ARQUITECTURA DE ORDENADORES

Práctica 3: Memoria Caché y Rendimiento

Grupo 1313, Pareja 04:

Florentino García Aznar
Alejandro Monterrubio Navarro

Ejercicio 0:

En este ejercicio, se pide la obtención de la configuración cache de un equipo que utilice Linux, para esto vamos a ejecutar una serie de comandos en los equipos del laboratorio.

Para ello empezamos ejecutando el comando:

> getconf -a | grep -i cache

Esto nos proporciona la siguiente información:

```
e420674@8A-20-8-20:~$ getconf -a | grep -i cache
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                 4
LEVEL2_CACHE_LINESIZE              64
LEVEL3_CACHE_SIZE                  8388608
LEVEL3_CACHE_ASSOC                 16
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC
LEVEL4_CACHE_LINESIZE
```

LEVEL 1:

- LEVEL1_ICACHE_SIZE: Indica el tamaño de la caché de instrucciones de nivel 1 (L1i), que en este caso es de 32 KB (32768 bytes).
- LEVEL1_ICACHE_LINESIZE: Especifica el tamaño de cada línea de la caché de instrucciones de nivel 1, que es de 64 bytes.
- LEVEL1_DCACHE_SIZE: Indica el tamaño de la caché de datos de nivel 1 (L1d), que también es de 32 KB.
- LEVEL1_DCACHE_LINESIZE: El tamaño de cada línea de la caché de datos de nivel 1 es igualmente de 64 bytes.
- LEVEL1_DCACHE_ASSOC: Muestra la asociatividad de la caché de datos de nivel 1, que es 8-way set associative.

Los puntos del 1 al 5 indican que la máquina tiene una separación entre las cachés de instrucciones y datos en el nivel 1 (L1i y L1d, respectivamente), lo cual es común en arquitecturas modernas para evitar conflictos y mejorar la eficiencia al acceder a las instrucciones y datos simultáneamente.

LEVEL 2:

- LEVEL2_CACHE_SIZE: Refiere al tamaño de la caché de nivel 2 (L2), que es compartida (unificada) para datos e instrucciones y tiene un tamaño de 256 KB (262144 bytes).
- LEVEL2_CACHE_LINESIZE: El tamaño de cada línea de la caché de nivel 2 es de 64 bytes.
- LEVEL2_CACHE_ASSOC: La asociatividad de la caché de nivel 2 es 4-way set associative.

La caché de nivel 2 es unificada, lo que significa que es utilizada tanto para datos como para instrucciones.

LEVEL 3:

- LEVEL3_CACHE_SIZE: Se refiere al tamaño de la caché de nivel 3 (L3), que también es compartida y tiene un tamaño considerable de 8 MB (8388608 bytes).
- LEVEL3_CACHE_LINESIZE: Al igual que L1 y L2, el tamaño de la línea de caché es de 64 bytes.
- LEVEL3_CACHE_ASSOC: La asociatividad de la caché de nivel 3 es 16-way set associative. La caché de nivel 3, al ser de mayor tamaño, permite una gran cantidad de datos e instrucciones para reducir los fallos de caché y mejorar el rendimiento general.

