UNIVERSIDAD DE CARABOBO
FACULTAD DE CIENCIAS Y TECNOLOGÍA
DEPARTAMENTO DE COMPUTACIÓN
ARQUITECTURA DEL COMPUTADOR

Simulación de Caché Asociativo por vías con anticipación de datos de memoria externa (Usando STXXL).

Integrantes:
Jesús Peñaloza Cedula:30532626
Jose Campos

Introducción

Este informe describe la implementación de un simulador de memoria caché asociativa por conjuntos con prefetching (anticipación de datos) utilizando la biblioteca STXXL para manejar memoria externa. El objetivo es analizar el comportamiento de diferentes configuraciones de caché (2, 4 y 8 vías) bajo un patrón de acceso realista con bloques calientes y fríos.

Definición de Caché

La memoria *caché* es un componente de hardware de alta velocidad que almacena copias temporales de datos e instrucciones frecuentemente accedidos desde la memoria principal (RAM). Su objetivo es **reducir la latencia** de acceso a datos y mejorar el rendimiento del sistema (Hennessy & Patterson, 2012).

Memoria Caché Asociativa por Conjuntos (Set-Associative Cache)

Definición

Una *caché asociativa por conjuntos* es un diseño intermedio entre la caché **directamente mapeada** y la **totalmente asociativa**. Combina la velocidad de acceso de la primera con la flexibilidad de la segunda, organizando la memoria caché en **conjuntos (sets)**, donde cada conjunto contiene múltiples líneas de caché (vías o *ways*) (Hennessy & Patterson, 2012).

Dirección de memoria:

[Tag (bits de identificación) | Index (bits del conjunto) | Offset (dentro del bloque)]

Características Clave

1. **Estructura:**
    - La cache se divide en **conjuntos** (ejemplo: 4, 8, 16 conjuntos).
    - Cada conjunto contiene **varias líneas de cache** (ejemplo: 2, 4, 8 vías).
    - Un bloque de memoria principal puede ubicarse en **cualquier línea dentro de un conjunto específico**.
2. **Mapeo:**
    - La dirección de memoria se divide en:
        - **Etiqueta (*tag*):** Identifica el bloque.
        - **Índice (*index*):** Selecciona el conjunto.
        - **Desplazamiento (*offset*):** Localiza el dato dentro del bloque.
3. **Ventajas:**
    - **Menos conflictos** que el mapeo directo (un bloque no está fijo a una única línea).
    - **Más eficiente** que una cache totalmente asociativa (requiere menos comparadores en paralelo).

4. **Desventajas:**
    - Mayor complejidad en hardware que el mapeo directo.
    - Puede presentar **mayor latencia** al comparar múltiples etiquetas en un conjunto.

Ejemplo de Funcionamiento

- **Cache de 2 vías por conjunto (2-way set-associative):**
    - Si un bloque puede ir en **2 líneas diferentes** dentro de su conjunto asignado.
    - Al buscar un dato, se compara la etiqueta con **todas las líneas del conjunto**.

Comparación con Otros Tipos de Cache

Tipo de Cache	Ventajas	Desventajas
Directamente mapeada	Simple y rápida	Alta tasa de conflictos
Totalmente asociativa	Máxima flexibilidad (sin conflictos)	Costosa (comparadores en paralelo)
Asociativa por conjuntos	Equilibrio entre velocidad y conflictos	Mayor complejidad que mapeo directo

---

Tipos de Fallos de Cache

Capacidad: El caché es demasiado pequeño para contener todos los datos necesarios.

Conflicto: Múltiples direcciones compiten por una misma ubicación en caché (Hennessy y Patterson, 2012).

Politica LRU (Least Recently Used)

El algoritmo LRU (Least Recently Used) es una política de reemplazo utilizada en memorias cache para decidir qué bloque debe ser eliminado cuando se necesita espacio para un nuevo dato. Su principio básico es:

> *"Reemplazar el bloque que no ha sido accedido durante el mayor tiempo."*

Prefetching en Memorias Cache

El **prefetching** (precarga) es una técnica utilizada en arquitecturas de computadoras para **anticipar** y cargar datos o instrucciones en la memoria cache **antes de que sean**

solicitados por el procesador. Su objetivo es **reducir la latencia** de acceso a memoria y minimizar los fallos de cache (*cache misses*), mejorando así el rendimiento del sistema (Hennessy & Patterson, 2012).Técnica que **precarga bloques adyacentes** al accedido para reducir *misses* y mejorar el rendimiento.

