ARQUITECTURA DE COMPUTADORES: PRÁCTICA 3

ALEJANDRO PASCUAL[1] Y VÍCTOR YRAZUSTA[2]

ÍNDICE

ÍNDICE DE FIGURAS

[1] *alejandro.pascualp@estudiante.uam.es*
[2] *victor.yrazusta@estudiante.uam.es*

1 EJERCICIO 0: INFORMACIÓN SOBRE LA CACHÉ DEL SISTEMA

1.1 Arquitectura de la caché de los equipos del laboratorio

Tras la ejecución del primer comando obtenemos información acerca del procesador y de su distribución de memorias caché:

```
vendor_id : GenuineIntel
cpu family : 6
model : 158
model name : Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz
cpu MHz : 900.029
cache size : 6144 KB
siblings : 4
cpu cores : 4
cache_alignment : 64
address sizes : 39 bits physical, 48 bits virtual
```

Recorte 1: cat /proc/cpuinfo (parte importante)

Como podemos observar contamos con direcciones de memoria físicas de 39 bits y direcciones de memoria virtuales de 48, así el tamño de los bloques de la caché, 64 bytes. La ejecución del siguiente comando muestra esto:

```
LEVEL1_ICACHE_SIZE              32768
LEVEL1_ICACHE_ASSOC             8
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               6291456
LEVEL3_CACHE_ASSOC              12
LEVEL3_CACHE_LINESIZE           64
LEVEL4_CACHE_SIZE               0
LEVEL4_CACHE_ASSOC              0
LEVEL4_CACHE_LINESIZE           0
```

Recorte 2: getconf -a | grep -i cache

Aquí podemos ver los diferentes tamaños de la caché del sistema, la cantidad de “conjuntos” o “vías” de esta y los tamaños de bloque para cada nivel (los equipos de laboratorio no cuentan con caché de nivel 4).

La ejecución del último comando nos proporciona una imagen en la que vemos reflejada esta información:

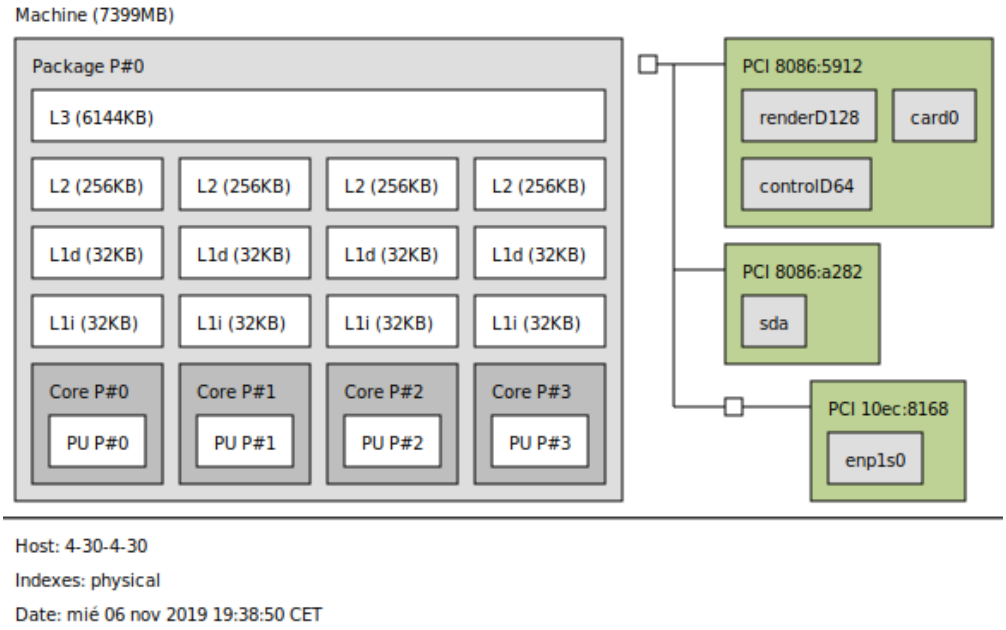

Figura 1: Arquitectura de la caché de los equipos del laboratorio.

Aquí podemos comprobar que contamos con una memoria caché de nivel 3 que comparten todos los núcleos del procesador, pero el resto de niveles son diferentes para cada núcleo.

2 EJERCICIO 1: MEMORIA CACHÉ Y RENDIMIENTO

2.1 Tomo de datos

Es necesario repetir varias veces la ejecución de las pruebas para conseguir información fiable. Nosotros hemos ejecutado cada test 10 veces de manera intercalada, y hemos realizado la media de todas estas métricas para la realización de las gráficas de los diferentes ejecicios mediante el siguiente código Python:

```python
import sys

if len(sys.argv) != 2:
    print('Numero incorrecto de argumentos.')
    exit()

file = open(sys.argv[1], 'r')

averages = {}

for line in file.readlines():
    words = line.split()
    n = int(words[0].replace(",", ""))
    values = []
    for word in words[1:]:
        values.append(float(word.replace(",", "")))
    if averages.get(n) is None:
        averages[n] = {
            'values': values,
            'samples': 1
        }
    else:
        values_averages = averages[n]['values']
        samples = averages[n]['samples']
        for i in range(0, len(values_averages)):
            values_averages[i] *= samples
            values_averages[i] += values[i]
            values_averages[i] /= samples+1
        averages[n]['samples'] += 1

file.close()
file = open(sys.argv[1], 'w')

for n, averages in averages.items():
    file.write(str(n))
    for average in averages['values']:
        file.write('\t' + str(average))
    file.write('\n')

file.close()
```

Recorte 3: calculo_media.py

2.2 Gráfica de ejecución de los programas "slow" y "fast"

Tras ejecutar nuestras pruebas, cálculo de medias y generación de gráficas automáticas obtenemos la siguiente gráfica:

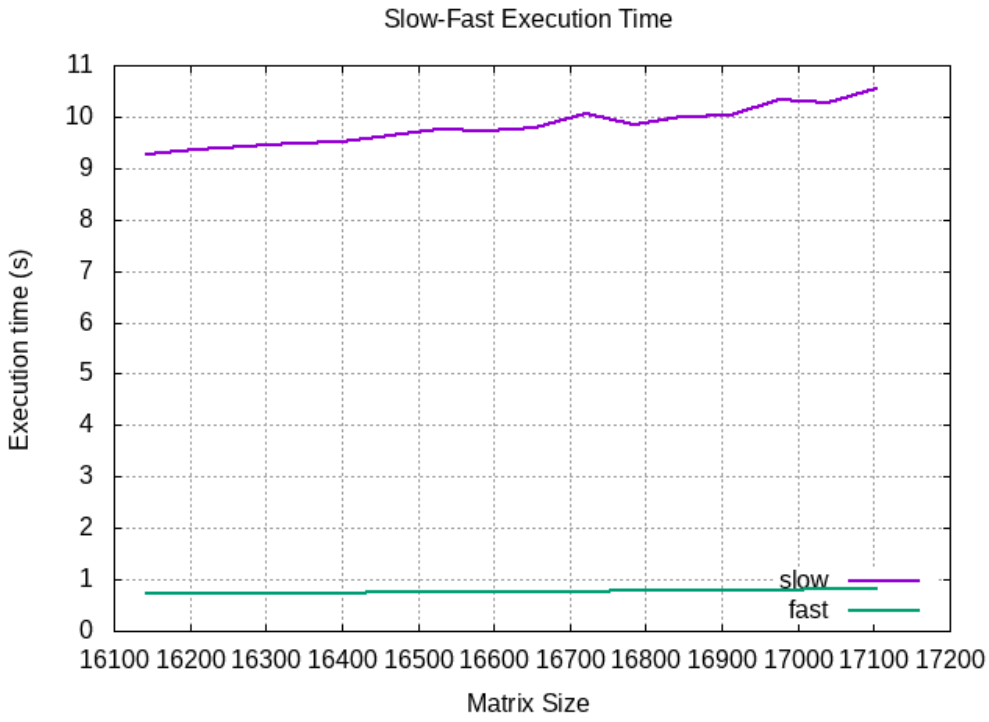

Figura 2: Gráfica time_slow_fast.png.

Podemos observar cómo el aumento del tamaño de la matriz aumenta el tiempo de computación de manera lineal, y mucho más acusadamente para el programa "slow" que para el programa "fast".

Esto se debe a que las matrices en memoria se almacenan como arrays de arrays, de manera que los elementos de cada fila se encuentran adyacentes en memoria, pero los elementos de cada columna se encuentran en zonas separadas de memoria, de manera que resulta mucho más eficiente recorrer las matrices por filas que por columnas.

Este efecto se manifiesta de manera más clara con tamaños de matriz más grandes, ya que con matrices de tamaños pequeños resulta prácticamente irrelevante el hecho de que la información de las columnas no sea adyacente entre sí.

3 EJERCICIO 2: TAMAÑO DE LA CACHÉ Y RENDIMIENTO

3.1 Ejecución de las pruebas

Para la realización de este ejercicio hemos usado la herramienta "cachegrind" para recolectar información sobre los fallos de memoria de los programas, usando la metodología explicada anteriormente [2,1]

3.2 Gráficas de fallos de escritura

Tras la ejecución automática de las pruebas y las gráficas obtenemos los siguientes resultados:

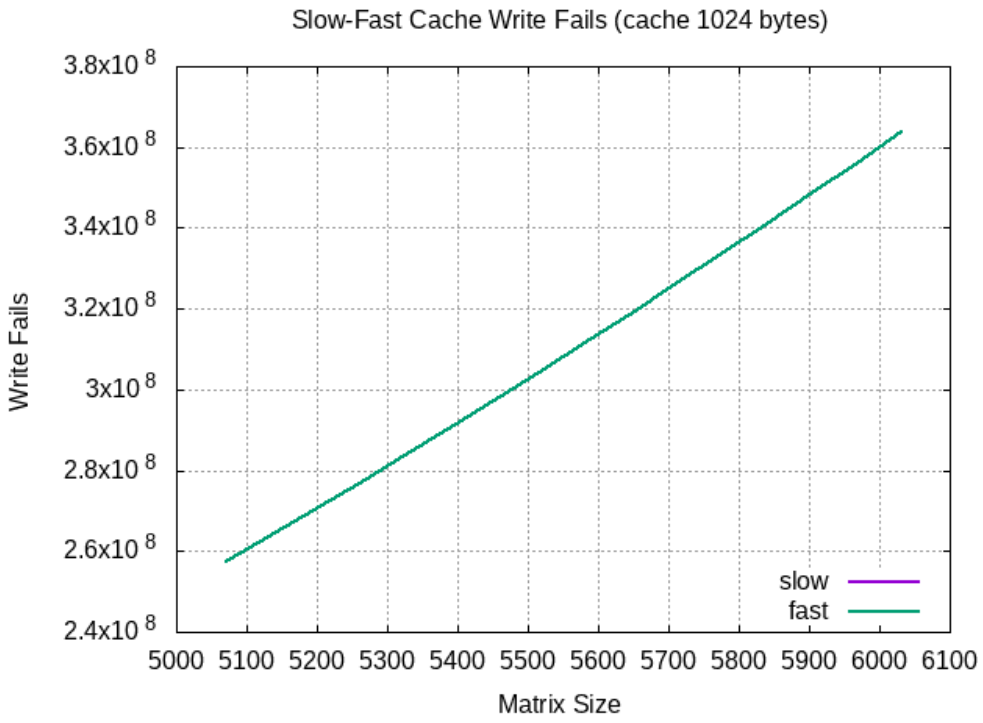

Figura 3: Gráfica de fallos de escritura para tamaño de caché de primer nivel 1024 bytes.

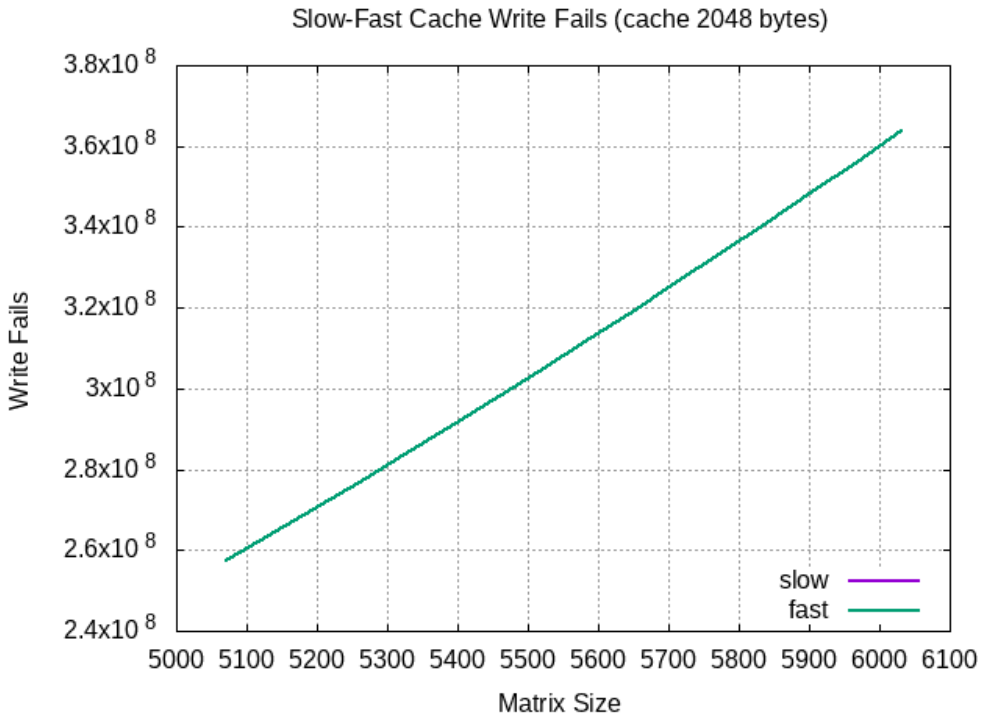

Figura 4: Gráfica de fallos de escritura para tamaño de caché de primer nivel 2048 bytes.