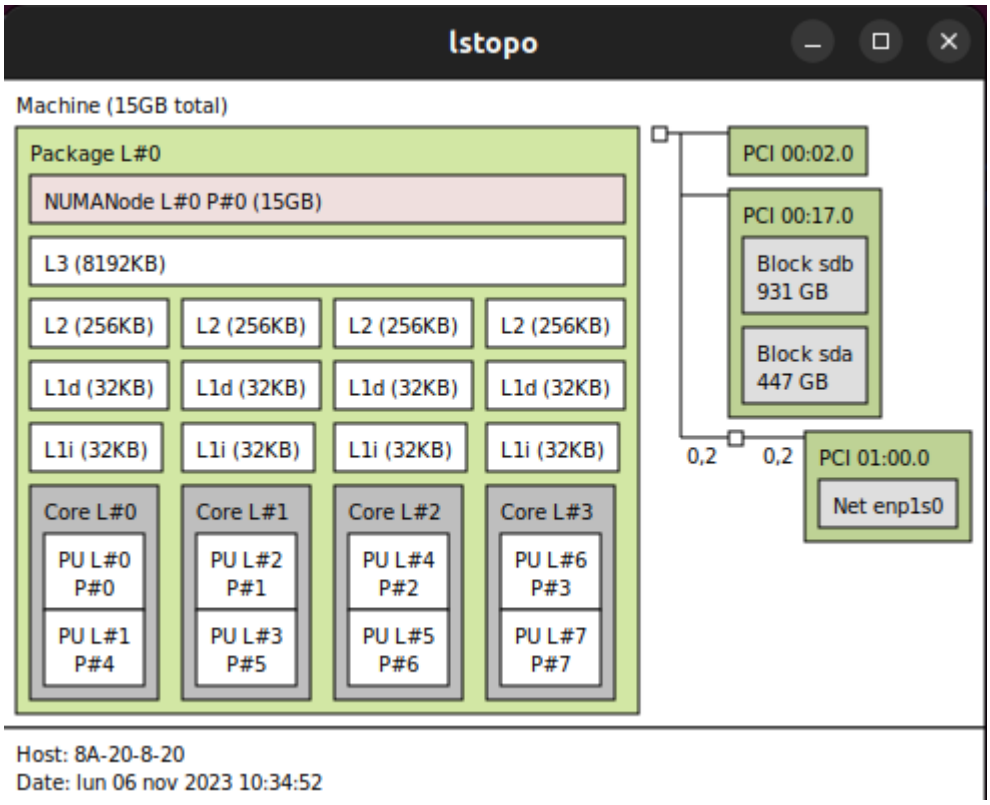
LEVEL 4:

- LEVEL4_CACHE_SIZE: No hay una caché de nivel 4 en esta arquitectura, como lo indica el tamaño de 0.

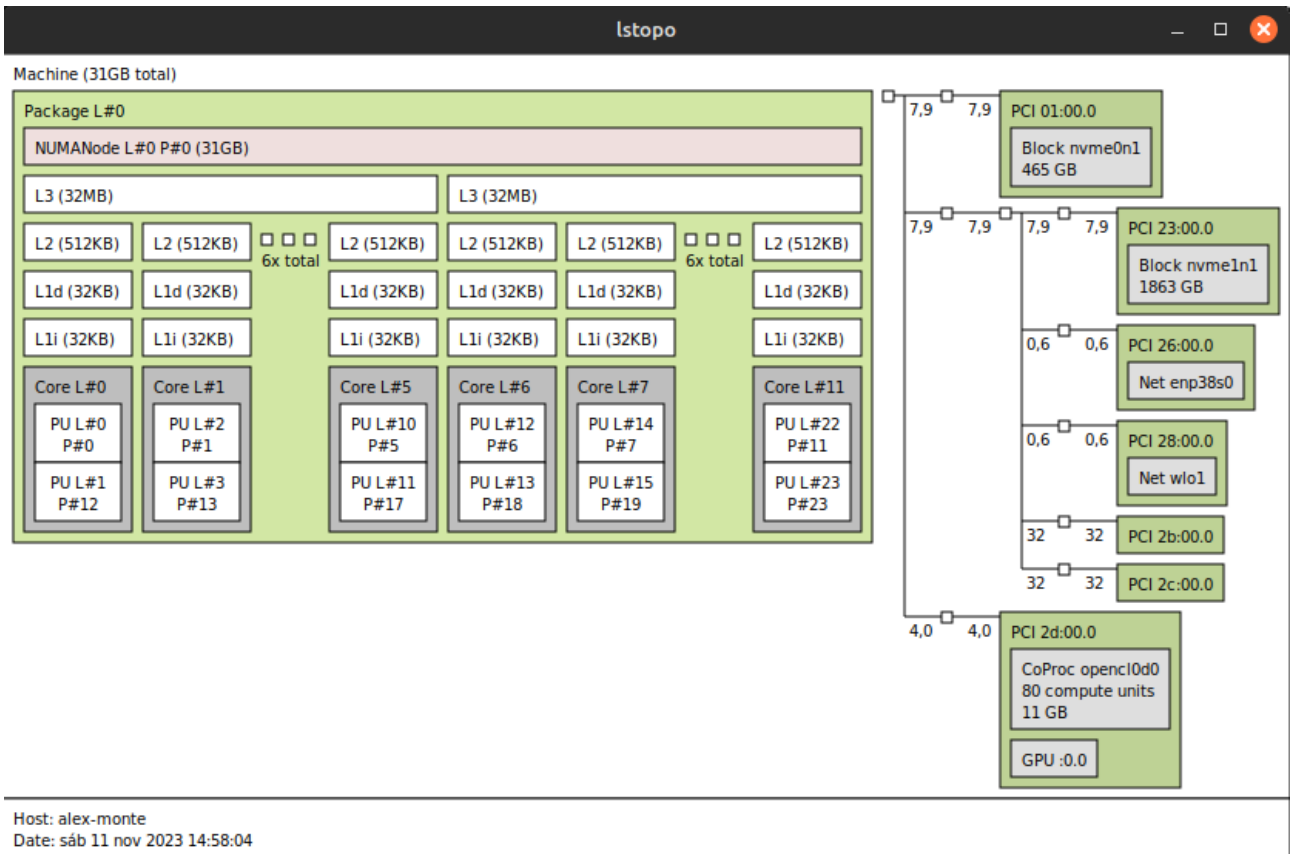
Al ejecutar en los laboratorios el comando:

> lstopo

Para obtener más información, se consigue:



Donde podemos ver la información adicional, así como la distribución de las cachés. Si lo ejecutamos en uno de nuestros equipos particulares se obtiene la siguiente información:



Si lo analizamos y después de investigar que eran distintos componentes, hemos conseguido la siguiente información:

Machine (31GB total): Este es el total de memoria RAM disponible en el sistema.

Package L#0: Se refiere al paquete físico de la CPU (a veces denominado "socket"), lo que indica que hay un solo procesador físico en esta máquina.

NUMANode L#0 (P#0 31GB): Indica que hay un nodo de memoria no uniforme (Non-Uniform Memory Access, NUMA) asociado con este paquete de CPU.

L3 (32MB): La caché de nivel 3 (L3) tiene 32MB y es compartida entre los núcleos del procesador.

L2 (512KB): Cada núcleo tiene su propia caché de nivel 2 (L2) de 512KB.

L1d (32KB) y L1i (32KB): Cada núcleo tiene su propia caché de nivel 1 (L1) dividida en dos partes: una caché de datos (L1d) y una caché de instrucciones (L1i), ambas de 32KB.

Cores and Processing Units (PUs): El diagrama muestra 12 núcleos (Core L#0 a Core L#11), y cada núcleo tiene dos unidades de procesamiento (PUs), lo que implica que cada núcleo ejecute dos hilos simultáneamente (por ejemplo, PU L#0 y PU L#12 en el Core L#0).

Otros dispositivos:

Blocks nvme0n1 y nvme1n1: Son dispositivos de almacenamiento NVMe, indicando conexión directa a las vías PCIe.

Net enp38s0 y wlo1: Representan interfaces de red.

CoProc opencl0d0: Un coprocesador.

GPU 0.0: Una unidad de procesamiento gráfico.

Ejercicio 1:

Para ejecutar este ejercicio se han utilizado varios comandos en shell y para hacer plot de la tabla, voy a poner aquí estos scripts:

Make:

> make

Este script de shell utiliza un bucle para ejecutar los programas con tamaños de matriz desde 1024 hasta 16384, en incrementos de 1024. Para cada tamaño, ejecuta ambos programas 10 veces, extrayendo el tiempo de ejecución (que suponemos es la tercera palabra en la línea que contiene 'Execution time') y lo añade al archivo “time_slow_fast.dat.”:

```
> for size in $(seq 1024 1024 16384); do
    for i in $(seq 1 10); do
        echo "Ejecutando slow con tamaño $size"
        time_slow=$(./slow $size | grep 'Execution time' | awk '{print $3}')
        echo "Ejecutando fast con tamaño $size"
        time_fast=$(./fast $size | grep 'Execution time' | awk '{print $3}')
        echo "$size $time_slow $time_fast" >> time_slow_fast.dat
    done
done
```

Para crear la gráfica con los tiempos de ejecucuón se asume que la primera columna de “time_slow_fast.dat” es el tamaño de la matriz, la segunda columna es el tiempo para slow y la tercera columna es el tiempo para fast (esto se encuentra en el archivo “plot_script.gnuplot”):

```
> set terminal png size 800,600
set output 'time_slow_fast.png'
set title 'Comparación de tiempos de ejecución - slow vs fast'
set xlabel 'Tamaño de la matriz (N)'
set ylabel 'Tiempo de ejecución (segundos)'
set key left top
set grid
plot 'time_slow_fast.dat' using 1:2 title 'Slow' with linespoints, \
     'time_slow_fast.dat' using 1:3 title 'Fast' with linespoints
```

