UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio 3

Integrantes: Juan Pablo Martínez

Curso: Organización de Computadores

Sección B2

Profesor(a): Daniel Wladdimiro Cottet

19 de Noviembre de 2017

Tabla de contenidos

1. Introducción

1.1. Contexto

Buscar datos en memoria principal es costoso en terminos de rendimiento para un computador, debido a varios factores arquitecturales del computador. Uno de ellos es la lejanía que tiene el procesador con la memoria principal, en donde mientras más lejos se encuentre la memoria del procesador, más lento es la recuperación de datos.
La memoria caché es una memoria pequeña, de rápido acceso y cercana al procesador, la cual permite gracias a su arquitectura y ubicación una mejora significativa en el rendimiento de un computador, ya que los datos más usados por el computador se encuentran alojados en esta memoria, facilitando el acceso a estos y mejorando el rendimiento del computador reduciendo las busquedas en memoria principal.

1.2. Problema

Entender cómo funciona una memoria caché en sus diferentes configuraciones, tales como mapeo directo, full asociativo y n-vías, así como también entender el funcionamiento de las políticas de reemplazo más usadas, como LRU, MRU y FIFO.

1.3. Motivación

Con el fin de entender como funcionan las memorias cachés en sus diferentes configuraciones y políticas de reemplazo es que se participará en este laboratorio, en donde se diseñará un programa que sea capaz de simular el funcionamiento de una memoria cache en sus diferentes configuraciones y políticas de reemplazo.

1.4. Objetivos

1.4.1. Objetivo General

1. Diseñar un programa que sea capaz de simular el comportamiento de una memoria caché, dado un archivo de texto con los datos a procesar y una configuración dada por

el usuario.

1.4.2. Objetivos específicos

1. Configurar una memoria caché usando los parámetros de ejecución del programa.
2. Diseñar una interfaz de usuario que permita ingresar el nombre del archivo a leer e ingresar los nombres de los archivos de salida.

1.5. Propuesta de solución

Se diseñará el programa utilizando el lenguaje de programaciión orientado a objetos C++.
El usuario, a través de una interfaz podrá indicar los nombres de los archivos de entrada y salida, para luego ejecutar el programa, con la configuración que indicó en el inicio del programa.

1.6. Herramientas

1. Sistema operativo Windows 10 Falls Creator Update.
2. Entorno de desarrollo MinGW.
3. Editor de texto Sublime Text 3.0.

1.7. Estructura del informe

El presente informe está separado en 5 capítulos, en donde el primer capítulo presenta el contexto del problema y los objetivos del laboratorio. El segundo capítulo resume la teoría que se necesitó para desarrollar el laboratorio. Los capítulos 3 y 4 muestran cómo fue diseñada y construída la aplicación para luego mostrar los resultados obtenidos. El último capítulo muestra las conclusiones que se obtuvieron de la experiencia, mostrando para ello cuáles fueron los objetivos que se cumplieron.

2. Marco teórico

2.1. Caché

Una memoria caché es una memoria en la que se almacena una serie de datos para su fácil acceso. Esta memoria es de capacidad muy limitada y se encuentra muy cerca del procesador, permitiendo de esta manera una comunicación más veloz entre procesador y datos.

Principalmente existen 3 configuraciones para una memoria caché, estas son: Mapeo directo, Full asociativa y Conjunto asociativo.

2.2. Mapeo Directo

La forma más sencilla de asignar una ubicación en el caché para cada palabra en la memoria es asignar la ubicación de caché basada en la dirección de la palabra en memoria. En este tipo de configuración, cada ubicación en memoria puede ser mapeada a sólo una localización en caché. Para ubicar una palabra se debe utilizar la siguiente ecuación:

$$pos = \frac{v * 4}{bxb} \%t \tag{1}$$

En donde v es el valor a buscar, bxb corresponde a la cantidad de bytes por bloque en la configuración de la caché y t es la cantidad de bloques que tiene la memoria caché.

Cabe destacar que para obtener la cantidad de bytes por bloque se debe saber por bloque hay 4 bytes, que corresponden a 32 bits, valor de una palabra en la arquitectura estudiada.

2.3. Full Asociativo

Esta configuración de caché permite guardar un bloque de datos en cualquier localización de la memoria caché.

Para localizar un bloque en la caché se debe buscar en todas las entradas del caché. Esta búsqueda se hace en forma paralela con el fin de reducir el tiempo de búsqueda.

