Silvia Tomey Prieto
Grupo 1323
Pareja 12

Práctica 3

Memoria caché y rendimiento

La práctica 3 se ha realizado sobre un equipo personal con multiprocesador Intel Core i5 de 2 núcleos.

Ej0: Información sobre la caché del sistema

Inspeccionar las características de la caché del procesador mediante la línea de comandos.

En este caso, nos encontramos ante un multiprocesador de 4 núcleos con una caché de 3072KB con una alineación de 64 bytes.

```
cpu MHz             : 800.000
cache size          : 3072 KB
physical id         : 0
```

```
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits phy
```

Podemos observar la situación de los 3 manejadores de la memoria caché dentro del procesador, así como su tamaño y funcionamiento.

```
Handle 0x0008, DMI type 7, 19 bytes
Cache Information
        Socket Designation: CPU Internal L2
        Configuration: Enabled, Not Socketed, Level 2
        Operational Mode: Write Back
        Location: Internal
        Installed Size: 512 kB
        Maximum Size: 512 kB
        Supported SRAM Types:
                Unknown
        Installed SRAM Type: Unknown
        Speed: Unknown
        Error Correction Type: Single-bit ECC
        System Type: Unified
        Associativity: 8-way Set-associative

Handle 0x0009, DMI type 7, 19 bytes
Cache Information
        Socket Designation: CPU Internal L1
        Configuration: Enabled, Not Socketed, Level 1
        Operational Mode: Write Back
        Location: Internal
        Installed Size: 128 kB
        Maximum Size: 128 kB
        Supported SRAM Types:
                Unknown
        Installed SRAM Type: Unknown
        Speed: Unknown
        Error Correction Type: Single-bit ECC
        System Type: Other
        Associativity: 8-way Set-associative
```

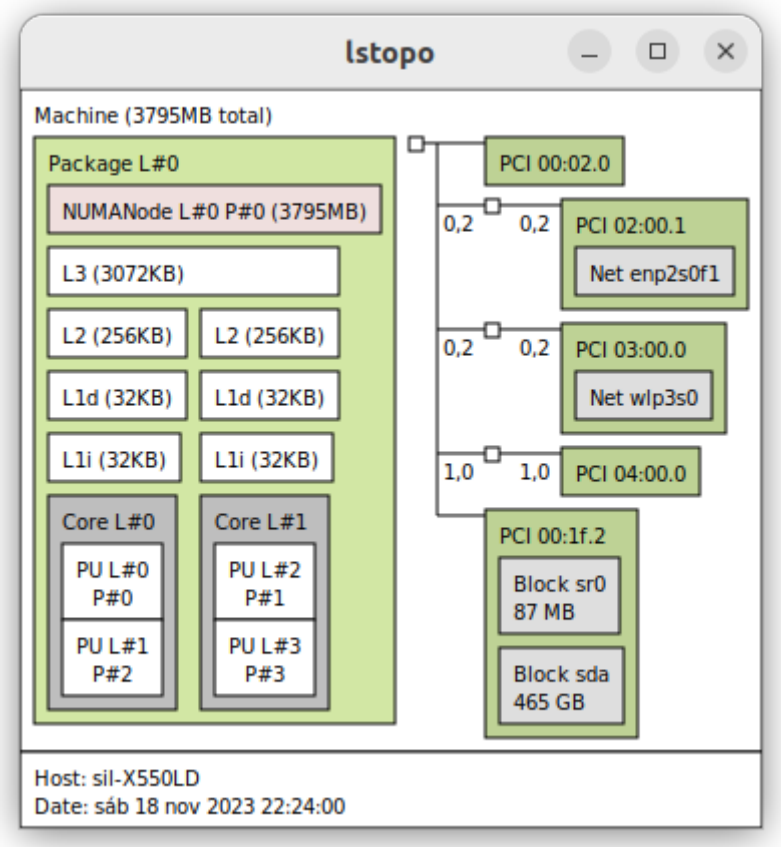
```
        Location: Internal
        Installed Size: 128 kB
        Maximum Size: 128 kB
        Supported SRAM Types:
                Unknown
        Installed SRAM Type: Unknown
        Speed: Unknown
        Error Correction Type: Single-bit ECC
        System Type: Other
        Associativity: 8-way Set-associative

Handle 0x000A, DMI type 7, 19 bytes
Cache Information
        Socket Designation: CPU Internal L3
        Configuration: Enabled, Not Socketed, Level 3
        Operational Mode: Write Back
        Location: Internal
        Installed Size: 3 MB
        Maximum Size: 3 MB
        Supported SRAM Types:
                Unknown
        Installed SRAM Type: Unknown
        Speed: Unknown
        Error Correction Type: Single-bit ECC
        System Type: Unified
        Associativity: 12-way Set-associative

Handle 0x000B, DMI type 16, 23 bytes
Physical Memory Array
        Location: System Board Or Motherboard
        Use: System Memory
```

