Examen 2: Parte práctica

Área Académica de Ingeniería en Computadores
Arquitectura de Computadores I
Profesor Luis Alberto Chavarría Zamora
Semestre II - 2023
Estudiantes:

- José Fernando Morales Vargas, 2019024270
- Jimena León Huertas, 2021016748

1. Cree un sistema base simple con caché de nivel L1 y L2. Use un ISA x86 (utilice un diagrama para describir la caché y otros aspectos relacionados a la arquitectura). Muestre que el sistema compila con un test simple. Indique los pasos seguidos y una captura del test.

Diagrama de memoria:

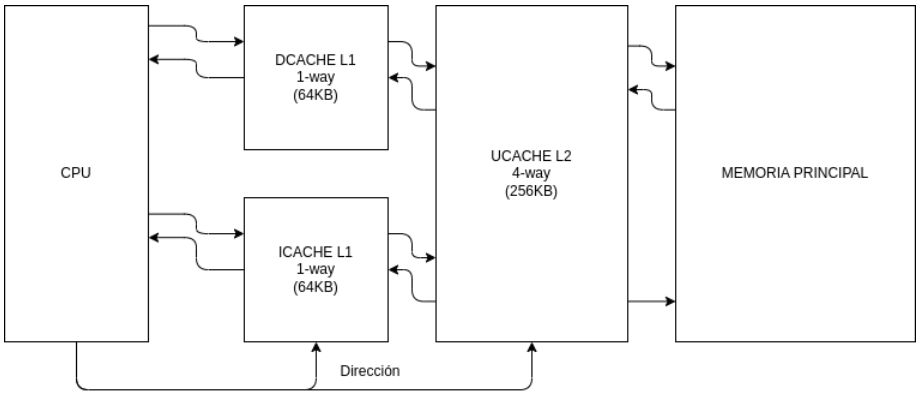
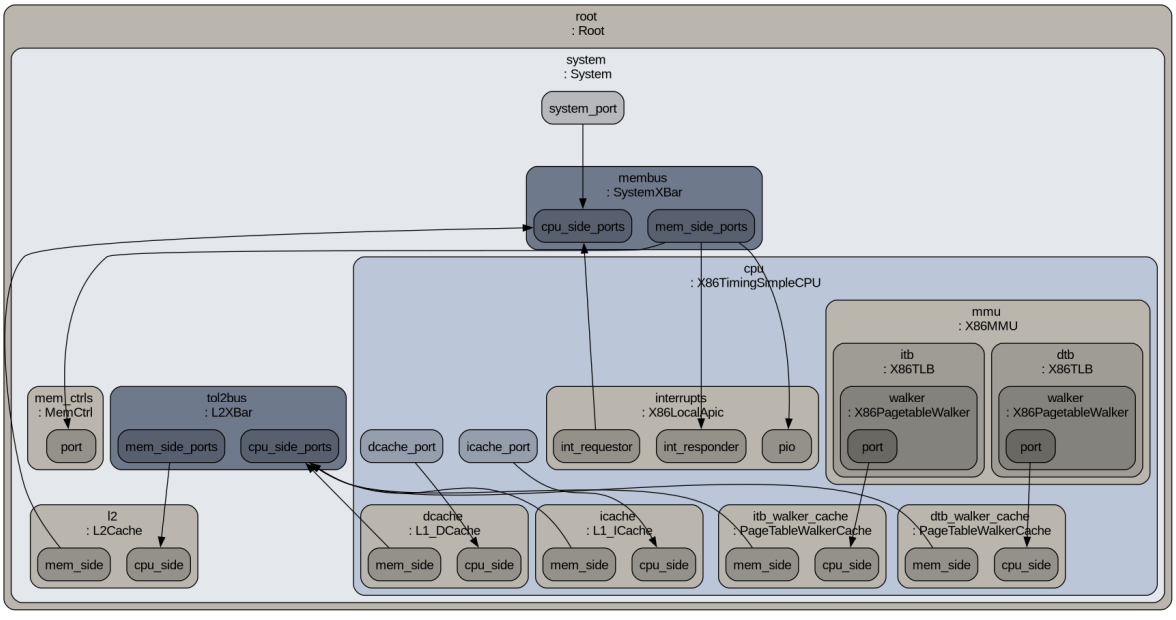


Diagrama de sistema generado por gem5



Pasos:

