INGENIERÍA INFORMÁTICA
Escuela Politécnica Superior
Universidad Autónoma De Madrid

Memoria, Cache y rendimiento

Práctica 3 de ARQO

David Teófilo Garitagoitia Romero
Daniel Cerrato Sánchez

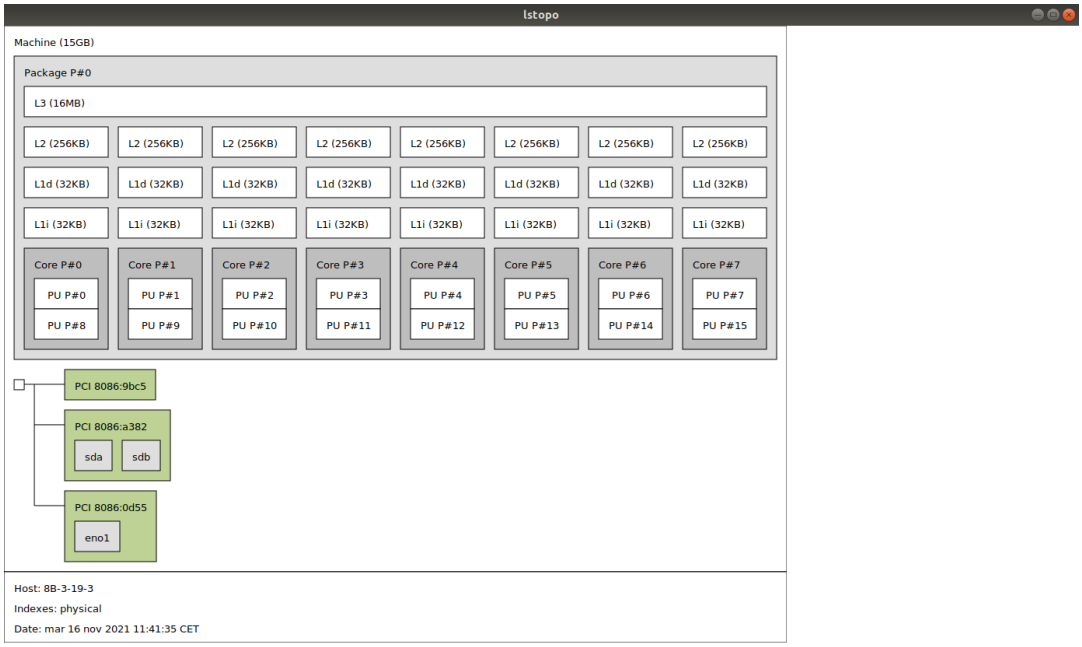
Pareja 3 Grupo 1322
11/18/2021

Índice de Contenidos

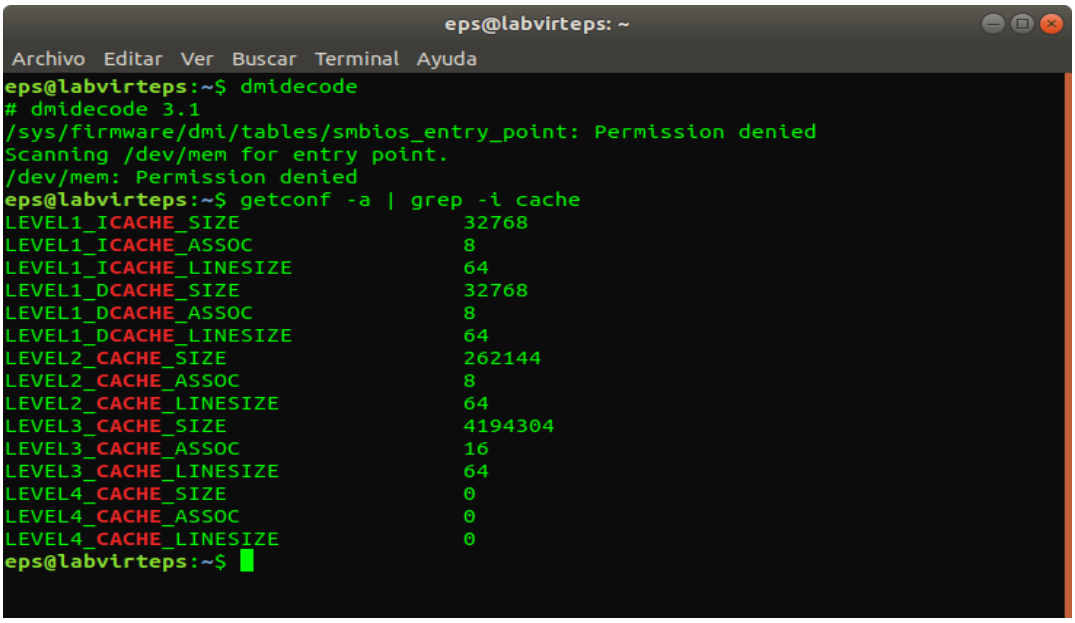
1. Ejercicio 0: Información sobre la caché del sistema

Se pide indicar en la memoria asociada a la práctica los datos relativos a las memorias caché presentes en los equipos del laboratorio. Se ha de resaltar en qué niveles de caché hay separación entre las cachés de datos e instrucciones, así como identificarlas con alguno de los tipos de memoria caché vistos en la teoría de la asignatura.

