MEMORIA P3

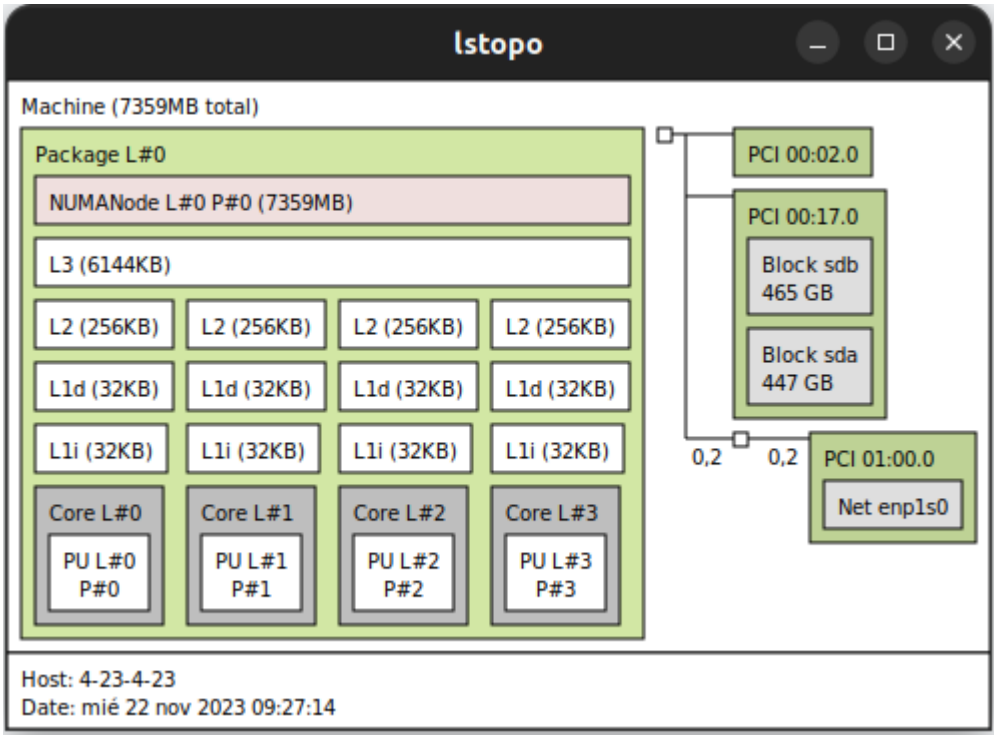
Daniel Aquino Santiago
Jorge Paniagua Moreno

Ejercicio 0: información sobre la caché del sistema

Resultado tras ejecutar: "*getconf -a | grep -i cache*"

```
e458432@4-23-4-23:~$ 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                  6291456
LEVEL3_CACHE_ASSOC                 12
LEVEL3_CACHE_LINESIZE              64
LEVEL4_CACHE_SIZE                  0
LEVEL4_CACHE_ASSOC
LEVEL4_CACHE_LINESIZE
```

Ejecutando el comando "lstopo" obtenemos:



1. **LEVEL1_ICACHE_SIZE:** Tamaño de la caché de instrucciones de nivel 1 (L1) en bytes. En este caso, es 32 KB (32768 bytes).

2. **LEVEL1_ICACHE_ASSOC:** Asociatividad de la caché de instrucciones de nivel 1 (L1).
3. **LEVEL1_ICACHE_LINESIZE:** Tamaño de línea de la caché de instrucciones de nivel 1 (L1) en bytes. En este caso, es 64 bytes.
4. **LEVEL1_DCACHE_SIZE:** Tamaño de la caché de datos de nivel 1 (L1) en bytes. También es de 32 KB (32768 bytes).
5. **LEVEL1_DCACHE_ASSOC:** Asociatividad de la caché de datos de nivel 1 (L1). En este caso, son 8 vías.
6. **LEVEL1_DCACHE_LINESIZE:** Tamaño de línea de la caché de datos de nivel 1 (L1) en bytes. Igualmente, es 64 bytes.
7. **LEVEL2_CACHE_SIZE:** Tamaño de la caché de nivel 2 (L2) en bytes. Específica 262144 bytes, lo que equivale a 256 KB.
8. **LEVEL2_CACHE_ASSOC:** Asociatividad de la caché de nivel 2 (L2). Aquí, hay 4 vías.
9. **LEVEL2_CACHE_LINESIZE:** Tamaño de línea de la caché de nivel 2 (L2) en bytes. Igualmente, es 64 bytes.
10. **LEVEL3_CACHE_SIZE:** Tamaño de la caché de nivel 3 (L3) en bytes. Específica 6291456 bytes, lo que equivale a 6 MB.
11. **LEVEL3_CACHE_ASSOC:** Asociatividad de la caché de nivel 3 (L3). En este caso, son 12 vías.
12. **LEVEL3_CACHE_LINESIZE:** Tamaño de línea de la caché de nivel 3 (L3) en bytes. Nuevamente, es 64 bytes.
13. **LEVEL4_CACHE_ASSOC:** Asociatividad de la caché de nivel 4 (L4).
14. **LEVEL4_CACHE_LINESIZE:** Tamaño de línea de la caché de nivel 4 (L4).

Observaciones:

- Hay separación entre las cachés de datos e instrucciones en el nivel 1 (L1), donde L1_ICache es la caché de instrucciones y L1_DCache es la caché de datos.
- Los tipos de caché son principalmente de tipo set-associative, donde el número de vías determina la asociatividad. Por ejemplo, L1_DCache es 8-way set-associative, lo que significa que cada conjunto tiene 8 líneas de caché entre las cuales se puede seleccionar para almacenar datos. Este es un diseño común para equilibrar el rendimiento y la complejidad de la caché.

Ejercicio 1: Memoria caché y rendimiento

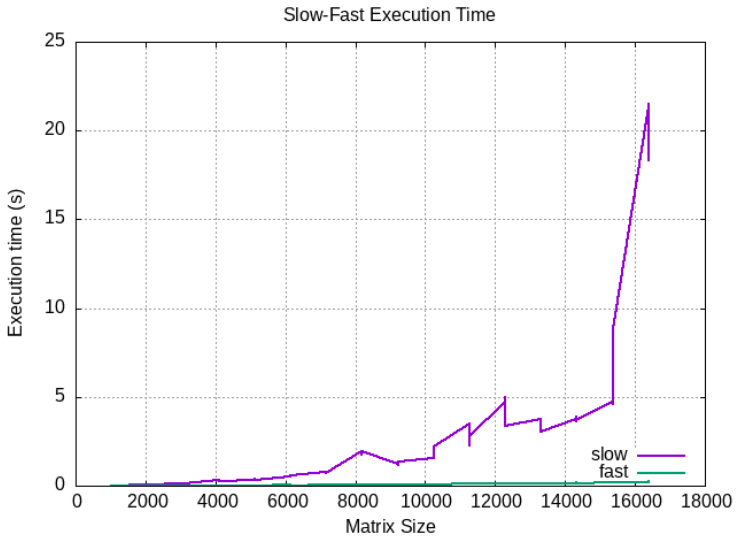
1) Se ha implementado un programa en bash que genera la media de el valor del tiempo de ejecución:

```
fDAT=time_slow_fast.dat

awk '{ sum += $3 } END { if (NR > 0) print sum / NR }' $fDAT
```

y el resultado obtenido es:

```
~/Do/i/t/p/ar/p/P3_1323_P11  bash slow_fast_avg.sh
2.93914
```

2) A grandes rasgos, realizar múltiples tomas de medidas de rendimiento es fundamental para obtener resultados más confiables y significativos. Proporciona una visión más completa del rendimiento del programa, teniendo en cuenta la variabilidad y los posibles factores aleatorios que pondrían afectar una única ejecución
3) Se ha generado el archivo correctamente.
4)



5) El código diseñado añadiendo el loop de limit 10:

```
for ((N = Ninicio ; N <= Nfinal ; N += Ninicio)); do
    for ((I = 0; I <= 10; I++)); do
        echo "N: $N / $Nfinal..."

        # ejecutar los programas slow y fast consecutivamente con tamaño de matriz N
        # para cada uno, filtrar la línea que contiene el tiempo y seleccionar la
        # tercera columna (el valor del tiempo). Dejar los valores en variables
        # para poder imprimirlos en la misma línea del fichero de datos
        slowTime=$(./slow $N | grep 'time' | awk '{print $3}')
        fastTime=$(./fast $N | grep 'time' | awk '{print $3}')

        echo "$N    $slowTime   $fastTime" >> $fDAT
    done
done
```

El loop secundario cuenta de un contador de 0 a 9, el cual es envuelto por otro contador que ejecuta los programas slow y fast, para tamaños que empiezan en N = 1024 y terminan en N = 16384, sumando 1024 en cada iteración.