Al ejecutar getconf sobre la búsqueda de grep sobre “caché”, obtenemos lo siguiente:

```
root@sil-X550LD:/home/sil/Escritorio/ArqO/
LEVEL1_ICACHE_SIZE                 32768
LEVEL1_ICACHE_ASSOC
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
LEVEL4_CACHE_LINESIZE
```

Que nos muestra toda la información disponible sobre la memoria caché de nuestro sistema.

Al ejecutar lstopo obtenemos de manera gráfica un análisis de la arquitectura del equipo similar al obtenido con el comando anterior:



Para los siguientes apartados se han generado 3 scripts que facilitan la ejecución del código de cada apartado, así como sus gráficas.

Ej1: Memoria caché y rendimiento

1) 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 las mismas.

Para la realización de este apartado se ha generado el script *slow_fast_time.sh*, que nos proporciona las medias de los tiempos para el cálculo de cada uno de los tamaños de las matrices, tanto con el programa slow como con el programa fast.

Se han hecho 10 ejecuciones de cada matriz con ambos programas.

2) Indique en la memoria una explicación razonada del motivo por el que hay que realizar múltiples veces la toma de medidas de rendimiento para cada programa y tamaño de matriz.

Se intercalan las ejecuciones para aprovechar mejor las memorias caché del sistema. Cada uno de los 2 programas puede cachearse en memorias distintas, de manera que pueden paralelizarse los procesos de cálculo para slow y fast. Si se ejecutaran primero todos los cálculos de slow y luego los de fast, estaríamos haciendo un uso en serie de las memorias, bajando considerablemente su rendimiento.

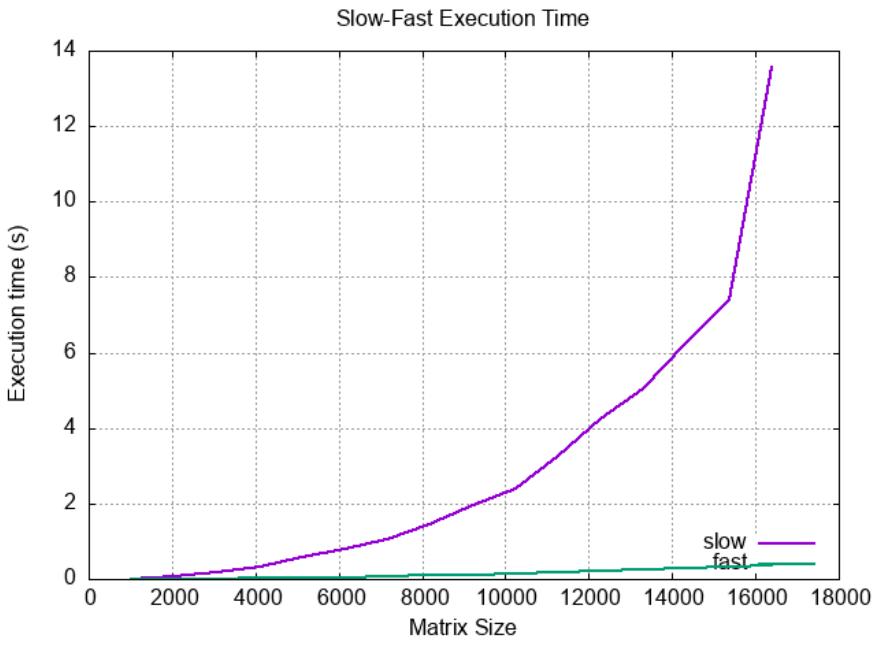
3) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero se llame time_slow_fast.dat y siga el formato:
<N> <tiempo "slow"> <tiempo "fast">

Se ha obtenido un fichero con los tiempos medios obtenidos con ambos programas para cada tamaño de matriz. El fichero también sigue el formato solicitado.

4) Pinte una gráfica con GNUplot en formato PNG que muestre los resultados obtenidos para los programas slow y fast, llámela time_slow_fast.png, e inclúyala en la memoria.