1. Se configura un entorno para uso de Gem5 tal como fue cubierto en el Taller 5. En este caso, se usará el archivo de descripción de sistema “se.py” utilizado para dicho taller.
2. Se determina el tipo de sistema a utilizar. En este caso, se quiere un sistema relativamente simple, por lo que se hace uso del “Timing Simple CPU”
3. Las opciones a configurar son las relacionadas a caches. Para activar los caches se hace uso de las opciones “--caches –l2cache” y luego se configuran los tamaños de cada caché junto con su asociatividad(configuración se muestra en captura). La línea de caché se toma de 64.
4. La prueba es simple, es tan solo un hello world. Esto se muestra en la captura en el resultado de llamar el comando “cat”
5. Se compila la prueba con un compilador para x86. Por conveniencia, se le da el nombre de “benchmark” al binario resultante. Se compila con -static y -O0 para asegurar que no haya optimizaciones del compilador.
6. La línea de comando a ejecutar para correr el simulador considera las variables de entorno $BENCHMARK y $GEM5_DIR. Ambas se definen haciendo uso del comando “export”. La primera debe apuntar al binario compilado en el paso anterior, la segunda debe apuntar al directorio en donde se clonó el repositorio de Gem5.
7. Se ejecuta la línea de comando mostrada en la captura. Dicha línea contiene todas las opciones de caché que simulan el sistema mostrado en los diagramas anteriores.

Captura del test:

```
root@718ebc86f51f:~/Project1_SPEC/examen# cat src/test.c
#include "stdio.h"

int main(){
    printf("\r\nHola desde Gem5\r\n");
    return 0;
}
root@718ebc86f51f:~/Project1_SPEC/examen# (cd src/; gcc test.c -o benchmark -O0 -static)
root@718ebc86f51f:~/Project1_SPEC/examen# export BENCHMARK=src/benchmark
root@718ebc86f51f:~/Project1_SPEC/examen# export GEM5_DIR=/root
root@718ebc86f51f:~/Project1_SPEC/examen#     $GEM5_DIR/build/X86/gem5.opt -d m5out/ \
        $GEM5_DIR/configs/deprecated/example/se.py -c $BENCHMARK -I 100000000 \
        --cpu-type=TimingSimpleCPU --caches --l2cache \
        --l1d_size=64kB --l1i_size=64kB --l2_size=256kB \
        --l1d_assoc=1 --l1i_assoc=1 --l2_assoc=4 --cacheline_size=64
```

Resultado:

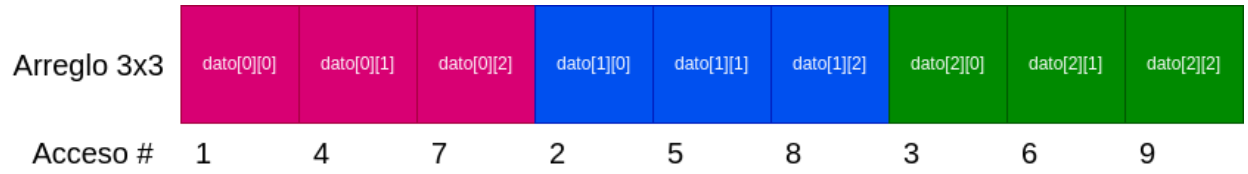
```
root@718ebc86f51f:~/Project1_SPEC/examen#     $GEM5_DIR/build/X86/gem5.opt -d m5out/ \
       $GEM5_DIR/configs/deprecated/example/se.py -c $BENCHMARK -I 100000000 \
       --cpu-type=TimingSimpleCPU --caches --l2cache \
       --l1d_size=64kB --l1i_size=64kB --l2_size=256kB \
       --l1d_assoc=1 --l1i_assoc=1 --l2_assoc=4 --cacheline_size=64
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Oct 30 2023 01:13:13
gem5 started Nov 13 2023 08:05:53
gem5 executing on 718ebc86f51f, pid 3707
command line: /root/build/X86/gem5.opt -d m5out/ /root/configs/deprecated/example/se.py -c src/benchmark -I 100000000 --cpu-type=TimingSimpleCPU --caches --l
2cache --l1d_size=64kB --l1i_size=64kB --l2_size=256kB --l1d_assoc=1 --l1i_assoc=1 --l2_assoc=4 --cacheline_size=64

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
Global frequency set at 1000000000000 ticks per second
warn: failed to generate dot output from m5out/config.dot
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is depr
ecated.
system.remote_gdb: Listening for connections on port 7000
**** REAL SIMULATION ****
src/sim/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)

Hola desde Gem5
Exiting @ tick 79472500 because exiting with last active thread context
```

2. Explique detalladamente mediante diagramas en qué consisten las siguientes optimizaciones:

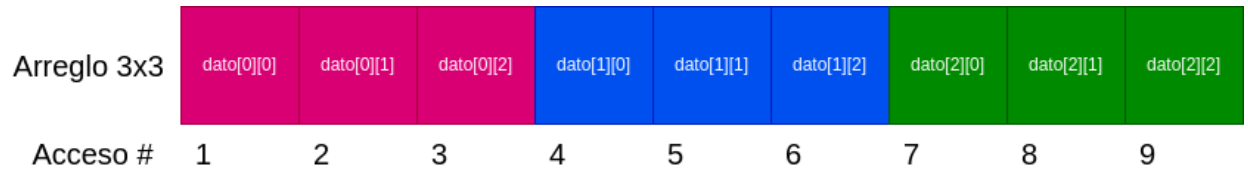
- **Loop Interchange o intercambio de ciclos**
- **Fusión de arreglos**

¿Qué pretenden mejorar respecto al desempeño de la caché?

