**UNIVERSIDAD DE COSTA RICA**

ESCUELA DE INGENIERÍA ELÉCTRICA

*Curso*: IE0521 | Estructuras de computadores digitales 2

*Tarea 1*: Simulación de varios tipos de memoria caché

*Profesor*: José Daniel Hernández

*Integrantes:*

Luis Fernando Mora Mora| B24449

Mario Castresana Avendaño | A41267

Luis Alvarado Villalta | 870193

29 de setiembre del 2016

***II-2016***

**Objetivo General**

Desarrollar conceptos relacionados a la memoria caché mediante su implementación y simulación en un lenguaje de alto nivel.

**Objetivos específicos**

* Utilización de interfaces en C++ para la implementación de la estructura del cache vista en el curso.
* Comparar los diferentes desempeños de cada tipo de memoria cache, variando parámetros de diseño como asociatividad, tamaño y cantidad de palabras por bloque de memoria.

**Problema propuesto**

Usando un lenguaje de programación de alto nivel (C++, en este caso) se desarrolló un programa que simula el comportamiento de una memoria caché con diferentes niveles de asociatividad.

1. De acuerdo a las especificaciones del enunciado, el programa deberá recibir como parámetros por consola la asociatividad, el tamaño del caché, y el tamaño del bloque.

Es decir, el programa deberá poder ser ejecutado con un comando similar a:

**$ ./cache <asociatividad> <tamaño del caché> <tamaño del bloque>**

2. El programa deberá ser capaz de procesar un archivo .trace que tiene todas las direcciones y datos de la simulación. El formato de cada línea del archivo sería

<dirección en hexadecimal> **{L,S}**

3. El programa será capaz de correr simulaciones para varios tipos de cache

Dicha simulación hará uso de un arreglo matricial al cual se le variará su tamaño con la idea de aumentar y disminuir la asociatividad al igual que variar el tamaño del bloque, asumiendo que las filas son los sets y las columnas son los bloques de cada set.

Primeramente se harán simulaciones para los siguientes niveles de asociatividad:

* Directo
* 2-way associative
* 4-way associative

Donde se medirán los *miss rate* correspondientes y de esta manera podremos ver las mejoras o desmejoras. Cada nivel tendrá 4 tamaños de bloques y 4 tamaños de set diferentes, teniéndose así 16 simulaciones.

**Introducción**

Como estamos aprendiendo en el curso es muy importante saber manejar las jerarquías de memorias ya que de esta manera, nosotros como futuros diseñadores de sistemas de computadoras podremos implementar ya sea los diseños más económicos, rápidos y confiables posibles.

Dentro de la jerarquía de memorias nos concentramos en la memoria caché la cual ocupa el segundo nivel desde el procesador (CPU) a las memorias más lentas como puede ser el disco magnético o disco duro.

Como la memoria caché es muy cara en relación con la memoria principal es muy necesario poder diseñar la memoria caché de la manera más óptima posible de forma que tengamos el mejor desempeño.

Para esto haremos uso de la mejor manera de aprender que es viendo cómo se comportan las cosas aprendidas por medio de la práctica.

En nuestro caso haremos 16 simulaciones para cada nivel de asociatividad (directo, 2-way, 4-way).

**Implementación**

En las clases de C++ implementadas,

* Cache.h
* Bloque.h

Se decidió implementar la memoria cache como una matriz especializada como la que se muestra en la figura 1. La idea es poder aprovechar al máximo las funciones definidas en cada clase para así poder trabajar con un caché de cualquier tamaño y asociatividad.



Fig. 1: matriz especializada para nuestra memoria cache implementada en C++.

**Clases principales y su código**

Para efectos de simplicidad se implementaron dos clases principales llamadas

* Bloque.h
* Cache.h

*Cache.h*

En Cache.h se implementaron funciones básicas para leer, buscar y escribir *tags*, así como también métodos encargados de leer bits de válido y también reportar la cantidad de misses y hits encontrados hasta el momento en la simulación. He aquí una lista de todos los métodos que se implementaron:

void writeTag(int, int, unsigned long);

unsigned long readTag(int, int);

void buscarTag(unsigned long);

int readMisses();

int readHits();

void reemplazarBloque(int, int, unsigned long);

void writeValid(int, int, bool);

bool readValid(int, int);

*Bloque.h*

Esta clase aparte, tiene las mismas funciones que la clase anterior, la idea detrás de este concepto es permitir manejar una interfaz más simple capaz de hacer las mismas operaciones por bloque y manejarlos más eficazmente.

**Simulaciones**

**Tabla 1. Simulación caché de Mapeo Directo**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tamaño de Bloque** | **Tamaño de Caché** | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Tabla 2. Simulación de caché de asociatividad 2-way**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tamaño de Bloque** | **Tamaño de Caché** | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Tabla 3. Simulación de caché de asociatividad 4-way**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tamaño de Bloque** | **Tamaño de Caché** | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Gráficas**

**Análisis de resultados**

*Para la gráfica 1 (Mapeo Directo):*

Como podemos apreciar en la gráfica 1, se observa que conforme el tamaño del bloque aumenta, los miss rate disminuyen en forma abrupta para el primer aumento de bloques. Y conforme el número de bloques aumenta, la pendiente de disminución es menor.

Esta tendencia es igual para los 4 tamaños de caché, siendo el caché más grande el que tiene el mayor miss rate de todos.

*Para la gráfica 2 (2-way associative):*

Como podemos apreciar en la gráfica 2, se observa que conforme el tamaño del bloque aumenta, los miss rate disminuyen en forma abrupta para el primer aumento de bloques. Y conforme el número de bloques aumenta, la pendiente de disminución es menor.

Esta tendencia es igual para los 4 tamaños de caché, siendo el caché más grande el que tiene el mayor miss rate de todos. Comparando el mapeo directo y la asociatividad 2-way hay una mejoría cuando los bloques son pequeños, y conforme aumenta el tamaño del bloque se parecen más las gráficas.

*Para la gráfica 3 (4-way associative):*

En esta gráfica la tendencia a la mejoría es igual que en el mapeo directo y el 2-way, pero se nota una clara mejoría para los bloques de menor tamaño. En general, se aprecia una mejoría del *miss rate*.

**Conclusiones**

Como esperábamos de acuerdo a la teoría aprendida en clase conforme aumentamos el número de bloques se da una disminución del miss rate, mejorando mucho nuestro diseño.

Además, conforme aumenta el tamaño del caché, se aumentan los miss rate para todas las asociatividades.

Se puede concluir también que conforme aumenta la asociatividad se tiene una gran mejoría en el miss rate para los bloques más pequeños.

En nuestro caso, el menor miss rate se da para la asociatividad 4-way con la caché más pequeña como era de esperarse.