La gráfica obtenida a partir del fichero del apartado anterior es la siguiente:

Tiempos medios



5) Además de la gráfica mencionada, se pide en la memoria explicar el método seguido para la obtención de los datos, y una justificación del efecto observado. ¿Por qué para matrices pequeñas los tiempos de ejecución de ambas versiones son similares, pero se separan según aumenta el tamaño de la matriz? ¿Cómo se guarda la matriz en memoria, por filas (los elementos de una fila están consecutivos) o bien por columnas (los elementos de una columna son consecutivos)?

Los elementos de una matriz se almacenan en memoria por filas. La diferencia de rendimiento entre un programa y otro está en el segundo bucle for:

slow.c

```
tipo compute(tipo **matrix,int n)
{
    tipo sum=0;
    int i,j;

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            sum += matrix[j][i];
        }
    }
    return sum;
}
```

fast.c

```
tipo compute(tipo **matrix,int n)
{
    tipo sum=0;
    int i,j;

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            sum += matrix[i][j];
        }
    }
    return sum;
}
```

En el programa slow, se accede a los elementos de la matriz en orden matrix[j][i]; esto es, primero por columnas y luego por filas. En el caso del programa fast, es justo al contrario: matrix[i][j]; primero por filas y luego por columnas.

Si tenemos en cuenta lo dicho en el primer párrafo (que los elementos se almacenan por filas), el programa slow está accediendo a los elementos de la matriz de una manera muy ineficiente (tendrá que recorrer los elementos de las matrices muchas más veces que el programa fast). Cuando estamos haciendo cálculos con matrices pequeñas, la diferencia no es tan grande entre ambos programas porque el número de elementos a recorrer es pequeño. Sin embargo, cuando trabajamos con matrices muy grandes, en las que hay millones de elementos, el programa slow recorrerá, efectivamente, millones de elementos más que el programa fast. Por eso la diferencia de rendimiento va creciendo más y más conforme aumenta el tamaño de la matriz.

Ej2: Tamaño de la caché y rendimiento

1) Tomar datos de la cantidad de fallos de caché producidos en la lectura y escritura de datos al ejecutar las dos versiones (fast y slow) del programa que calcula las sumas de los elementos de una matriz para matrices de tamaño NxN, con N variando entre 1024+128*P y 5120+128*P y con un incremento en saltos de 1024 unidades. Utilizar tamaños de caché de primer nivel (tanto para datos como para instrucciones) de tamaño variable de 1024, 2048, 4096 y 8192 Bytes (note que deben ser potencias de 2) y una caché de nivel superior de tamaño fijo igual a 8 Mbytes. Para todas las cachés asumir que son de correspondencia directa (es decir, con una única vía) y con tamaño de línea igual a 64 bytes. Como valgrind es un emulador, solo es necesario ejecutar 1 vez el programa para cada configuración solicitada.

Para tomar los datos solicitados, se han programado en el script *cache_tamCache.sh* los comandos:

valgrind --tool=cachegrind --cachegrind-out-file=slow_out.dat –LL=8192,1,64 --I1=$C,1,64 --D1=$C,1,64 ./slow $N

valgrind --tool=cachegrind --cachegrind-out-file=fast_out.dat –LL=8192,1,64 --I1=$C,1,64 --D1=$C,1,64 ./fast $N

que nos proporcionarán los errores de lectura y escritura de ambos programas para cada tamaño de matriz($N) y de caché($C).

Además de los ficheros solicitados en el siguiente apartado, se van a generar 2 ficheros auxiliares que se corresponden con la salida del comando de valgrind, llamados *slow_out.dat* y *fast_out.dat*.

2) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que los datos se almacenen en ficheros llamados cache_<tamCache>.dat (esto es, se generará un fichero para cada tamaño de caché) siguiendo el formato:

