Universidad de Costa Rica
Escuela de Ingeniería Eléctrica

IE0521 Estructuras de computadoras digitales II

Proyecto programado: Simulación de un cache

Objetivos

Analizar el redimiendo de distintas configuraciones de memoria caché por medio de simulaciones en un lenguaje de alto nivel.

Descripción del Proyecto

Cada grupo de estudiantes deberán realizar un simulador de memoria caché, en C\C++, que permita obtener métricas de rendimiento utilizando un *trace* real (lista de accesos a memoria). Además de obtener estos resultados, los estudiantes deberán realizar pruebas a nivel de unidad utilizando el framework gtest de google, para garantizar el correcto funcionamiento del sistema.

Como punto se partida, se cuenta con el siguiente repositorio de git (cache-git). Dicho repositorio cuenta con las descripciones de las funciones, pruebas, estructuras, cmakes entre otros. La solución propuesta deberá ajustarte a este marco de trabajo y cumplir con el estándar de código establecido en clase.

Parte 1: Simulación del trace para un cache L1

El cache a simular debe ser de tipo *write-allocate*, *write-back* y soportar tres políticas de remplazo: LRU, NRU y SRRIP(HP) [1]. Para RRIP el intervalo de re-refencia se tomará M=1 si la asociatividad es menor o igual a dos y M=2 si la asociatividad es mayor a dos. Nuevos bloques en el cache se deben insertar utilizando un valor de RRPV de $2^M - 2$.

El diseño deberá ser parametrizable, es decir, por medio de argumentos se indicará las características del cache a simular.

Los parámetros de entrada serán los siguientes:

1. Tamaño del cache en KB (-t)
2. Tamaño de la línea en bytes (-l)
3. Asociatividad (-a)
4. Política de remplazo (-rp)

Como entrada a la simulación se utilizará el trace mcf.trace.gz que es parte del benchmark SPEC CPU 2006. Este puede ser descargado del siguiente enlace: (TraceLink).

El formato del *trace* es el siguiente:

- LS : un valor de cero indica un *load* y un uno un *store*.
- Dirección: La direcciones son valores de 8 caracteres en hexadecimal.

LS	Dirección	IC
# 0	7fffed80	1
# 0	10010000	10
# 0	10010060	3
# 0	10010030	4
# 0	10010004	6
# 0	10010064	3
# 0	10010034	4

- **IC:** es el número de instrucciones que se ejecutaron entre la referencia a memoria anterior y la actual (contando la instrucción actual). Este valor se utilizará para para calcular el CPU time.

El programa deberá poder ejecutarse de la siguiente forma:

gunzip -c mcf.trace.gz | cache -t < # > -a < # > -l < # > -rp < # >

La salida de la simulación deberá imprimirse en consola con el siguiente formato:

Cache parameters:	
Cache Size (KB):	16
Cache Associativity:	2
Cache Block Size (bytes):	32
Simulation results:	
Overall miss rate:	0.00
Read miss rate:	0.00
Dirty evictions:	00000
Load misses:	00000
Store misses:	00000
Total misses:	00000
Load hits:	00000
Store hits:	00000
Total hits:	00000

Parte 2: Pruebas L1

Además de poder correr sobre un trace real, los bloques funcionales deberán ser probados utilizando el framework gtest de google. La descripción del test plan se encuentra en el archivo L1cache_test.cpp. Todas las pruebas deberán realizar lo solicitado y retornar los valores esperados.

Parte3: Optimizaciones avanzadas y pruebas

Se deberán implementar dos optimizaciones avanzadas, se podra elegir entre las siguientes tres: un *Victim cache*, un cache multinivel y un *prefetcher OBL tagged*.

Las optimizaciones avanzadas no se evaluarán sobre el trace real, sino sobre casos de prueba provistos por la profesora, por cada optimización avanzada el grupo de estudiantes deberá generar al menos dos pruebas para validar el funcionamiento.

