Práctica 2

Arquitectura de Computadoras
Simulador de cache y evaluación de su desempeño

Fernanda Mora, ITAM 8/03/2016

1. Objetivos

El objetivo de esta práctica es implementar un simulador de caché en algún lenguaje de programación y verificar la correctud del mismo mediante una evaluación de su desempeño.

2. Implementación

La práctica se divide en dos partes:

1. Simular un cache con las siguientes funcionalidades:
   - *Cambiar parámetros:* tamaño cache, tamaño del bloque, estructura de caché (unificado o separado), asociatividad, política de escritura (write-back o write-through).
   - *Reportar estadísticas:* número de referencias a instrucciones, datos, misses de instrucción y datos, número de palabras cargadas de memoria y copiadas de regreso a memoria.
2. Realizar una evaluación del desempeño del caché respondiento una serie de preguntas.

El ejercicio se implementó en el lenguaje de programación C por dos razones:

1. Es un lenguaje que permite asignación dinámica de memoria y por lo tanto va de la mano con la práctica y
2. Se nos proporcionaron archivos esqueleto del ejercicio en C, por lo que era más fácil desarrollarlo en este lenaguaje.

3. Desarrollo

3.1. Primera parte: simulación de caché

Para simulación del caché se usaron los archivos esqueleto *main.c* y *cache.c*. El único archivo que se modificó fue el *cache.c*. Se fue desarrollando la práctica incrementando funcionalidad paulatinamente.

A continuación se muestran las funciones principales del archivo y qué hace cada una:

- set_cache_param: Se inicializan los parámetros del caché: tamaño del bloque, tamaño del cache, la asociatividad, las políticas de lectura y misses de lectura. Son los parámetros que en la parte dos de la práctica se variarán para hacer un análisis del desempeño del caché.
- init_cache: Aquí se inicializó el caché, las estadísticas y las estructuras de datos para el caché unificado y dividido.
- insert_head: se inserta en la cabeza un bloque de caché
- perform_access: es la rutina que hace el acceso a cache y asigna nueva línea de cache, según sean instrucciones o datos. Es la rutina más larga de la práctica.
- flush_cache: es la rutina para limpiar el caché.
- delete, insert: inserta o elimina bloque de cache.
- dump_settings, print_stats: imprime la configuración elegida de caché (parámetros) y las estadísticas del caché.

3.2. Segunda parte: evaluación de desempeño

Caracterización del conjunto a trabajar

En este ejercicio, para instrucciones y datos, se graficó la tasa de hit como función del tamaño del caché. El rango del tamaño del caché es de 8 bytes hasta que ya no haya cambio en la tasa de hit. Se usó un caché tipo fully-associative, se usaron políticas de write-back y write-allocate.

A continuación se muestran las gráficas para cada uno de los conjuntos SPICE, TEX y CC para instrucciones y datos.

En la Figura 1 podemos ver que para los datos SPICE, a partir de un tamaño de caché de 256 bytes ya no se alcanza un aumento significativo en la tasa de hit. Sin embargo para 256 bytes en las instrucciones la tasa de hit alcanzada es mayor. No obstante, la tasa de hit converge más rápido para los datos, es decir, necesitamos menos bytes para alcanzar niveles razonablemente altos de tasa de hit.