1. Datos de tiempo de ejecución: De la gráfica proporcionada se observa que los tiempos de ejecución de los programas 'slow' y 'fast' aumentan con el tamaño de la matriz. Para matrices pequeñas, los tiempos de ejecución son similares, pero a medida que el tamaño de la matriz crece, el tiempo de ejecución del programa 'slow' aumenta significativamente en comparación con el 'fast'.

Medias:

N	Time_Slow (s)	Time_Fast (s)

1024	0.006030	0.001225
2048	0.022484	0.003550
3072	0.049088	0.006388
4096	0.094594	0.011232
5120	0.139479	0.017577
6144	0.227540	0.025594
7168	0.285053	0.034826
8192	0.848969	0.046298
9216	0.476925	0.057229
10240	0.607212	0.071058
11264	0.754722	0.087425
12288	1.300784	0.102027
13312	1.005635	0.118912
14336	1.184110	0.136822
15360	1.327620	0.158859
16384	3.566662	0.181196

2. Razón para múltiples ejecuciones: Las pruebas de rendimiento se realizan múltiples veces para obtener un resultado más confiable y reducir la varianza causada por las fluctuaciones transitorias en el rendimiento del sistema. Estas fluctuaciones pueden deberse a la carga del sistema, procesos en segundo plano, o variaciones en la gestión del caché y memoria. Al realizar varias ejecuciones y tomar la media, se minimiza el impacto de estos factores aleatorios, proporcionando una estimación más precisa del rendimiento real del programa.

3. Archivo con resultados: El archivo time_slow_fast.dat con los resultados de las ejecuciones se encuentra en la carpeta de la práctica. Este archivo debe sigue el formato especificado, que cada línea contenga el tamaño de la matriz seguido de los tiempos promedio de ejecución de los programas 'slow' y 'fast'.

4. Gráfica en GNUplot: La gráfica time_slow_fast.png muestra los resultados obtenidos y ha sido creada siguiendo las instrucciones proporcionadas para GNUplot.

5. Método y justificación del efecto observado:

- Método de obtención de datos: Los datos se obtuvieron ejecutando iterativamente los programas 'slow' y 'fast' con tamaños de matriz incrementales y registrando los tiempos de ejecución. Cada tamaño de matriz se ejecutó múltiples veces y se calculó la media de los tiempos de ejecución para cada tamaño.

- Justificación del efecto observado: La diferencia en el rendimiento entre 'slow' y 'fast' con matrices más grandes se debe al uso del caché. Las matrices pequeñas caben en los niveles superiores de la caché, lo que permite un acceso rápido tanto para 'slow' como para 'fast'. Sin embargo, a medida que el tamaño de la matriz excede la capacidad de la caché, 'slow' sufre más penalizaciones debido a su patrón de acceso a la memoria. Si 'slow' accede a la memoria por filas y 'fast' por columnas (o viceversa), uno de ellos puede estar accediendo a la memoria de manera no secuencial, lo que resulta en más fallos de caché y, por tanto, un tiempo de ejecución mayor.

Importante: Se ha realizado este ejercicio dos veces cambiando el orden de los bucles para obtener los resultados de distinta manera y poder estudiarlo de otra manera, esto fue sugerido por el profesor en clase. El script es:

```
> declare -A times_slow times_fast

# Inicializa los acumuladores de tiempo
for size in $(seq 1024 1024 16384); do
   times_slow[$size]=0
   times_fast[$size]=0
done

# Ejecuta los programas y acumula los tiempos
for i in $(seq 1 10); do
   for size in $(seq 1024 1024 16384); do
      echo "Ejecutando slow con tamaño $size"
      time_slow=$(./slow $size | grep 'Execution time' | awk '{print $3}')
      times_slow[$size]=$(echo "${times_slow[$size]} + $time_slow" | bc)

      echo "Ejecutando fast con tamaño $size"
      time_fast=$(./fast $size | grep 'Execution time' | awk '{print $3}')
      times_fast[$size]=$(echo "${times_fast[$size]} + $time_fast" | bc)
   done
done

# Calcula los promedios y escribe en un archivo
for size in $(seq 1024 1024 16384); do
   avg_time_slow=$(echo "${times_slow[$size]} / 10" | bc -l)
   avg_time_fast=$(echo "${times_fast[$size]} / 10" | bc -l)
   echo "$size $avg_time_slow $avg_time_fast" >> time_slow_fast_avg.dat
done
```