El intercambio de ciclos busca un reacomodo de los bloques iterativos, de forma que la localidad temporal y la localidad espacial mejoren. Esto se da pues los accesos a memoria pasan de ser accesos discontinuos a accesos continuos. En el caso que no se realiza el intercambio, el orden de accesos termina siendo similar a como se muestra en la siguiente figura:

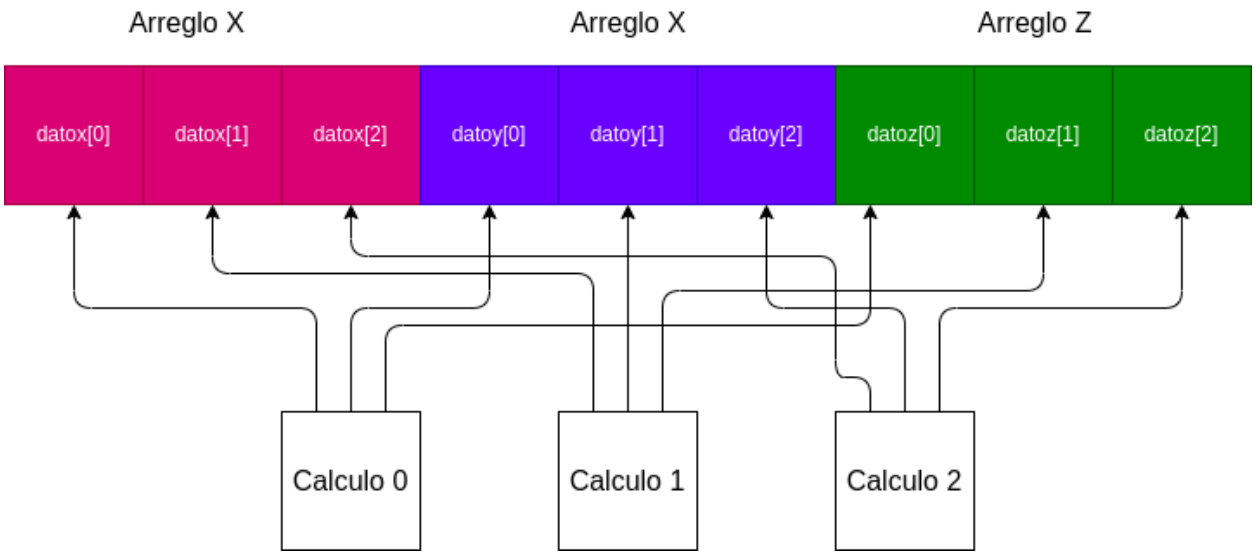


Como puede notarse, el orden de acceso es discontinuo. Al aplicar el intercambio, el acceso es ordenado:

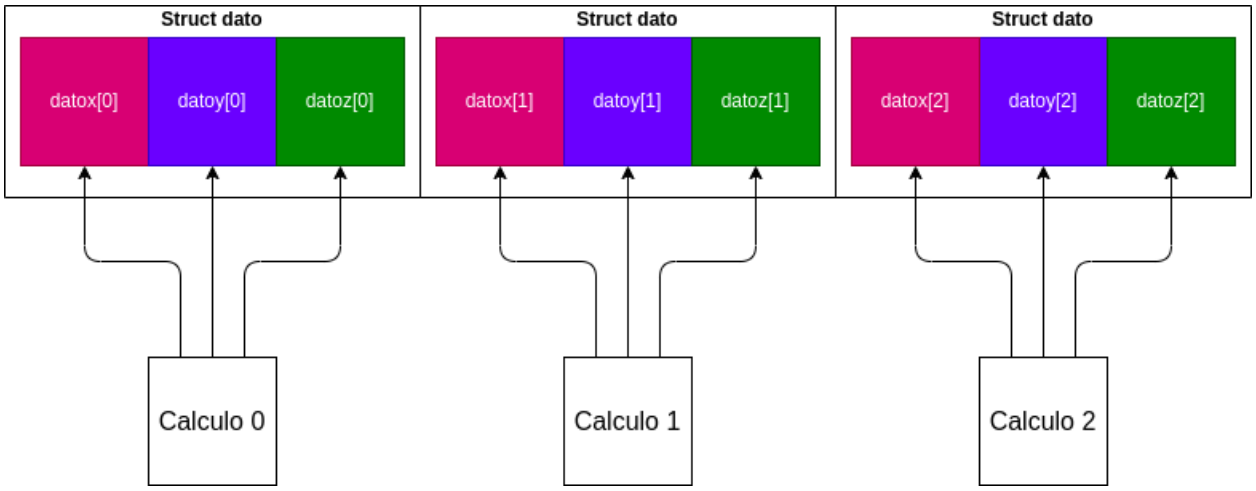


De esta forma, se puede observar que los datos con mayor localidad espacial terminan entonces aumentando su localidad temporal pues el uso de los datos cercanos requiere menos tiempo cuando el acceso es continuo.