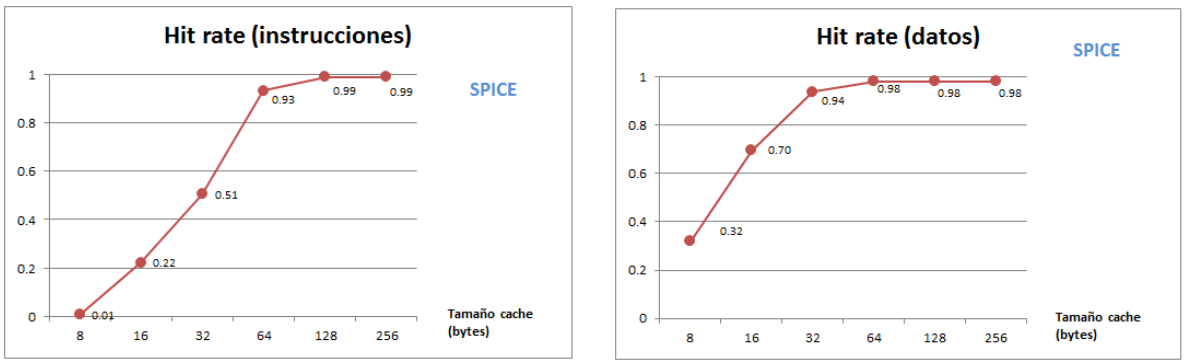


Figura 1: Tasa de hit por tamaño del cache en Spice

En la Figura 2 podemos ver que para los datos TEX, un tamaño de caché más pequeño es suficiente para alcanzar tasas altas de hit. Nuevamente notamos que se alcanzan tamaños más altos de hit para las instrucciones que para los datos pero que los datos convergen más rápidamente.

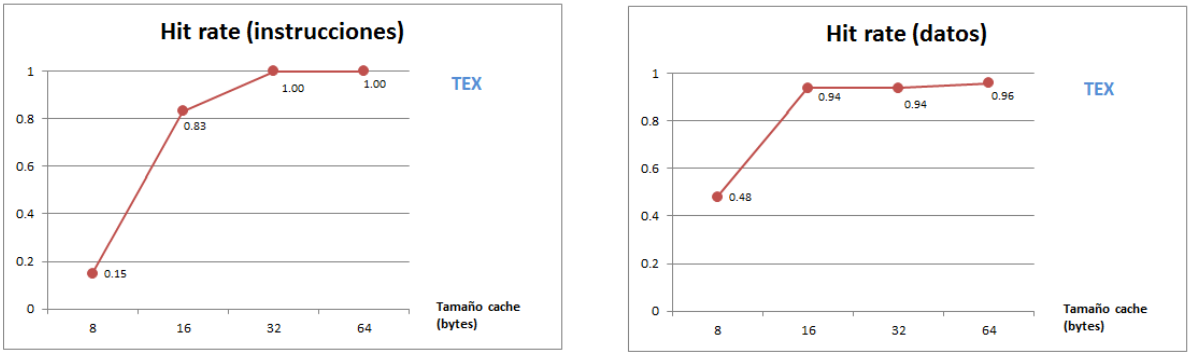


Figura 2: Tasa de hit por tamaño del cache en TEX

Finalmente en la Figura 3 podemos ver que para el conjunto CC todavía se necesitan tamaños más grandes caché para alcanzar tasas altas de hit y nuevamente las instrucciones presentan tasas de hit más altas.

