**Impacto de la Asociatividad en el Rendimiento de Caché**

Elaborado por:

## Gabriel Enrique Pinto Sermeño

**C.I. 26.364.594**

**Arquitectura del Computador**

Marzo, 2025

**Impacto de la Correspondencia/Asociatividad en el Rendimiento de Caché**

Elaborado por:

## Gabriel Enrique Pinto Sermeño

**C.I. 26.364.594**

**Arquitectura del Computador**

Diseñar experimentos con distintas configuraciones de caché para evaluar la eficiencia en la tasa de aciertos. Implementar cachés con diferentes grados de asociatividad. Analizar tasa de aciertos y tiempo de acceso. Comparar resultados con cachés totalmente asociativas. Ajustar políticas de reemplazo en cachés con distintas asociatividades.

Objetivos Secundarios

-Implementar Modelos de Caché: Desarrollar en C++ modelos funcionales de simuladores para los tres tipos principales de organización de caché: correspondencia directa, asociativa por conjuntos (con N configurable) y totalmente asociativa.

- Diseñar y Ejecutar Experimentos: Configurar y ejecutar sistemáticamente las simulaciones de caché, variando el parámetro de asociatividad mientras se mantienen constantes otros parámetros (como el tamaño total de la caché y, implícitamente, el tamaño del bloque según la implementación actual) sobre una misma traza de direcciones de memoria.

- Medir y Registrar el Rendimiento: Calcular y registrar de forma precisa la tasa de aciertos (y fallos) para cada configuración de caché y asociatividad simulada.

- Analizar y Comparar Resultados: Comparar las tasas de acierto obtenidas entre las diferentes configuraciones de asociatividad y políticas de reemplazo, identificando tendencias, ventajas y desventajas de cada enfoque en el contexto de la traza utilizad

**Introducción:**

En el marco del estudio de la arquitectura de computadoras, hemos desarrollado un simulador en C++ para analizar y comparar el rendimiento de políticas de correspondencia en memoria caché. El programa en cuestión pretende simular el comportamiento de la caché ante una secuencia de accesos a la memoria, lo que nos permitió evaluar algunas métricas clave como la tasa de fallos y la de aciertos. Por otro lado, por lo que respecta a la lectura de las distintas direcciones de memoria, hemos implementado una técnica eficiente basada en mmap para leer sobre un fichero CSV, lo que nos permitió procesar un mayor volumen de datos de forma más rápida que de que dadas otras alternativas.

**Funcionalidad General y Preparación:**

El programa comienza creando una interfaz de usuario sencilla (ShowMenu) en la que elegimos el tipo de simulación de correspondencia que queremos ejecutar (Directa, Asociativa por Conjuntos, Completamente Asociativa, o Todas) y también definimos los parámetros de la caché: el número total de bloques (líneas) y, en el caso de la asociativa por conjuntos, la cantidad de conjuntos.

Una característica clave de nuestra implementación es el uso de mmap. Abrimos el archivo ENTRADA ("dataset.csv") y lo mapeamos directamente en el espacio de direcciones de nuestro proceso. Esto nos proporciona un puntero (direccionMemoria) que podemos tratar como si fuera un gran arreglo de caracteres, representando el contenido del archivo. Recorremos este espacio de memoria mapeada para extraer las direcciones (asumimos que son las cadenas de caracteres que aparecen antes de las comas). Usamos la función DirHexToDec para convertir estas cadenas, que interpretamos como direcciones hexadecimales, a valores numéricos enteros (unsigned long int).

El programa lleva la cuenta de los aciertos (aciertos) y fallos (fallos) para cada tipo de correspondencia y, al finalizar cada simulación, mostramos los resultados (printResultados), incluyendo el porcentaje de aciertos, utilizando una función auxiliar (formatNumber) para hacer que los números grandes sean más legibles.

**Correspondencia Directa (correspondenciaDirecta)**

En este tipo de sistema, cada bloque de la memoria principal solo puede ser cargado en una única línea de la caché.

Implementación:

Comenzamos creando un vector de caché que representa los bloques de la caché, con un tamaño igual al tamanoCache (número de bloques) que el usuario ha especificado. Cada entrada se inicializa en -1, lo que indica que está vacía.

Recorremos el archivo mapeado (direccionMemoria). Cuando encontramos una coma, procesamos la cadena acumulada, s.

Convertimos la dirección (s) a decimal (direccionDecimal) utilizando DirHexToDec.

Calculamos el bloque de caché de destino usando la operación módulo: numBloque = direccionDecimal % cache.size(). Esta es la clave de la correspondencia directa.