Explicación del script:

1. Inicialización de Acumuladores de Tiempo:

El script crea dos arrays asociativos, times_slow y times_fast, para acumular los tiempos de ejecución de los programas slow y fast, respectivamente.

Inicializa estos arrays para cada tamaño de matriz desde 1024 hasta 16384, con un incremento de 1024.

2. Ejecución de los Programas y Acumulación de Tiempos:

El script ejecuta un bucle doble. El bucle externo se ejecuta 10 veces (de 1 a 10), y el bucle interno itera a través de los diferentes tamaños de matriz (de 1024 a 16384, en incrementos de 1024).

En cada iteración del bucle interno, el script ejecuta los programas slow y fast con el tamaño de matriz actual, captura el tiempo de ejecución de cada uno y lo acumula en los arrays correspondientes (times_slow y times_fast).

3. Cálculo de Promedios y Escritura en Archivo:
Después de completar todas las iteraciones, el script itera sobre los tamaños de matriz una vez más.

Para cada tamaño, calcula el tiempo promedio de ejecución tanto para slow como para fast dividiendo el tiempo acumulado por 10 (número de ejecuciones).

Escribe estos tiempos promedio en un archivo llamado time_slow_fast_avg.dat.

4. Formato del Archivo de Salida:

El archivo time_slow_fast_avg.dat contiene tres columnas: tamaño de la matriz, tiempo promedio de ejecución de slow, y tiempo promedio de ejecución de fast.

Este archivo se utiliza luego para generar un gráfico en Gnuplot que muestra las dos líneas de tiempo promedio para slow y fast.

Las gráficas y scripts por separado para cada resolución, se encuentran en la carpeta Ejercicio1a el primer caso visto y en la carpeta Ejercicio1b el segundo.

Ejercicio 2:

Gráfica de Fallos de Escritura de Caché:

Programa Slow: A medida que el tamaño de la caché aumenta, el número de fallos de escritura disminuye. Esto es esperado ya que una caché más grande puede almacenar más datos, lo que reduce la necesidad de recuperar esos datos desde un nivel superior de memoria, que es más lento.

Programa Fast: Similar al programa slow, el programa fast también muestra una disminución en los fallos de escritura a medida que el tamaño de la caché aumenta.

Gráfica de Fallos de Lectura de Caché:

Programa Slow: Se observa un patrón similar al de los fallos de escritura, donde un tamaño de caché más grande reduce los fallos de lectura. A medida que la matriz crece, las lecturas de la memoria se hacen más frecuentes, y una caché más grande puede contener más de estos datos, reduciendo los fallos.

Programa Fast: También muestra una reducción en los fallos de lectura con el aumento del tamaño de la caché, aunque la tasa de disminución puede ser diferente en comparación con el programa slow debido a posibles diferencias en la eficiencia del acceso a los datos.

Comparación entre Slow y Fast:

En ambas gráficas, se observa que para un tamaño de caché fijo, el programa fast generalmente tiene menos fallos de caché que el programa slow. Esto indica que el programa fast es más eficiente en el manejo de la caché o en el acceso a los datos, lo que resulta en una menor cantidad de fallos de caché.

Razones de los Efectos Observados:

Localidad Temporal: Los programas que reutilizan datos o instrucciones recientemente referenciados se benefician de una caché más grande, ya que esta puede almacenar más de estos datos recientemente usados.

Localidad Espacial: Si un programa accede a datos que tienden a estar agrupados, una caché más grande puede almacenar estos grupos de datos, reduciendo los fallos.

Eficiencia del Programa: El programa fast puede estar mejor optimizado para hacer uso de la caché, tal vez a través de un mejor uso de la localidad o un diseño de algoritmo que minimice los accesos a memoria que no están en caché.