Al ejecutar el comando lstopo se puede observar la información mostrada en la siguiente imagen:



Adicionalmente empleando el comando > getconf -a | grep -i cache se muestra por pantalla la siguiente información



De esta información se deduce que tenemos:

3 niveles de cache y 8 cores con Hyper-Threading (el nivel 4 aparece como 0 por terminal y el lstopo no lo muestra por tanto no existe dicho nivel)

Los niveles 1 y 2 son propios de cada core y el nivel 3 restante es compartido.

A medida que aumentamos de nivel el tamaño de la cache aumenta, los primeros niveles son más pequeños para ser más rápidos mientras que los últimos son más grandes para abarcar más información.

1. El primer nivel está compuesto por:
   Cache de Datos y de instrucciones de 32KB asociativa con 8 vías y un tamaño de bloque de 64B.

2. El segundo nivel está compuesto por:
   Cache de 256KB asociativa con 4 vías y un tamaño de bloque de 64B.

3. El tercer y ultimo nivel está compuesto por:
   Cache de 16MB con 16 vías y 64B de bloque.

2. Ejercicio 1: Memoria Caché y rendimiento

Para este ejercicio se nos pide comprobar de manera empírica como el patrón de acceso a los datos puede mejorar el aprovechamiento de las memorias caché del sistema. Para ello, se utilizarán los dos programas de prueba que se aportan con el material de la práctica (slow.c y fast.c).

Se nos pide tomar datos de tiempo de ejecución de los dos programas de ejemplo que se proveen (slow y fast) para matrices de tamaño NxN, con N variando entre 1024 y 16384 con un incremento en saltos de 1024 unidades. Deberá tomar resultados para todas las ejecuciones múltiples veces (se recomiendan al menos 10) y de forma intercalada, como se ha indicado anteriormente, y calcular la media de estas.

Para esta tarea hemos creado el script ej1.sh encargado de ejecutar tanto slow como fast con N entre los valores solicitados 15 veces para obteniendo la media de estos escribir los resultados en el fichero time_slow_fast.dat además de emplear gnuplot para generar time_slow_fast.dat con el gráfico.

Es necesario la ejecución varias veces de cada programa para cada tamaño de matriz para medir el tiempo de acceso a cache evitando que el sistema operativo utilice el cache de una ejecución en la siguiente reduciendo de esta forma el tiempo de ejecución.

Para obtener los datos se hace un doble bucle para ejecutar los programas para los diversos tamaños de matriz, se guardan en arrays las sumas de los tiempos y finalmente se hace la media simplemente dividiendo las sumas de las ejecuciones entre los bucles.

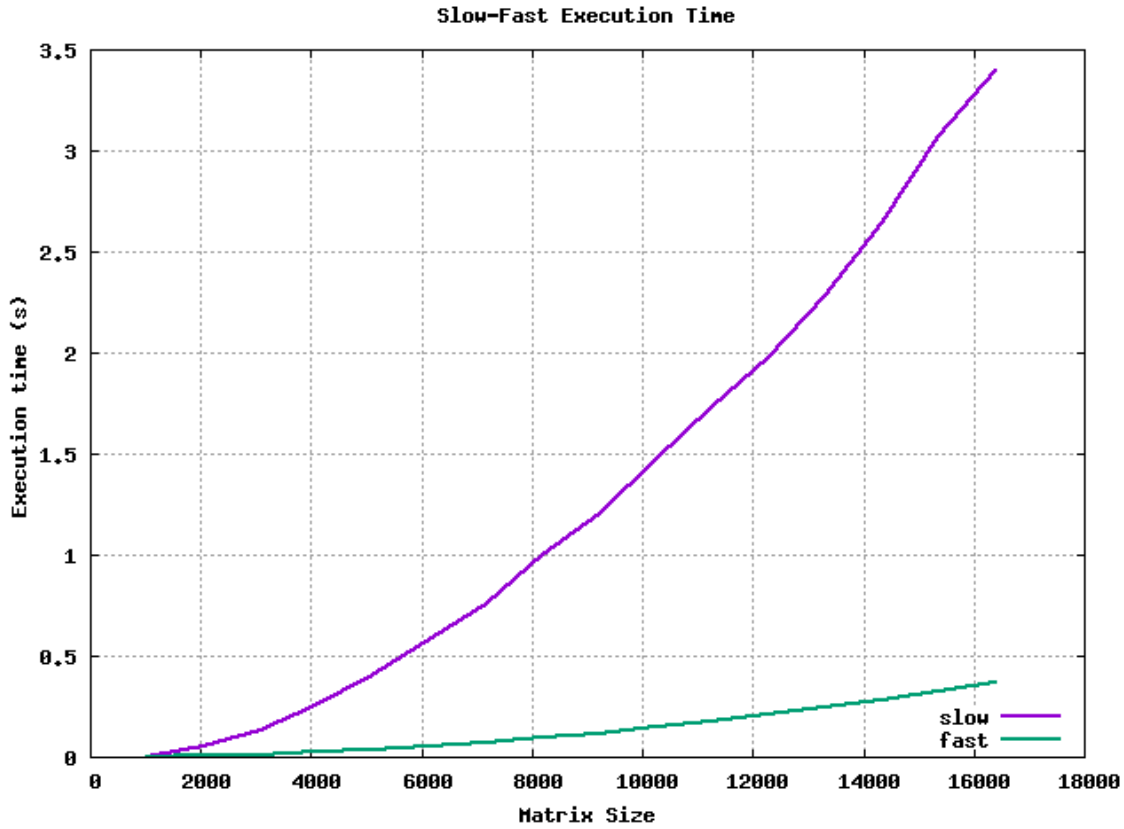
En un principio, cuando el tamaño de la matriz es menor, los accesos a memoria son reducidas por lo que las diferencias de tiempo no son muy acusadas, sin embargo, a medida que aumenta su tamaño las diferencias comienzan a ser considerables debido a que también aumentan los accesos a memoria.

