Práctica3: Memoria caché y rendimiento

**Ejercicio 0: información sobre la caché del sistema.**

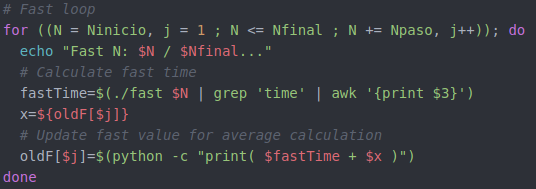


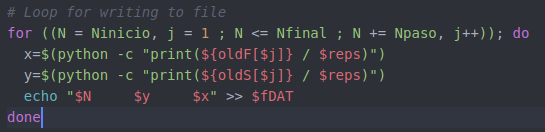
En esta primera imagen podemos ver que la cache de los equipos docente posee 3 niveles de cache (aunque aparezcan 4 se ve esa no existe, pues su tamaño y todo es 0). A su vez, el nivel 1 esta dividido en 2, I y D. el atributo SIZE nos indica el tamaño de cada nivel, ASSOC nos indica que es asociativa,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

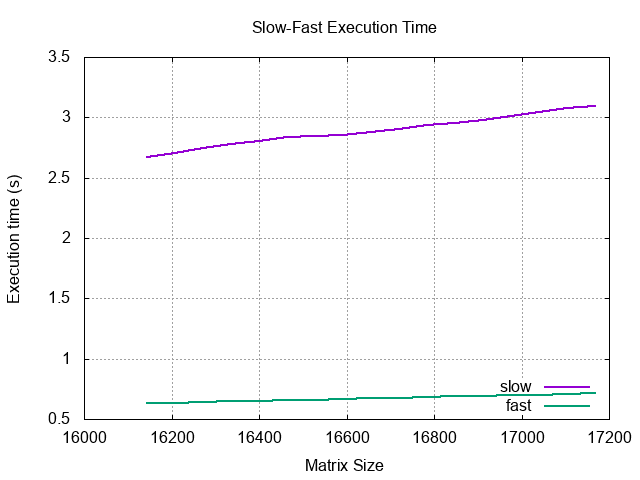
|  |  |
| --- | --- |
|  | Con este comando podemos observar también que hay 3 niveles de cache, pero de esta manera se ve que cada Core tiene cierta porción de cada nivel menos el nivel 3, que es común para todos los Cores. |

**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)

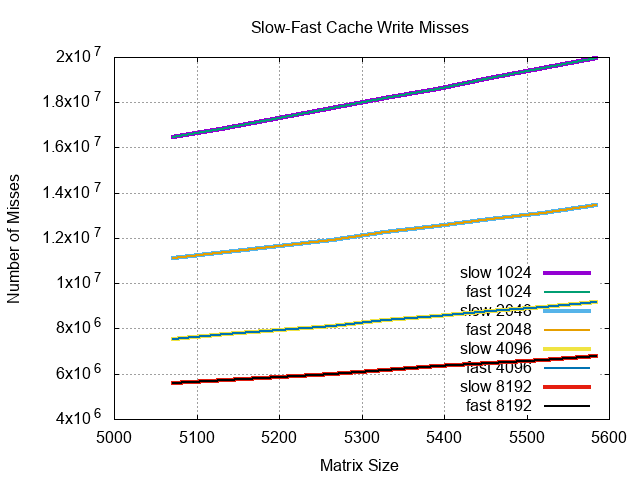
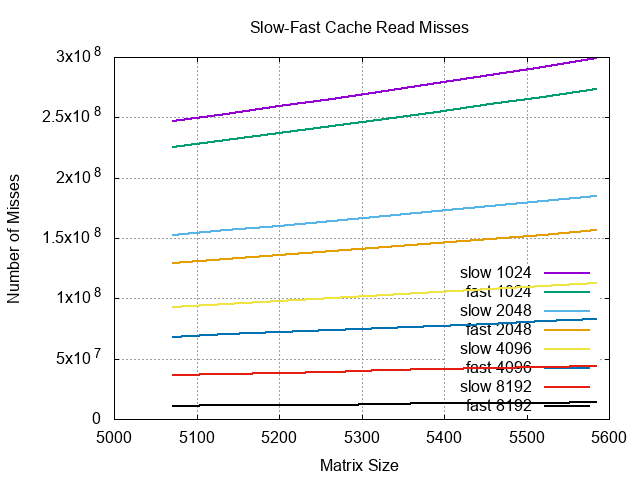


La grafica (slow\_fast\_time.png) tiene ese aspecto porque la manera de acceder a las matrices de slow es a través de columnas que es mucho mas lento que acceder a través de filas (como hace fast) ya que las caches lo que devuelven son filas, no columnas. Los datos que se guardan/extraen en las filas 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.**

