Informe Técnico

Simulador de Arquitecturas de Memoria Caché

Diseño, Implementación y Análisis

Autores: Samuel Primera C.I: 31.129.684 Samuel Reyna C.I.:30.210.759

Fecha: October 16, 2025

Informe de Simulación de Arquitecturas de Memoria Caché

Samuel Primera C.I: 31.129.684 Samuel Reyna C.I.:30.210.759

October 16, 2025

1 Descripción del Problema y Objetivos

El objetivo principal del proyecto es **diseñar e implementar un simulador de caché en C++ (utilizando STL) para modelar y comparar arquitecturas de memoria caché**.

1.1 Arquitecturas a Simular

El simulador debe modelar las siguientes arquitecturas:

- Correspondencia Directa.
- Asociativa por N vías.

1.2 Entrada y Métricas de Evaluación

El simulador procesa una secuencia de direcciones de memoria como entrada. Los resultados clave para la comparación de rendimiento son las **Estadísticas de Aciertos y Fallos**.

1.3 Requisito Avanzado

El diseño debe considerar la evaluación de **técnicas de anticipación de carga de datos** para explotar la localidad de referencia y determinar su impacto en la tasa de fallos de la caché.

2 Manejo y Estructuras de Datos

La implementación de la simulación de caché se basa en una jerarquía de clases que utiliza tipos de datos específicos para la configuración, las estadísticas y la representación interna de las memorias.

2.1 Comparativa de Estructuras

La siguiente tabla resume las estructuras principales para cada tipo de caché:

Característica	**Caché de Correspondencia Directa**	**Caché Asociativa por N Vías (LRU)**
Estructura Principal	Vector simple de enteros (*etiquetas*)	Dos Matrices Bidimensionales (*Vector de Vectores*): *etiquetas* y *tiempoLRU*
Cálculo del Índice	Posición única basada en el número total de bloques: *indice* = (etiqueta) mod (numBloques) .	Posición del conjunto basada en el número de conjuntos: *indice* = (etiqueta) mod (numConjuntos)
Dimensiones	Tamaño: *numBloques*.	Dimensiones: *numVias* (filas) × *numConjuntos* (columnas).

Característica	Caché de Correspondencia Directa	Caché Asociativa por N Vías (LRU)
Propósito de etiquetas	Almacenar la etiqueta del bloque cargado .	Almacenar la etiqueta del bloque cargado (*etiquetas[via][conjunto]*).
Inicialización de etiquetas	Se llena con -1 (vacío).	Se llena con -1 (vacío).
Estructura de Reemplazo	No aplica: la posición es fija.	Matriz *tiempoLRU* (inicializada con 0).
Lógica de Reemplazo	El bloque existente es siempre reemplazado por el nuevo.	LRU: Se busca la vía dentro del conjunto con el menor valor en *tiempoLRU* (el menos usado) para reemplazar.
Actualización en Acierto	No hay seguimiento de tiempo.	El valor en *tiempoLRU* se actualiza al contador actual de accesos.

2.2 Jerarquía de Clases y Estructuras Específicas

2.2.1 Clase Base `Cache`

Gestiona parámetros de configuración (`numBloques`, `tamBloque`, tipo `int`) y métricas de rendimiento (contadores de accesos, aciertos y fallos, tipo `long long`, inicializados a cero). Recibe direcciones de memoria como cadenas de texto (`string`) denominadas `direccionBin`.

2.2.2 Clase `CacheDirecta`

Implementación específica para Correspondencia Directa.

- Estructura: Un **vector de enteros** (`vector<int>`) llamado `etiquetas`, con tamaño igual a `numBloques`.
- Inicialización: Todas las posiciones se inicializan con el valor -1 (vacío).

2.2.3 Clase `CacheAsociativa`

Implementación para N Vías con política LRU.

