Evaluación de desempeño de memoria cache mediante simulación por trazas

Arquitectura de Computadoras

Saúl Caballero
Ricardo Figueroa

Introducción

En esta práctica se realiza un simulador de cache que permite el uso de distintos parámetros en términos de tamaño de cache, asociatividad, tamaño de bloque, uso de caches separados para datos e instrucciones (I-D) y políticas de escritura (write-back o write-through y write-allocate o no write-allocate). La política de reemplazo utilizada en caso de tener una asociatividad mayor a uno es Last Recently Used. Los resultados obtenidos muestran que el simulador funciona y son similares a lo que se espera de una memoria cache.

Marco Teórico

A continuación se explican los parámetros que el revisor de este documento debe conocer y que impactan en el desempeño de una memoria cache y su costo:

El tamaño de una memoria cache influye directamente en el hit rate debido a que se puede guardar un mayor número de localidades de memoria principal, sin embargo tamién incrementa el hit time. La asociatividad también ayuda en el hit rate, sin embargo es costosa en términos de recursos en las búsquedas. El tamaño de bloque reduce el miss rate hasta cierto punto afectado por el principio de localidad. Las combinaciones más comunes en términos de política de escritura son write-back y write-allocate y write-through con write no allocate.

Durante un hit en un acceso de escritura, la política write-back escribe únicamente en cache y marca como dirty para posteriormente ser escrita en memoria principal durante un reemplazo. La politica write-through escribe directamente en memoria principal y en cache, esto implica mayor tráfico y ancho de banda en memorio pero su diseño es más sencillo y la consistencia con la memoria principal en permanente.

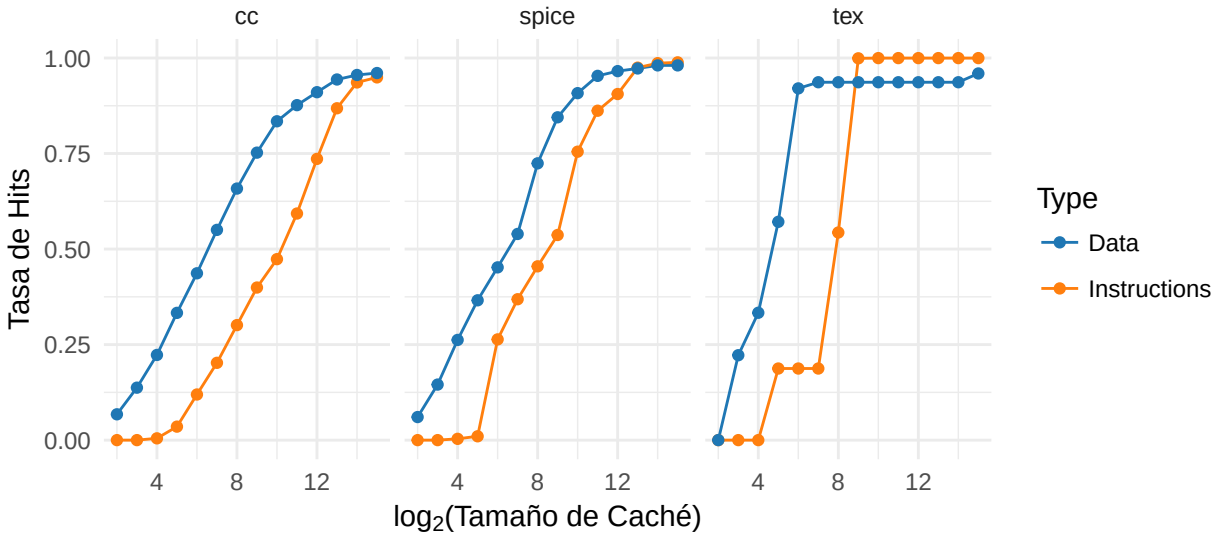
Durante un miss en un acceso de escritura, la política write-allocate trae el bloque completo a cache y modifica únicamente en cache, mientras que el no-write-allocate solo escribe en memoria.

Evaluación de desempeño

A continuación se evaluará el desempeño de las propiedades que puede tener un Caché.