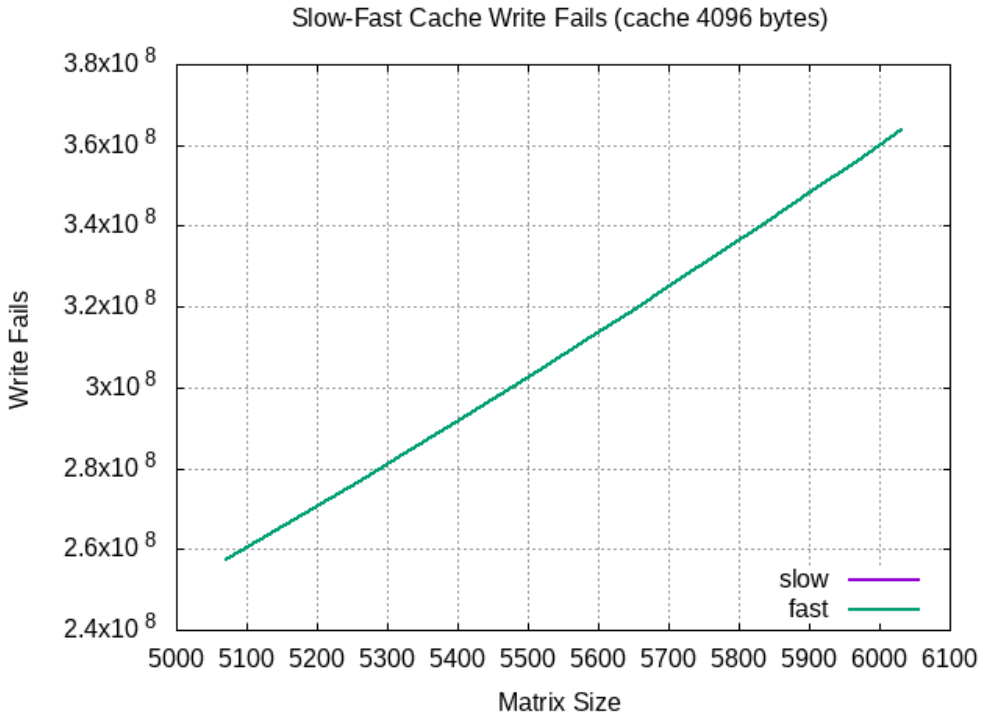

Figura 5: Gráfica de fallos de escritura para tamaño de caché de primer nivel 4096 bytes.

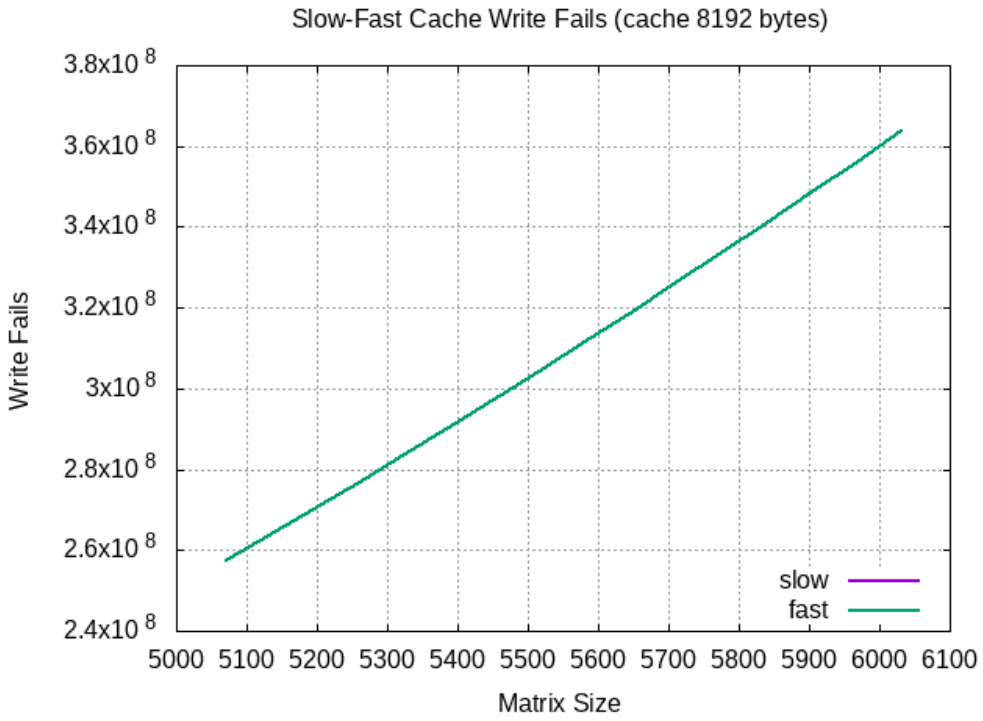

Figura 6: Gráfica de fallos de escritura para tamaño de caché de primer nivel 8192 bytes.

El principal efecto observado sobre las gráficas es el hecho de que los fallos de escritura en la memoria caché son proporcionales al tamaño de las matrices, creciendo de manera lineal para tamaños grandes.

3.3 Gráficas de fallos de lectura

Tras la ejecución automática de las pruebas y las gráficas obtenemos los siguientes resultados:

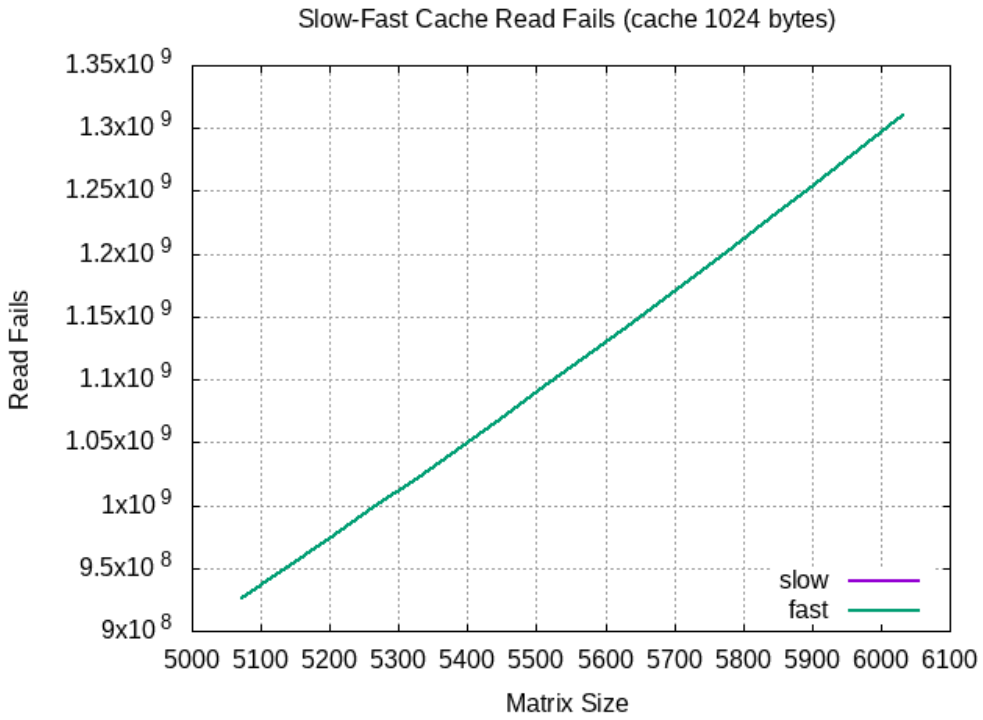

Figura 7: Gráfica de fallos de lectura para tamaño de caché de primer nivel 1024 bytes.