En matrices pequeñas, la influencia de la caché puede hacer que las diferencias en el patrón de acceso sean menos significativas, y ambas versiones pueden tener tiempos de ejecución similares. Sin embargo, a medida que el tamaño de la matriz

aumenta, la diferencia en cómo se accede a los datos pueden contribuir a una divergencia en los tiempos de ejecución.

En slow.c se accede a la matriz transpuesta utilizando ‘matrix[j][i]’, lo que implica acceder a elementos por columnas. Esto es menos eficientes si la matriz está almacenada por filas

En fast.c, se accede a la matriz de manera convencional utilizando ‘matrix[i][j]’

Ejercicio 2: Tamaño de la caché y rendimiento

Se ha implementado “slow_fast_fallos_cache_variante..sh” que genera la información de cachegrind la cual se procesa con “cg_annotate”, obteniendo respectivamente D1mr y D1mw, variando como se explica el enunciado los tamaños de caché y de N, siendo N la dimension de la matriz a multiplicar por los programas slow.c y fast.c

```
#!/bin/bash

# Inicializar variables
Ninicio=1024
Npaso=1024
Nfinal=5120
P=4
CACHE_SIZES=(1024 2048 4096 8192)  # Tamaños de caché en Bytes
CACHE_SIZE_UPPER=8388608  # 8 MBytes
POSICION_D1MR=9
POSICION_D1MW=15

# Bucle para N desde P hasta Q
for ((N = Ninicio + 128 * P; N <= 5120 + 128 * P; N += Npaso)); do
    for CACHE_SIZE in "${CACHE_SIZES[@]}"; do
        OUTPUT_FILE="cache_${CACHE_SIZE}.dat"

        # Escribir el valor de N en el archivo
        echo -n "$N " >> $OUTPUT_FILE

        for PROGRAM in "slow" "fast"; do
            # Ejecutar Valgrind con cachegrind y guardar la salida en un archivo temporal
            valgrind --tool=cachegrind --I1=$CACHE_SIZE,1,64 --D1=$CACHE_SIZE,1,64 --LL=$CACHE_SIZE_UPPER,1,64 --cachegrind-out-file=pr

            echo ""
            echo "CG_ANNOTATE:"
            echo ""
            cg_annotate pr_out.dat | head -n 20

            LINE=$(cg_annotate pr_out.dat | head -n 20| awk 'NR==18 {print}')
            D1MR=$(echo "$LINE" | awk -v col="$POSICION_D1MR" '{print $col}')
            D1MW=$(echo "$LINE" | awk -v col="$POSICION_D1MW" '{print $col}')

            # Escribir los valores en el archivo
            echo -n "$D1MR $D1MW " >> $OUTPUT_FILE

            rm pr_out.dat
        done

        # Agregar un salto de línea al final de la línea después de escribir los resultados para cada conjunto de caché
        if [ "$((N + Npaso))" -le "$((5120 + 128 * P))" ]; then
            echo "" >> "$OUTPUT_FILE"
        fi

        sed -i 's/,//g' "$OUTPUT_FILE"
    done
done
```

En esta parte se recoge los valores de D1mr y D1mw de cada salida, los cuales se encuentran respectivamente en la posición 9 y 15 de la fila 18. Se insertan ambos valores separados por un espacio en el archivo “`cache_${CACHE_SIZE}.dat`” de cada ejecución, primero para el programa slow y después para el programa fast. Cuando se han insertado

ambos valores, se inserta un salto de línea si no hemos llegado a la última ejecución para cada "`cache_${CACHE_SIZE}.dat`".

Finalmente, para cada archivo, se eliminan las comas, para no generar gráficas erróneas a la hora de crear las gráficas.

comando valgrind completo en "slow_fast_avg.sh":

```
valgrind --tool=cachegrind --I1=$CACHE_SIZE,1,64 --D1=$CACHE_SIZE,1,64
--LL=$CACHE_SIZE_UPPER,1,64 --cachegrind-out-file=pr_out.dat ./$PROGRAM
$N
```

Por último, se generan las gráficas con Gnuplot:

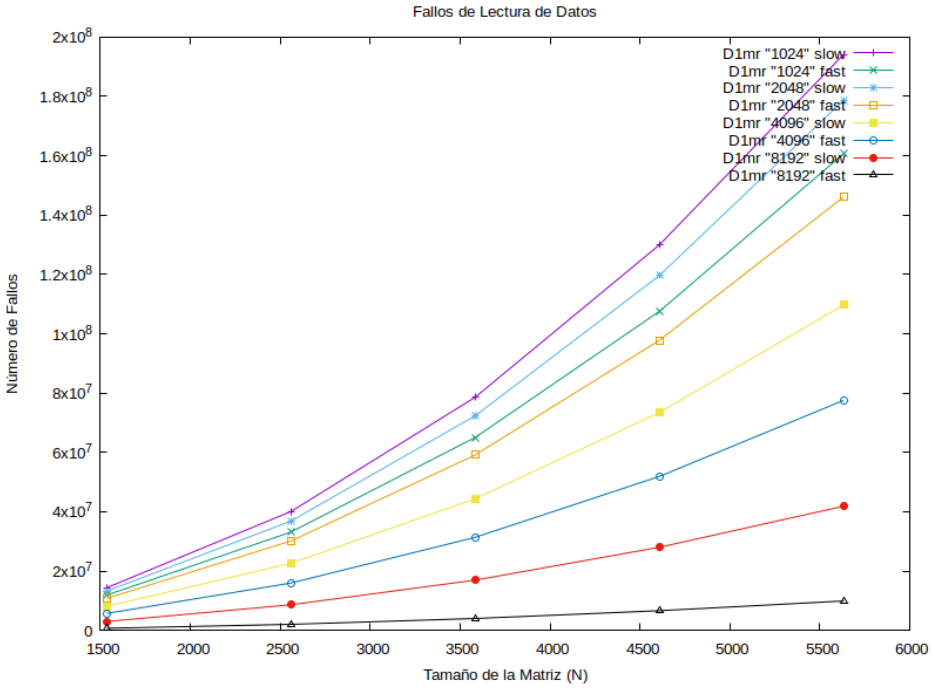
```
# Llamar a gnuplot para generar el gráfico y pasarle directamente por la entrada
# estándar el script que está entre "<< END_GNUPLOT" y "END_GNUPLOT"
gnuplot << END_GNUPLOT

# Configuración del terminal y salida de las gráficas
set terminal pngcairo enhanced font 'arial,10' size 800, 600
set output 'cache_lectura.png'

# Configuración de la primera gráfica (lectura de datos)
set title 'Fallos de Lectura de Datos'
set xlabel 'Tamaño de la Matriz (N)'
set ylabel 'Número de Fallos'
plot "cache_1024.dat" using 1:2 with linespoints title 'D1mr "1024" slow', \
     "cache_1024.dat" using 1:4 with linespoints title 'D1mr "1024" fast', \
     "cache_2048.dat" using 1:2 with linespoints title 'D1mr "2048" slow', \
     "cache_2048.dat" using 1:4 with linespoints title 'D1mr "2048" fast', \
     "cache_4096.dat" using 1:2 with linespoints title 'D1mr "4096" slow', \
     "cache_4096.dat" using 1:4 with linespoints title 'D1mr "4096" fast', \
     "cache_8192.dat" using 1:2 with linespoints title 'D1mr "8192" slow', \
     "cache_8192.dat" using 1:4 with linespoints title 'D1mr "8192" fast'

# Cambiar el nombre de salida para la segunda gráfica
set output 'cache_escritura.png'

# Configuración de la segunda gráfica (escritura de datos)
set title 'Fallos de Escritura de Datos'
set ylabel 'Número de Fallos'
plot "cache_1024.dat" using 1:3 with linespoints title 'D1mw "1024" slow', \
     "cache_1024.dat" using 1:5 with linespoints title 'D1mw "1024" fast', \
     "cache_2048.dat" using 1:3 with linespoints title 'D1mw "2048" slow', \
     "cache_2048.dat" using 1:5 with linespoints title 'D1mw "2048" fast', \
     "cache_4096.dat" using 1:3 with linespoints title 'D1mw "4096" slow', \
     "cache_4096.dat" using 1:5 with linespoints title 'D1mw "4096" fast', \
     "cache_8192.dat" using 1:3 with linespoints title 'D1mw "8192" slow', \
     "cache_8192.dat" using 1:5 with linespoints title 'D1mw "8192" fast'
quit
END_GNUPLOT
echo "Plots generated successfully."
```

