UNIVERSIDAD DE COSTA RICA

EIE Escuela de Ingeniería Eléctrica

IE 0521 – Estructuras de Computadoras Digitales II

Tarea #4: Simulador de Memoria Caché

La nota final asignada al rubro de tarea, se asignará mediante un promedio ponderado de las calificaciones obtenidas en todas las tareas asignadas. El peso asignado a esta tarea será: **5**.
Esta tarea se realizará en grupos de máximo 3 personas.

Objetivos

- Analizar el rendimiento de distintas configuraciones de caché por medio de simulaciones en un lenguaje de alto nivel.
- Analizar el rendimiento de configuraciones de caché multinivel por medio de simulaciones en un lenguaje de alto nivel.

Descripción de la asignatura

Cada grupo deberá programar un simulador de memoria caché, en `C\C++` o `Python`, que permita obtener métricas de rendimiento utilizando como entrada un "`trace`" donde se indica los accesos a memoria y si es una lectura o escritura.

Datos a utilizar

Se utilizarán varios archivos "`trace`" que contienen información de alrededor 1.000.000 de accesos de memoria cada uno. Estos datos fueron obtenidos mediante la ejecución de aplicaciones que forman parte de la suite de benchmarks SPEC CPU2000. Los trace pueden ser descargados desde Mediación Virtual o usando este link. Es recomendable descargar y procesar el archivo sin descomprimir (en formato `.gz`).
El archivo sigue el siguiente formato:

```
r 56ecd8
r 47f639
r 7ff0001ff
w 47f63e
r 4817ef
r 7d5ab8
```

En cada línea, se encuentran dos datos: el primero es un carácter indica si el acceso fue lectura (**r**ead) o escritura (**w**rite), y el segundo indica la dirección accedida en memoria (en hexadecimal).

Aunque el trace indica de forma diferente las lecturas de las escrituras, usted no deberá hacer ninguna diferencia entre ellas.

Parte 1. Caché de un único nivel

En esta parte se implementará una clase “Cache” la cual hace el manejo de las lecturas y escrituras que se encuentran en el trace. La clase debe utilizar parámetros, que permitan modificar el tamaño y configuración del caché.

Parametrización del código

El programa deberá ser diseñado de forma parametrizable, es decir, las características del caché a simular deberán ser indicadas por medio de argumentos (no es necesario utilizar los argumentos con “-<Nombre del parámetro>” siempre y cuando se especifique claramente en la documentación la forma correcta de correr el programa).
Los argumentos de entrada serán los siguientes:

1. **Capacidad del caché en kB (-s).** Será un número entero que indica la capacidad total del caché. Se probará el código únicamente con potencias de 2 (2, 4, 8,16, 64, 128, etc)
2. **Asociatividad del caché (-a).** Será un número entero que indica la cantidad de “ways” en cada set. Por ejemplo 1 indica mapeo directo, 8 indica 8-way set associative, etc. Se probará el código únicamente con potencias de 2 (1, 2, 4, 8, 16, etc).
3. **Tamaño del bloque en bytes (-b).** Será un entero que indica cuántos bytes se tienen por bloque en el caché. Se probará el código únicamente con potencias de 2 (2, 4, 8,16, 64, 128, etc).
4. **Política de reemplazo (-r).** Un char que indica la política para expulsar bloques del caché. Una “*l*” (ele) indica LRU, una “*r*” (erre) indica aleatoria.

Código Base

Para facilitar el inicio de la tarea se incluyen varios archivos que sirven de base. Estos son:

1. `cache_sim.py:` Es el archivo a ejecutar. Contiene comandos para leer los argumentos desde la consola, leer el trace y procesarlo, dependiendo de la configuración de caché seleccionada.
2. `cache.py:` Es el esqueleto de una clase que implementa el caché.

Cada grupo puede modificar estos archivos para implementar su caché o puede empezar su código desde cero.

Despliegue de Resultados

Al ejecutar el programa, se deberá imprimir la siguiente información en una sóla línea, separada por espacios:

1. Cantidad total de misses.
2. Miss rate total (Total de misses / Total de accesos).

En esta parte de la tarea deberá recolectar resultados de ejecutar su código alrededor de 450 veces, por lo que se sugiere buscar formas de automatizar el lanzamiento de las corridas y/o lanzar múltiples simulaciones en paralelo, pero sobre todo, se sugiere iniciar la tarea con suficiente antelación a la fecha de entrega. Aproveche el formato de impresión de los resultados.