La matriz se guarda por filas como se puede ver en arg.c

```
tipo ** generateMatrix(int size)
{
        tipo *array=NULL;
        tipo **matrix=NULL;
        int i=0,j=0;

        matrix=(tipo **)malloc(sizeof(tipo *)*size);
        array=(tipo *)malloc(sizeof(tipo)*size*size);
        if( !array || !matrix)
        {
                printf("Error when allocating matrix of size %d.\n",size);
                if( array )
                        free(array);
                if( matrix )
                        free(matrix);
                return NULL;
```

```
        }

        srand(0);
        for(i=0;i<size;i++)
        {
                matrix[i] = &array[i*size];
                for(j=0;j<size;j++)
                {
                        matrix[i][j] = (1.0*rand()) / (RAND_MAX/10);
                }
        }

        return matrix;
}
```

El resultado de la ejecución es la siguiente gráfica:



3. Ejercicio 2: Tamaño de la cache y rendimiento

Para este ejercicio se nos pide la medición de cantidad de fallos producidos en lectura y escritura para unos tamaños de matriz y de caché, para ello creamos el script cacheMisses encargado de generar los cache_[tam].dat siendo [tam] el tamaño correspondiente de la caché (1024, 2048, 4096 y 8192 Bytes) y el tamaño de la matriz variando entre 1024+1024*P y 1024+1024*(P+1), (al ser la pareja 3, P=3), incrementando el tamaño de la matriz en saltos de 256 unidades.

Para ello el script genera archivos auxiliares llamando a valgrind con los comandos siguientes, donde Cache es una variable que va tomando los distintos tamaños de cache de primer nivel y N los tamaños de matriz:

valgrind --tool=cachegrind --cachegrind-out-file=$auxSlow --I1=$Cache,1,64 --D1=$Cache,1,64 --LL=$cacheSuperior,1,64 ./slow $N

valgrind --tool=cachegrind --cachegrind-out-file=$auxFast --I1=$Cache,1,64 --D1=$Cache,1,64 --LL=$cacheSuperior,1,64 ./fast $N

Ahora para obtener los fallos que deseemos bastará con recoger dicha información de la salida de cachegrind en los ficheros auxiliares para ello podriamos obtener la información relativa al uso de caché guardado en el fichero y después emplear un grep pero utilizaremos head y tail para seleccionar la línea en la que se encuentra la información que queremos y simplemente usar print $x siendo x la posición del numero de fallos del tipo deseado.

Por último, con todos los datos obtenidos generaremos las gráficas.

Pata facilitar la comprensión de estas se ha optado por dividir las dos gráficas cache_escritura.png y cache_lectura.png en dos subgráficas para así mostrar en unas las relativas al programa fast y en otras las de slow.

La tendencia es al alza en ambas gráficas ya que al aumentar el tamaño de la matriz aumentan los accesos a memoria por lo que los fallos también aumentan.

En escritura los resultados son idénticos para las ejecuciones de ambos programas ya que ambos acceden para escribir el resultado en la matriz en el mismo orden, solo cambian los accesos para lectura como se observa claramente en las gráficas.