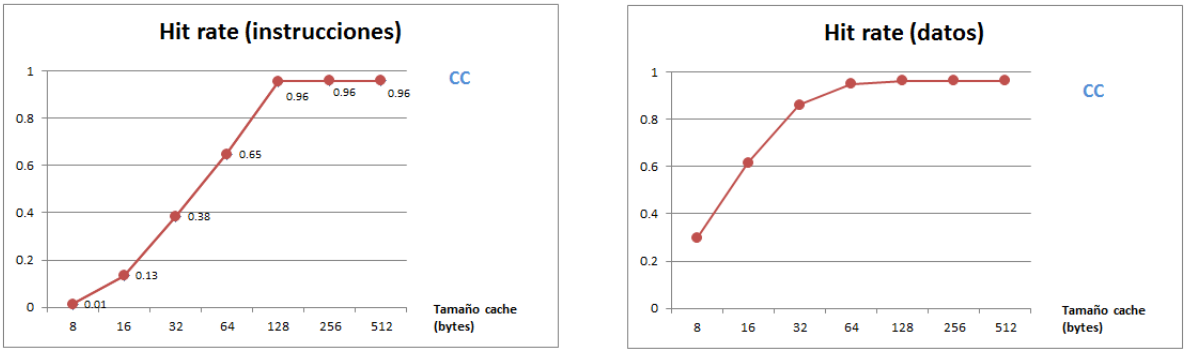


Figura 3: Tasa de hit por tamaño del cache en CC

1. Explicar qué hace este experimento y cómo funciona. Explicar la significancia de los factores en las gráficas de tasa de hit vs tamaño de los cachés.
   R = Este experimento lo que hace es ver hasta qué punto añadir un tamaño de caché más grande aumenta la tasa de hit. Uno esperaría que a más tamaño de caché, siempre tendremos más tasa de hit (obviamente hasta llegar al límite superior de tasa de hit, i.e. 1), pues estamos aumentando la capacidad de datos que se pueden almacenar. Sin embargo, aumentar el tamaño del cache también tiene desventajas graves, como lo son aumentar el hit time, el aumento en costo y la energía requerida.

2. ¿Cuál es el tamaño del conjunto total de instrucciones y de datos para cada una de las trazas?
   R = Para el conjunto SPICE, con 128 bytes se alcanzan niveles muy altos de hit rate en las instrucciones, mientras que para los datos necesitamos 64, es decir, menos. Para el

conjunto TEX con 32 bytes es suficiente para las instrucciones, mientras que para los datos necesitamos al menos 16 bytes. Finalmente para el conjunto CC necesitamos 128 para las instrucciones y 64 para los datos. Esto muestra que la decisión en cuanto al tamaño del caché varia tanto por conjunto de datos, como para instrucciones y datos; no hay un tamaño que siempre funcione bien, de modo que para escoger el tamaño del caché hay que realizar este tipo de ejercicios con nuestros datos, tomar en cuenta el presupuesto que tenemos y también el tamaño del bloque, la asociatividad y el bandwidth.

Impacto del tamaño del bloque

Ahora simularemos un caché de tamaño 8 K-bytes, asociatividad de 2, write-back cache, y política de write-allocate. Lo que queremos ver ahora es cómo impacta el tamaño del bloque de caché en la tasa de hit.

1. Explicar porqué la gráfica de tasa de hit vs el tamaño del bloque tiene la forma que tiene. En particular, explicar la relevancia de la localidad espacial en esta curva.
   R = La tasas de hit en las Figuras 4, 5 y 6 aumentan a una tasa muy pequeña y de hecho, en algún punto empiezan a disminuir. Esto se debe a que a pesar de que la localidad espacial nos ayuda a aumentar la tasa de hit (pues datos contiguos a un dato o instrucción que fue recientemente usado tenderán a usarse pronto, aumentando así la tasa de hit) si el tamaño del bloque es demasiado grande entonces pocos bloques cabrán en el caché, lo cual disminuirá eventualmente la tasa de hit. Podemos ver que para estos 3 conjuntos de datos, la localidad espacial no es tan relevante y por eso la tasa de hit no es muy sensible a cambios en el tamaño del bloque.

2. ¿Cuál es el tamaño óptimo del bloque para cada traza? (considerar instrucciones y datos por separado)
   R = No hay un tamaño de bloque óptimo para cada conjunto, pues esto también depende del tamaño del caché que tengamos y el miss penalty que se genere con el aumento en el tamaño del bloque (incluso también bloques de caché más grandes aumentan las conflict misses, especialmente en cachés pequeños). En los tres conjutnos vemos que no vale mucho la pena aumentar demasiado el tamaño del bloque, pues la ganancia en hit rate es mínima. Con un tamaño de bloque de 32 se alcanzan tasas de hit de +99 % en los 3 conjuntos y este tamaño parece razonable, sin embargo estas decisiones no se pueden tomar aisladas, hay que tomar en cuenta y medir los factores que ya mencionamos.

3. ¿Es diferente dicho tamaño óptimo del bloque para instrucciones y datos? ¿Qué te dice esto de la naturaleza de las referencias a instrucción vs a datos?
   R = No existe mucha diferencia entre el tamaño del bloque óptimo para instrucciones y datos, lo cual muestra que la localidad espacial se comporta de manera similar en ambos.