Ejercicio 3:

Tiempo de Ejecución (Execution Time):

Se observa que el tiempo de ejecución para ambos métodos aumenta exponencialmente con el tamaño de la matriz.
El método transpuesto es consistentemente más rápido que el método normal para tamaños de matriz más grandes.
La razón de esta mejora en el tiempo de ejecución se debe a una mejor localidad espacial y de caché en el método transpuesto, ya que acceder a los elementos de una matriz por filas (como se hace con la matriz transpuesta) es más eficiente en términos de caché que hacerlo por columnas debido a cómo se almacenan los datos en memoria.

Fallos de Caché de Lectura (Cache Read Misses):

Los fallos de caché de lectura también aumentan con el tamaño de la matriz, pero la tasa de incremento es mucho mayor para el método normal en comparación con el transpuesto.
Esto se debe a que el método transpuesto mejora la localidad de los datos al acceder a la memoria de una manera que es más consistente con el patrón de acceso de la caché. En contraste, el método normal sufre más fallos de caché debido a los accesos dispersos en la memoria al acceder a los elementos de la matriz B por columnas.

Fallos de Caché de Escritura (Cache Write Misses):

Los fallos de escritura aumentan con el tamaño de la matriz, aunque el incremento es más leve en comparación con los fallos de lectura.
El patrón de los fallos de escritura es similar para ambos métodos, lo que sugiere que las operaciones de escritura en la matriz resultante no se ven tan afectadas por la transposición de la matriz B. Esto podría deberse a que las escrituras en la matriz de resultados C se realizan por filas en ambos métodos y, por lo tanto, disfrutan de una buena localidad espacial.

Conclusión:

El cambio de tendencia observado al variar el tamaño de las matrices refleja la importancia de la localidad de datos en la eficiencia de los accesos a la memoria. El método transpuesto demuestra mejoras significativas en el tiempo de ejecución y en los fallos de caché de lectura debido a que accede a la memoria de una manera que coincide mejor con el diseño y la optimización de los sistemas de caché modernos.

Ejercicio 4:

Las gráficas proporcionadas muestran los resultados de un estudio sobre el impacto de la configuración de la caché y el tamaño de las matrices en la multiplicación de matrices normal y traspuesta. A partir de estos datos, se pueden realizar varias observaciones y análisis:

Fallos de Lectura de Caché (Read Misses): La gráfica muestra un incremento significativo en el número de fallos de lectura a medida que aumenta el tamaño de la matriz. Esto es esperable porque matrices más grandes requerirán más acceso a la memoria, lo cual puede exceder la capacidad de la caché y resultar en fallos de caché. La multiplicación traspuesta tiende a tener menos fallos de lectura, lo que sugiere que acceder a los datos de una matriz traspuesta puede resultar en un patrón de acceso más favorable para la caché debido a la localidad espacial mejorada.

Fallos de Escritura de Caché (Write Misses): Los fallos de escritura también aumentan con el tamaño de la matriz, aunque la variabilidad es mayor, lo que podría deberse a la variabilidad en el comportamiento de escritura de la caché y cómo las escrituras son manejadas por el algoritmo de escritura de caché (write-back o write-through, por ejemplo).

Tiempos de Ejecución: Como se espera, los tiempos de ejecución aumentan con el tamaño de la matriz para ambas versiones de la multiplicación. La multiplicación normal muestra tiempos de ejecución más largos en comparación con la traspuesta, lo cual es coherente con el número de fallos de lectura de caché: un mayor número de fallos puede resultar en tiempos de ejecución más largos debido a la necesidad de recuperar datos de niveles más bajos y más lentos de la jerarquía de memoria.

Conclusiones del Experimento:

La localidad de los datos es un factor crucial en la eficiencia de la caché. La versión traspuesta de la multiplicación de matrices mejora la localidad de los datos, lo que reduce los fallos de caché y, por tanto, mejora el tiempo de ejecución.

Los fallos de escritura parecen ser menos predecibles que los fallos de lectura, lo que sugiere que los patrones de acceso de escritura pueden no ser tan consistentemente optimizados por la traspuesta de la matriz.

La eficiencia de la caché y el rendimiento del algoritmo están fuertemente influenciados por el tamaño de la matriz en relación con el tamaño de la caché. A medida que las matrices se vuelven demasiado grandes para caber en la caché, el número de fallos aumenta dramáticamente.