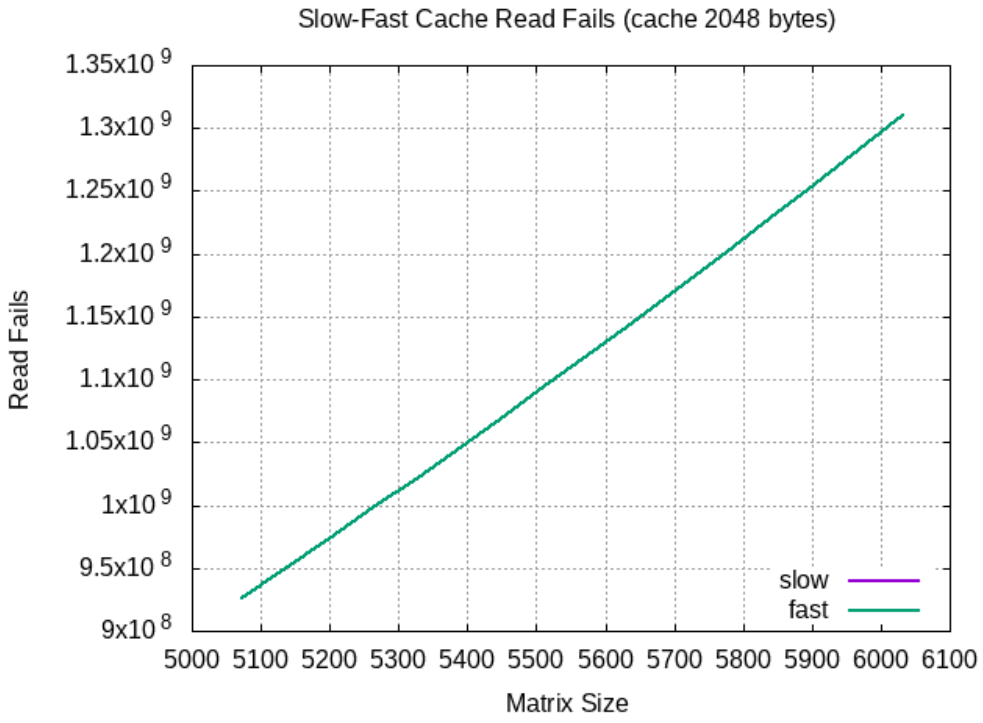

Figura 8: Gráfica de fallos de lectura para tamaño de caché de primer nivel 2048 bytes.

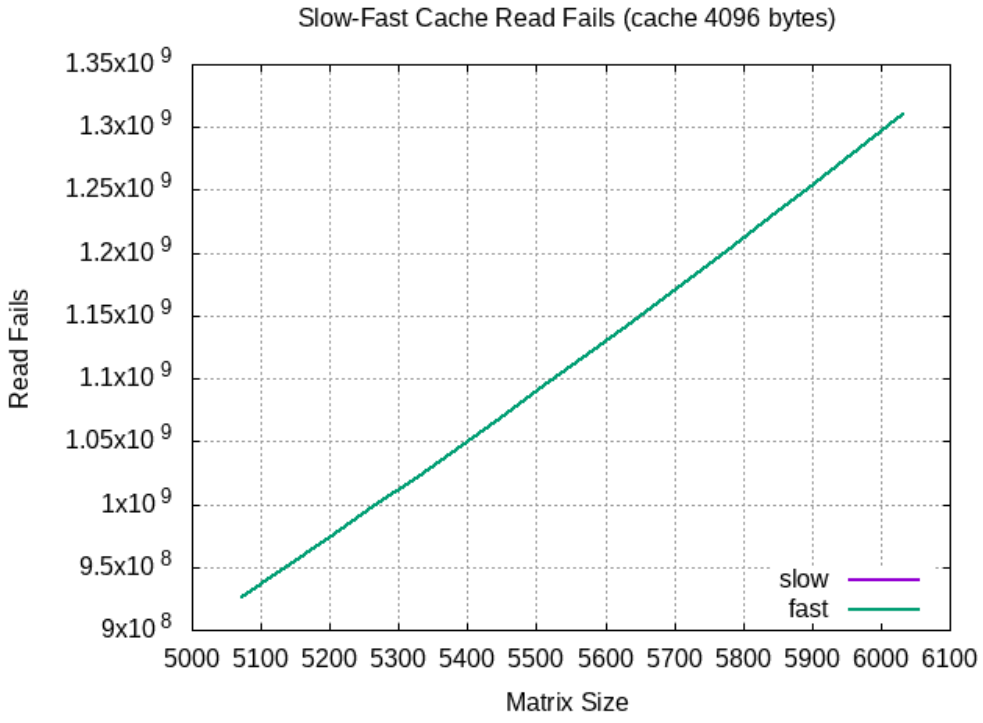

Figura 9: Gráfica de fallos de lectura para tamaño de caché de primer nivel 4096 bytes.

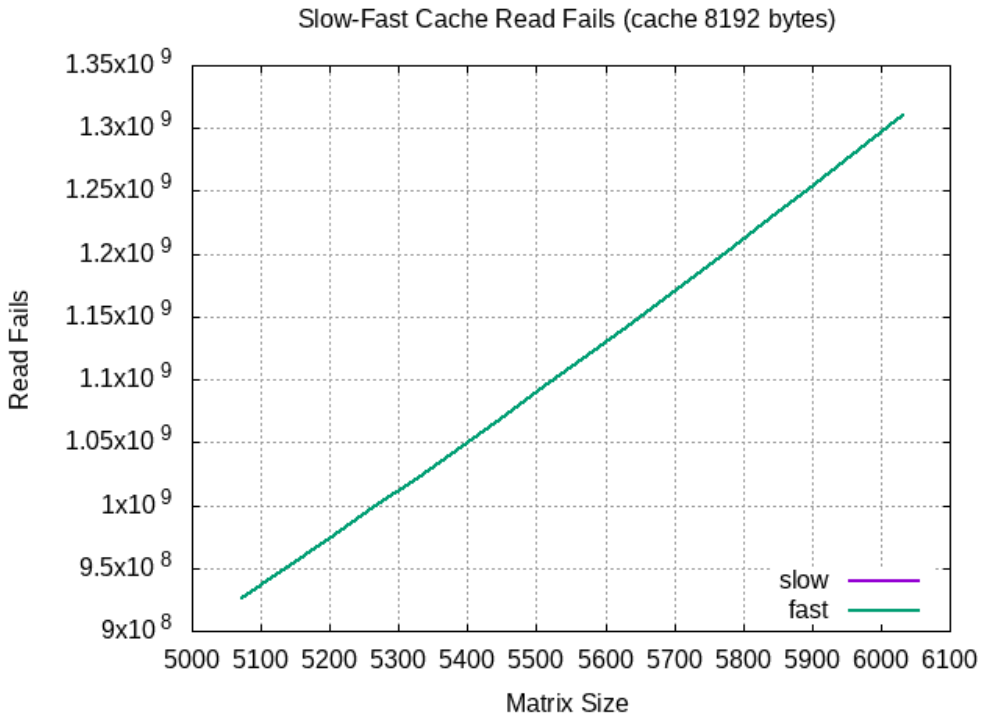

Figura 10: Gráfica de fallos de lectura para tamaño de caché de primer nivel 8192 bytes.