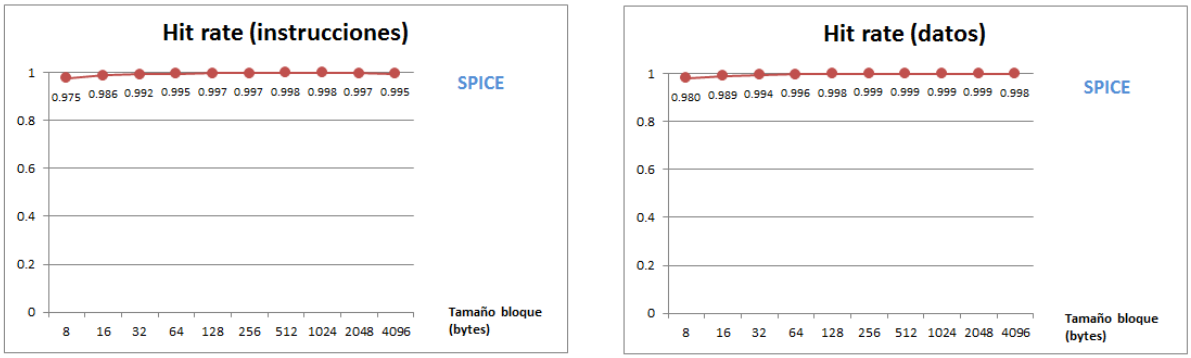


Figura 4: Tasa de hit por tamaño del bloque en Spice

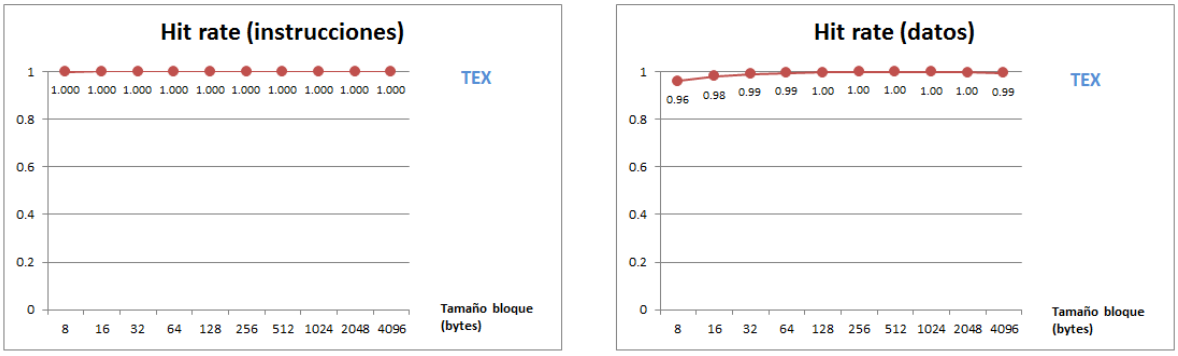


Figura 5: Tasa de hit por tamaño del bloque en TEX

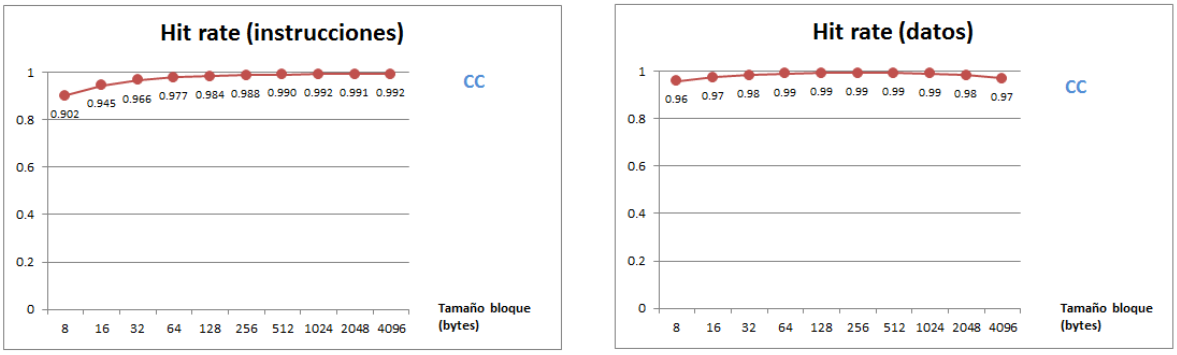


Figura 6: Tasa de hit por tamaño del bloque en CC

Impacto de la asociatividad

Ahora vamos a considerar el impacto de la asociatividad en la tasa de hit. Consideramos un tamaño de caché de 8K-bytes. El tamaño del bloque de 128 bytes, un caché write-back y una política de write-allocate. Se varió la asociatividad de 2-64 en potencias de 2.

1. Explicar porqué la gráfica de la tasa de hit vs la asociatividad tiene la forma que tiene.
   R = Mayor asociatividad implica tasas de hit más altas, sin embargo también puede aumentar el tiempo de hit y el consumo de energía. En la Figura 7, 8 y 9 podemos ver que la

asociatividad tiene poco impacto en la tasa de hit, en decir, es casi indistinguible cuánta asociatividad usemos.

2. ¿Existe diferencia entre las gráficas de referencia a instrucciones y datos? ¿Qué te dice acerca de la diferencia en el impacto de la asociatividad en instrucciones vs datos?
   R = Vemos que la diferencia es muy imperceptible, pero las instrucciones alcanzan niveles de hit ligeramente más altos que los datos en los tres conjuntos de datos. Además, para una asociatividad de 2, vemos que las instrucciones prácticamente ya alcanzaron el nivel más alto de hit, mientras que los datos todavía pueden incrementar su tasa de hit al aumentar el nivel de asociatividad.