2.4. Conjunto Asociativo

Configuración de la memoria caché, la cual intenta unir los métodos descritos anteriormente. La memoria caché se divide en q conjuntos, cada uno de n bloques, llamando así la caché n-asociativo. Para encontrar un dato en uno de los bloques se utiliza la siguiente eucuación:

$$pos = \frac{v * 4}{bxp} \%t \tag{2}$$

En donde v es el valor buscado, bxp corresponde a la división entre la cantidad de bloques por set y t la cantidad de bloques en la caché.
Para calcular el valor de bxp se hace: t/n.

2.5. Hit y Miss

Hit es el término que se indica cuando un dato ha sido encontrado en memoria caché, si el dato no es encontrado entonces se indica el nombre de miss.
Tasa de hit corresponde a la fracción de accesos de memoria encontrados, y de manera contraria la tasa de miss es la resta entre la unidad y la tasa de hit. Estos datos son importantes para el análisis de rendimiento de la memoria caché, ya que un gran número de miss indica un mal rendimiento de memoria caché.

2.6. Politicas de Reemplazo

Estas son políticas para reemplazar un bloque cuando el caché se encuentra lleno. Estas se usan tanto en cachés conjunto asociativo y full asociativo.

2.6.1. FIFO

FIFO son las siglas en inglés de First-in-First-out, la cual señala que lo primer que entra es lo primero que sale. Esta política de reemplazo indica que los datos que llevan más tiempo en la caché son reemplazados por los nuevos.

2.6.2. MRU

Esquema de reemplazo en donde el bloque que se reemplaza es el que ha sido usado más recientemente.

2.6.3. LRU

Un esquema de reemplazo en donde el bloque que es reemplazado es el que tiene más tiempo de haber sido ocupado.

3. Desarollo

3.1. Diseño

Se diseñaron las clases Word, Block y Procesor. La clase Procesor contiene todas los métodos necesarios para ejecutar el programa. Contiene un arreglo de dos dimensiones tipo Block, en donde la fila representa el set del bloque y la columna reprenta el bloque en donde se encuentra la palabra. La clase Block contiene un arreglo de la clase Word, la cual contiene el valor a guardar. De esta manera, la columna del arreglo Block apunta al arreglo de tipo Word, el cual representa de 1 hasta las palabras que puede contener el bloque.
Se utilizarón las bibliotecas iostream para la interacción de la aplicación con el usuario (mostrar mensajes por pantalla y tomar datos desde el teclado), fstream (lectura y escritura de archivos), string (manipulación de cadena de carácteres), cstdlib (manejo de memoria) y list(lista dinámica) de C++ para desarrollar el programa.

3.2. Configuración de caché

Debido a que se utilizó el sistema operativo Windows 10, los parámetros configuración de la memoria caché deben ser ingresados como parámetros de ejecución del programa. Para ello se utilizan los argumentos argc y argv.
El argumento argc, de tipo entero, guarda la cantidad de parámetros que se ejecutan, es decir, si el programa se ejecuta con los parámetos **cache.exe LRU 2 2 8**, el valor de argc será igual a 5. El argumento argv, de tipo cadena de carácteres, guarda los parámetros que se indicaron junto a la ejecución del programa. Utilizando el ejemplo anterior, argv[0] contiene el valor çache.exe", argv[1]contiene el valor "LRU", argv[2] contiene el valor 2, argv[3] contiene el valor 2 y argv[4] contiene el valor 8.
Con estas variables se puede verificar si los parámetros ingresados son correctos para configurar la memoria caché o no. Si los valores son correctos, entonces se crea un objeto de tipo Procesor, con los parámetros ingresados.

3.3. Políticas de Reemplazo

Para las poder utilizar las políticas de reemplazo se agregó el atributo "edad."[en] cada objeto de tipo Block. Este atributo va aumentando cada vez que el bloque es consultado o cambiado.

El algoritmo LRU diseñado recibe el dato a guardar y la posición del set del bloque. Luego se busca el bloque que tenga el menor valor en el atributo "edad"de manera iterativa, cuando encuentra un bloque que tiene una "edad"menor, la variable auxiliar que guarda el valor se cambia por la encontrada y se guarda la posición del bloque en donde se encuentra este valor. Una vez la búsqueda termina el bloque es modificado según los parámetros ingresados en el algoritmo y los datos encontrados en la búsqueda realizada.

Se diseñó una estrategia similar para MRU, en donde la diferencia se encuentra en que se busca el bloque que tenga más "edad".