Observamos que los fallos en la memoria caché, tanto de lectura como de escritura, son proporcionales al tamaño de las matrices, creciendo de manera lineal para tamaños grandes.

También podemos observar que los fallos de lectura y escritura de la caché para los programas "slow" y "fast" resultan prácticamente idénticos (aunque no exactamente idénticos). Hemos obtenido estos datos tras muchas simulaciones en diferentes equipos, tanto con la herramienta "cachegrind" como con la herramienta "callgrind", y concluimos que esto se debe de deber a algún tipo de fallo en nuestra metodología[2,1] o en la ejecución de las pruebas en sí.

4 EJERCICIO 3: CACHÉ Y MULTIPLICACIÓN DE MATRICES

4.1 Algoritmo

Pese a existir algoritmos de multiplicación de matrices significativamente más eficientes [1], hemos decidido implementar una algoritmo que se limita a recorrerse ambas matrices, multiplicando y sumando números para obtener el resultado de la multiplicación de matrices (similar a como lo haríamos a mano). De esta manera la complejidad computacional del algoritmo es de $O(N^3)$, siendo N el tamaño de las matrices.

Para el programa que calcula primero la transpuesta de la matriz usamos otra matriz auxiliar para almacenarla, pese a que podríamos haberlo realizado con tan solo $O(1)$ memoria adicional, realizando "swaps" dentro de la propia matriz.

4.2 Realización de las pruebas

Para la realización de las pruebas hemos usado scripts de generación de ficheros, medias y gráficas de manera automática tal y como explicamos en la sección 2.1.

4.3 Gráficas de tiempo de ejecución

Tras las realización de las pruebas obtenemos la siguiente gráfica:

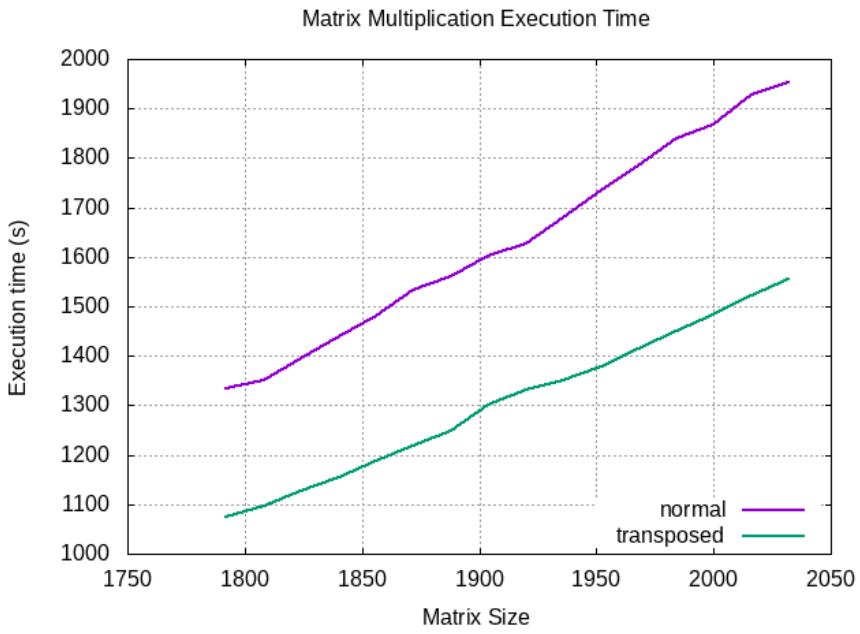

Figura 11: Gráfica de tiempo de ejecución de ambos programas de multiplicación de matrices.

Como podemos observar, el tiempo de ejecución del programa que calcula primero la transpuesta de la matriz es sistemáticamente más rápido, aunque necesite copiar uno a uno los valores de una matriz en otra. Tambien podemos observar que el tiempo de ejecución aumenta en mayor medida en el programa normal, en contraposición del transpuesto, que crece más levemente.

4.4 Gráficas de fallos de memoria

Tras las realización de las pruebas automáticas obtenemos la siguiente gráfica:

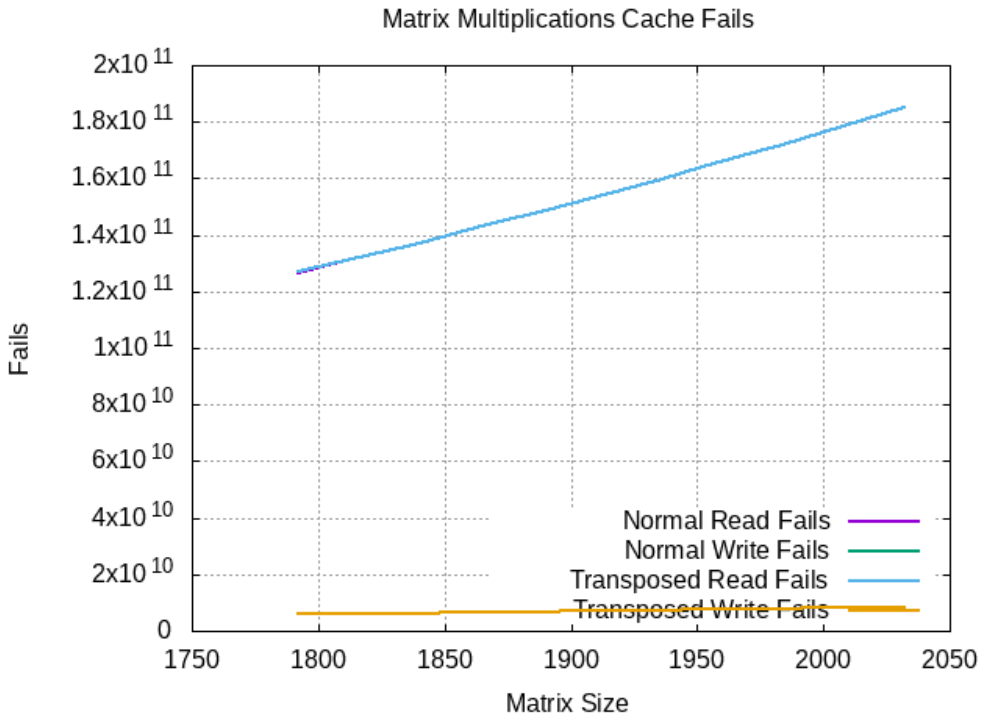

Figura 12: Gráfica de fallos de caché de ambos programas de multiplicación de matrices.