En el caso de la fusión de arreglos, el mismo busca mejorar la localidad espacial, siendo esto de manera que unifican varios arrays (de varias o una dimensión) en uno solo. Esto puede ser mediante el guardado de los mismos en un solo struct, o bien unir el contenido de ambos en un solo arreglo. En el caso sin optimización, un diagrama que ilustra el acceso a memoria es el siguiente:



Como se puede observar, se tienen 3 cálculos, cada uno requiere 3 datos con el mismo índice: x,y,z. Si se mantienen en arreglos separados, los accesos a memoria para cada cálculo serán discontinuos, resultando en una degeneración del desempeño. El caso de la optimización se muestra en el siguiente diagrama:



Como puede observarse, al aplicar la fusión, los accesos a memoria para cada cálculo resultan en accesos a bloques continuos. Claramente el ejemplo de 3 arrays de 3 miembros es minimalista, la optimización es más provechosa entre más grande sean los arreglos que se fusionan y entre máyor localidad temporal tengan los accesos a memoria (cálculos dependen de datos vecinos).

3. Implemente un código fuente que muestren optimizaciones del punto 2. Utilice tamaños de arreglos grandes con el fin de evaluar las mejoras. Debe implementar el código fuente del caso sin optimización. Todo código debe venir comentado y justificado, si no, no se calificará el rubro.

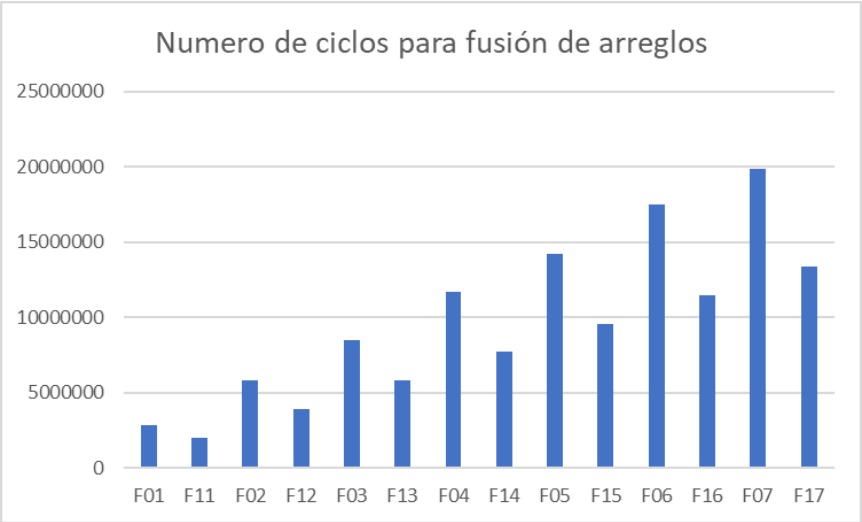
El código se adjunta al entregable. Se provee un makefile que permite compilar el test, el caso de intercambio de ciclos sin optimización (i0) y con optimización(i1), así como el caso de fusión de arreglos sin optimización (f0) y con optimización (f1). Todos los casos se compilan con linkeo estático y nivel de optimización 0.

4. Cree pruebas para cada caso y recolecte los resultados que considere necesarios para mostrar que efectivamente sí hay mejora (Discuta las estadísticas con base a lo explicado en 2, para la creación de gráficas puede usar Excel, Python o cualquier otro).

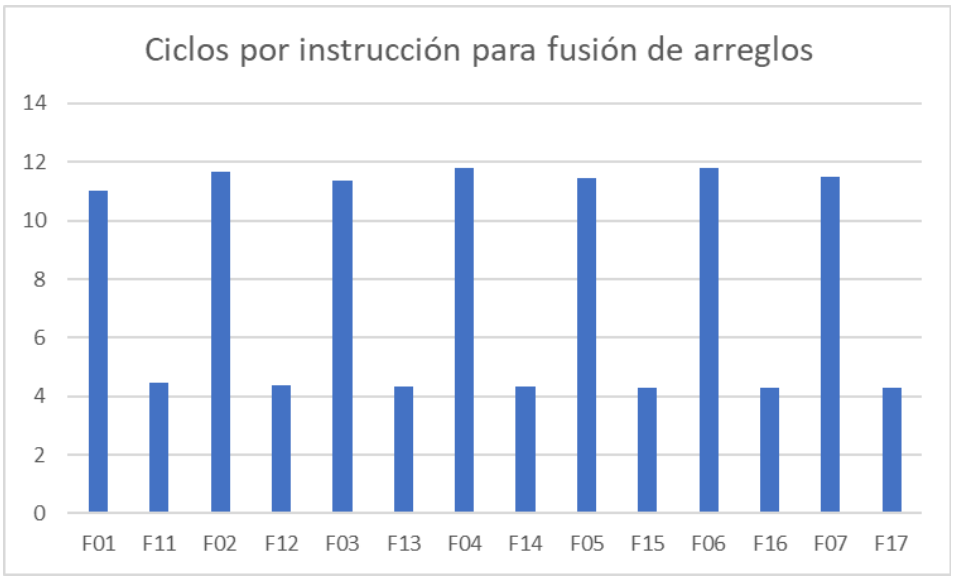
Para las pruebas se creó un script de bash que ejecuta todas las pruebas a considerar con una sola llamada (“runGem5.sh”, adjuntado en entregable). Para cada optimización se ejecuta el caso sin optimizaciones y con optimizaciones para arreglos de tamaño que varía con cada iteración de la prueba. La cantidad de datos máxima se varía de 0x1000 a 0x7000 y se generan archivos de texto que contienen las estadísticas más relevantes a resaltar para caracterizar la mejora.