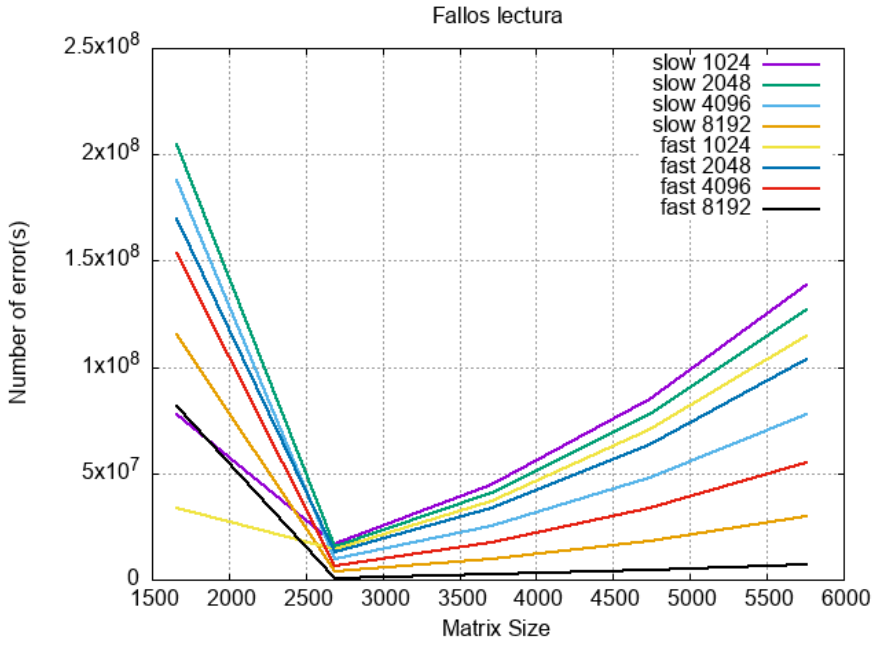
<N> <D1mr "slow"> <D1mw "slow"> <D1mr "fast"> <D1mw "fast">

Se han obtenido un total de 4 ficheros, 1 para cada uno de los tamaños de caché especificados en el enunciado. Cada uno de los ficheros sigue también el formato solicitado.

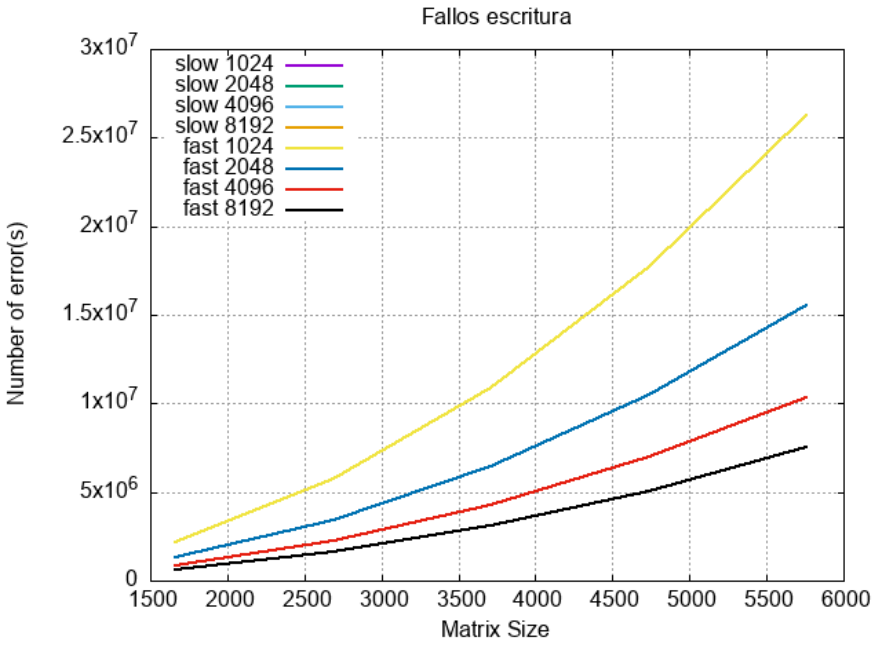
3) Se han de representar los resultados sobre dos gráficas generadas con GNUplot en formato PNG: una gráfica para los fallos de lectura de datos en un fichero cache_lectura.png y otra gráfica para los fallos de escritura de datos en un fichero cache_escritura.png. En cada caso, en el eje de las abscisas se indicará el tamaño de la matriz, mientras que en eje de las ordenadas se indicará el número de fallos. Deberá pintar una serie de datos por cada tamaño de caché indicado.

A partir de los datos obtenidos en el apartado 2, se han generado las siguientes gráficas:

Fallos de lectura



Fallos de escritura



4) Justifique el efecto observado. ¿Se observan cambios en la gráfica al variar los tamaños de caché para el programa slow? ¿Y para el programa fast? ¿Varía la gráfica cuando se fija en un tamaño de caché concreto y compara el programa slow y fast? ¿A qué se debe cada uno de los efectos observados?