Verificamos si la dirección direccionDecimal ya está en cache[numBloque].

Acierto (Hit): Si hay coincidencia, incrementamos el contador de aciertos.

Fallo (Miss): Si no hay coincidencia (o si el bloque estaba vacío), aumentamos el contador de fallos y actualizamos cache[numBloque] con la nueva direccionDecimal. El contenido anterior (si existía) se pierde.

Análisis: Esta es la política más sencilla, pero puede enfrentar "colisiones de mapeo", donde direcciones que se utilizan con frecuencia compiten por el mismo bloque de caché, lo que genera fallos innecesarios, incluso cuando otros bloques están disponibles.

**La Correspondencia Completamente Asociativa (correspondenciaCompletamenteAsociativa)**

Es un esquema donde un bloque de la memoria principal puede ser almacenado en cualquier línea (bloque) de la caché.

Implementación:

Primero, inicializamos el vector de caché, llamado tamanoCache, con -1. Luego, para cada dirección, que llamamos direccionDecimal, que extraemos del archivo mapeado, buscamos esa direccionDecimal en todas las posiciones del vector de caché.

Si encontramos la dirección en alguna posición, lo llamamos un acierto (Hit) y aumentamos el contador de aciertos. Por otro lado, si no la encontramos después de revisar toda la caché, se considera un fallo (Miss) y aumentamos el contador de fallos. En este caso, seguimos con la política de reemplazo:

Primero, buscamos un bloque vacío (-1) en la caché. Si encontramos uno en la posición x, colocamos la direccionDecimal en cache[x]. Si no hay bloques vacíos, aplicamos una política de reemplazo. En nuestro código, implementamos un reemplazo aleatorio: seleccionamos un índice al azar (numBloque = rand() % cache.size()) y sobrescribimos cache[numBloque] con la direccionDecimal.

Análisis: Este enfoque ofrece una gran flexibilidad y tiene el potencial de lograr altas tasas de acierto, ya que evita las colisiones que se producen en la correspondencia directa. Sin embargo, la búsqueda en toda la caché y la lógica de reemplazo (especialmente si es más compleja, como LRU) pueden hacer que sea más costosa en términos de hardware. Aunque nuestro método de reemplazo aleatorio es sencillo, no siempre es el más óptimo.

**Correspondencia Asociativa por Conjuntos (correspondenciaAsociativoPorConjuntos)**

Esta política es una mezcla de las dos anteriores. La caché se divide en numConjuntos, y cada bloque de la memoria principal se asigna a un conjunto específico, aunque puede estar en cualquiera de los bloques (vías) dentro de ese conjunto.

Implementación:

Utilizamos un array 2D llamado cache[numConjuntos][tamanoCache], donde numConjuntos representa el número de conjuntos y tamanoCache (que se pasa desde main) indica cuántas vías (bloques) hay por conjunto. Es crucial entender esta interpretación de tamanoCache en este contexto particular.

Además, empleamos otro array 2D llamado tiempoDeUso[numConjuntos][tamanoCache] para llevar a cabo la política de reemplazo LRU (Least Recently Used - Menos Usado Recientemente). Ambas estructuras se inicializan en -1.

Para cada dirección direccionDecimal:

Primero, calculamos el conjunto destino: numConjunto = direccionDecimal % numConjuntos. Luego, incrementamos el contador de tiempo para todos los bloques válidos dentro de ese conjunto (aumentarTiempoDesdeQueSeUso).

Buscamos direccionDecimal solo en las vías de cache[numConjunto].

Acierto (Hit): Si la encontramos en la vía x, aumentamos el contador de aciertos y reiniciamos el contador de uso para esa vía: tiempoDeUso[numConjunto][x] = 0.

Fallo (Miss): Si no la encontramos en el conjunto, incrementamos el contador de fallos. Aplicamos la política de reemplazo LRU dentro del conjunto:

Buscamos una vía vacía (-1) en cache[numConjunto]. Si encontramos una en la posición x, colocamos direccionDecimal en cache[numConjunto][x] y establecemos tiempoDeUso[numConjunto][x] = 0.

Si no hay vías vacías en el conjunto, buscamos la vía x dentro del conjunto que tenga el valor más alto en tiempoDeUso (la menos recientemente usada). Sobrescribimos cache[numConjunto][x] con direccionDecimal y reiniciamos tiempoDeUso[numConjunto][x] = 0.