Podemos observar que los fallos de lectura son mucho más acusados que los de escritura, ya que mientras los de escritura se mantienen prácticamente constantes, los de lectura crecen de manera lineal.

También destaca el hecho de que la cantidad de fallos en memoria caché es prácticamente idéntico entre programas, como ya observamos anteriormente [3,3].

5 EJERCICIO 4

Para la realización del ejercicio 4 hemos decidido ejecutar la multiplicación de matrices con diferentes tamaños de caché, de una manera similiar a la expuesta en el ejercicio 2 [3,1], en la que tambien recogemos información sobre los tiempos de ejecución, para los diferentes tamaños de matrices y caché.

5.1 Gráficas de tiempos de ejecución

Estos son los tiempos de ejecución obtenidos tras la realización de las pruebas:

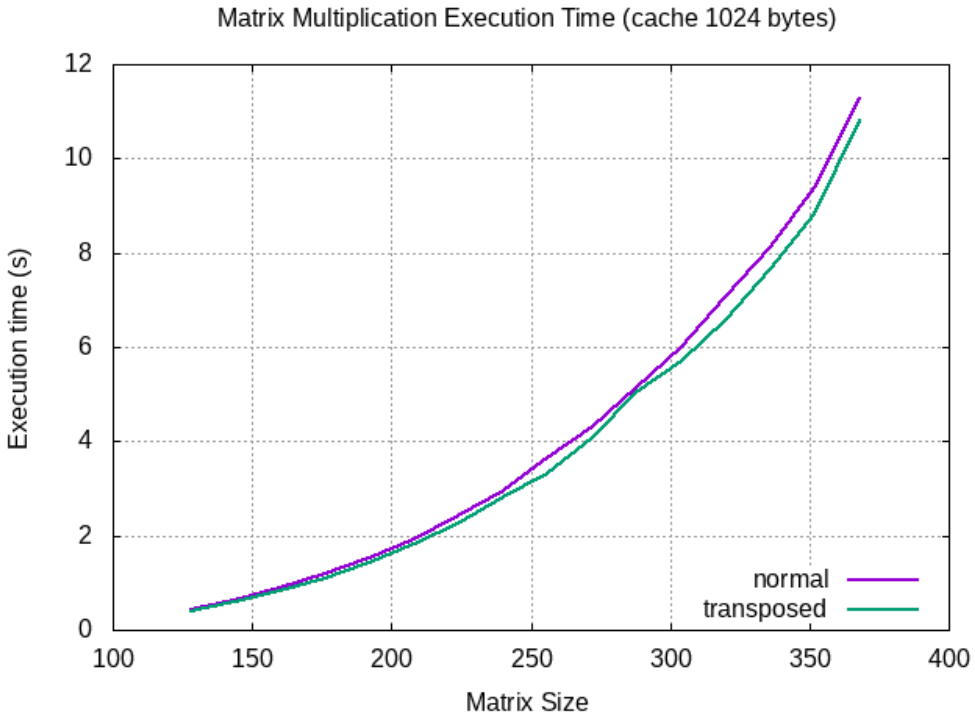

Figura 13: Gráfica de tiempo de ejecución para tamaño de caché de primer nivel 1024 bytes.

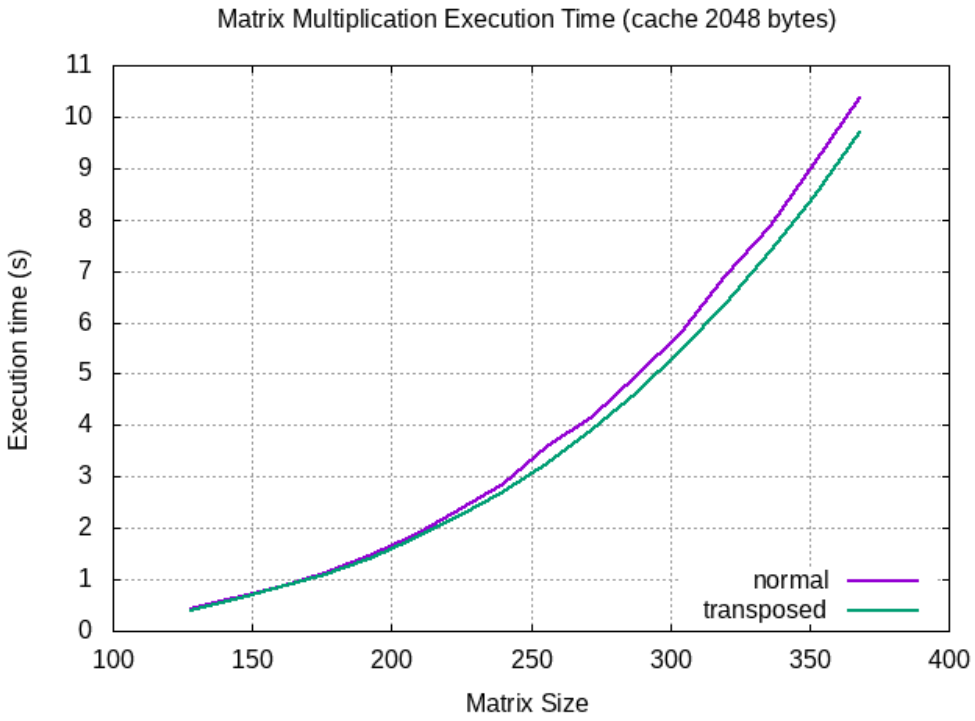

Figura 14: Gráfica de tiempo de ejecución para tamaño de caché de primer nivel 2048 bytes.