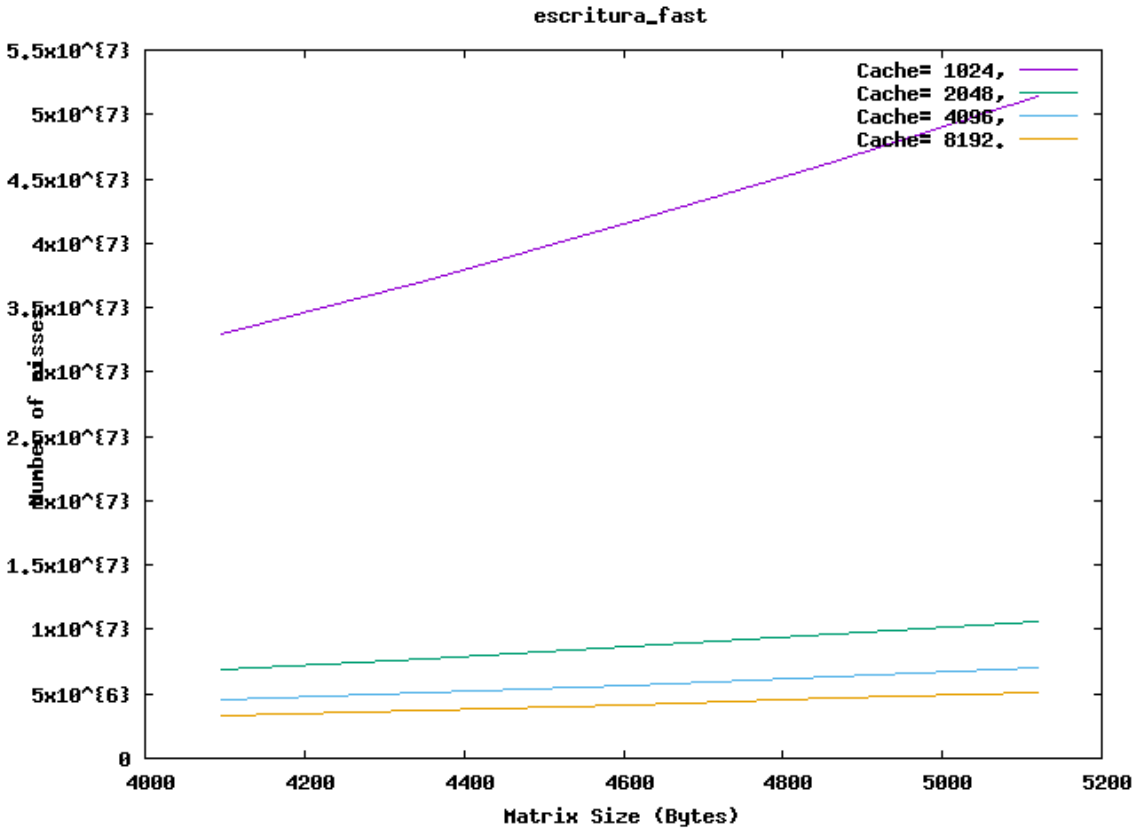
En lectura el programa fast tiene resultados muy superiores a los del programa slow con un valor aproximado de 7.5*10^7 fallos para el tamaño de matriz de 4096B y de cache de 1024B, mientras que para los mismo datos, el programa slow tiene 9.3*10^7 fallos lo que son aproximadamente 1.8*10^7 fallos de diferencia, para esos mismos tamaños de cache, cuando la matriz aumenta hasta los 5120 (que es el máximo valor para P=3) el programa slow marca 1.45*10^8 fallos mientras que el programa fast no llega a los 1.2*10^8 fallos lo que es una diferencia de 2.7*10^7 fallos.

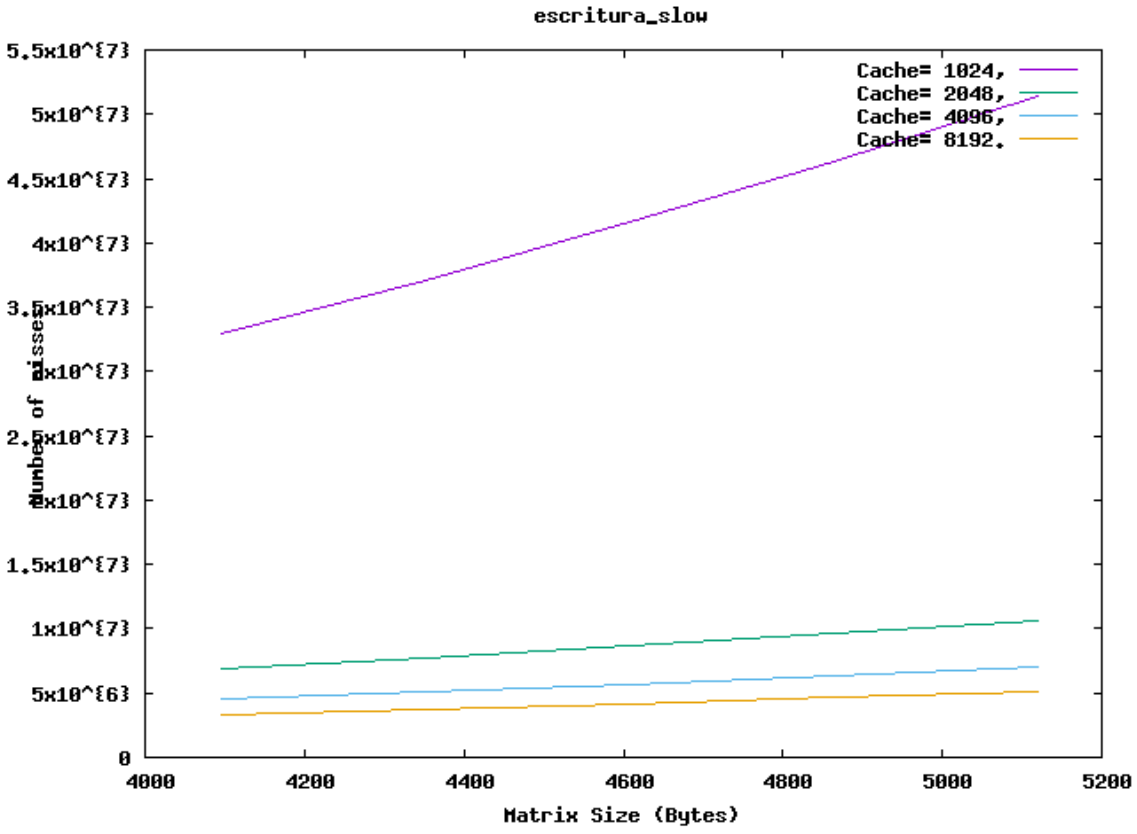
Esta diferencia entre los resultados de slow y fast se debe a que slow realiza las sumas empleando las columnas; lo que es menos eficiente teniendo en cuenta que en C, las matrices son guardadas como arrays continuos.