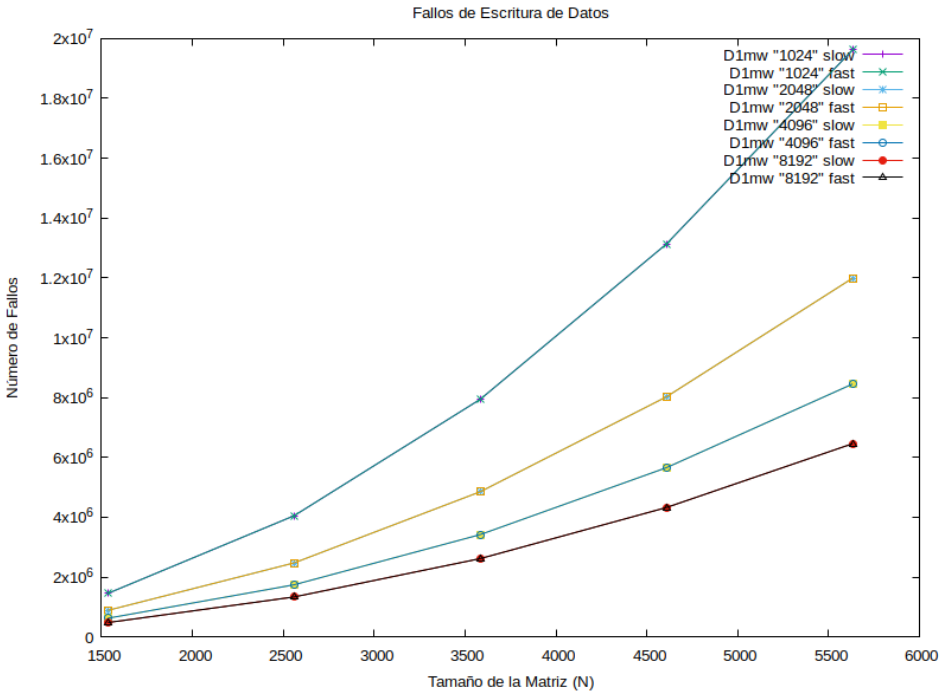
Primero se genera la gráfica “cache_lectura.png”.

Se inicializa el valor del título, del eje x y del eje y y finalmente se recogen los valores de todos los “.dat” de la columna 2 y la columna 4

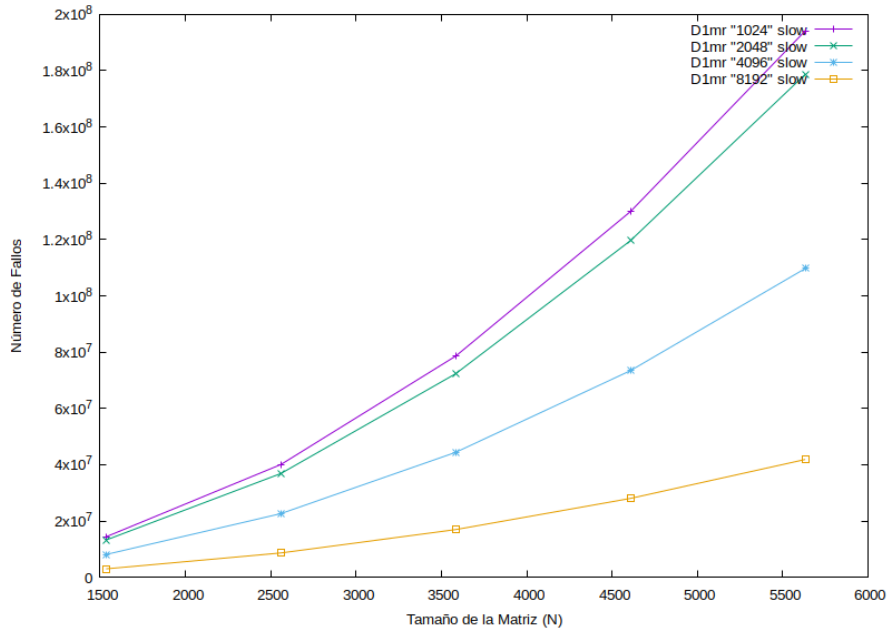
De igual manera se genera la gráfica “cache_escritura.png”.

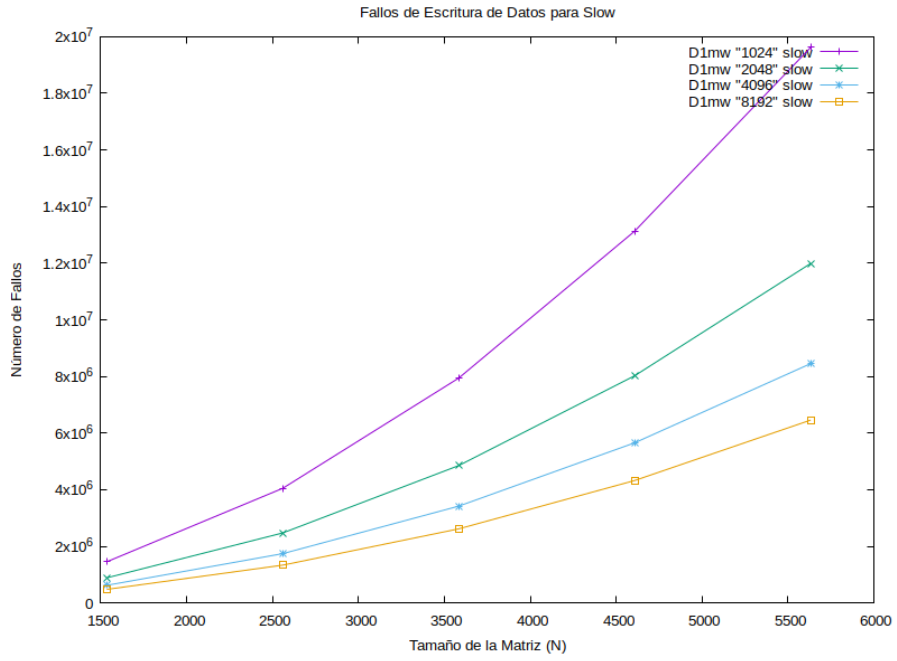
Los resultados son los siguientes:

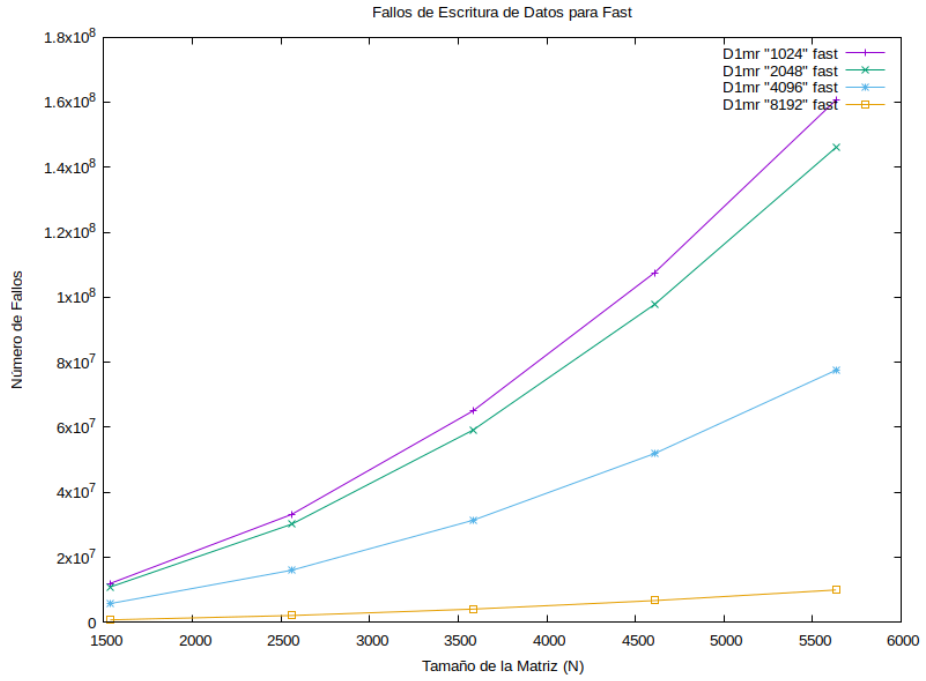


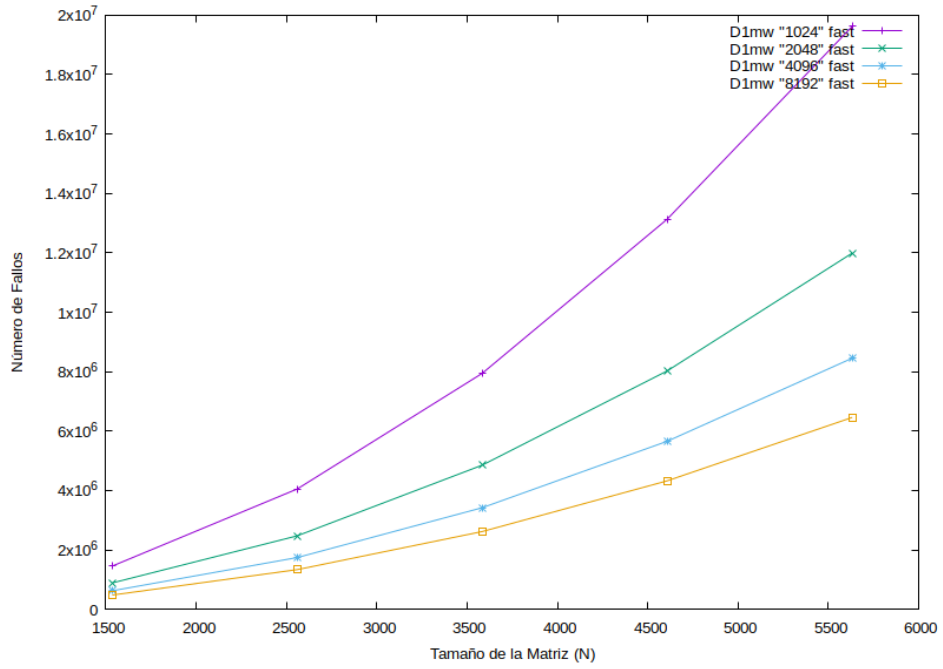


Por temas de comprensión, aunque no se especifica en el enunciado, hemos creado las gráficas donde se encuentran los valores de D1mr y D1mw de cada programa slow y fast.