Análisis de resultados

Se procederá a analizar el comportamiento del caché ante variaciones en sus distintos parámetros. En cada experimento incluya las gráficas solicitadas y analice los resultados obtenidos.

Efecto del tamaño del caché

Varíe la capacidad del caché entre 8kB y 128kB, utilizando las potencias de 2 que hay en medio únicamente *i.e* 8, 16, 32, 64 y 128. Mantenga la asociatividad en 8-way, el tamaño de bloque en 64B y la política de reemplazo en LRU.
Para cada una de esas 5 configuraciones, simule el caché para **todos** los traces brindados, tabule los datos en una hoja de cálculo y **grafique el miss rate total promedio, usando la media geométrica** (a través de todos los trace) vs tamaño del caché. Analice si los datos obtenidos concuerdan con el comportamiento esperado.
En específico, grafique y analice el comportamiento del trace "465.tonto-1769B.trace.txt.gz".

Efecto de la asociatividad del caché

Varíe la asociatividad del caché entre mapeo directo y 16-way, utilizando las potencias de 2 que hay en medio únicamente *i.e* 1, 2, 4, 8, 16. Mantenga la capacidad en 32kB, el tamaño de bloque en 64B y la política de reemplazo en LRU.
Para cada una de esas 5 configuraciones, simule el caché para **todos** los traces brindados, tabule los datos en una hoja de cálculo y **grafique el miss rate total promedio, usando la media geométrica** (a través de todos los trace) vs asociatividad del caché. Analice si los datos obtenidos concuerdan con el comportamiento esperado.
En específico, grafique y analice el comportamiento del trace "470.lbm-1274B.trace.txt.gz".

Efecto del tamaño del bloque en el caché

Varíe el tamaño del bloque en el caché entre 16B y 128B, utilizando las potencias de 2 que hay en medio únicamente *i.e* 16, 32, 64, 128. Mantenga la capacidad en 32kB, la asociatividad en 8-way y la política de reemplazo en LRU.

Para cada una de esas 4 configuraciones, simule el caché para **todos** los traces brindados, tabule los datos en una hoja de cálculo y **grafique el miss rate total promedio, usando la media geométrica** (a través de todos los trace) vs tamaño del bloque. Analice si los datos obtenidos concuerdan con el comportamiento esperado.

En específico, grafique y analice el comportamiento del trace "401.bzip2-226B.trace.txt.gz".

Efecto de la política de reemplazo del caché

Varíe la política de reemplazo del caché (LRU y aleatoria). Mantenga la capacidad en 32kB, la asociatividad en 8-way y el tamaño de bloque en 64B.

Para cada una de esas 2 configuraciones, simule el caché para todos los traces brindados, tabule los datos en una hoja de cálculo y **grafique el miss rate total promedio, usando la media geométrica** (a través de todos los trace) vs política de reemplazo. Analice si los datos obtenidos concuerdan con el comportamiento esperado.

Guía de resultados

Para verificar si su implementación es correcta, abajo se presentan los resultados obtenidos para un caché de 128kB, con asociatividad de 16 ways, bloques de 64B y utilizando LRU:

Trace	Total Misses	Miss rate total	Trace	Total Misses	Miss rate total
400.perlbench-41B	234	0.023%	**453.povray-887B**	2032	0.203%
401.bzip2-226B	3912	0.391%	**454.calculix-104B**	418	0.042%
403.gcc-16B	21331	2.133%	**456.hmmer-191B**	2927	0.293%
410.bwaves-1963B	11523	1.152%	**458.sjeng-1088B**	1195	0.120%
416.gamess-875B	1050	0.105%	**459.GemsFDTD-1169B**	17753	1.775%
429.mcf-184B	92791	9.279%	**462.libquantum-1343B**	17649	1.765%
433.milc-127B	22083	2.208%	**464.h264ref-30B**	555	0.056%
435.gromacs-111B	3087	0.309%	**465.tonto-1769B**	28139	2.814%
436.cactusADM-1804B	2854	0.285%	**470.lbm-1274B**	24494	2.449%
437.leslie3d-134B	3968	0.397%	**471.omnetpp-188B**	16488	1.649%
444.namd-120B	991	0.099%	**473.astar-153B**	2866	0.287%
445.gobmk-17B	1411	0.141%	**481.wrf-1170B**	2760	0.276%
447.dealII-3B	7	0.001%	**482.sphinx3-1100B**	10652	1.065%
450.soplex-247B	54998	5.500%	**483.xalancbmk-127B**	20637	2.064%

Parte 2. Caché multinivel

Utilizando la clase “Cache” de la primera parte (aunque será necesario modificarla ligeramente), se implementará un simulador de caché multinivel.