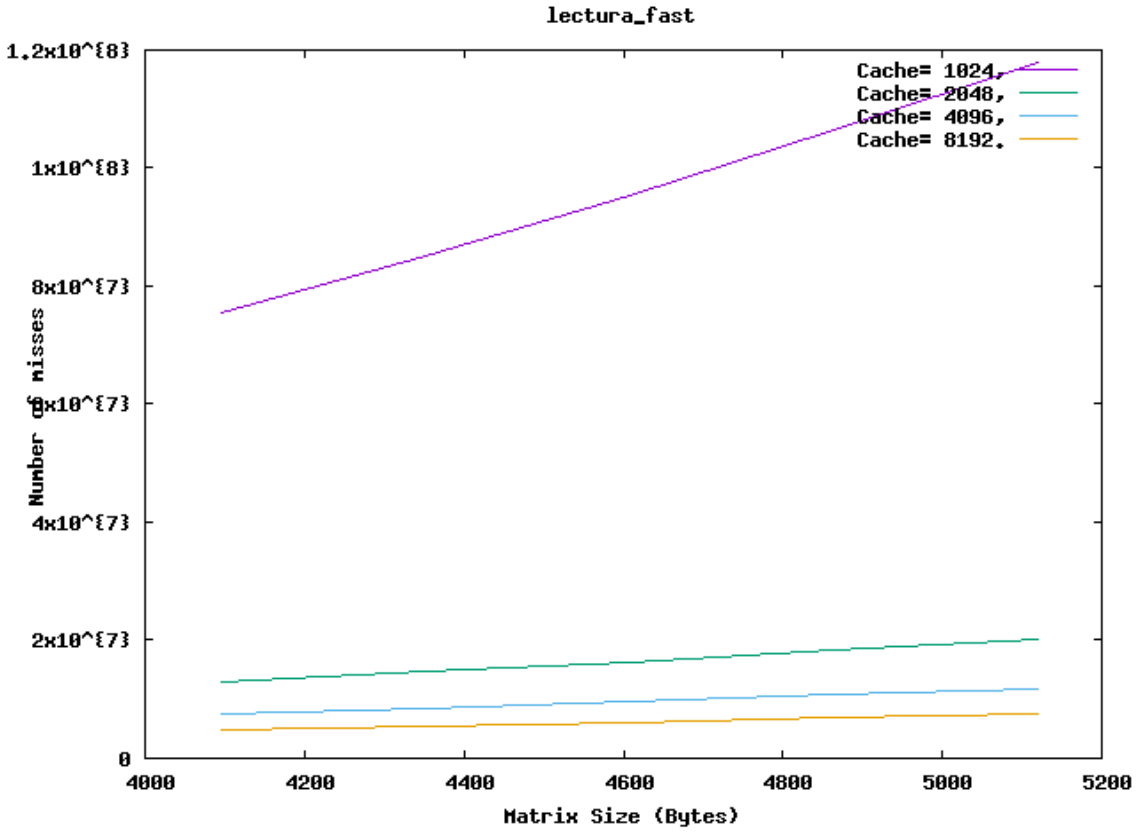
Para acceder a las columnas debe acceder a una única posición del array para cada array, lo que supone tener que cargar el bloque del array para únicamente acceder a un valor.