¿Se observan cambios en la gráfica al variar los tamaños de caché para el programa slow?

Como podemos observar, para tamaños de caché mayores se producen menos fallos, produciéndose una gran diferencia entre los fallos que se generan para tamaño de caché igual a 1024 y para 8192. Esto puede deberse a que al ampliar el tamaño de la caché se genera más capacidad de almacenar datos en la memoria De igual manera, se observa que a medida que crece el tamaño de la matriz, se producen más fallos, tantos como de

escritura como de lectura en la caché. Esto se produce ya que el número de datos a almacenar se incrementa con respecto a sus ejecuciones anteriores con tamaños de matriz menor.

¿Y para el programa fast?

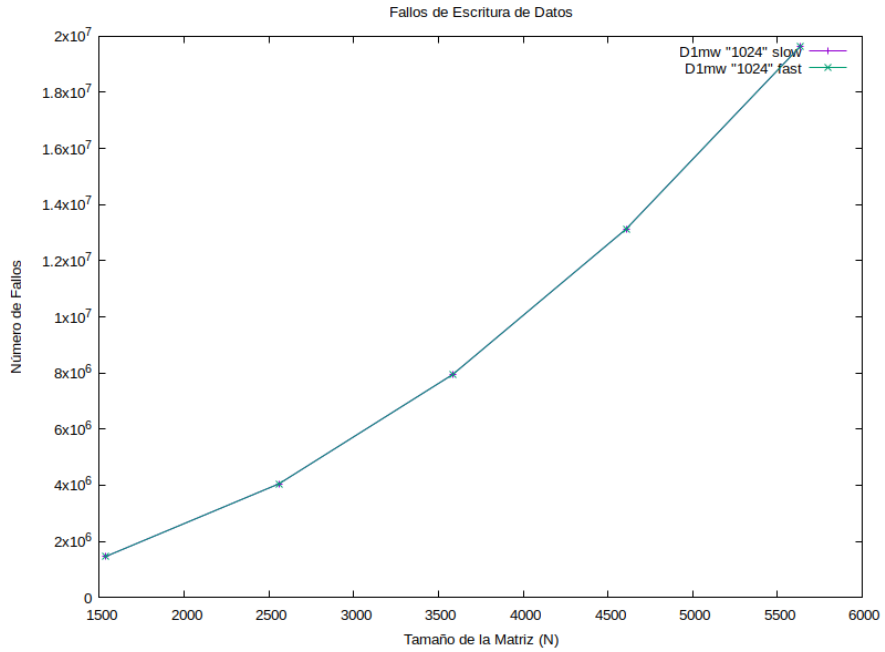
Se observan los mismos cambios que para el programa slow, pero el número de fallos se reducen considerablemente,llegando a observar muy pocos fallos de escritura para tamaños de caché de 8192

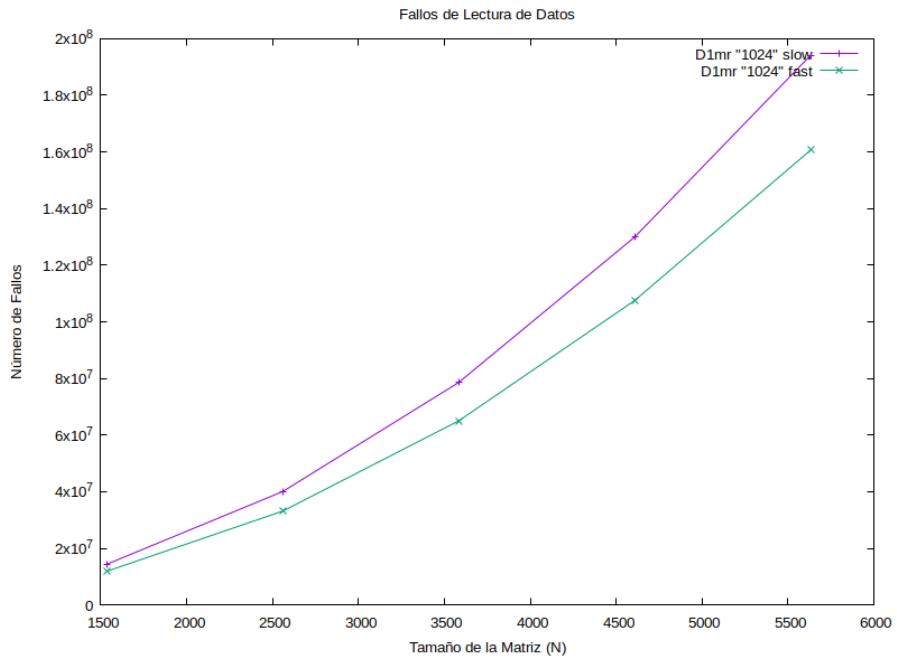
¿Varía la gráfica cuando se fija en un tamaño de caché concreto y compara el programa slow y fast?

Para ello se ha creado un nuevo script que funciona igual que el exterior pero sin modificar el tamaño de la cache: “slow_fast_cache_fija.sh”

```
#!/bin/bash

# Inicializar variables
Ninicio=1024
Npaso=1024
Nfinal=5120
P=4
CACHE_SIZE=1024 # Tamaño de caché en Bytes
CACHE_SIZE_UPPER=8388608  # 8 MBytes
POSICION_D1MR=9
POSICION_D1MW=15

# Bucle para N desde P hasta Q
for ((N = Ninicio + 128 * P; N <= 5120 + 128 * P; N += Npaso)); do

    OUTPUT_FILE="cache_${CACHE_SIZE}_fix.dat"
    # Escribir el valor de N en el archivo
    echo -n "$N " >> "$OUTPUT_FILE"

    for PROGRAM in "slow" "fast"; do
        # Ejecutar Valgrind con cachegrind y guardar la salida en un archivo temporal
        valgrind --tool=cachegrind --I1="$CACHE_SIZE",1,64 --D1="$CACHE_SIZE",1,64 --LL="$CACHE_SIZE_UPPER",1,64 --cachegrind-out-file=pr_out.dat "./$PROGRAM" "$N"

        echo ""
        echo "CG_ANNOTATE:"
        echo ""
        cg_annotate pr_out.dat | head -n 20

        LINE=$(cg_annotate pr_out.dat | awk 'NR==18 {print}')
        D1MR=$(echo "$LINE" | awk -v col="$POSICION_D1MR" '{print $col}')
        D1MW=$(echo "$LINE" | awk -v col="$POSICION_D1MW" '{print $col}')

        # Escribir los valores en el archivo
        echo -n "$D1MR $D1MW " >> "$OUTPUT_FILE"

        rm pr_out.dat
    done

    # Agregar un salto de línea al final de la línea después de escribir los resultados para cada conjunto de caché
    if [ "$((N + Npaso))" -le "$((5120 + 128 * P))" ]; then
        echo "" >> "$OUTPUT_FILE"
    fi

    sed -i 's/,//g' "$OUTPUT_FILE"
done
```

```
echo "Generating plot..."
# Llamar a gnuplot para generar el gráfico y pasarle directamente por la entrada
# estándar el script que está entre "<< END_GNUPLOT" y "END_GNUPLOT"
gnuplot << END_GNUPLOT

# Configuración del terminal y salida de las gráficas
set terminal pngcairo enhanced font 'arial,10' size 800, 600
set output 'cache_lectura_fix.png'

# Configuración de la primera gráfica (lectura de datos)
set title 'Fallos de Lectura de Datos'
set xlabel 'Tamaño de la Matriz (N)'
set ylabel 'Número de Fallos'
plot "cache_1024_fix.dat" using 1:2 with linespoints title 'D1mr "1024" slow', \
     "cache_1024_fix.dat" using 1:4 with linespoints title 'D1mr "1024" fast'

# Cambiar el nombre de salida para la segunda gráfica
set output 'cache_escritura_fix.png'

# Configuración de la segunda gráfica (escritura de datos)
set title 'Fallos de Escritura de Datos'
set ylabel 'Número de Fallos'
plot "cache_1024_fix.dat" using 1:3 with linespoints title 'D1mw "1024" slow', \
     "cache_1024_fix.dat" using 1:5 with linespoints title 'D1mw "1024" fast'
quit
END_GNUPLOT
echo "Plots generated successfully."
```





Es observable que para tamaños fijos de caché fast produce menos fallos de lectura que slow, ya que en el ámbito de la escritura ambos fallan prácticamente las mismas veces.