Tanto para el programa slow como para el programa fast la tendencia del número de fallos de caché es aumentar, tanto para la lectura como para la escritura, conforme aumenta el tamaño de la matriz. Esto es debido a que, cuanto mayor es la matriz, mayor el número de datos que se deben leer y escribir.

También podemos observar que, conforme aumenta el tamaño de la caché, menor es el número de fallos. Esto tiene mucha lógica, ya que una caché más grande almacenará más datos, disminuyendo el número de veces en las que no se encuentran los datos.

Es necesario destacar cómo exclusivamente al comienzo de los programas los fallos de lectura son muy grandes. Esto se debe a que aún no hay datos almacenados en la memoria caché, y, cuando se accede a ella para proceder a la lectura, no se puede hallar el dato buscado. El gran descenso en los fallos se produce a partir del momento en el que tenemos los datos almacenados en nuestra memoria.

Con respecto a la gráfica de fallos de escritura, lo destacable es cómo el número de fallos es prácticamente similar tanto para slow como para fast (las líneas de ambos están superpuestas y por eso sólo se observan 4 en vez de 8). Esto es porque el número de fallos es independiente del programa que se use; con cualquiera de los dos necesitaremos escribir en la memoria.

Ej3: Caché y multiplicación de matrices

1) Tomar datos de los tiempos de ejecución obtenidos al ejecutar las dos versiones del programa que calcula la multiplicación de dos matrices de tamaño NxN, con N variando entre 128+16*P y 2176+16*P y con un incremento en saltos de 256 unidades. Deberá tomar resultados para todas las ejecuciones múltiples veces y de forma intercalada (como se indica en el ejercicio 1) y calcular la media de las mismas.

Para obtener los datos de este apartado hemos ejecutado nuestro script *matrix_mult.sh*, que nos proporcionará las medias de tiempo necesarias para ejecutar las multiplicaciones de las matrices de distintos tamaños.

2) Tomar datos de la cantidad de fallos de caché producidos en la lectura y escritura de datos al ejecutar las dos versiones del programa que calcula la multiplicación de dos matrices de tamaño NxN, con N variando entre 128+16*P y 2176+16*P y con un incremento en saltos de 256 unidades. En este caso se usará con cachegrind con la configuración de cachés por defecto. Recuerda que como valgrind es un emulador, solo es necesario ejecutar 1 vez el programa para cada configuración solicitada.

En el mismo script, hemos vuelto a ejecutar los comandos de *cachegrind* para obtener los fallos de lectura y escritura de la caché, esta vez con un tamaño de memoria por defecto.

3) Guardar para la entrega el fichero con los resultados obtenidos. Por criterios de unificación, se pide que el fichero se llame mult.dat y siga el formato:

<N> <tiempo “normal”> <D1mr “normal”> <D1mw “normal”>
<tiempo “trasp”> <D1mr “trasp”> <D1mw “trasp”>

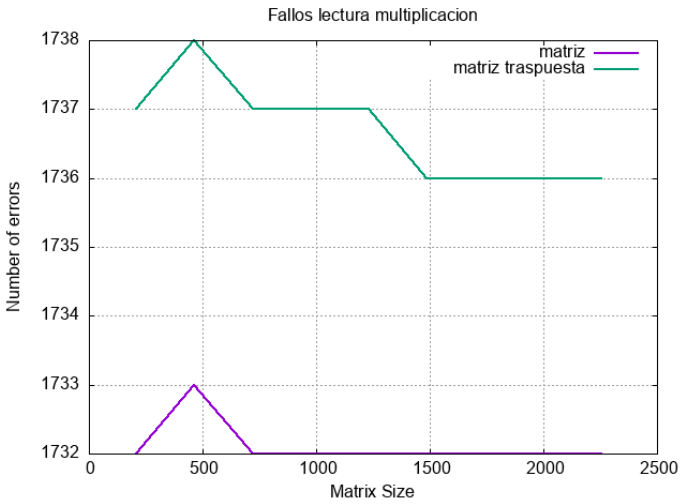
Nota: Es posible ejecutar los apartados 1) y 2) en un mismo script, lo que permitirá unificar la salida de ambos programas de forma más sencilla que si se ejecutan en scripts separados. Pero recordad que solo se requiere 1 ejecución con valgrind.

Se ha obtenido el fichero solicitado a partir de la ejecución del script mencionado. El fichero cumple con el formato requerido en el enunciado.