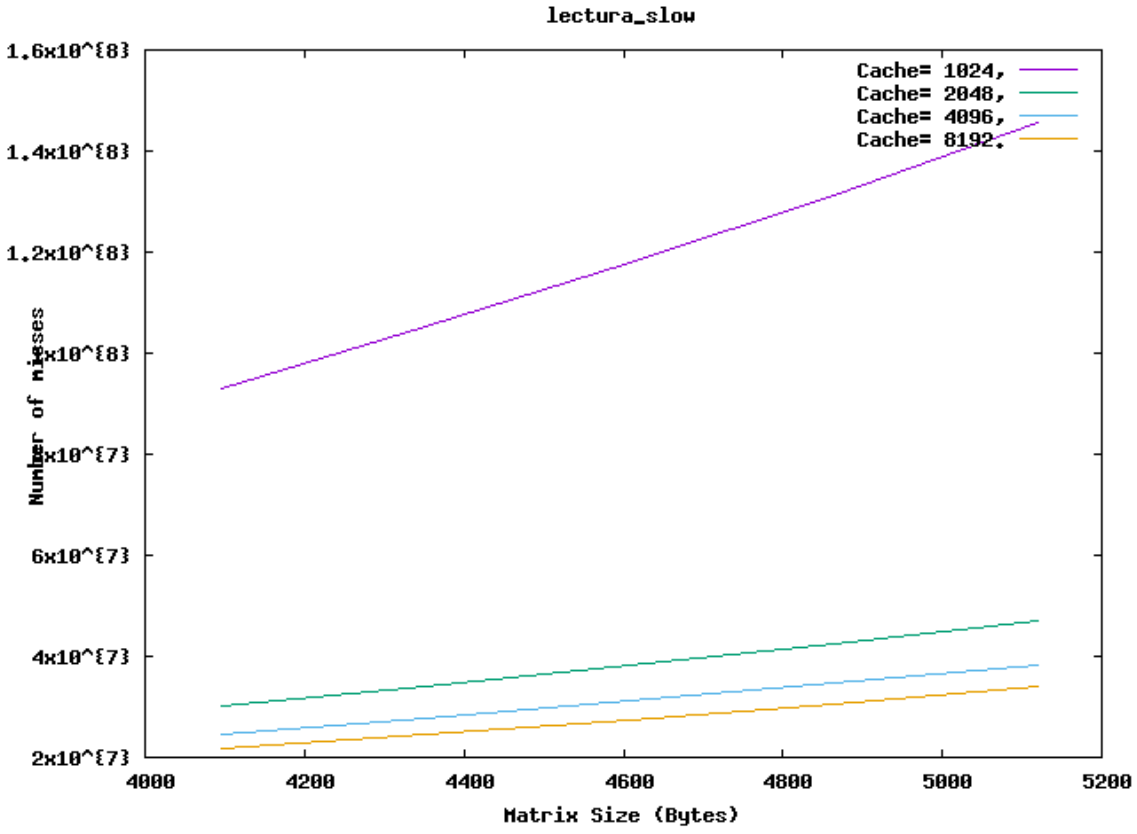
En fast realiza las sumas por filas, por ende, los fallos en cache son producidos porque la fila no está en memoria, o porque la fila entera está en varios bloques, y a diferencia de slow, fast aprovecha la carga del bloque de la fila operando sobre ella y no sobre un único elemento de esta.

Las gráficas de los resultados se muestran en las siguientes páginas.


escritura_fast
Cache= 1024,
Cache= 2048,
Cache= 4096,
Cache= 8192.
Number of misses
Matrix Size (Bytes)
5.5x10^{7}
5x10^{7}
4.5x10^{7}
4x10^{7}
3.5x10^{7}
3x10^{7}
2.5x10^{7}
2x10^{7}
1.5x10^{7}
1x10^{7}
5x10^{6}
0
4000
4200
4400
4600
4800
5000
5200


escritura_slow
Cache= 1024,
Cache= 2048,
Cache= 4096,
Cache= 8192.
Number of misses
Matrix Size (Bytes)
5.5x10^{7}
5x10^{7}
4.5x10^{7}
4x10^{7}
3.5x10^{7}
3x10^{7}
2.5x10^{7}
2x10^{7}
1.5x10^{7}
1x10^{7}
5x10^{6}
0
4000
4200
4400
4600
4800
5000
5200


lectura_fast
Cache= 1024,
Cache= 2048,
Cache= 4096,
Cache= 8192.
Number of misses
Matrix Size (Bytes)
1.2x10^{8}
1x10^{8}
8x10^{7}
6x10^{7}
4x10^{7}
2x10^{7}
0
4000
4200
4400
4600
4800
5000
5200


lectura_slow
Cache= 1024,
Cache= 2048,
Cache= 4096,
Cache= 8192.
Number of misses
Matrix Size (Bytes)
1.6x10^{8}
1.4x10^{8}
1.2x10^{8}
1x10^{8}
8x10^{7}
6x10^{7}
4x10^{7}
2x10^{7}
4000
4200
4400
4600
4800
5000
5200

4. Ejercicio 3: Caché y multiplicación de matrices

Se crean los programas mult_matrices y mult_matrices_transpuesta encargados de multiplicar dos matrices, siendo en el último en el que se ha empleado la matriz transpuesta para dicha operación.