En el caso de la fusión de arreglos se obtuvieron las siguientes gráficas mediante Excel. Se adjunta el archivo en el entregable. La nomenclatura es F(prueba)(tamaño de array) al tratarse de fusión de arreglos e i(prueba)(tamaño de array) al tratarse de intercambio de ciclos, en donde prueba es igual a 0 si no está optimizado y 1 si sí está optimizado. El tamaño del array puede tomar alguno de los siguientes valores: 0x1000, 0x2000, 0x3000, 0x4000, 0x5000, 0x6000 y 0x7000. Se toma el dígito más relevante para distinguirlo en las gráficas.

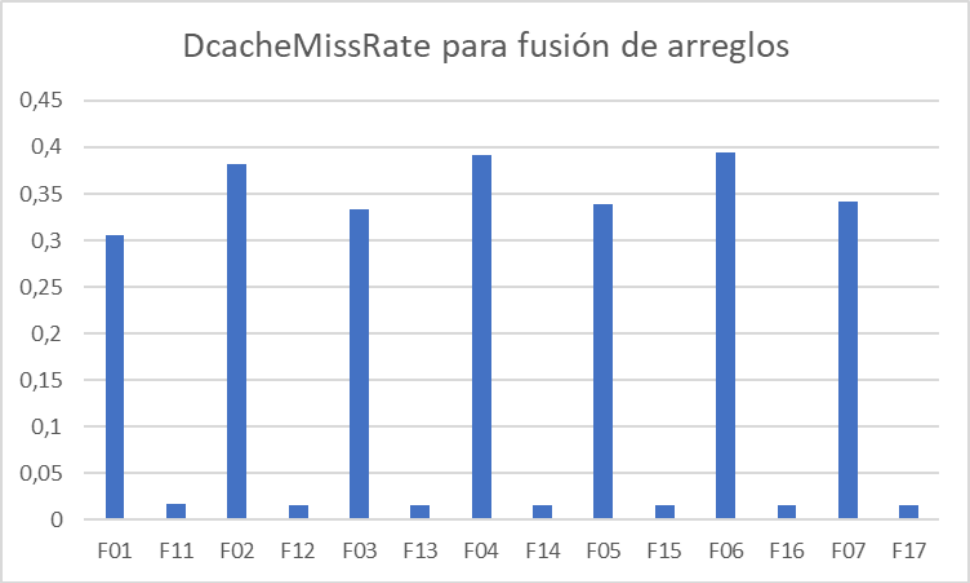
Primero se analizarán gráficas respectivas a la fusión de arreglos. Como muestra la siguiente gráfica, el número de ciclos fue menor en cada prueba optimizada, lo cual resulta beneficioso para el rendimiento del programa. Nótese que conforme aumentaba el tamaño del array, el número crecía, ya que al haber más datos se necesitan más ciclos para acceder a todos ellos. Sin embargo, en cada prueba optimizada se mantiene el hecho de que disminuye la cantidad de ciclos. Esto concuerda con la teoría estudiada.



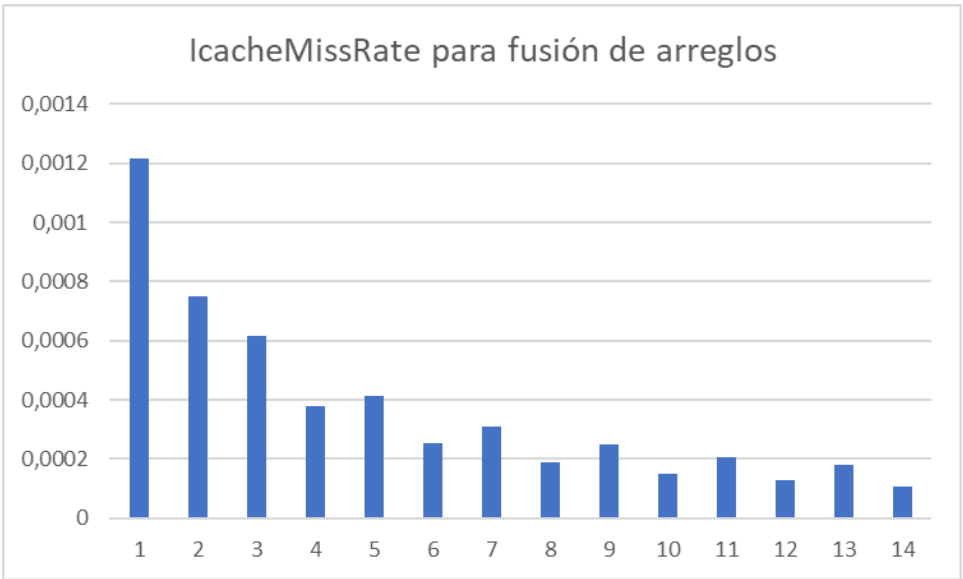
Ahora, en cuanto a la cantidad de ciclos por instrucción, se puede notar en la gráfica siguiente que este número también disminuye con cada prueba optimizada. En este caso, el aumentar el tamaño del array no modifica significativamente la cantidad de ciclos.