Tamaño de Cache

El experimento consiste en evaluar el desempeño con distintos tamaños de caché. Se usa un caché con memoria separada, tamaño de bloque de 4 bytes, política **write-back** y **write-allocate**, y para evitar que en el análisis exista un impacto de misses por conflicto, se usa una caché que es totalmente asociativa. Los tamaños de caché van desde 2^2 hasta 2^{13} bytes.



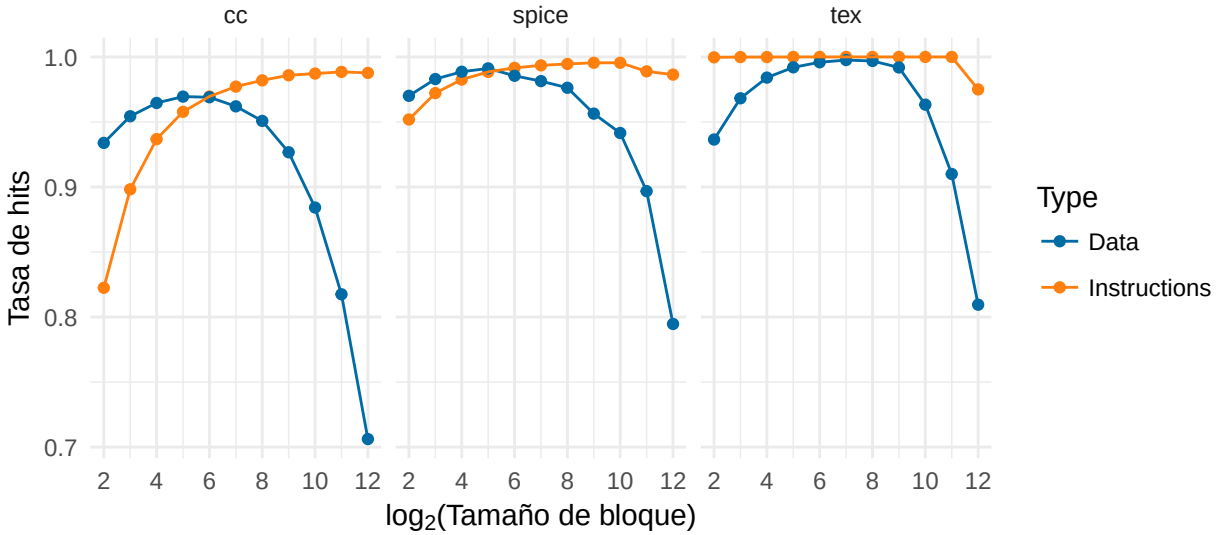
Se puede observar que el impacto del tamaño de caché tiene una relación no decreciente con la tasa de hits, sin embargo, a partir de un cierto tamaño, ya no tiene impacto sobre la tasa de hits.

En la siguiente tabla se puede observar las localidades totales usadas por las trazas para datos e instrucciones:

Archivo	Tipo	Localidades
cc	Dato	242,661
cc	Instrucción	757,341
spice	Dato	217,237
spice	Instrucción	782,764
tex	Dato	235,168
tex	Instrucción	597,309

Tamaño del Bloque

En este experimento, se juega con el tamaño del bloque el cual va desde 2^2 bytes hasta 2^{12} bytes. Se usa un caché con memoria separada cada una de 8 Kbytes, asociatividad de 2 y política **write-back** y **write-allocate**. A continuación se muestran el impacto del tamaño del bloque en la tasa de hits:



Se puede observar que la relación del tamaño de bloque con la tasa de hits tiene una función concava, por lo que podemos decir que existe un tamaño óptimo de tamaño de bloque, dependiendo de la traza usada. Esta forma se

puede explicar de la siguiente forma:

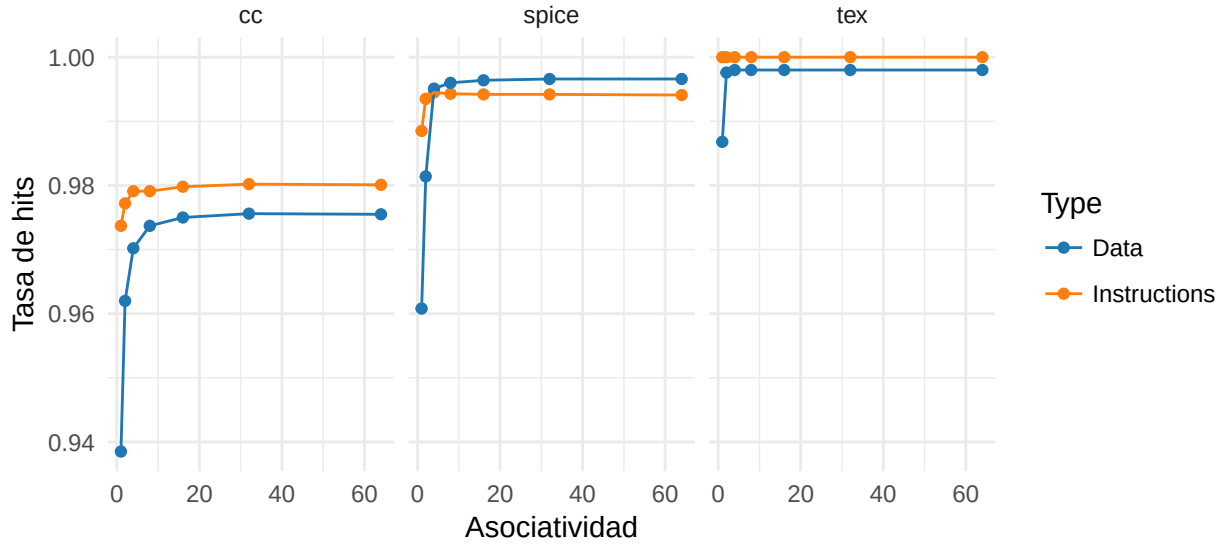
- Parte creciente: el principio de localidad nos dice que localidades cercanas se van a usar de forma muy seguida, por lo que entre más cercanas las localidades mayor probabilidad de que sean usadas. Por lo tanto cuando aumenta el tamaño de bloque aumenta la tasa de hits.
- Parte decreciente: cuando el tamaño de bloque es demasiado grande, estamos metiendo localidades al caché que es probable que no vayamos a usar en un futuro cercano, por lo que la tasa de hits comienza a disminuir.

El tamaño óptimo de bloque para las trazas es el siguiente:

Traza	Tipo	Tamaño Óptimo
cc	Datos	32
cc	Instrucciones	2048
spice	Datos	32
spice	Instrucciones	1024
tex	Datos	128
tex	Instrucciones	4-2048

Asociatividad

En este experimento, se juega con la asociatividad del caché la cual va desde 1 bytes hasta 64 bytes. Se usa un caché con memoria separada cada una de 8 Kbytes, tamaño de bloque de 128 bytes y política **write-back** y **write-allocate**. A continuación se muestran el impacto de la asociatividad en la tasa de hits:



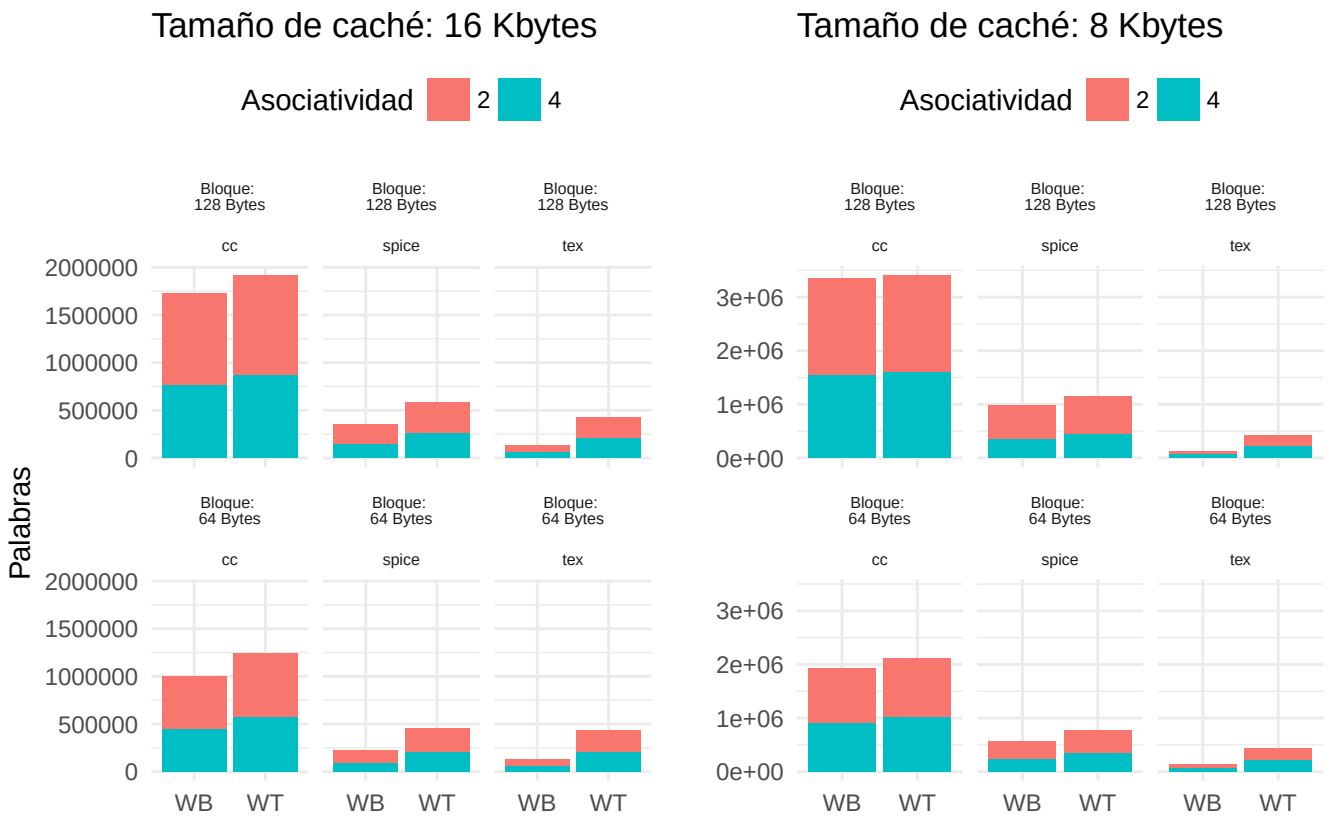
Se observa que al aumentar la asociatividad se encuentra una relación no decreciente con la tasa de hits; sin embargo, llega un punto donde la asociatividad ya no ayuda mucho. Además al aumentar la asociatividad, se vuelve muy costosa la búsqueda.

También se puede apreciar que la tasa de hits de las instrucciones es mayor que la de los datos, excepto en la traza de *spice*. Además se observa que la diferencia entre datos e instrucciones se queda estable a partir de un cierto punto.