3.4. Lectura de archivos

Luego de que el programa entra en ejecución y el usuario ingresa en la sección de lectura de archivo el nombre del archivo a leer, el programa verifica si el archivo existe, en caso de que exista y cumpla con el formato necesario, los datos leídos son guardados en una lista dinámica de la plantilla de C++, para luego ser ocupados en cualquier parte del programa.

3.5. Archivos de salida

Los nombres de los archivos de salida son indicados por el usuario. Debido a que son dos archivos que se guardarán en la misma carpeta, se verifica que los nombres de estos dos archivos no sean iguales, con el fin de no perder un archivo.

En primer archivo de salida se muestra el contenido final de la memoria caché, es decir, se muestra por set, bloque y palabra el contenido que se guardó en la última iteración. Para ello se recorre el arreglo de dos dimensiones y el arreglo de tipo Word del objeto Block, de forma iterativa, escribiendo en cada iteración los datos encontrados en la caché.En el segundo archivo se muestra en porcentaje la tasa de hit y miss de la ejecución del programa.

4. Experimentos

4.1. Resultados obtenidos

Para probar el programa se utilizaron los archivos de prueba y ejemplos de salida que fueron entregados en el archivo ejemplosLab3.zip. Esta carpeta contiene un archivo de texto con los datos de entrada y 12 ejemplos de como serían las salidas según las distintas configuraciones de la memoria caché.

Los resultados para las pruebas, bajo la configuración LRU y MRU para políticas de reemplazo y con distintos niveles de asociatividad con varias palabras por bloque fueron iguales a los ejemplos de salida. No obstante, los resultados para la configuración FIFO para política de reemplazo y de mapeo directo no fueron iguales, en particular, habían varios bloques vacíos que si debieron haber sido ocupados.

Las tasas de hit y miss que se escriben, luego de ejecutar el programa son diferentes a las mostradas en los archivos de ejemplo, las cuales son porcentajes enteros. Las tasas escritas en las pruebas son más exactas a nivel decimal que las mostradas en los archivos de ejemplo.

4.2. Análisis de resultados

Se encontró que la configuración de mapeo directo no funcionaba correctamente debido a que no se calculó bien el rango para un caché de multi palabra. Debido a esto es que habían más palabras que bloques para guardar, lo cual generaba errores en la ejecución del programa.

Uno de los problemas más significativos que se encontró fue las comparaciones que se pueden realizar si uno de los datos a leer en el archivo es 0. En el diseño del programa, especificamente en el arreglo de dos dimensiones de tipo Block, se crea un objeto con valor 0, esto en terminos de diseño, significa un estado inicial, no un valor inicial. Entonces si uno de los datos leídos era el 0, se podía indicar un hit, cuando no el 0 que estaba en ese bloque era solo un estado inicial, no un dato. Para solucionar esto se agregó la edad del bloque. La edad del bloque es una variable entera que indica cuando fue señalado ese bloque, entonces si el bloque tiene valor 0 y además la edad de ese bloque es distinta de 0, entonces si es el valor 0 y no el estado

inicial.

Las tasas de hit y miss que se obtienen son porcentajes decimales, esto se debe a que se considera que es más exacto trabajar con cierto nivel de exactitud para obtener un análisis más profundo en rendimiento, a grandes niveles.

5. Conclusiones

En esta experiencia se logró cumplir tanto como el objetivo general como los objetivos específicos. Esto es, Diseñar un programa que sea capaz de simular el comportamiento de una memoria caché en las distantas configuraciones dadas por el usuario y diseñar una interfaz de usuario que le permita al usuario interactuar con la apliación para ver los resultados obtenidos.

Se logró entender como funcionan las políticas de reemplazo LRU, MRU y FIFO, diseñando para ello distintos algoritmos que las ejecuten.

Se diseñaron clases que permiten la interacción entre objetos de las clases Block y Word para guardar datos en los bloques, según el nivel de asociatividad y cantidad de palabras por bloque, diseñando para ello arreglos de dos dimensiones y una dimensión respectivamente.

Se diseñó una interfaz que le permita al usuario ingresar tanto el nombre de archivo de entrada y archivos de salida, así como también poder visualizar la configuración de la memoria caché.

Con el fin de evitar tener el archivo de lectura abierto durante la ejecución del programa, se diseñó una estrategia de solución, en la cual los datos leídos son guardados en una lista dinámica para posteriormente ocuparlos cuando sea necesario.

Bibliografía

Patterson, D. A. and Hennessy, J. L. (2011). *Computer Organization and Design*, volume 4. Morgan Kauffman.