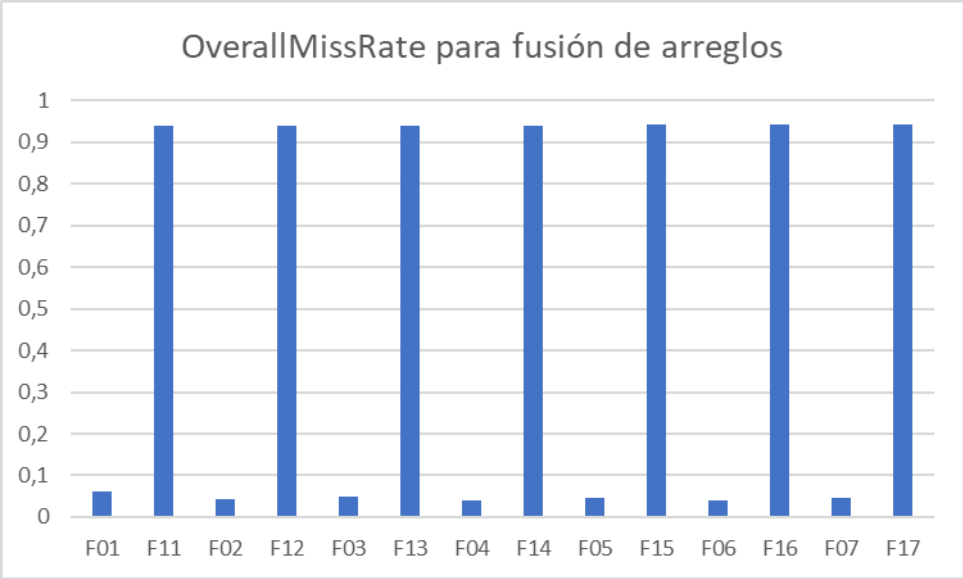
El miss rate para la memoria caché de datos fue menor con cada prueba optimizada. Esto brinda un acceso más conveniente y fácil a los datos, lo cual es precisamente el propósito de la fusión de arreglos: mejorar la localidad espacial.



Por otro lado, el miss rate para la memoria caché de instrucciones fue significativamente menor conforme se aumentaba el tamaño del array. Al usar las pruebas optimizadas el miss rate disminuyó, por lo cual a pesar de aumentar el tamaño del array, se comprueba que la optimización logró su objetivo.

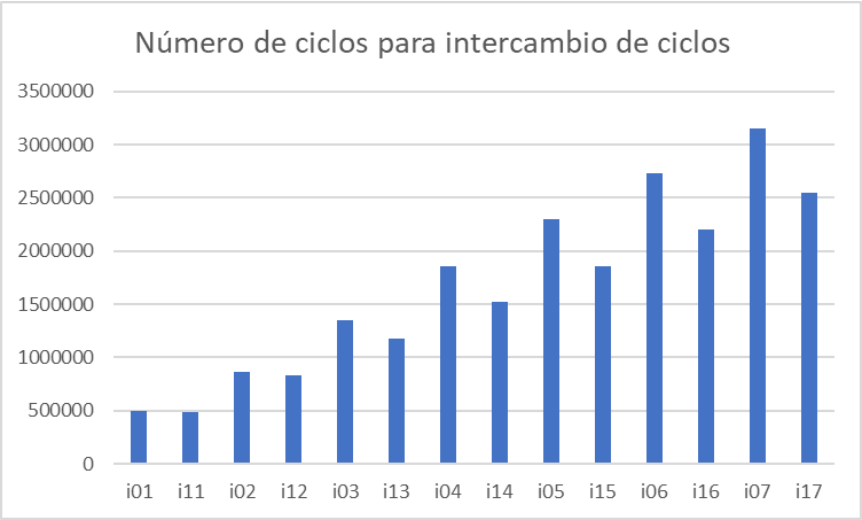


Finalmente, para fusión de arreglos, se tiene que el miss rate general aumentó en cada prueba optimizada. El aumento en el tamaño del array no fue significativo para este caso. Debido a que los arreglos se fusionan y ahora los datos serán más fácilmente accesibles, el acceso a L2 no es tan necesario. Por ende, el valor aumenta.