Ancho de Banda en Memoria

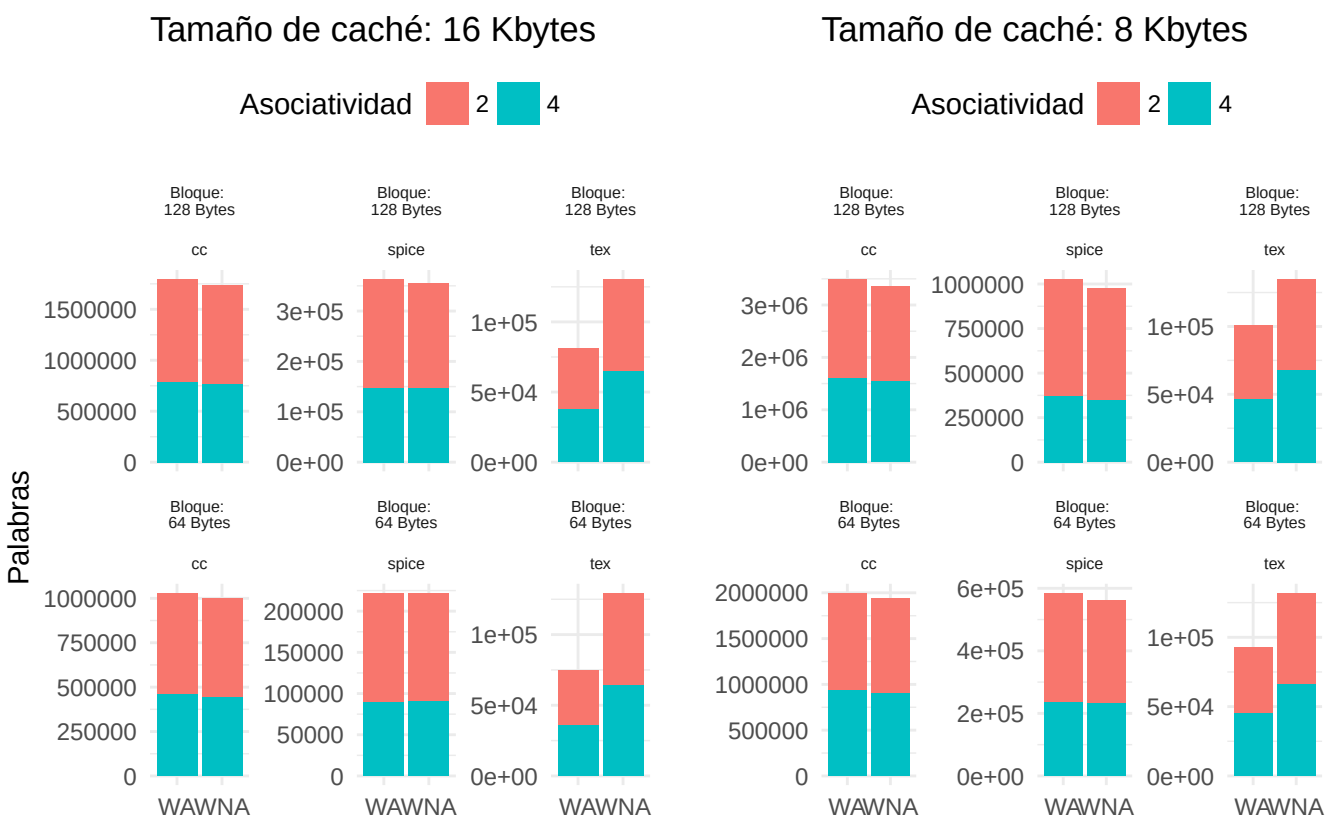
En este experimento, se ve que ocurre con el ancho de banda en memoria. Se usa un caché con memoria separada cada una de 8 Kbytes o 16 Kbytes, tamaño de bloque de 64 y 128 bytes, asociatividad 2 y 4, y política **write-no-allocate**. Se usa la suma de copies back y demand fetch como media de ancho de banda en memoria.

Comparación write-through contra write-back



Se puede observar que la política **write-through** es la que tiene le tránsito de palabras más alta independientemente de los casos analizados. La explicación es sencilla, pues en este caso siempre que hace un proceso guarda directamente en memoria por lo que los copies back aumentan. Esta respuesta puede ser con programas donde la mayoría de operaciones fueran de escritura en lugar de lectura.

Comparación write-allocate contra write-no-allocate



En este caso no se puede observar un patrón claro de cuál tiene menor tránsito. e puede esperar una mayor cantidad de demand fetches en el caso de **write-allocate** porque en los misses se escribe todo un bloque y en **write-no-allocate** solo escribe una palabra. La volatilidad viene de la política de **writes-back**.

Conclusión

El aprendizaje derivado de la práctica fue muy valioso y ayudo a los autores a entender con profundidad el funcionamiento de una memoria cache y el impacto que tiene los distintos parámetros. Consideramos que la implementación también fue exitosa en términos de desarrollo de habilidades generales de computación (programación en C).