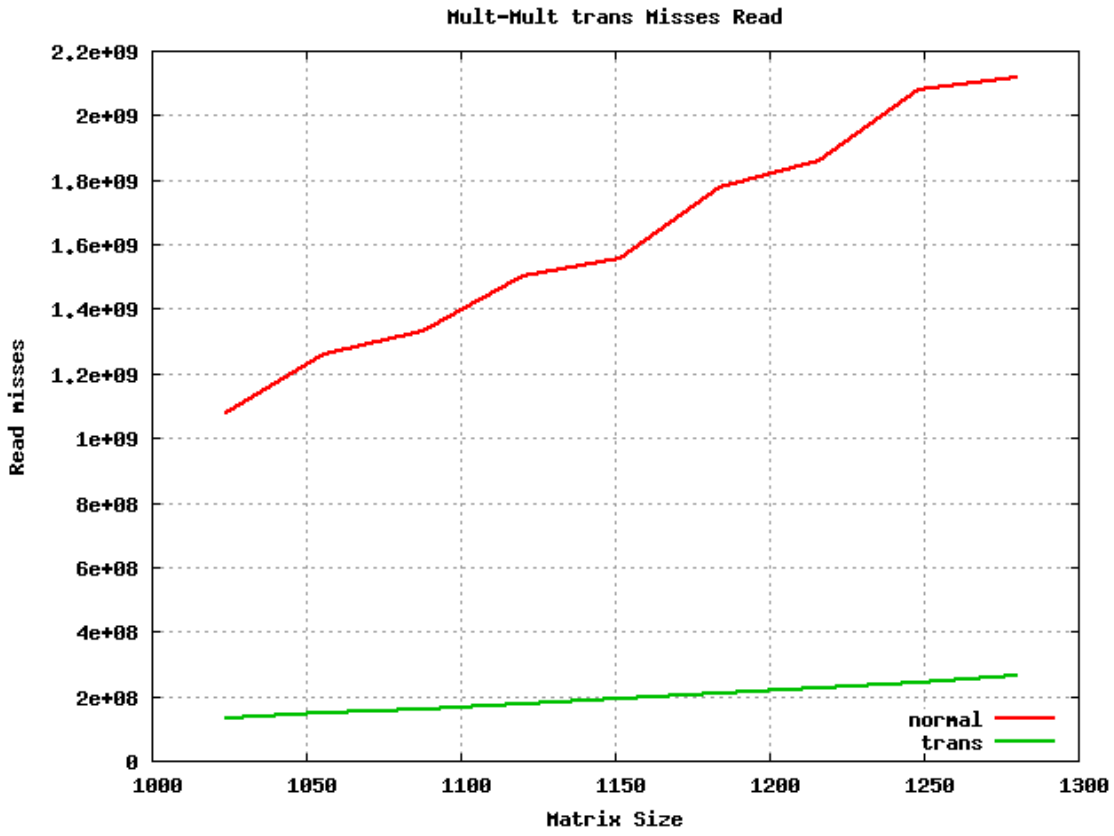
Para lo toma de datos y realización de gráficas se ha creado el script ej3.sh que, de forma similar a lo realizado en ejercicios anteriores, se encarga de ejecutar ambos programas para distintos tamaños de matriz y medir tiempos y fallos.

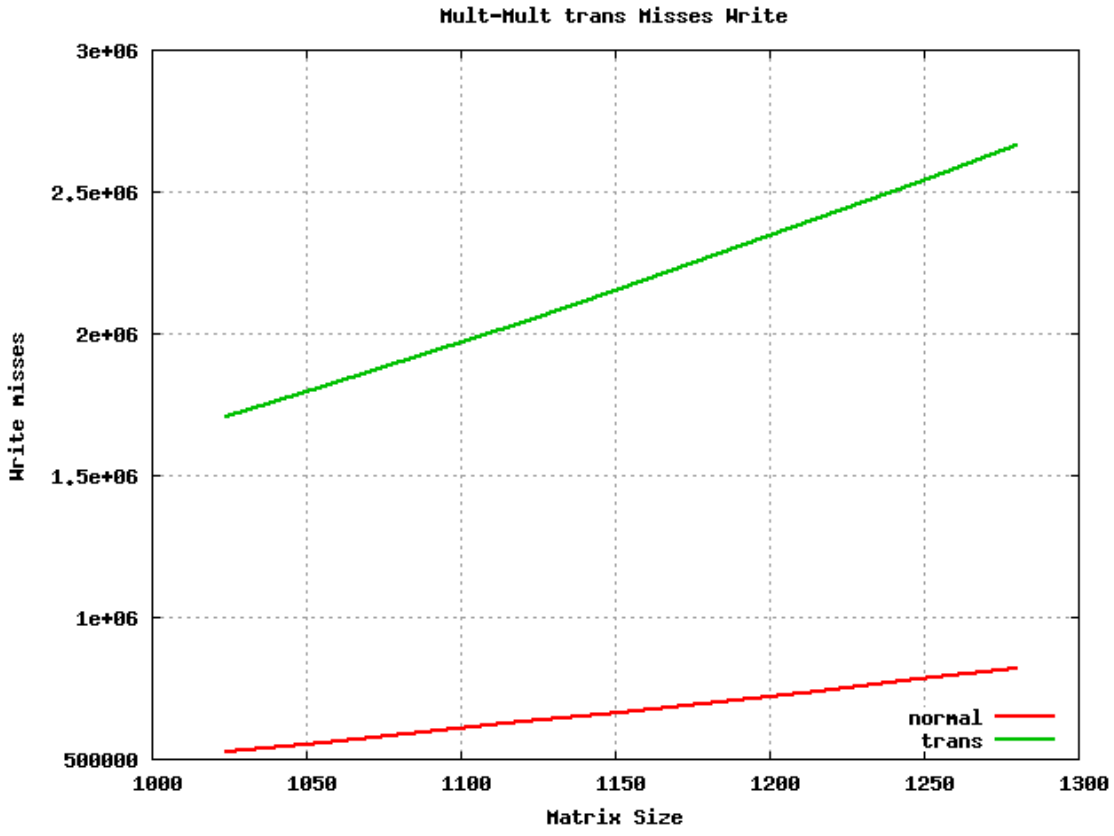
De los resultados y las gráficas podemos observar que la función de multiplicación normal tiene muchos más fallos de lectura (esto es debido a lo mismo estudiado en el ejercicio 2, la multiplicación 2 requiere recorrer las columnas mientras que la transpuesta emplea las filas)