Mirar *cache\_escritura.png* y *cache\_lectura.png*

****

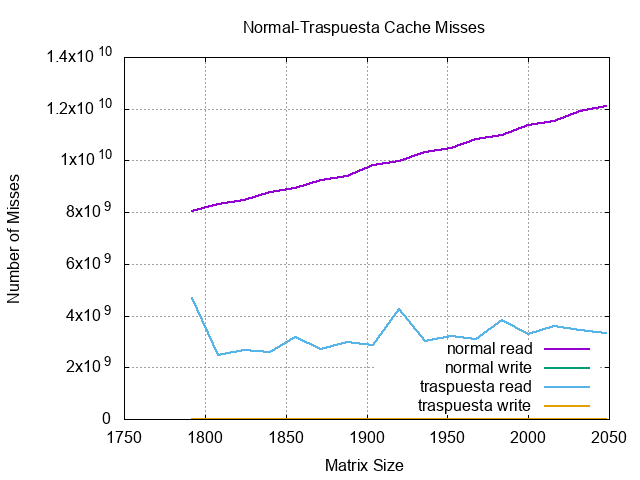
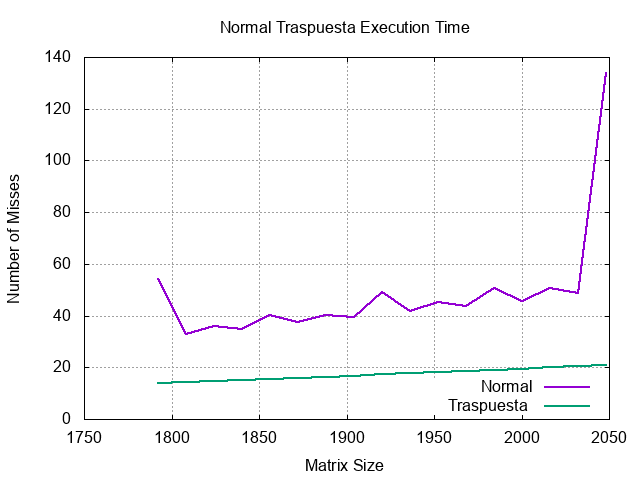
Para los errores de lectura, cuanto mas pequeña es la cache mas errores se producen tanto para slow como para fast, siempre tardando mas slow por lo comentado en el anterior ejercicio.

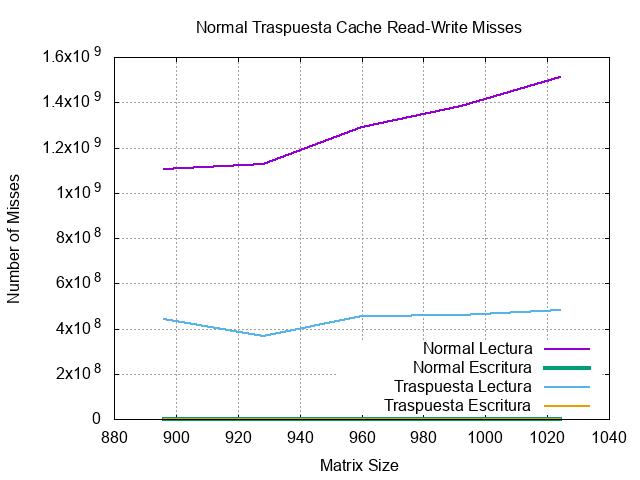
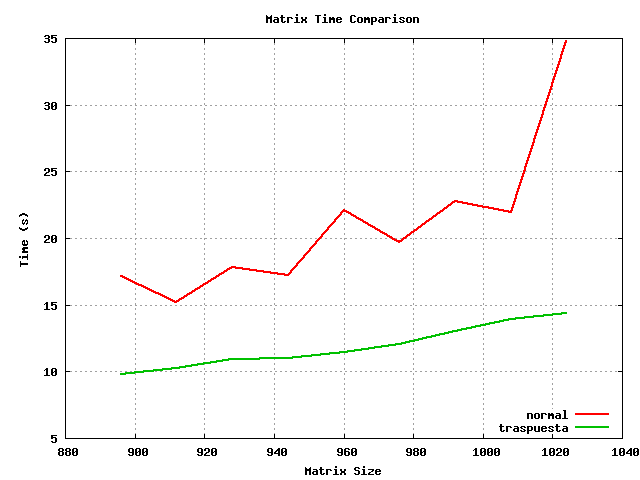
Para los errores en escritura, para un mismo tamaño de cache, slow y fast tienen los mismos errores. Y, al igual que en lectura, para menos tamaño de cache, más errores se producen.

Al ser la cache mas 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 y encontrarlos es mas pequeña que si la cache tuviese más capacidad.

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

Para este ejercicio, hemos hecho dos versiones de cada grafica ya que el cluster tiene un tiempo limite de 1 hora, lo ejecutado en el cluster el tamaño de las matrices es la mitad y el salto entre tamaños es el doble de lo esperado. 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:

Así, hemos obtenido las siguientes graficas