¿A qué se debe cada uno de los efectos observados?

Un tamaño de caché más grande, como ya he dicho, significa una mayor capacidad para almacenar datos, esto mejora la probabilidad de que los datos necesarios están ya almacenados en la caché, reduciendo así la tasa de fallos. Además, tamaños de caché más grandes mejoran la localidad temporal,es decir la tendencia de acceder a los datos mejora.

Por otra parte, el aumento del tamaño de la matriz, aumenta la necesidad de almacenamiento, ya que se generan más datos. Esto puede exceder la capacidad de la caché, lo que genera más fallos en dicha caché. Antes, se ha mencionado la localidad temporal, y en este caso, los tamaños más grandes de caché, ralentizan el acceso a los datos.

Indagando ahora en los programas slow y fast, llegamos a la conclusión de que una manera más eficiente de acceder a la memoria caché, en este caso el programa fast sobre slow, puede estar diseñado para aprovechar mejor la jerarquía de memoria, reduciendo los fallos.

Ejercicio 3: Caché y multiplicación de matrices

mult_matrix.c:

```c
#include <stdio.h>
#include "arqo3.h"

int main(int argc, char *argv[]){
    int n, i, j, k;
    tipo **a = NULL, **b = NULL, **c = NULL;
    struct timeval fin, ini;

    printf("Word size: %ld bits\n", 8*sizeof(tipo));

    if(argc != 2){
        printf("Error: ./%s <matrix size>\n", argv[0]);
        return -1;
    }

    n = atoi(argv[1]);
    a = generateMatrix(n);
    b = generateMatrix(n);
    c = generateEmptyMatrix(n);

    gettimeofday(&ini, NULL);

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            for(k = 0; k < n; k++){
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    gettimeofday(&fin, NULL);
    printf("Execution time: %f\n", ((fin.tv_sec*1000000+fin.tv_usec)-(ini.tv_sec*1000000+ini.tv_usec))*1.0/1000000.0);

    freeMatrix(a);
    freeMatrix(b);
    freeMatrix(c);
    return 0;
}
```

mult_trasp.c:

```c
#include <stdio.h>
#include "arqo3.h"

int main(int argc, char *argv[]){
    int n, i, j, k;
    tipo **a = NULL, **b = NULL, **bt = NULL, **c = NULL;
    struct timeval fin, ini;

    printf("Word size: %ld bits\n", 8*sizeof(tipo));

    if(argc != 2){
        printf("Error: ./%s <matrix size>\n", argv[0]);
        return -1;
    }

    n = atoi(argv[1]);
    a = generateMatrix(n);
    b = generateMatrix(n);
    bt = generateEmptyMatrix(n);
    c = generateEmptyMatrix(n);

    gettimeofday(&ini, NULL);

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            bt[i][j] = b[j][i];
        }
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            for(k = 0; k < n; k++){
                c[i][j] += a[i][k] * bt[j][k];
            }
        }
    }

    gettimeofday(&fin, NULL);
    printf("Execution time: %f\n", ((fin.tv_sec*1000000+fin.tv_usec)-(ini.tv_sec*1000000+ini.tv_usec))*1.0/1000000.0);

    freeMatrix(a);
    freeMatrix(b);
    freeMatrix(bt);
    freeMatrix(c);
    return 0;
}
```

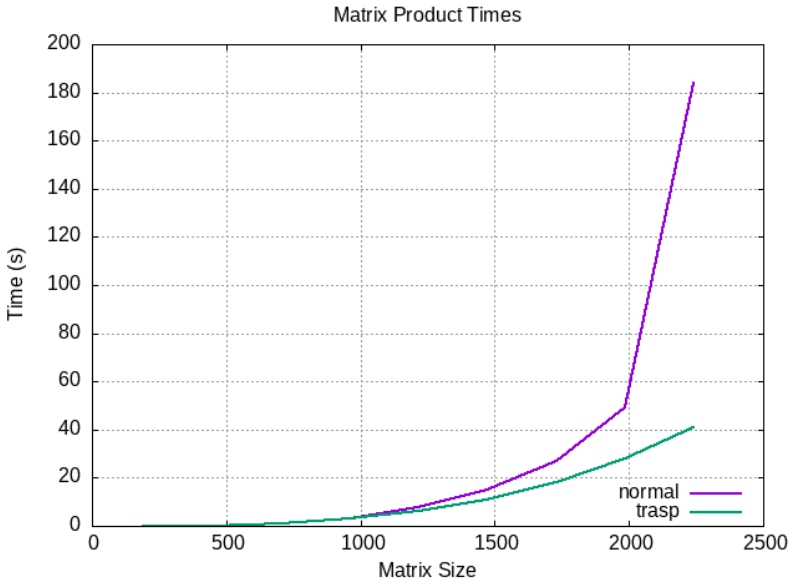
P = 11 -> 128 + 16 * 4 = 192 // 2176 + 16 * 4 = 2240

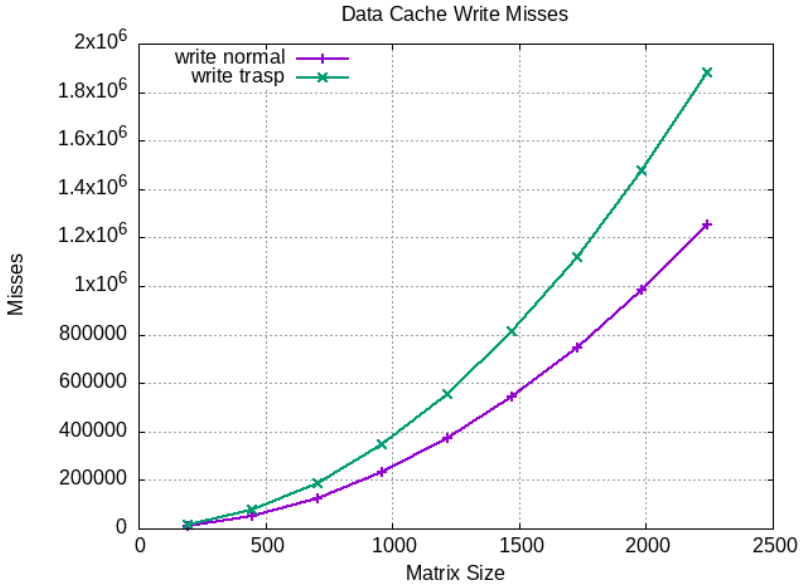
Se ha generado el script “ej3.sh” para llevar a cabo los apartados 1) y 2). En el apartado 1) han habido problemas para generar bien los tiempos y realizar la media con la última actualización del código presentada, la gráfica corresponde a una versión anterior en la que los tiempos si se medían correctamente (debido al gran tiempo que supone la ejecución del script no se ha podido perfeccionar esta parte). Para el apartado 3) se ha seguido las directrices del enunciado para generar el “mult.dat”: echo "$Nsize $normTime $normD1mr $normD1mw $traspTime $traspD1mr $traspD1mw" >> "$Ndata"

