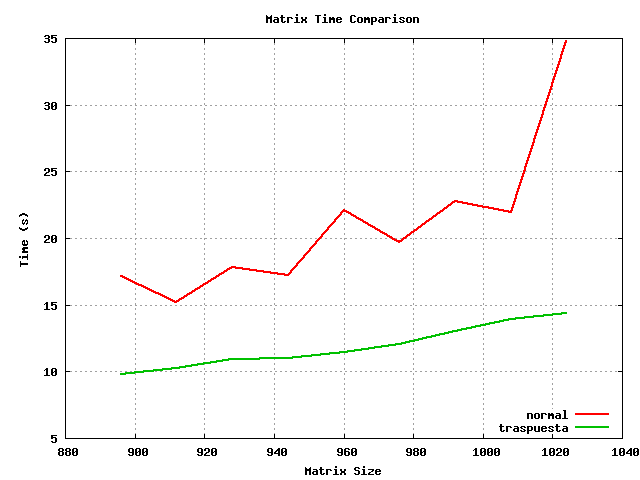
Para lectura, para un mismo tamaño de cache, se producen mas errores en slow que en fast, mientras que, para los errores en escritura, para un mismo tamaño de cache, slow y fast tienen los mismos errores.

**Ejercicio 3.** **Caché y multiplicación de matrices**

Para este apartado, primero generamos dos programas en C que se encargase de multiplicar dos matrices (para la primera parte del ejercicio) y otro que se encargase de generar la traspuesta de una matriz y luego multiplicar de la manera especificada en el enunciado (segunda parte del ejercicio). Una vez tenemos estos programas, al igual que para los ejercicios anteriores, utilizamos scripts escritos en bash que se encargasen de ejecutar estos programas repetidas veces con diferentes tamaños de matriz (pasados como argumento a los programas) para generar las medias, y también ejecutarlos con valgrind para obtener los errores en cache.

Para este ejercicio, hemos hecho dos versiones de cada grafica ya que el cluster tiene un tiempo límite de 1 hora de ejecución. Así, lo que hemos ejecutado en el cluster, el tamaño de las matrices es la mitad y el salto entre tamaños es el doble de lo esperado. Mientras que en los equipos de los laboratorios ejecutamos la versión con tamaños esperados. (Mirar los archivos *mult\_time.png, mult\_cache.png, mult\_time\_cluster.png, mult\_cache\_cluster.png*. Los que tienen “\_cluster” en el nombre son los que hemos dicho que hemos ejecutado en el cluster con tamaños reducidos).

Graficas no ejecutadas en el cluster, con el tamaño completo:

Graficas de lo ejecutado en el cluster, con tamaños menores:

A pesar de las diferencias de tamaño, se puede ver que siguen el mismo comportamiento (así salimos de dudas).

Para las graficas de tiempo se puede ver que el tiempo en generar una matriz traspuesta y multiplicar sigue una escala lineal, que a mayor tamaño mayor tiempo, mientras que, para realizar una multiplicación normal, el tiempo es exponencial. Esto se explica con lo mencionado anteriormente, para la multiplicación normal, una de las matrices hay que recorrerla por columnas, mientras que para el segundo tipo ambas matrices se recorren por filas, siendo esta manera mucho más rápida. Así, aunque gastemos tiempo en generar una matriz traspuesta, es conveniente pues el tiempo usado en recorrer las matrices para multiplicar es mucho mejor.

En cuanto a los fallos de cache, los fallos de escritura para ambos son muy reducidos y siguen un modelo lineal. Para los fallos de lectura ambos siguen un modelo lineal, solo que los de la traspuesta tiene una pendiente mucho mas reducida (casi horizontal) lo cual quiere decir que se mantiene, aunque se aumente el tamaño de la matriz, mientras que para la multiplicación normal podemos una mayor pendiente, siendo mayor los errores cuanto mayor es el tamaño de la matriz.

En cuanto a la entrega, tenemos 3 carpetas correspondientes a cada ejercicio.

* En la carpeta del ejercicio 1 (EX1):
  + El script *slow\_fast\_time.sh* utilizado para calcular los tiempos de ejecución. Para este script son necesarios los programas escritos en C (slow, fast y arqo3) que nos proporcionaban.
  + El fichero *slow\_fast\_time.dat* con los datos obtenidos de tiempos de cada programa (slow y fast)
  + La imagen *slow\_fast\_time.png* de las graficas generada por el script
* En la carpeta del ejercicio 2 (EX2):
  + El script *slow\_fast\_cache.sh* utilizado para calcular los errores en cache. Para este script son necesarios los programas escritos en C (slow, fast y arqo3) que nos proporcionaban.
  + Los ficheros *.dat* con los datos obtenidos errores de cache, donde el numero que hay en el nombre del fichero indica el tamaño de la cache para esos errores.
  + Las imágenes *cache\_escritura.png* y *cache\_lectura.png* de las gráficas generada por el script mostrando los errores de escritura y lectura, respectivamente, producidos para cada tamaño de cache por cada programa
* En la carpeta del ejercicio 3(EX3)
  + Los programas escritos en C necesarios mara la multiplicación de matrices. Multiplica\_matriz.c (junto a multiplica\_matriz.h) contiene todas las funciones base utilizadas por los
  + El script
  + Los ficheros *.dat*
  + Las imágenes