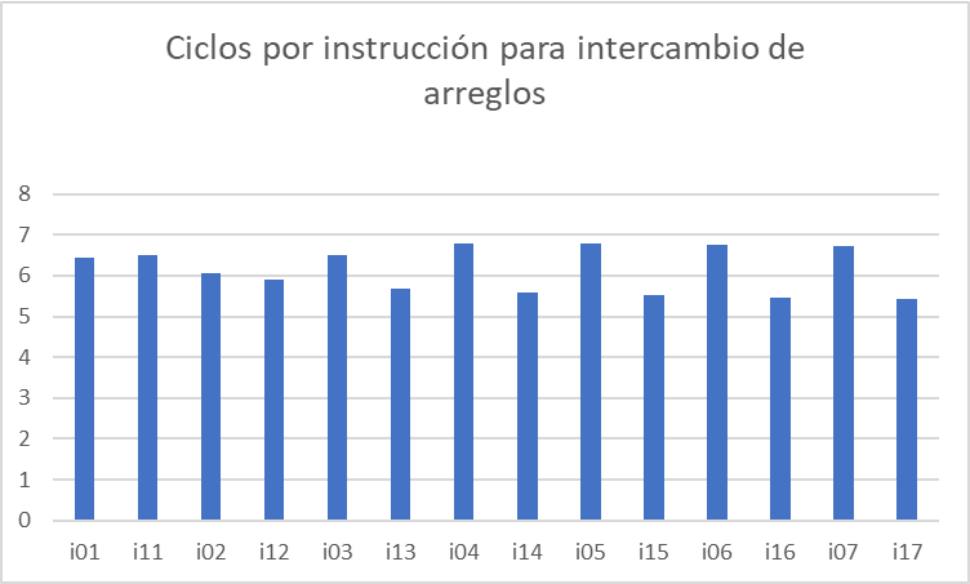


Ahora se analizarán gráficas para el código que ejemplifica el intercambio de ciclos. Se cuenta con las mismas variables y las imágenes se tomaron del mismo Excel mencionado anteriormente.

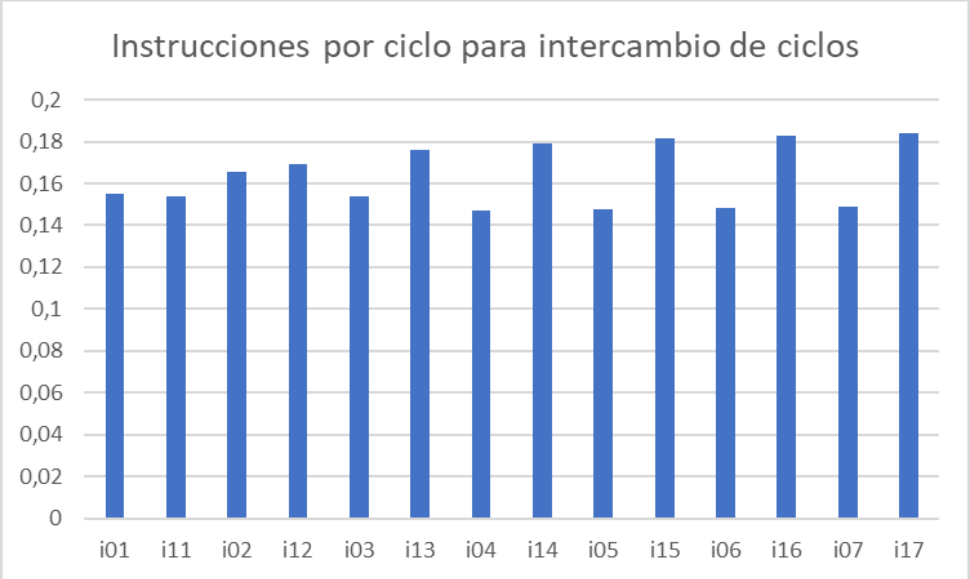
En cuanto al número de ciclos en el programa, se nota mediante la gráfica siguiente que conforme se aumenta el tamaño del array, se aumenta la cantidad de ciclos utilizados. Sin embargo, al igual que en el análisis anterior, al correr las pruebas optimizadas se tiene una disminución con respecto al código sin optimizar, esto se da porque el acceso optimizado en .el código utiliza un acceso más eficiente a los datos ya que los mismos se encontrarían contiguos en las pruebas optimizadas. Por ello, se demuestra que el intercambio de ciclos cumple con su propósito de mejorar el rendimiento del programa.



