Universidad de Costa Rica
Escuela de Ingeniería Eléctrica

IE0521 Estructuras de computadoras digitales II

Proyecto programado I Parte

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.

Características del cache

El cache a simular debe ser de tipo *write-allocate*, *write-back* y soportar dos políticas de remplazo: LRU 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 $2^M - 2$.

Simulación del trace

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*.

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

- Dirección: La direcciones son valores de 8 caracteres en hexadecimal.
- 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:	
CPU time (cycles):	0000
AMAT(cycles):	0000
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

Para los cálculos de redimiendo asuma lo siguiente:

- Todas las instrucciones excepto los *loads* toman un ciclo de reloj.
- El miss penalty es de 20 ciclos de reloj.
- Un load puede tomar un ciclo de reloj, en caso de un *hit*, o 1 + n ciclos en caso de un *miss*, donde n es el *miss penalty*.

Para el *trace* mostrado anteriormente las 31 instrucciones tomarían 111 ciclos de reloj, suponiendo un *miss penalty de 20*, cuatro *misses* y tres *hits*.

Pruebas

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.

Evaluación y entrégales

Esta primera parte del proyecto tiene un valor de un 12 % y se realizará en grupos de 2 personas, como máximo. Se calificará: que el diseño sea parametrizable, que los resultados de la simulación sean correctos y que el programa sea eficiente (si el programa tarda más de 30 s realizando la simulación mcf la nota máxima será de 85). Las pruebas deberán estar completas, apegarse al test plan y evidenciar el correcto funcionamiento del programa, un proyecto sin las pruebas tendrá una nota máxima de 70 %.

La entrega se realizará por medio de un repositorio de git, este deberá incluir una sección *Readme* con la descripción del programa, así como las instrucciones o dependencias para ejecutarlo. En mediación virtual, se habilitará un espacio donde el estudiante deberá proveer la dirección a su repositorio, antes del día **2 de mayo** a medio día.

Parte de la revisión del proyecto se hará de forma presencial, el 2 de mayo 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 16.04 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.
- Cada función del programa debe contener un encabezado indicando una descripción general, los parámetros de entrada y los de salida.
- Tal como lo indica la carta al estudiante, la nota máxima para un programa que no compila sera de 40 %.
- Cada día de retraso en la entrega reducirá la nota máxima en 5 %.

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.