Universidad de Costa Rica
Escuela de Ingeniería Eléctrica

IE0521 Estructuras de computadoras digitales II

Proyecto Programado III 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 deberá realizar un simulador de memoria caché, en C\C++, que permita obtener métricas de rendimiento utilizando un *trace* real (lista de accesos a memoria). El sistema deberá ser validado a nivel de unidad utilizando el framework gtest de google. Además, se deberá optimizar la solución propuesta por medio de optimizaciones de paralelización.

Características del caché

Se simulará sistema de memoria para dos procesadores. Cada procesador contará con un caché L1 privado y compartirán un caché L2. Los cachés de L1 y L2 se asumen como inclusivos y deberán implementar un protocolo de coherencia MESI y MSI.

Caché multinivel

Se deberá simular dos niveles de caché: primer nivel (L1) y segundo nivel (L2). El usuario podrá definir el tamaño y la asociatividad de L1, mientras que, el tamaño y asociatividad de L2 serán definidos a partir de las características del caché de primer nivel. 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*.

Simulación del trace

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

Los parámetros de entrada serán los siguientes:

1. Tamaño del caché en KB (-t)
2. Tamaño de la línea en bytes (-l)
3. Asociatividad (-a)
4. Protocolo de coherencia (MSI — MESI) (-cp)

Como entrada a la simulación se utilizará los *traces* mcf.trace.gz y art.trace.gz que son parte del benchmark SPEC CPU 2006. Estos pueden ser descargados del siguiente enlace: (TraceLink).

Asuma que el trace corresponde a las instrucciones ejecutadas por los dos CPUs, si el **(número de acceso) mod 4 == 0** corresponde a una solicitud a memoria del CPU1, sino es un acceso del CPU2.

El formato del *trace* es el siguiente:

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

- LS : un valor de cero indica un *load* y un uno un *store*.
- 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).

El programa deberá poder ejecutarse de la siguiente forma:

gunzip -c mcf.trace.gz | cache -t < # > -a < # > -l < # > -cp < $msi|mesi$ >

La salida de la simulación deberá imprimirse en consola. La tabla **??** muestra los resultados esperados para la simulación.

Tabla 1: Formato para los resultados de simulación de un caché multinivel

Cache parameters:	
L1 Cache Size (KB):	16
L2 Cache Size (KB):	64
Cache L1 Associativity:	2
Cache L2 Associativity:	4
Cache Block Size (bytes):	32
Coherence protocol:	MESI
Simulation results:	
Overall miss rate:	0.00
CPU1 L1 miss rate:	0.00
CPU2 L1 miss rate:	0.00
Coherence Invalidation CPU1	0.00
Coherence Invalidation CPU2	0.00

Pruebas

Además de poder correr sobre un trace real, los estudiantes deberán plantear dos pruebas, utilizando gtest, que permitan validar el correcto funcionamiento del sistema. Se deberá entregar un documento en pdf con el plan de pruebas.

Paralelización

Por medio de la librería *pthreads*, se deberá optimizar la solución planteada y demostrar que obtiene una mejora en el redimiendo de la ejecución de los traces.

Evaluación y entrega

Esta tercera parte del proyecto tiene un valor de un 25 % 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. Las pruebas deberán estar completas, apegarse al test plan planteado y evidenciar el correcto funcionamiento del programa, un programa sin 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 los estudiantes deberán adjuntar el plan de pruebas y proveer la dirección a su repositorio, antes del día **27 de junio** a medio día.

Parte de la revisión del proyecto se hará de forma presencial, el 27 de junio en horario a convenir con la profesora, en esta 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 30 %.
- Cada día de retraso en la entrega reducirá la nota máxima en 5 %.