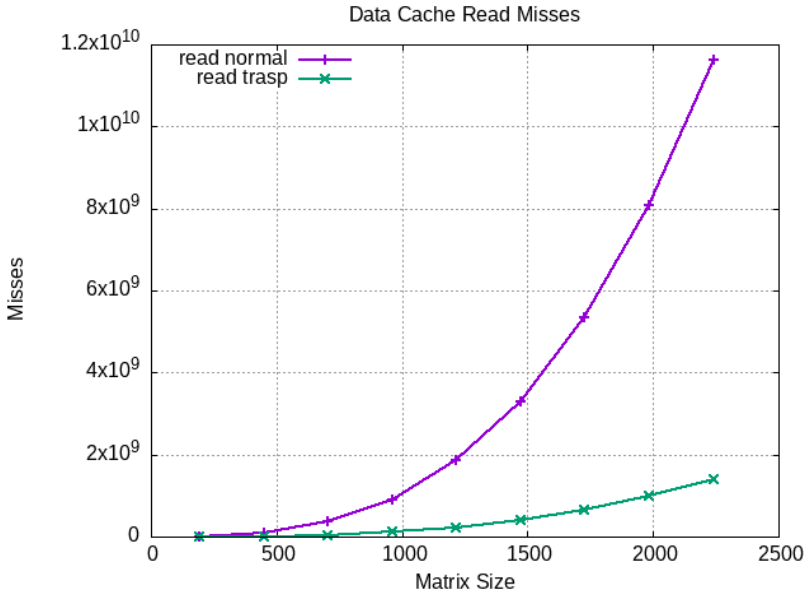
```bash
#!/bin/bash

#Variables generales
P=4

#Variables ej 3
N1=$((128 + 16 * P))
N2=$((2176 + 16 * P))
N=0
Nstep=256
Ndata=mult.dat

rm -f $Ndata $Nplot
touch $Ndata

for ((N = N1 ; N <= N2 ; N += Nstep)); do
    if ((i=="0")); then
        times_norm[$i]=0
        times_trasp[$i]=0
        normD1mr[$i]=0
        normD1mw[$i]=0
        traspD1mr[$i]=0
        traspD1mw[$i]=0
    fi;
    for((i = 0; i <= 10; i++)); do
        echo "N: $N / $N2..."
        normTime_t=$(./mult_matrix "$N" | grep 'time' | awk '{print $3}')
        traspTime_t=$(./mult_trasp "$N" | grep 'time' | awk '{print $3}')


        times_norm[$i]=$(echo "${times_norm[$i]}" "$normTime_t" | awk '{print $1 + $2}')
        times_trasp[$i]=$(echo "${times_trasp[$i]}" "$traspTime_t" | awk '{print $1 + $2}')
    done
done

length=${#times_norm[@]}

for ((N = 0; N < length; N += 1 )); do
    Nsize=$((N1 + Nstep * N))
    echo "$N"

    normTime=$(echo "${times_norm[$N]}" | awk '{print $1/5}')
    traspTime=$(echo "${times_trasp[$N]}" | awk '{print $1/5}')

    valgrind --tool=cachegrind --cachegrind-out-file=norm.out ./mult_matrix $Nsize
    valgrind --tool=cachegrind --cachegrind-out-file=trasp.out ./mult_trasp $Nsize

    normD1mr=$(cg_annotate norm.out | grep 'PROGRAM' | awk '{print $9}' | tr -d ',')
    normD1mw=$(cg_annotate norm.out | grep 'PROGRAM' | awk '{print $15}' | tr -d ',')
    traspD1mr=$(cg_annotate trasp.out | grep 'PROGRAM' | awk '{print $9}' | tr -d ',')
    traspD1mw=$(cg_annotate trasp.out | grep 'PROGRAM' | awk '{print $15}' | tr -d ',')

    echo "$Nsize $normTime $normD1mr $normD1mw $traspTime $traspD1mr $traspD1mw" >> "$Ndata"
done
```

Para el apartado 4) para generar las 3 gráficas que requiere el enunciado se ha realizado de la siguiente manera:

```
#!/bin/bash

echo "Generating plot..."
# llamar a gnuplot para generar el gráfico y pasarle directamente por la entrada
# estándar el script que está entre "<< END_GNUPLOT" y "END_GNUPLOT"
gnuplot << END_GNUPLOT
set title "Data Cache Read Misses"
set ylabel "Misses"
set xlabel "Matrix Size"
set key left top
set grid
set term png
set output "mult_cache_read.png"
plot "mult.dat" using 1:3 with linespoints lw 2 title "read normal", \
     "mult.dat" using 1:6 with linespoints lw 2 title "read trasp"
replot
set title "Data Cache Write Misses"
set ylabel "Misses"
set xlabel "Matrix Size"
set key left top
set grid
set term png
set output "mult_cache_write.png"
plot "mult.dat" using 1:4 with linespoints lw 2 title "write normal", \
     "mult.dat" using 1:7 with linespoints lw 2 title "write trasp"
replot
set title "Matrix Product Times"
set ylabel "Time (s)"
set key left top
set output "mult_time.png"
plot "mult.dat" using 1:2 with linespoints lw 2 title "normal", \
     "mult.dat" using 1:5 with linespoints lw 2 title "trasp"
replot
quit
END_GNUPLOT
```


Matrix Product Times
Time (s)
Matrix Size
normal
trasp
0
20
40
60
80
100
120
140
160
180
200
0
500
1000
1500
2000
2500


Data Cache Write Misses
Misses
Matrix Size
write normal
write trasp
0
200000
400000
600000
800000
1x10^6
1.2x10^6
1.4x10^6
1.6x10^6
1.8x10^6
2x10^6
0
500
1000
1500
2000
2500


Data Cache Read Misses
Misses
Matrix Size
read normal
read trasp
0
2x10^9
4x10^9
6x10^9
8x10^9
1x10^10
1.2x10^10
0
500
1000
1500
2000
2500

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

En las gráficas de fallos de lectura de caché podemos ver cómo a medida que las matrices crecen en tamaño, aumenta la probabilidad de que los datos necesarios para la multiplicación no estén completamente contenidos en la caché, lo que resulta en más fallos de caché y un rendimiento más lento debido al acceso frecuente a la memoria principal. En el caso de escritura la traspuesta presenta más fallos.

Y en la gráfica de tiempos de ejecución es algo lógico, al ser más grande la matriz, se tardará más tiempo en llevar a cabo la multiplicación de estas.

Lo que sí hay que comentarles es que se producen menos fallos de lectura de caché para la traspuesta porque alguno de estos datos ya están en la caché y, además, se tarda menos tiempo en realizar operaciones con estos datos.

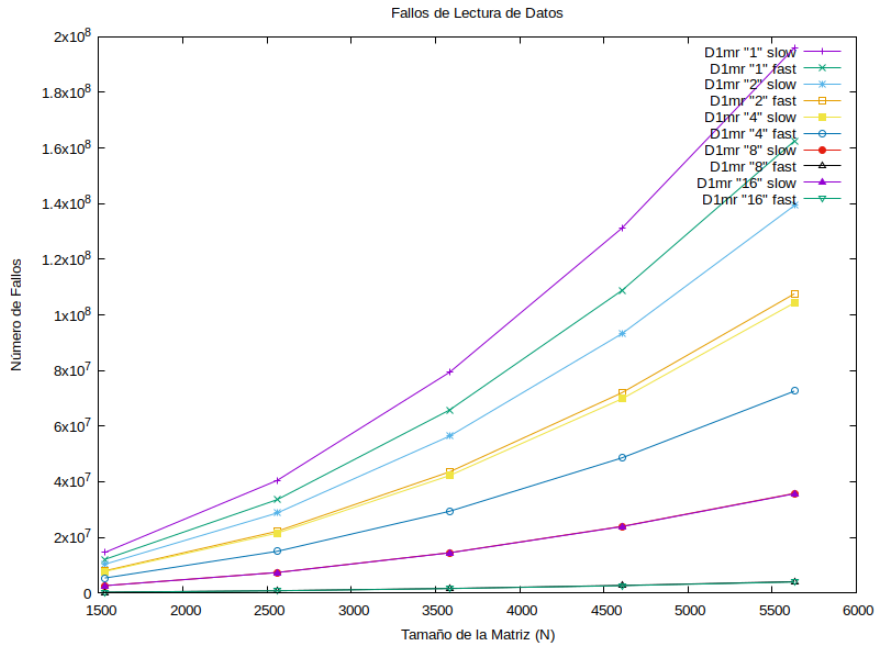
Ejercicio 4 (Opcional): Configuraciones de Caché en la multiplicación de matrices