Parametrización del código

El programa deberá ser diseñado de forma parametrizable, es decir, las características del caché a simular deberán ser indicadas por medio de argumentos (no es necesario utilizar los argumentos con “-<Nombre del parámetro>” siempre y cuando se especifique claramente en la documentación la forma correcta de correr el programa).
Los argumentos de entrada serán los siguientes:

1. **Capacidad del caché L1 en kB (--l1_s).** Será un número entero que indica la capacidad total del caché L1. Se probará el código únicamente con potencias de 2 (2, 4, 8,16, 64, 128, etc)
2. **Asociatividad del caché L1 (--l1_a).** Será un número entero que indica la cantidad de “ways” en cada set del caché L1. Por ejemplo 1 indica mapeo directo, 8 indica 8-way set associative, etc. Se probará el código únicamente con potencias de 2 (1, 2, 4, 8, 16, etc).
3. **Presencia de nivel de caché L2 (--l2) y (--l3).** Estas son banderas que, en caso de no ser incluidas en el comando de ejecución, indican que no habrá L2 o L3, respectivamente. No debería suceder el caso de L3 sin L2, su programa deberá generar un error si este caso ocurriese.
4. **Capacidad y asociatividad de los cachés L2 y L3 (--l2_s, --l2_a, --l3_s y --l3_a).** Definidos de la misma forma que para L1.
5. **Tamaño del bloque en bytes (-b).** Será un entero que indica cuántos bytes se tienen por bloque en el caché, el tamaño de cada bloque será el mismo para todos los niveles de caché. Se probará el código únicamente con potencias de 2 (2, 4, 8,16, 64, 128, etc).
6. **Política de reemplazo (-r).** Un char que indica la política para expulsar bloques del caché. Únicamente se probará con la política LRU.

Para facilitar el inicio de la tarea se incluye un archivo que sirve de base:

1. `cache_sim.py:` Es el archivo a ejecutar. Contiene comandos para leer los argumentos desde la consola, leer el trace y procesarlo, dependiendo de la configuración de caché seleccionada..

El otro archivo que será necesario es el `cache.py` que desarrollaron para la Parte 1. Para esta parte se recomienda modificar la función `access`, la cual recibe el tipo de acceso y la dirección de memoria buscada, y realiza todo el procesamiento del acceso (búsqueda y traída de los datos al caché en caso de no estar presentes), para que devuelva una variable tipo booleana

que indica si el acceso fue un miss (True = miss, False = hit). Esto le permitirá saber si debe continuar en L2 o en L3 la búsqueda.

Cada acceso al caché seguirá el siguiente proceso:

1. Revisar si está en L1, si está es un hit, sino deberá buscarlo en L2.
2. Si está en L2 es un hit (en L2) y el dato debe guardarse en el caché L1, si no está deberá buscarlo en L3.
3. Si está en L3 es un hit (en L3) y el dato debe guardarse en el caché L2 y L1, si no está deberá traerlo de memoria.

Maneje todos los accesos sin diferenciar reads de writes. Cuando se realiza una victimización, en cualquier nivel del caché, no es necesario actualizar los demás niveles.

Análisis de resultados

Se procederá a analizar el comportamiento del caché ante variaciones en sus distintos parámetros. En cada experimento incluya las gráficas solicitadas y analice los resultados obtenidos.

Caché de un único nivel

Recolecte los datos de todos los traces al utilizar un caché L1 de 32kB, asociatividad en 8-way, el tamaño de bloque en 64B y la política de reemplazo en LRU. Asuma que el hit time de este caché es 4 ciclos de reloj, mientras que el miss penalty son 500 ciclos.

Incluya en su reporte una tabla con el AMAT de cada una de los traces para esta configuración. Calcule el AMAT promedio, usando media geométrica, a través de todos los traces.