Análisis: Se busca un equilibrio entre la simplicidad de la caché directa y la flexibilidad de la caché asociativa completa.

Análisis: Es importante encontrar un balance entre la simplicidad de la caché directa y la flexibilidad de la completamente asociativa. Esta opción reduce las colisiones en comparación con la caché directa y resulta más fácil de implementar en hardware que la completamente asociativa. La política LRU se encarga de mantener en caché los bloques que se han utilizado más recientemente dentro de cada conjunto.

|  |  |  |  |
| --- | --- | --- | --- |
| **48.5 MB** | | | |
| **NYDP\_Arrest\_Data.csv** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **8** | **8** | **8(4 conjuntos)** |
| **Tiempo de ejecución** | **15 seg** | **11 seg** | **8 seg** |
| **Aciertos** | **1.989.060** | **1.953.961** | **1.471.920** |
| **Fallos** | **3.222.244** | **3.257.343** | **3.739.384** |
| **% de Aciertos** | **61.83%** | **71.76%** | **62.51%** |

|  |  |  |  |
| --- | --- | --- | --- |
| **48.5 MB** | | | |
| **NYDP\_Arrest\_Data.csv** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **4** | **4** | **4( 2 conjuntos)** |
| **Tiempo de ejecución** | **6 seg** | **3 seg** | **2 seg** |
| **Aciertos** | **2.211.528** | **2.229.646** | **1.742.577** |
| **Fallos** | **2.999.776** | **2.981.658** | **3.468.727** |
| **% de Aciertos** | **57.56%** | **57.22%** | **66.56%** |

|  |  |  |  |
| --- | --- | --- | --- |
| **212 MB** | | | |
| **final\_dataset.csv** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **8** | **8** | **8(4 conjuntos)** |
| **Tiempo de ejecución** | **30 seg** | **12 seg** | **10 seg** |
| **Aciertos** | **18.878.247** | **19.545.929** | **23.064.301** |
| **Fallos** | **19.818.378** | **19.150.696** | **15.632.324** |
| **% de Aciertos** | **48.79%** | **50.51%** | **59.60%** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **212 MB** | | | | |
| **final\_dataset.csv** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **4** | **4** | **4(2 conjuntos)** |
| **Tiempo de ejecución** | **25seg** | **10 seg** | **8 seg** |
| **Aciertos** | **13.466.295** | **18.158.291** | **20.489.608** |
| **Fallos** | **24.986.955** | **20.538.334** | **18.207.017** |
| **% de Aciertos** | **45.52%** | **46.92%** | **52.95%** |

|  |  |  |  |
| --- | --- | --- | --- |
| **354 MB** | | | |
| **credit\_card\_transactions** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **8** | **8** | **8(4 conjuntos)** |
| **Tiempo de ejecución** | **42 seg** | **22 seg** | **13 seg** |
| **Aciertos** | **20.891.643** | **21.207.742** | **23.219.106** |
| **Fallos** | **10.745.784** | **10.429.685** | **8.418.321** |
| **% de Aciertos** | **66.03%** | **67.03%** | **73.39%** |

|  |  |  |  |
| --- | --- | --- | --- |
| **354 MB** | | | |
| **credit\_card\_transactions** | **Directa** | **Asociativa** | **Asociativa x Conjunto** |
| **Nro de Bloques** | **4** | **4** | **4(2 conjuntos)** |
| **Tiempo de ejecución** | **38 seg** | **15 seg** | **10 seg** |
| **Aciertos** | **20.032.085** | **20.046.173** | **22.327.013** |
| **Fallos** | **11.605.342** | **11.591.254** | **9.310.414** |
| **% de Aciertos** | **63.32%** | **63.36%** | **70.57%** |

## Conclusión

A través de este proyecto, logramos implementar y simular con éxito tres políticas clave de correspondencia de caché: Directa, Completamente Asociativa con reemplazo aleatorio, y Asociativa por Conjuntos con reemplazo LRU. Para manejar de manera eficiente un archivo CSV de gran tamaño como fuente de direcciones de memoria, utilizamos la técnica de mapeo de memoria (mmap). El simulador nos permite comparar de manera directa el número de aciertos y fallos en diferentes configuraciones de caché (tamaño total, número de conjuntos/vías), lo que se convierte en una herramienta valiosa para comprender cómo estas políticas afectan el rendimiento del sistema de memoria.

Todos los tipos de correspondencia pueden ser usado, la correspondencia asociativa x conjunto es la más efectiva en relación a rendimiento/recursos, teniendo una alta tasa de acierto con un consumo de recursos promedio