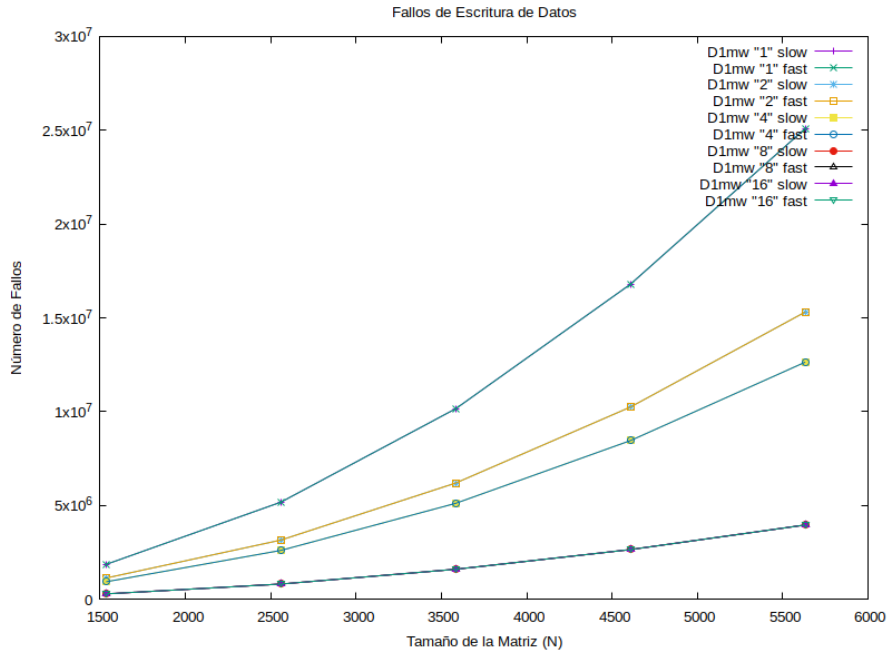
Primero probaremos en variar el valor de “associativity”, el script es el siguiente “slow_fast_asociatividad_variada.sh”:

```bash
#!/bin/bash

# Inicializar variables
Ninicio=1024
Npaso=1024
Nfinal=5120
P=4
CACHE_SIZE=1024 # Tamaños de caché en Bytes
ASSOCIATIVITYS=(1 2 4 8 16)
CACHE_SIZE_UPPER=8388608  # 8 MBytes
POSICION_D1MR=9
POSICION_D1MW=15

# Bucle para N desde P hasta Q
for ((N = Ninicio + 128 * P; N <= Nfinal + 128 * P; N += Npaso)); do
    for ASSOCIATIVITY in "${ASSOCIATIVITYS[@]}"; do
        OUTPUT_FILE="cache_as_${ASSOCIATIVITY}.dat"
        # Escribir el valor de N en el archivo
        echo -n "$N " >> $OUTPUT_FILE

        for PROGRAM in "slow" "fast"; do
            # Ejecutar Valgrind con cachegrind y guardar la salida en un archivo temporal
            valgrind --tool=cachegrind --I1=$CACHE_SIZE,$ASSOCIATIVITY,64 --D1=$CACHE_SIZE,$ASSOCIATIVITY,64 --LL=$CACHE_SIZE_UPPER,$ASSOCIATIVITY,64 --cachegrind-out-fil

            echo ""
            echo "CG_ANNOTATE:"
            echo ""
            cg_annotate pr_out.dat | head -n 20

            LINE=$(cg_annotate pr_out.dat | head -n 20| awk 'NR==18 {print}')
            D1MR=$(echo "$LINE" | awk -v col="$POSICION_D1MR" '{print $col}')
            D1MW=$(echo "$LINE" | awk -v col="$POSICION_D1MW" '{print $col}')

            # Escribir los valores en el archivo
            echo -n "$D1MR $D1MW " >> $OUTPUT_FILE

            rm pr_out.dat
        done

        # Agregar un salto de línea al final de la línea después de escribir los resultados para cada conjunto de caché
        if [ "$((N + Npaso))" -le "$((Nfinal + 128 * P))" ]; then
            echo "" >> "$OUTPUT_FILE"
        fi

        sed -i 's/,//g' "$OUTPUT_FILE"
    done
done
```

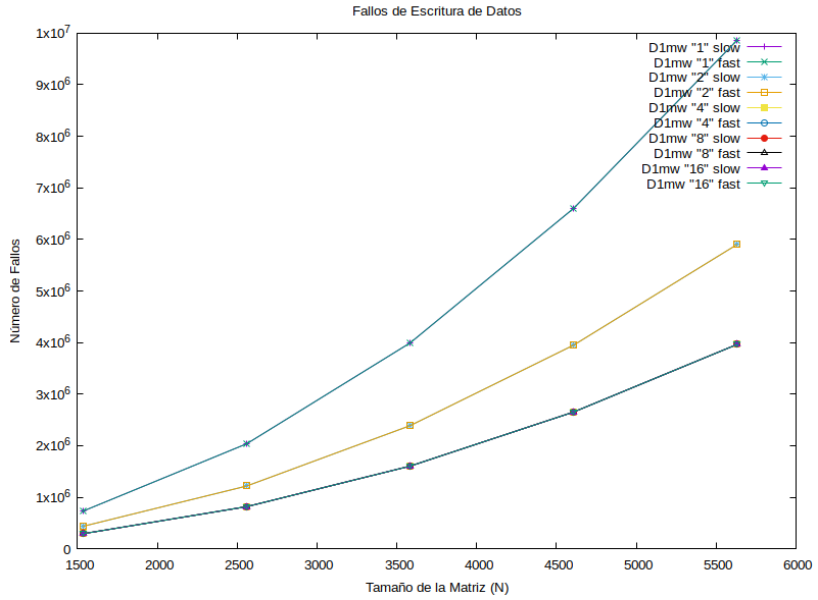
```bash
echo "Generating plot..."
# Llamar a gnuplot para generar el gráfico y pasarle directamente por la entrada
# estándar el script que está entre "<< END_GNUPLOT" y "END_GNUPLOT"
gnuplot << END_GNUPLOT

# Configuración del terminal y salida de las gráficas
set terminal pngcairo enhanced font 'arial,10' size 800, 600
set output 'cache_lectura_associativity.png'

# Configuración de la primera gráfica (lectura de datos)
set title 'Fallos de Lectura de Datos'
set xlabel 'Tamaño de la Matriz (N)'
set ylabel 'Número de Fallos'
plot "cache_as_1.dat" using 1:2 with linespoints title 'D1mr "1" slow', \
     "cache_as_1.dat" using 1:4 with linespoints title 'D1mr "1" fast', \
     "cache_as_2.dat" using 1:2 with linespoints title 'D1mr "2" slow', \
     "cache_as_2.dat" using 1:4 with linespoints title 'D1mr "2" fast', \
     "cache_as_4.dat" using 1:2 with linespoints title 'D1mr "4" slow', \
     "cache_as_4.dat" using 1:4 with linespoints title 'D1mr "4" fast', \
     "cache_as_8.dat" using 1:2 with linespoints title 'D1mr "8" slow', \
     "cache_as_8.dat" using 1:4 with linespoints title 'D1mr "8" fast', \
     "cache_as_16.dat" using 1:2 with linespoints title 'D1mr "16" slow', \
     "cache_as_16.dat" using 1:4 with linespoints title 'D1mr "16" fast'

# Cambiar el nombre de salida para la segunda gráfica
set output 'cache_escritura_associativity.png'

# Configuración de la segunda gráfica (escritura de datos)
set title 'Fallos de Escritura de Datos'
set ylabel 'Número de Fallos'
plot "cache_as_1.dat" using 1:3 with linespoints title 'D1mw "1" slow', \
     "cache_as_1.dat" using 1:5 with linespoints title 'D1mw "1" fast', \
     "cache_as_2.dat" using 1:3 with linespoints title 'D1mw "2" slow', \
     "cache_as_2.dat" using 1:5 with linespoints title 'D1mw "2" fast', \
     "cache_as_4.dat" using 1:3 with linespoints title 'D1mw "4" slow', \
     "cache_as_4.dat" using 1:5 with linespoints title 'D1mw "4" fast', \
     "cache_as_8.dat" using 1:3 with linespoints title 'D1mw "8" slow', \
     "cache_as_8.dat" using 1:5 with linespoints title 'D1mw "8" fast', \
     "cache_as_16.dat" using 1:3 with linespoints title 'D1mw "16" slow', \
     "cache_as_16.dat" using 1:5 with linespoints title 'D1mw "16" fast'

quit
END_GNUPLOT
echo "Plots generated successfully."
```

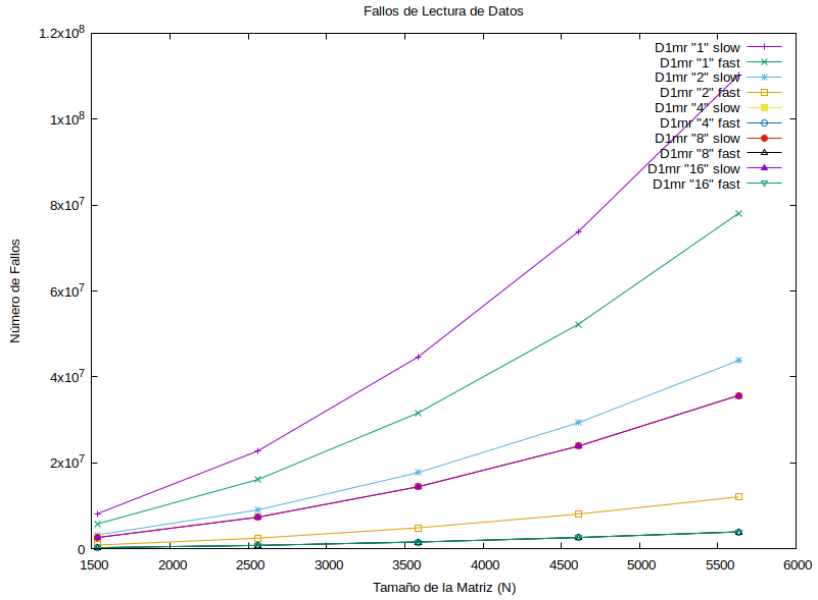
Las gráficas resultantes son:





Se ha probado con un tamaño de caché más grande = 4096





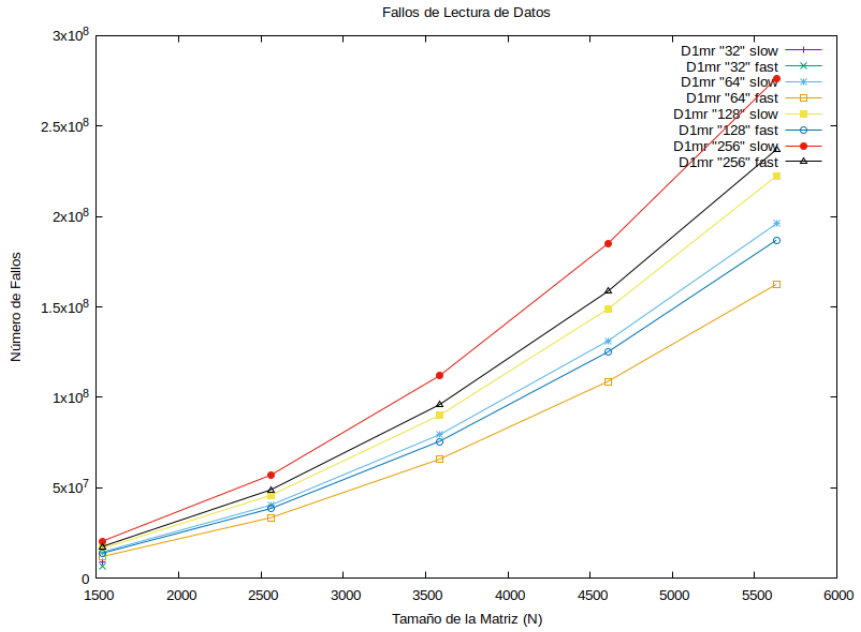
Y como podemos observar se generan mejores resultados que con un tamaño menor de caché para asociaciones más altas.

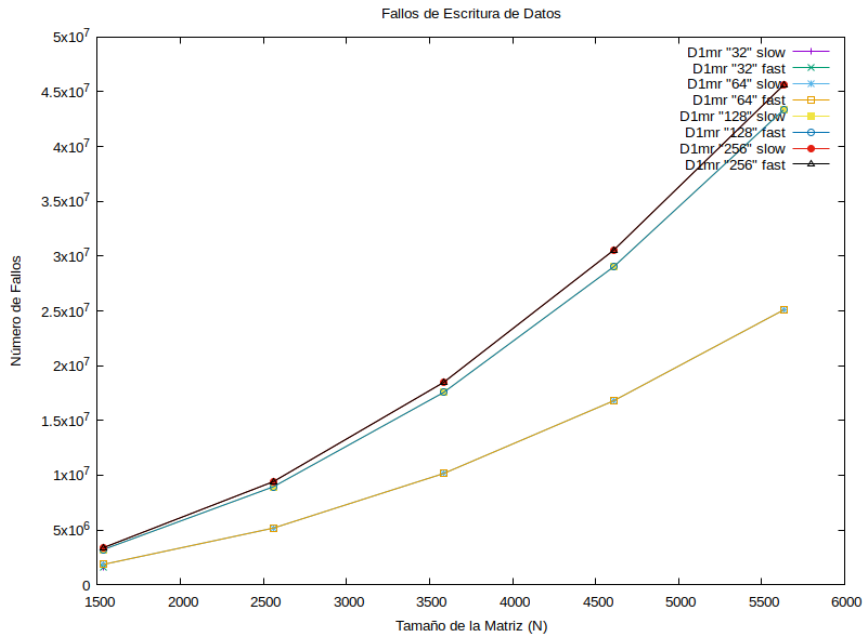
Variando el tamaño de la línea “slow_fast_line.sh”:

```bash
#!/bin/bash

# Inicializar variables
Ninicio=1024
Npaso=1024
Nfinal=5120
P=4
CACHE_SIZE=1024 # Tamaños de caché en Bytes
LINES=(32 64 128 256)
CACHE_SIZE_UPPER=8388608  # 8 MBytes
POSICION_D1MR=9
POSICION_D1MW=15

# Bucle para N desde P hasta Q
for ((N = Ninicio + 128 * P; N <= Nfinal + 128 * P; N += Npaso)); do
    for LINE in "${LINES[@]}"; do
        OUTPUT_FILE="cache_ln_${LINE}.dat"
        # Escribir el valor de N en el archivo
        echo -n "$N " >> $OUTPUT_FILE

        for PROGRAM in "slow" "fast"; do
            # Ejecutar Valgrind con cachegrind y guardar la salida en un archivo temporal
            valgrind --tool=cachegrind --I1=$CACHE_SIZE,1,$LINE --D1=$CACHE_SIZE,1,$LINE --LL=$CACHE_SIZE_UPPER,1,$LINE --cachegrind-out-file=pr_out.dat ./$PROGRAM $N

            echo ""
            echo "CG_ANNOTATE:"
            echo ""
            cg_annotate pr_out.dat | head -n 20

            LINEA=$(cg_annotate pr_out.dat | head -n 20| awk 'NR==18 {print}')
            D1MR=$(echo "$LINEA" | awk -v col="$POSICION_D1MR" '{print $col}')
            D1MW=$(echo "$LINEA" | awk -v col="$POSICION_D1MW" '{print $col}')

            # Escribir los valores en el archivo
            echo -n "$D1MR $D1MW " >> $OUTPUT_FILE

            rm pr_out.dat
        done

        # Agregar un salto de línea al final de la línea después de escribir los resultados para cada conjunto de caché
        if [ "$((N + Npaso))" -le "$((Nfinal + 128 * P))" ]; then
            echo "" >> "$OUTPUT_FILE"
        fi

        sed -i 's/,//g' "$OUTPUT_FILE"
    done
done
```

```bash
echo "Generating plot..."
# Llamar a gnuplot para generar el gráfico y pasarle directamente por la entrada
# estándar el script que está entre "<< END_GNUPLOT" y "END_GNUPLOT"
gnuplot << END_GNUPLOT

# Configuración del terminal y salida de las gráficas
set terminal pngcairo enhanced font 'arial,10' size 800, 600
set output 'cache_lectura_line.png'

# Configuración de la primera gráfica (lectura de datos)
set title 'Fallos de Lectura de Datos'
set xlabel 'Tamaño de la Matriz (N)'
set ylabel 'Número de Fallos'
plot "cache_ln_32.dat" using 1:2 with linespoints title 'D1mr "32" slow', \
     "cache_ln_32.dat" using 1:4 with linespoints title 'D1mr "32" fast', \
     "cache_ln_64.dat" using 1:2 with linespoints title 'D1mr "64" slow', \
     "cache_ln_64.dat" using 1:4 with linespoints title 'D1mr "64" fast', \
     "cache_ln_128.dat" using 1:2 with linespoints title 'D1mr "128" slow', \
     "cache_ln_128.dat" using 1:4 with linespoints title 'D1mr "128" fast', \
     "cache_ln_256.dat" using 1:2 with linespoints title 'D1mr "256" slow', \
     "cache_ln_256.dat" using 1:4 with linespoints title 'D1mr "256" fast'

# Cambiar el nombre de salida para la segunda gráfica
set output 'cache_escritura_line.png'

# Configuración de la segunda gráfica (escritura de datos)
set title 'Fallos de Escritura de Datos'
set ylabel 'Número de Fallos'
plot "cache_ln_32.dat" using 1:3 with linespoints title 'D1mr "32" slow', \
     "cache_ln_32.dat" using 1:5 with linespoints title 'D1mr "32" fast', \
     "cache_ln_64.dat" using 1:3 with linespoints title 'D1mr "64" slow', \
     "cache_ln_64.dat" using 1:5 with linespoints title 'D1mr "64" fast', \
     "cache_ln_128.dat" using 1:3 with linespoints title 'D1mr "128" slow', \
     "cache_ln_128.dat" using 1:5 with linespoints title 'D1mr "128" fast', \
     "cache_ln_256.dat" using 1:3 with linespoints title 'D1mr "256" slow', \
     "cache_ln_256.dat" using 1:5 with linespoints title 'D1mr "256" fast'


quit
END_GNUPLOT
echo "Plots generated successfully."
```

Las gráficas resultantes son:





Los resultados que podemos observar es que la mejor solución la obtenemos con tamaño de línea igual a 64 para el programa fast.c

Pero en general los tamaños de línea más altos empeoran los resultados. El tamaño de línea en una caché hace referencia a la cantidad de datos transferidos entre la caché y la memoria principal en una sola operación. Por lo tanto genera que se produzcan más carga o lectura de datos en una sola operación.También estos datos compiten entre ellos por

espacio en caché, si la política de reemplazo de la caché no es eficiente para manejar líneas más grandes, puede resultar en mayor número de fallos.

Observando todo esto, vamos a probar a combinar los mejores resultados de cada ejecución para asociatividad, tamaño de línea y tamaño de memoria caché, esto se implementara en “slow_fast_best.sh”