Caché con dos niveles

Utilizando el mismo caché L1 descrito en la sección anterior agregue ahora un caché L2 a la simulación. Utilizando bloques de 64B y LRU como política de reemplazo varíe la capacidad y la asociatividad del caché y encuentre la configuración que minimiza el AMAT. Debe probar todas las combinaciones de:

- Capacidad de L2: 64kB y 128kB
- Asociatividad de L2: 8-way y 16-way

Asuma que el hit time de L1 es 4 ciclos de reloj, el de L2 es 12 ciclos y el miss penalty son 500 ciclos.

Incluya en su reporte una tabla con el AMAT promedio, usando media geométrica, (obtenido a través de todos los traces) para cada una de las configuraciones solicitadas.

Presencia de L3

Utilizando el mismo caché L1 y un L2 de 256kB y asociatividad de 8-ways, agregue un caché L3 a la simulación. Utilizando bloques de 64B y LRU como política de reemplazo varíe la capacidad y la asociatividad del caché y encuentre la configuración que minimiza el AMAT. Debe probar todas las combinaciones de:

- Capacidad de L3: 512kB y 1024kB
- Asociatividad de L3: 16-way y 32-way

Asuma que el hit time de L1 es 4 ciclos de reloj, el de L2 es 12 ciclos, el de L3 es 60 ciclos y el miss penalty son 500 ciclos.

Incluya en su reporte una tabla con el AMAT promedio, usando media geométrica, (obtenido a través de todos los traces) para cada una de las configuraciones solicitadas.

Ordene los traces de menor a mayor AMAT en el caso de solamente L1. Usando ese orden, grafique el AMAT de sólo usar L1, el mejor caso de L1+L2 y el mejor caso de L1+L2+L3.

Con base en los resultados obtenidos en los experimentos anteriores, conteste lo siguiente:

- Analice el comportamiento observado. Discuta cómo varía el AMAT ante cambios en las configuraciones (más niveles de caché, más grandes, más asociatividad).

Evaluación

Se utilizará la rúbrica descrita a continuación

	Rúbrica	**Puntaje Máximo**
I Parte	Construcción del caché	8
	Implementación de asociatividad	4
	Implementación de políticas de reemplazo	4
	Análisis del efecto del tamaño del caché	5
	Análisis del efecto de asociatividad	5
	Análisis del efecto del tamaño del bloque	5
	Análisis del efecto del reemplazo	5
	Eficiencia (ejecución en menos de 2 minutos)	2
	Orden, formato y documentación del código (incluir README)	2
	Total I Parte	**40**
II Parte	Implementación de caché multinivel	20
	Datos de sólo L1	6
	Datos de L1+L2	6
	Datos de L1+L2+L3	6
	Gráfica AMAT para todos los traces	5
	Análisis comportamiento	6
	Análisis confiabilidad AMAT	6
	Orden, formato y documentación del código (incluir README)	5
	Total II Parte	**60**

Otras consideraciones

- Se asume que la persona estudiante tiene los conocimientos de programación necesarios para realizar esta tarea, cualquier refrescamiento en estos conceptos será responsabilidad del estudiante.
- El código debe ser legible y estar debidamente comentado, sea consistente con los formatos de comentarios y nombres de variables.
- Cada función del programa debe contener un encabezado indicando una descripción general de la misma, así como los parámetros de entrada y salida.
- La entrega será por medio de la plataforma Mediación Virtual. Cada persona estudiante deberá subir una carpeta comprimida (.zip u otro formato adecuado) que contenga todos los archivos necesarios para la compilación (en caso de hacerlo en C++) y ejecución del código, así como un README con instrucciones para compilar y ejecutar correctamente el programa, deberá incluir además un archivo PDF con las gráficas y el análisis de los resultados obtenidos en los experimentos solicitados.
- No descomprima el trace para ejecutar el programa:
    - Si utiliza Python, la librería `gzip` permite leer archivos sin descomprimirlos
    - Si utiliza C/C++ lea el archivo del standard input (por ejemplo, utilizando cin), y utilice zcat para enviar los datos del trace al programa, por ejemplo:

      zcat <Archivo Trace> | ./<Su ejecutable> [Parámetros]
- Se aceptarán entregas tardías con una penalización de **25 puntos** por día tarde.