- Estructuras: Dos **matrices bidimensionales** (`vector<vector<int»`) paralelas:
  1. `etiquetas`: Almacena la etiqueta del bloque en la posición `etiquetas[via][conjunto]`. Se inicializa con -1.
  2. `tiempoLRU`: Almacena el valor del último acceso. Se inicializa con **0**.
- Parámetros Internos: `numVias` y `numConjuntos` (este último calculado como numBloques/vias).

3 Justificaciones de Diseño

3.1 Uso de Herencia

Se optó por utilizar la herencia para simplificar y reutilizar código.

- **Diferencia en Complejidad y Estructuras**: Los métodos de acceso son lo suficientemente diferentes para requerir implementaciones separadas. El direccionamiento asociativo requiere dos matrices paralelas (etiquetas y seguimiento de accesos), mientras que el directo solo necesita un único vector para etiquetas. El método asociativo presenta un **orden de complejidad algorítmica y estructural significativamente mayor** que el directo.
- **Simplificación y Reutilización**: Una **clase abstracta** (`Cache`) unifica funciones y atributos comunes (estadísticas y parámetros de configuración). Las clases hijas (`CacheDirecta` y `CacheAsociativa`) solo implementan las funciones y estructuras de datos específicas de su método.

3.2 Uso de Números Enteros vs. Binarios

Se optó por números enteros ya que resultan **más fáciles de comprender y operar** beneficiosos para el equipo.

- Trabajar con binarios requeriría nuevas clases y estructuras auxiliares para facilitar su manejo y operaciones como el desplazamiento de bits.
- Las estructuras de datos nativas de C++ (vectores y matrices) están optimizadas para índices enteros, lo que haría necesaria una **transformación de binario a entero** de todos modos para acceder a dichas estructuras.

3.3 Parámetros de Entrada

Se toman como entrada el número de vías, el número de bloques y su tamaño (además de las direcciones). Esto se debe a que son necesarios para determinar el método de mapeo (directo si *vias*=1, asociativo si es mayor) y el diseño de la caché, lo cual afecta significativamente a la tasa de aciertos. Permite al usuario configurar y probar el simulador sin manipular el código fuente .

4 Algoritmos Implementados

4.1 `Cache` (Clase Abstracta)

- `Cache(int nBloques, int tBloque)`: Constructor que inicializa `numBloques` y `tamBloque`.
- `accederDireccion(const string &direccionBin)`: **Función virtual abstracta** para manejar el acceso a memoria.

- `getEtiqueta(const string &direccionBin)`: Calcula la **etiqueta (tag)** del bloque. Convierte la dirección binaria a entero y la divide por `tamBloque` .
- `estadisticas()`: Imprime las estadísticas de rendimiento (accesos, aciertos, fallos, tasas).

4.2 `CacheDirecta` (Direccionamiento Directo)

- `CacheDirecta(int nBloques, int tBloque)`: Inicializa el vector de etiquetas al tamaño de `numBloques` con el valor -1.
- `accederDireccion(const string &direccionBin)`: Implementa el Algoritmo de Mapeo Directo. Calcula el índice (etiqueta mod numBloques) y verifica acierto/fallo. En caso de fallo, **reemplaza forzosamente** la etiqueta.

4.3 `CacheAsociativa` (N Vías - LRU)

- `redimensionar(int vias, int nBloques)`: Calcula numConjuntos = nBloques/vias y redimensiona las matrices `etiquetas` ($\rightarrow -1$) y `tiempoLRU` ($\rightarrow 0$).
- `CacheAsociativa(int vias, int nBloques, int tBloque)`: Constructor que llama a la función `redimensionar`.
- `accederDireccion(const string &direccionBin)`: Implementa el Algoritmo de Acceso y Reemplazo LRU. Calcula el índice del conjunto (etiqueta mod numConjuntos). Si es acierto, actualiza `tiempoLRU`. Si es fallo, busca la vía con el `tiempoLRU` más bajo para reemplazar y actualiza el valor.

4.4 `main.cpp`

Implementa el **Algoritmo de Selección de Caché y Simulación**. La lógica de selección es: si vias = 1, usa `CacheDirecta`; si es mayor, usa `CacheAsociativa`. Incluye validación de entradas para parámetros positivos y para asegurar que el número de vías no exceda el número de bloques.