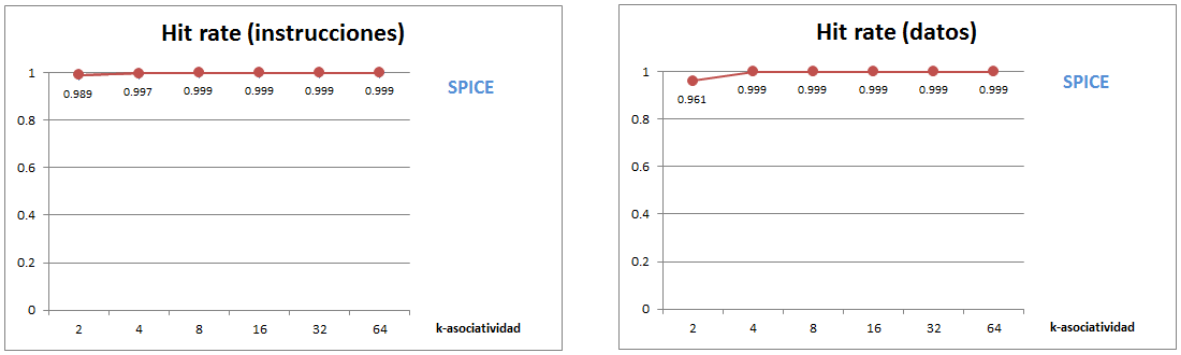


Figura 7: Tasa de hit por nivel de asociatividad en Spice

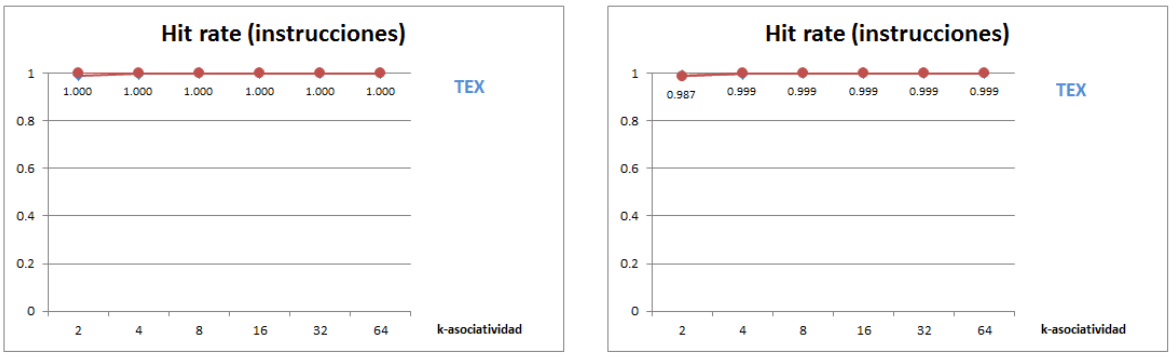
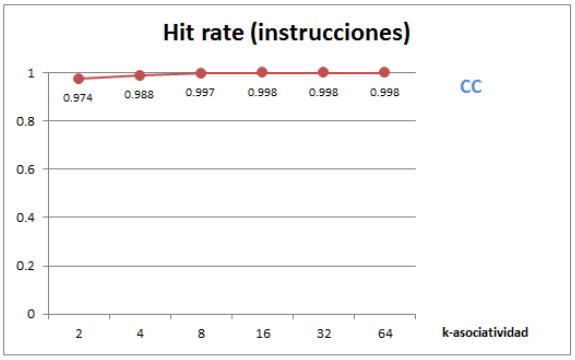


Figura 8: Tasa de hit por nivel de asociatividad en TEX



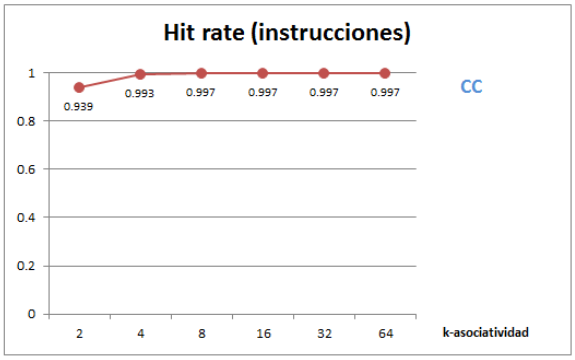


Figura 9: Tasa de hit por nivel de asociatividad en CC

Bandwidth de la memoria

Ahora simular un caché y comparar el tráfico generado por una política de write-through vs write-back.

La siguiente Figura 10 muestra las políticas de escritura cuando fijamos la política de asignación a *not write allocate* y podemos ver que no hay una diferencia en el tráfico (bandwidth).