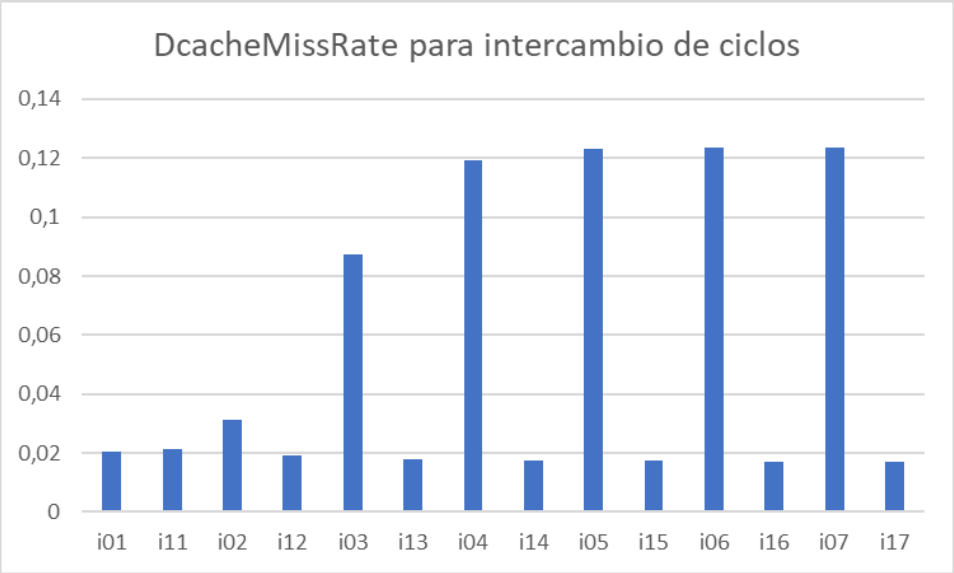
Ahora bien, los ciclos por instrucción toman un comportamiento similar al anterior. Se disminuye el valor al correr las pruebas optimizadas, mientras que el tamaño del array no cambia significativamente el resultado.



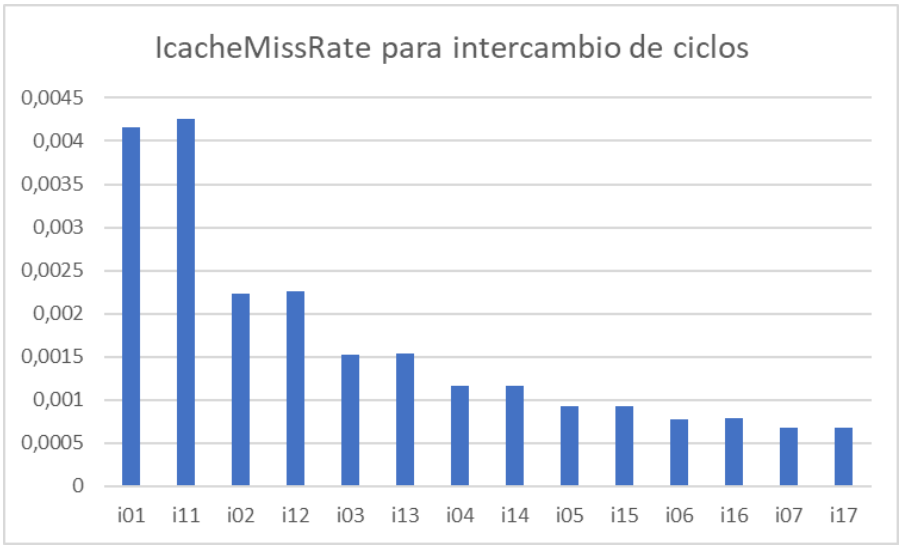
La cantidad de instrucciones por ciclo aumenta en este caso, pero solo en las pruebas optimizadas. Por lo cual, se puede afirmar que mejora el rendimiento gracias a la optimización realizada.



El miss rate para la memoria caché de datos presenta una disminución significativa al utilizar las pruebas optimizadas, lo cual se refleja en la siguiente gráfica. Conforme aumenta el tamaño del array, aumenta el miss rate; sin embargo, la disminución debida a la optimización se mantiene muy similar.



Por otro lado, el miss rate de la memoria caché de instrucciones disminuye significativamente cuando aumenta el tamaño del array y se utilizan las pruebas optimizadas. Al intercambiarse los ciclos, se deben hacer menos instrucciones dentro de los mismos ya que evitan redundancia y con ello la cantidad de accesos es menor. Es decir, la manera en que se accede a un arreglo influye en el rendimiento y la cantidad de misses en caché, tal y como se esperaba.



Finalmente, el miss rate general para L2 con intercambio de ciclos aumentó al utilizar las pruebas optimizadas. El tamaño del array no fue significativo en estos resultados. Al ser cercano a 1, es posible observar que es bastante alto. Esto sucede porque al utilizar el código optimizado, se vuelve menos recurrente acceder a la caché L2 y, con ello, el valor de la razón de misses aumenta. Los datos estarán contiguos, por lo cual no será tan necesario acceder a L2.