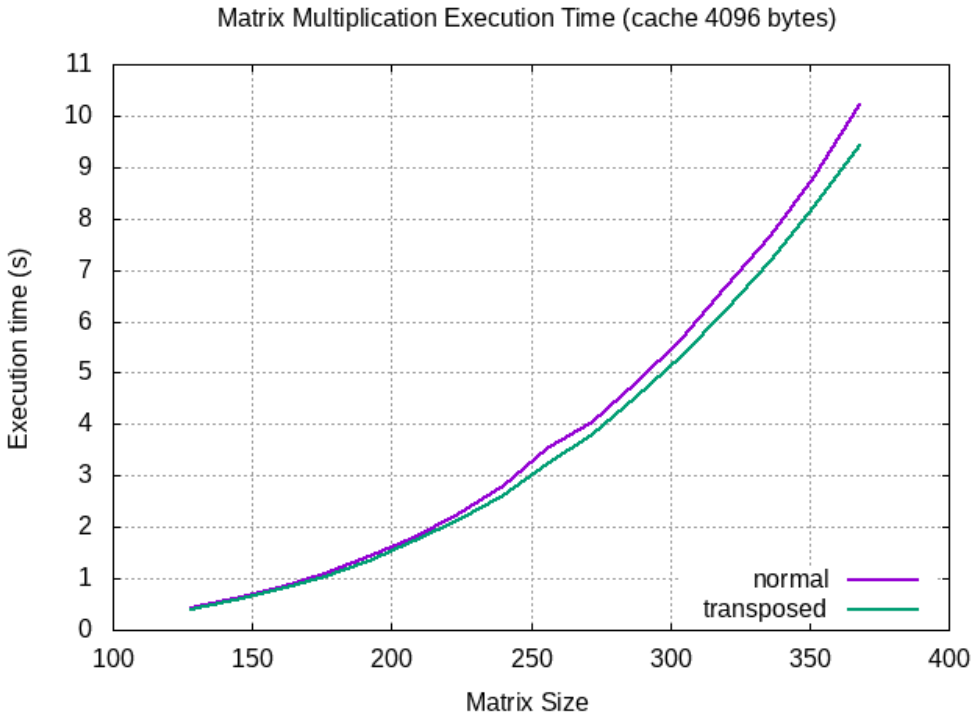

Figura 15: Gráfica de tiempo de ejecución para tamaño de caché de primer nivel 4096 bytes.

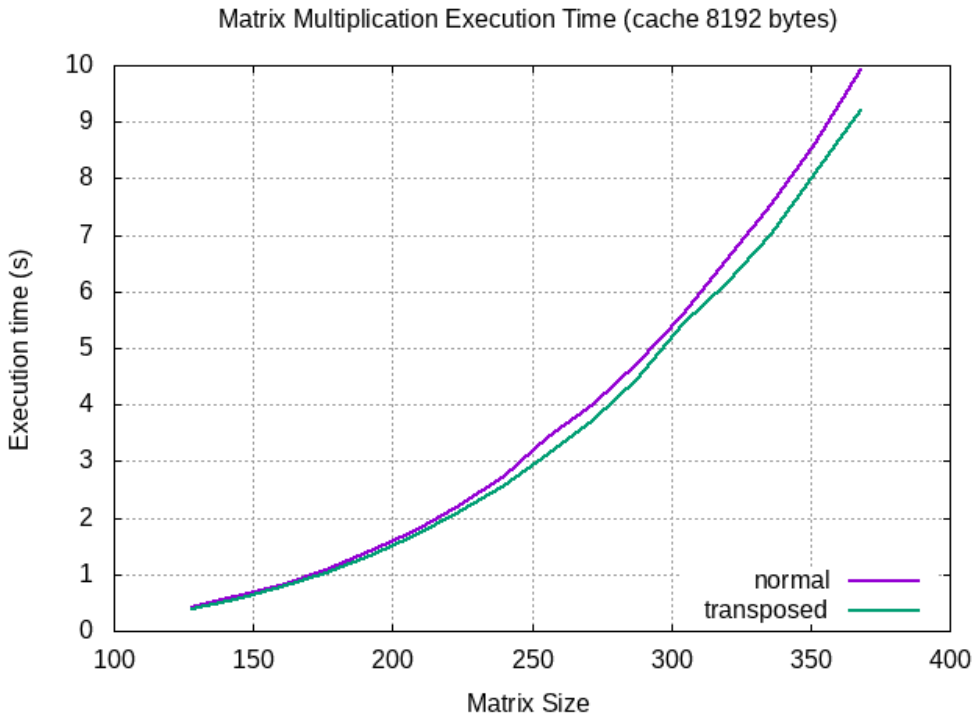

Figura 16: Gráfica de tiempo de ejecución para tamaño de caché de primer nivel 8192 bytes.

Los tiempos de ejecución de ambos programas son similares para todos los casos, creciendo de manera polinómica con el tamaño de la matriz, como ya explicamos en el ejercicio 3 [4,1]. Sí que es notable el hecho de que ambos programas se ejecutan más rápido conforme más tamaño de caché se configura para cachegrind, y que el programa que realiza primero la transposición de la matriz es constantemente más rápido que el que no la hace.

5.2 Gráficas de fallos de lectura y escritura

Las gráficas obtenidas para la información de los fallos de caché tras la ejecución de los tests son estas:

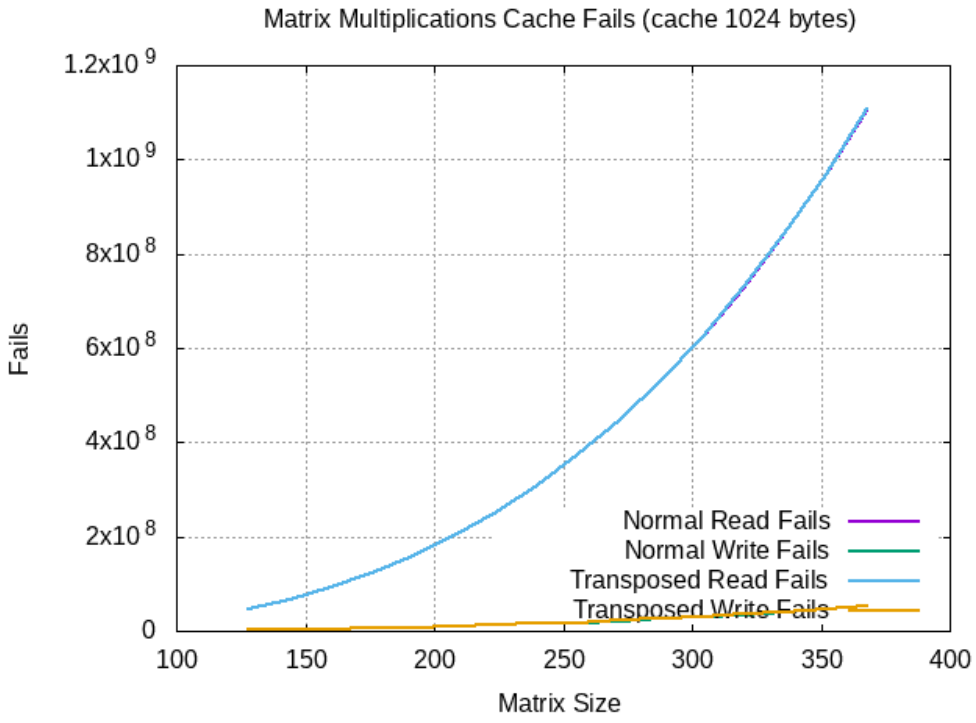

Figura 17: Gráfica de fallos de caché para tamaño de caché de primer nivel 1024 bytes.