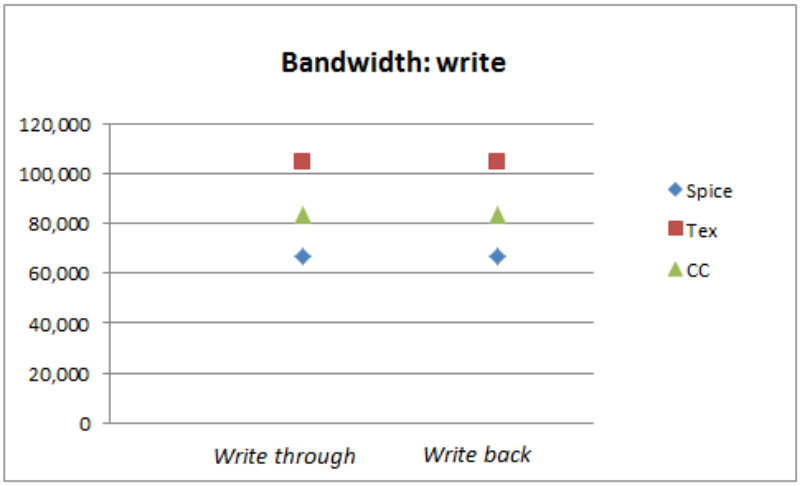


Figura 10: Bandwidth (copies back) por política de escritura

Fijó la política de escritura a write-back y se variarion las políticas de asignación y podemos ver que los tres conjuntos de datos aumentaron su tráfico en la poítica de no-write-allocate. Esto se debe a que los accesos a memoria a las escrituras se están duplicando cuando hay miss. Entonces cuando hay alto número de misses esta combinación de write-back y no-write-allocate producirá mayor tráfico o bandwidth. .