- **Secuencial**: Precarga bloques contiguos (implementado en este trabajo).
- **Estríado**: Para patrones con saltos regulares.
- **Adaptativo**: Ajusta dinámicamente la distancia de prefetch.

Resumen del Código

El código proporcionado implementa una simulación de un sistema de caché con diferentes grados de asociatividad (2, 4 y 8 vías). Utiliza una política de reemplazo LRU (Least Recently Used) y simula un mecanismo de prefetching para cargar bloques adyacentes anticipadamente.

Características Principales

1. **Configuración del Caché**:
    - Tamaño total del caché: 1024 elementos
    - Tamaño de bloque: 32 elementos
    - Número total de elementos accedidos: 2048
2. **Mecanismos Implementados**:
    - Política de reemplazo LRU
    - Prefetching con distancia configurable (2 bloques)
    - Diferentes grados de asociatividad (2, 4 y 8 vías)
3. **Bibliotecas Utilizadas**:
    - STXXL para manejo de estructuras de datos externas
    - Biblioteca estándar de C++ para estructuras de datos y generación de números aleatorios

Análisis del Código

Estructuras de Datos

1. **Bloque (`Block`)**:
    - Contiene un array de datos de tamaño `My_BLOCK_SIZE`
    - Representa la unidad básica de transferencia entre memoria principal y caché
    - Inicialización con valores secuenciales para simular datos reales.
2. **Clase `Cache`**:
    - Implementada como plantilla parametrizada por el número de vías (`NUM_WAYS`)
    - Usa un vector de listas para representar los conjuntos del caché
    - Utiliza STXXL para simular almacenamiento externo
    - Métodos clave:

    - `access()`: Gestiona hits/misses y activa prefetching.
    - `prefetch_adjacent_blocks()`: Precarga bloques futuros.
3. **Clase `ExternalMemory`**
    - Usa **STXXL** para simular memoria externa (disco).
    - contiene un `stxxl::vector<Block>` para almacenar bloques.
4. **Clase `CacheSet`**
    - Implementa un conjunto de caché con política **LRU**.
    - Usa una `std::list` para manejar el orden de acceso.
5. **Clase `RealisticAccessPattern`**
    - Genera direcciones con **50% de probabilidad** de acceder a bloques calientes (25% del espacio de direcciones).

Algoritmos

Acceso al Caché:

- Calcula dirección de bloque y conjunto
- Busca el bloque en el conjunto correspondiente
- Si es un hit, mueve el bloque al frente (LRU)
- Si es un miss, carga el bloque y aplica prefetching

Prefetching:

- Carga anticipadamente bloques adyacentes al accedido
- Distancia configurable mediante `PREFETCH_DISTANCE`

Ventajas del Prefetching enfoque actual:

- Simpleza de implementación
- Bajo overhead computacional

Limitaciones Del Prefetching enfoque actual :

- No se adapta a patrones de acceso variables
- Podría precargar datos innecesarios

4. Resultados y Análisis

4.1. Métricas Obtenidas

Configuración	Tasa de Aciertos (%)	Tasa Efectiva (con Prefetch) (%)	Prefetch Hits	Tiempo (s)

2-vías	81.05%	85.46%	620	0.00037
4-vías	82.42%	86.14%	549	0.00031
8-vías	83.35%	86.62%	500	0.00029

4.2. Conclusiones

1. **Impacto de la asociatividad**:
    - **4-vías** ofrece el mejor equilibrio entre tasa de aciertos y complejidad.
    - **8-vías** mejora ligeramente el rendimiento, pero con mayor sobrecarga.
2. **Efectividad del prefetching**:
    - Aumenta la tasa efectiva de aciertos en **~4-5%**.
    - Más útil en configuraciones con **menor localidad** (50% accesos calientes).
3. **Tiempos de ejecución**:
    - Todos los casos se ejecutan en **menos de 1 milisegundo**.
    - Diferencias mínimas entre configuraciones.

Referencia

Hennessy, J. L., & Patterson, D. A. (2012). *Computer Architecture: A Quantitative Approach* (5th ed.). Morgan Kaufmann.

STXXL Library Documentation (2023). *Standard Template Library for Extra Large Data Sets*