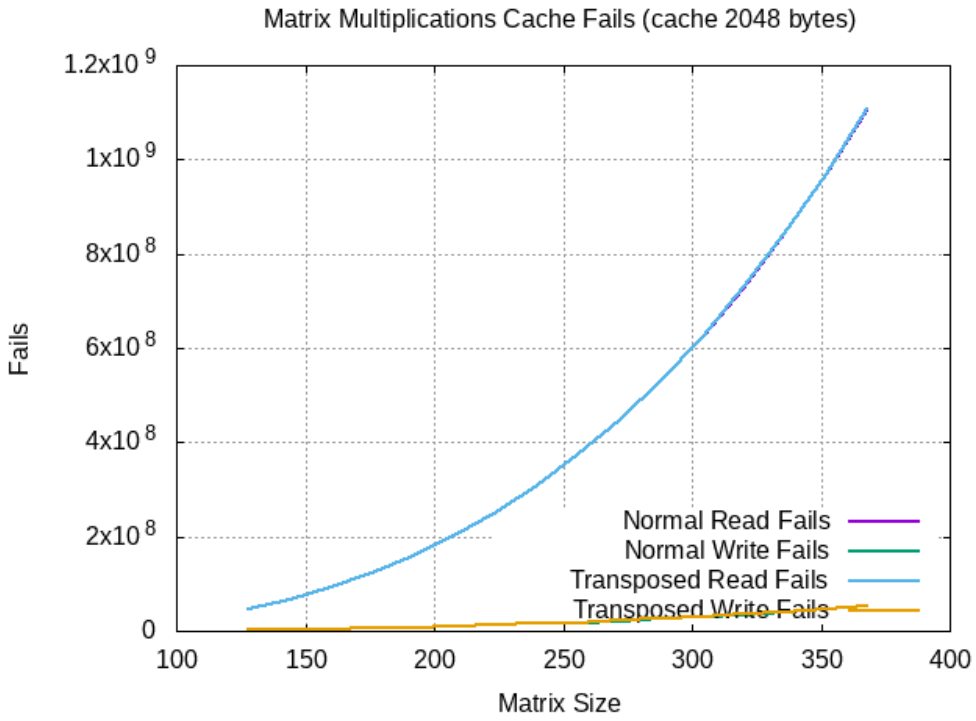

Figura 18: Gráfica de fallos de caché para tamaño de caché de primer nivel 2048 bytes.

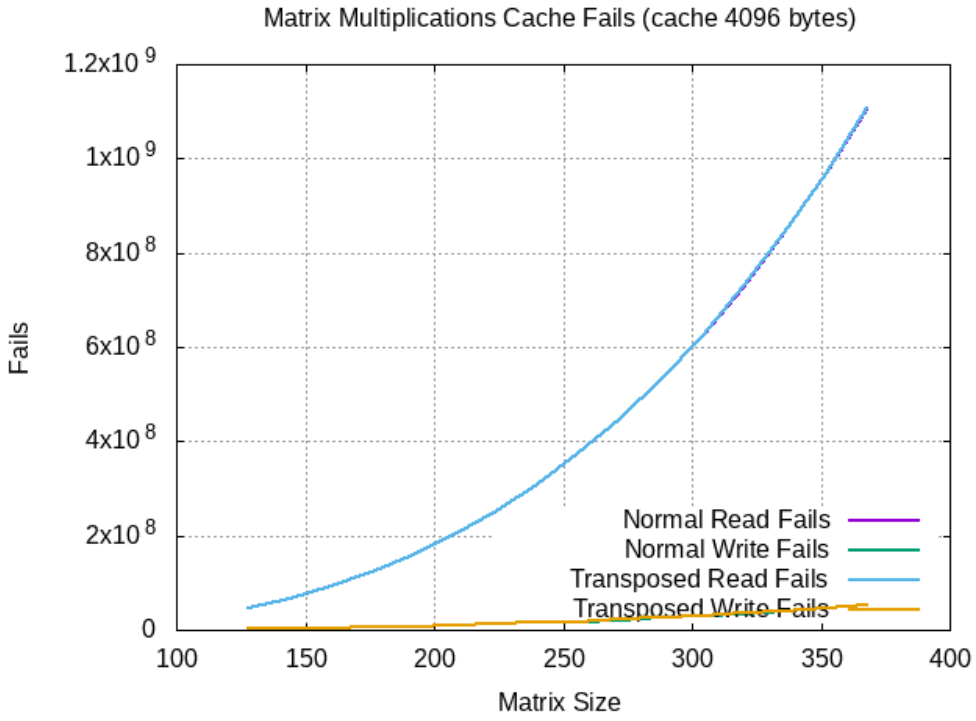

Figura 19: Gráfica de fallos de caché para tamaño de caché de primer nivel 4096 bytes.

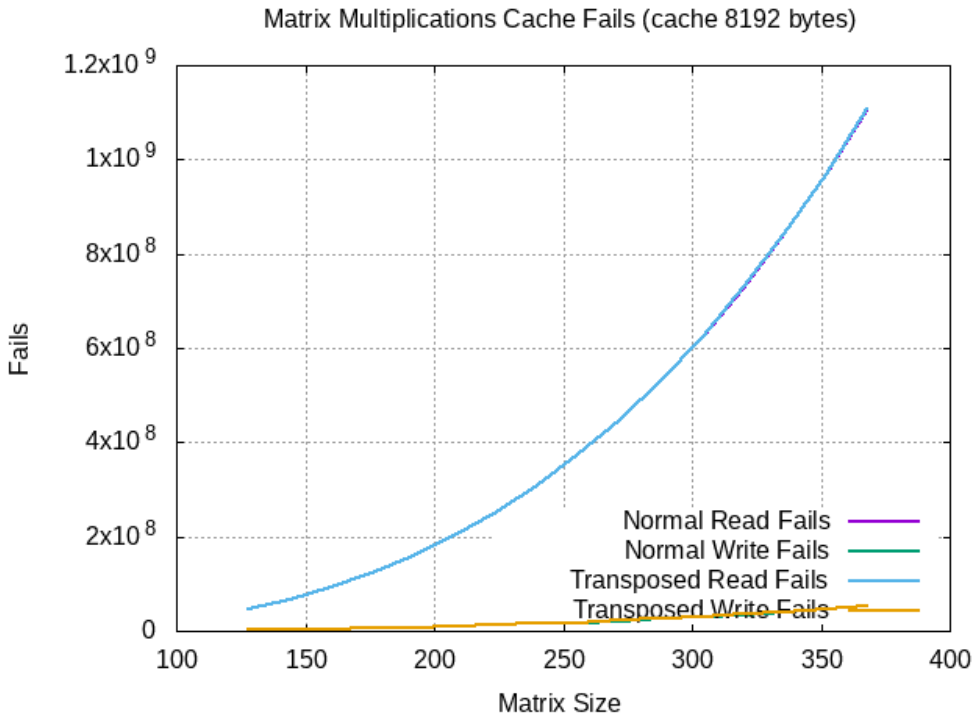

Figura 20: Gráfica de fallos de caché para tamaño de caché de primer nivel 8192 bytes.

Las gráficas muestran que los fallos de escritura son mucho menores a los de lectura independientemente del tamaño de la caché de nivel 1. Así mismo los fallos de escritura crecen de manera lineal con el tamaño de la matriz, mientras que los de lectura lo hacen de manera polinómica.

Observamos una problemática parecida a la que ya nos encontramos en el ejercicio 2[3,3], y es que ambos programas poseen gráficas extrañamente similares.

REFERENCIAS

[1] François Le Gall. Powers of Tensors and Fast Matrix Multiplication. *arXiv e-prints*, art. arXiv:1401.7714, Jan 2014.