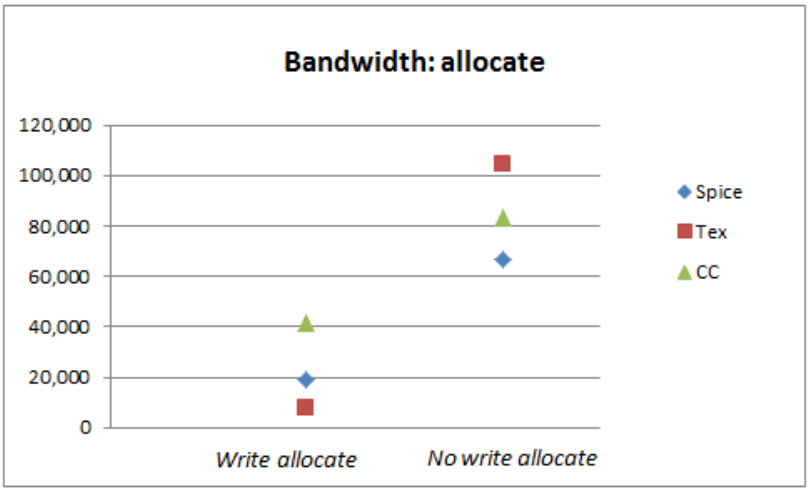


Figura 11: Bandwidth (copies back) por política de asignación

1. ¿Qué caché tiene el menor tráfico a memoria: el write-through o el write-back? ¿Por qué?

   R = Igual, pues las políticas sólo difieren en el timing de la escritura a la memoria principal, pero eventualmente ambas políticas lo hacen generando el mismo tráfico.

2. ¿Existe un escenario bajo el cual tu respuesta a la pregunta de arriba sería lo contrario? Explica.

   R = en el write-back, si los bloques de caché no se reemplazan pues este es el punto en el que se copia en la memoria. Entonces si no son reemplazados el tráfico en el write-back sería menor que en el write-through.

Ahora usa los mismos parámetros pero fija la política de escritura a write-back. Realiza el mismo experimento pero esta vez compara la política de write-allocate vs write-no-allocate.

1. ¿Cuál cache tiene el menor tráfico de memoria: el write-allocate o el write-no-allocate? **R** = El no write allocate tiene mayor tráfico.

2. ¿Hay un escenario bajo el cual la respuesta a la pregunta de arriba sería distinta? Explica. **R** Cuando hay muy pocos misses (casi inexistenteS) ambas políticas se comportarán igual, pues como vimos, lo que hace que el tráfico aumente es la cantidad de misses.

4. Conclusiones

El desarrollo de esta práctica ayudó a reafirmar e internalizar los conceptos teóricos y prácticos que ya habíamos visto sobre memoria caché.

Además ayudó a entender que la optimización del caché depende de muchos factores diferentes y que no hay una función de optimización objetivo ni una configuración *estrella*. En este sentido pudimos ver de una manera más tangible el impacto que los diferentes parámetros

del caché tiene en el desempeño del mismo y cómo existen trade-offs entre todos los factores significativos.

Sin embargo, más que los conceptos de caché y su diseño, la parte más dura personalmente fue la programación en C.

5. Referencias

[1] Patterson, David A., and John L. Hennessy. *Computer organization and design: the hardware/software interface*. Newnes, 2013.

[2] Patt, Yale N., Sanjay J. Patel, and J. Patel. *Introduction to computing systems: from bits and gates to C and beyond*. 2004.