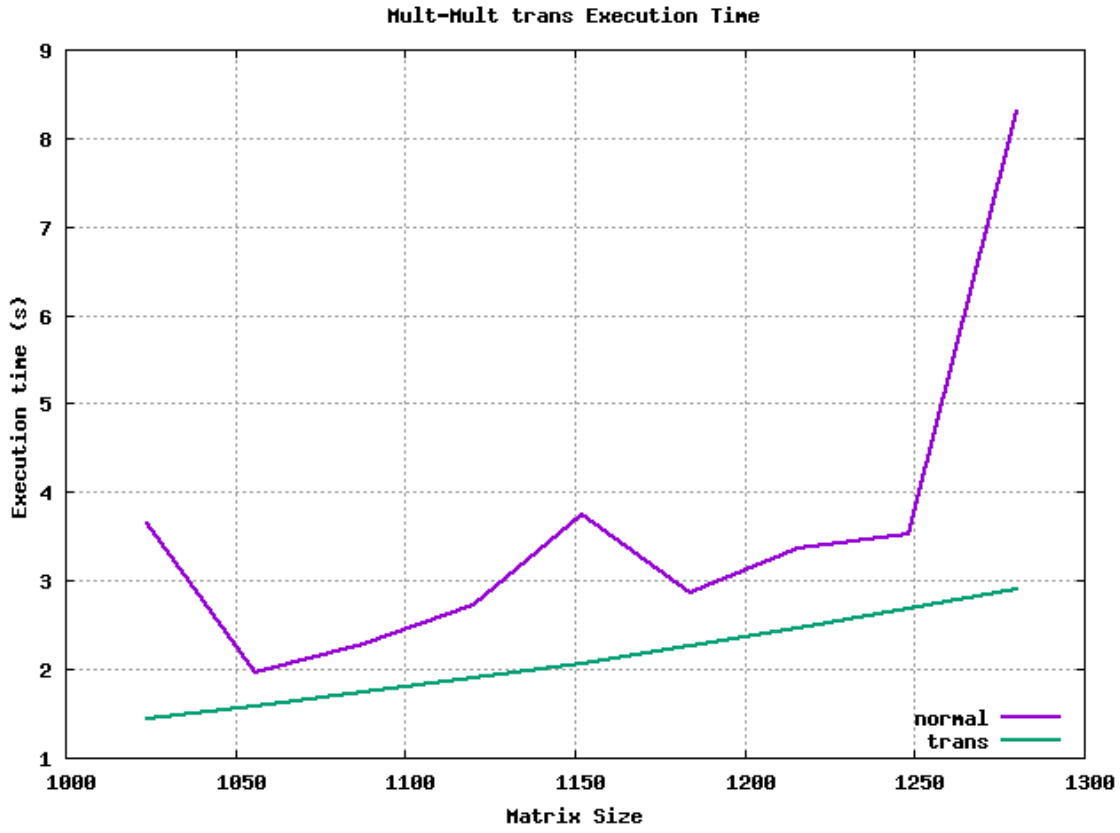
Esto a su vez repercute en el rendimiento donde claramente se aprecia la superioridad en el tiempo de ejecución de la función normal frente a la trans, debido a que, aunque la función trans requiere del cálculo de la transpuesta, esto es despreciable frente a los consumos de la multiplicación (transponer supone un doble bucle O(N^2), mientras que la multiplicación un bucle triple O(N^3)).

Por último, se observa un mayor número de errores de escritura en trans frente a normal, esto se debe a que la escritura del resultado es idéntica para ambos, sin embargo, trans se debe ocupar también de escribir la matriz transpuesta.

Los resultados gráficos se muestran en las siguientes diapositivas.


Mult-Mult trans Misses Read
Read misses
2.2e+09
2e+09
1.8e+09
1.6e+09
1.4e+09
1.2e+09
1e+09
8e+08
6e+08
4e+08
2e+08
0
1000
1050
1100
1150
1200
1250
1300
Matrix Size
normal
trans


Mult-Mult trans Misses Write
Write misses
3e+06
2.5e+06
2e+06
1.5e+06
1e+06
500000
1000
1050
1100
1150
1200
1250
1300
Matrix Size
normal
trans


Mult-Mult trans Execution Time
Execution time (s)
Matrix Size
normal
trans
1
2
3
4
5
6
7
8
9
1000
1050
1100
1150
1200
1250
1300

[FINAL DE DOCUMENTO]