Caché multinivel

Se deberá simular dos niveles de caché: primer nivel (L1) y segundo nivel (L2). El tamaño del caché de segundo nivel deberá ser cuatro veces el tamaño de L1 y su asociatividad se definirá como el doble de la del primer nivel. Además, los niveles L1 y L2 utilizan una política de remplazo LRU, son inclusivos y *write-throuth*, mientras que entre L2 y memoria se utiliza una política writeback.

Victim Cache

El *victim cache* será de 16 entradas totalmente asociativo y con una política de remplazo FIFO.

OBL

Se implementará un mecanismo de *prefetch* de tipo "un bloque adelante con etiqueta" o OBL (por siglas en inglés) en un cache de primer nivel con un política de remplazo LRU.

Bajo este esquema, cada bloque de memoria tiene una etiqueta que determina si una referencia al bloque causa o no el *prefetch* del siguiente bloque $B + 1$.

Cuando un bloque, B, es solicitado por el CPU y no se encuentra en el cache (*cache miss*), este se trae al cache con la etiqueta de *prefetch* en 0, este *miss* dispara la solicitud del bloque $B+1$, pero en este caso su etiqueta se marca como 1. Si el bloque B se solicita nuevamente (*cache hit*), como la etiqueta de *prefetch* es 0 no se dispara ninguna acción de prefetch adicional, solo se actualiza los bits asociados a la política de remplazo. Ahora, si el CPU solicita el bloque $B+1$, como su etiqueta de prefetch era 1 esta se actualiza a 0 y dispara el prefetch del siguiente bloque en este caso $B+2$.

En caso de que el bloque solicitado por medio del mecanismo de prefetch ya se encuentre en el cache, no se deberá realizar ninguna acción sobre este bloque.

Evaluación y entrégables

El proyecto tiene un valor de un 25 % y se realizará en grupos de 3 personas, como máximo. El desglose de la calificación se muestra en el Cuadro 1.

La entrega se realizará de dos formas, primero por medio de un *merge request* de un solo *commit* al repositorio base, y segundo a través de mediación virtual, ambos tienen que estar actualizados antes del día **4 de noviembre** a media noche.

Parte de la revisión del proyecto se hará por medio de sesiones virtuales, el 5 y 7 de noviembre en horario a convenir con la profesora, en esta revisión se revisará el código de forma visual, así como en ejecución. Los estudiantes deberán demostrar la compresión del tema en general, así como de la solución planteada al problema.

Otras consideraciones

- Se asume que el estudiante tiene conocimientos de programación, por lo que cualquier refrescamiento del lenguaje y herramientas es responsabilidad del estudiante.
- Se utilizará como sistema operativo base Ubuntu 18.04 en adelante, por lo que el código y sus instrucciones de construcción deben poder ser ejecutadas en este sistema operativo.
- El código debe ser legible y estar comentado apropiadamente. Elija un formato para los comentarios y un formato para el nombre de las variables. Sea consistente con los formatos elegidos a lo largo de su programa.

Cuadro 1: Desglose de la evaluación

Item	Puntaje Máximo
P1: Resultados correctos LRU	8
P1: Resultados correctos NRU	8
P1: Resultados correctos RRIP	8
P1: Apego a las restricciones iniciales	2.5
P1: Eficiencia (Ejecución menor a dos minutos)	5
P1: Código	2.5
P2: Ejecución exitosa de las pruebas	10
P2: Pruebas acordes a lo especificado	10
P3: Implementación Opt Avanzada 1	12
P3: Implementación Opt Avanzada 2	12
P3: Pruebas Opt Avanzada 1	5
P3: Pruebas Opt Avanzada 2	5
P3: Pruebas Opt Avanzada 1	6
P3: Pruebas Opt Avanzada 2	6

- La nota máxima para un programa que no compila sera de 40 %.
- **No se aceptaran entregas luego de la fecha establecida**.

Referencias

[1] Jaleel, Aamer, Kevin B Theobald, Simon C Steely Jr y Joel Emer: *High performance cache replacement using re-reference interval prediction (RRIP)*. En *ACM SIGARCH Computer Architecture News*, volumen 38, páginas 60–71. ACM, 2010.