5 Análisis de Resultados

Se realizaron pruebas con entradas generadas aleatoriamente.

5.1 Comparativa de Rendimiento con 100.000 y 1.000.000 de Casos

Table 2: Resultados con Bloques = 16 y Palabras = 8

Métrica	**Directa**	**Asociativa**	
	Vías=1	**Vías=8**	**Vías=16**
100.000 Casos			
Tiempo	$0,019s$	$0,024s$	$0,028s$
Fallos	50121	49938	50060
Aciertos	49879	50062	49940
Tasa de Aciertos	49.879%	50.062%	50.062%
1.000.000 Casos			
Tiempo	$0,178s$	$0,228s$	$0,267s$
Fallos	500920	500341	500323
Aciertos	499080	499659	499677
Tasa de Aciertos	49.908%	49.966%	49.968%

Observaciones A mayor asociatividad, el porcentaje de aciertos tiende a ser mayor, pero el consumo de tiempo también se incrementa. La diferencia en el porcentaje de aciertos entre las distintas configuraciones (directa, asociativa y completamente asociativa) no es significativa.

5.2 Invariabilidad con Mayor Número de Bloques y Alto % de Aciertos

En los siguientes ejemplos, la mejora en el porcentaje de aciertos es casi invariable a pesar de la asociatividad.

Table 3: Resultados con Alto Número de Bloques (1.000.000 Casos, Palabras = 8)

Métrica	**Bloques=64**		**Bloques=32**	
	Vías=1 (Directa)	**Vías=16** (Asociativa)	**Vías=1** (Directa)	**Vías=16** (Asociativa)
Tiempo	$0,166s$	$0,19s$	$0,163s$	$0,205s$
Fallos	32	32	32	32
Aciertos	999968	999968	999968	999968
Tasa de Aciertos	99.997%	99.997%	99.997%	99.997%

6 Conclusión

Existen grandes diferencias entre el mapeo directo y el asociativo, tanto en resultados como en el proceso de ubicación de un bloque de caché.

6.1 Diferencia de Complejidad Algorítmica

- **Mapeo Directo**: Es sencillo y rápido de ejecutar. Implica calcular la etiqueta y su única ubicación. Si no coincide la etiqueta, hay un fallo y se reemplaza directamente. Esto representa un algoritmo de orden $O(1)$.
- **Mapeo Asociativo**: Requiere encontrar la etiqueta y el índice del conjunto, además de encontrar la vía dentro del conjunto, lo que obliga a **iterar sobre todo el conjunto**. En caso de fallo, se recorre el conjunto para buscar el acceso más antiguo (LRU). Esto resulta en un algoritmo de orden $O(n)$, donde n es el número de vías.
- El mapeo asociativo es **mucho más costoso en tiempo** que el directo, especialmente si la caché es completamente asociativa y el número de bloques es grande.

6.2 Tasa de Fallos y Flexibilidad

Aunque más costoso en tiempo, el mapeo asociativo tiende a tener un **menor índice de fallos**. Esto se debe a que es más flexible, permitiendo que direcciones con el mismo índice ocupen cualquier vía dentro de su conjunto, reduciendo los **conflictos en la memoria**.

6.3 Impacto de los Parámetros de Diseño

Además del método de direccionamiento, el **número de bloques** y el **tamaño del bloque** afectan la tasa de aciertos:

- **Número de Bloques**: Un mayor número de bloques suele aumentar la tasa de aciertos. Sin embargo, si el tamaño de la caché es muy superior a los datos procesados, puede ser un desperdicio de potencia por una mejora marginal.
- **Tamaño del Bloque**: Un bloque más grande mejora la **localidad espacial**, pero esto solo es beneficioso si los accesos siguientes siguen un patrón secuencial.

7 Referencias Bibliográficas

- Patterson, D. A, y Hennessy, J. L. (2011) Estructura y diseño de computadores: La interfaz hardware/software (4ta ed.) Reverté.