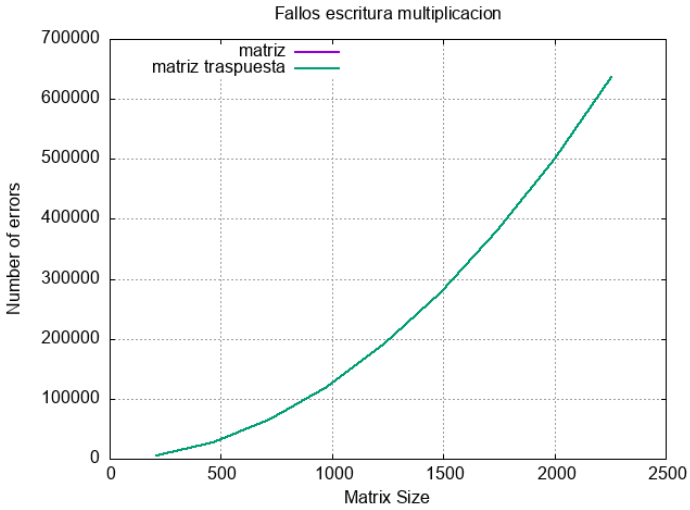
4) Se han de representar los resultados obtenidos de las 3 medidas realizadas para cada una de las dos versiones del programa de multiplicación de matrices sobre tres gráficas generadas con GNUplot en formato PNG: una para los fallos de caché en lectura en un fichero llamado mult_cache_read.png, otra para los fallos de caché en escritura en un fichero llamado mult_cache_write.png y otra para los tiempos de ejecución en un fichero llamado mult_time.png e incluirlas en la memoria.

Las gráficas obtenidas a partir de los datos de los ficheros obtenidos en el apartado anterior son las siguientes:

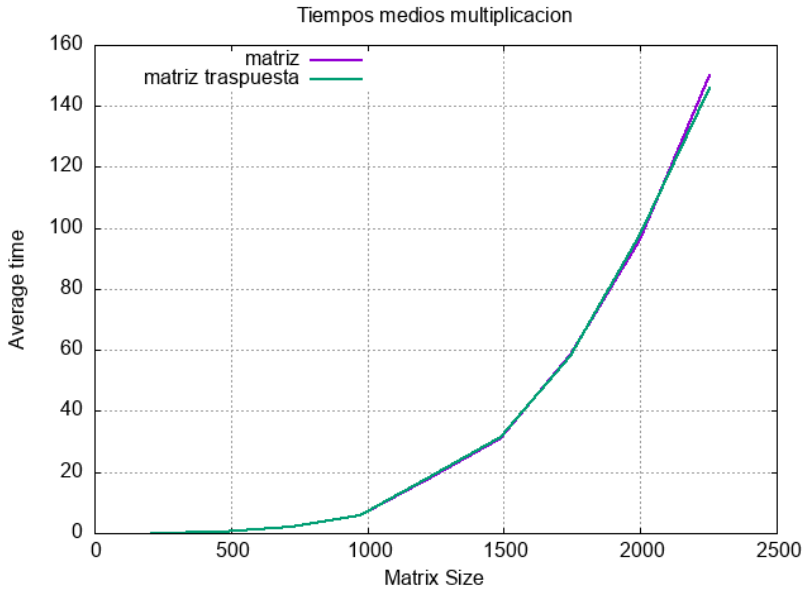
Fallos de lectura



Fallos de escritura



Tiempos medios



5) Justifique el efecto observado. ¿Se observan cambios de tendencia al variar los tamaños de las matrices? ¿A qué se deben los efectos observados?

Lo más destacable de las gráficas obtenidas es cómo aumentan los tiempos de cálculo conforme aumenta el tamaño de las matrices. También cabe mencionar que los tiempos son prácticamente iguales tanto para la multiplicación normal como para la multiplicación con la matriz traspuesta. Esto nos indica que la complejidad de nuestro cálculo radica en la multiplicación en sí, y no en la trasposición de la segunda matriz.

Esto también explicaría los datos de las gráficas de los fallos de memoria. Puesto que hemos observado que no aumenta la complejidad del algoritmo con respecto a la forma en la que se almacena la matriz, los errores de escritura y de lectura también son prácticamente iguales en ambos casos.

Los fallos de escritura aumentan conforme aumenta el tamaño de la matriz porque cada vez almacenamos más elementos en la memoria, por lo que el número de veces que se escribe y sobreescribe también es mayor. Los fallos de lectura, en cambio, no son muchos porque ya hemos procesado y cacheado las matrices a la hora de medir sus tiempos, por lo que cuando procedemos a medir fallos no es necesario computarlas de nuevo.