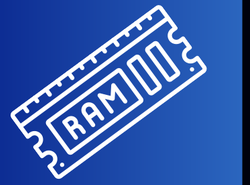
# Simulador de Caché Asociativa por Vías de Memoria Externa

Arquitectura del computador

Jesús Peñaloza 30562626
José Campos 25110846

# Introducción
## Definiciones

Este proyecto 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.




### Memoria 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).

### Caché asociativa

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)

# Comparación
## Otros tipos de memoria caché

| 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 |

# Elementos y técnicas utilizadas

## Librería STXXL

STXXL es una librería de C++ que extiende la funcionalidad de la librería estándar de plantillas (STL) para el procesamiento de conjuntos de datos extremadamente grandes.

## 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.

## Prefetching

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.

# Modelo de bloques de caché caliente y fríos

Los términos "bloques calientes" (hot blocks) y "bloques fríos" (cold blocks) se refieren a cómo se distribuyen los accesos a la memoria en un programa o sistema. Esta clasificación es clave para entender el comportamiento de la caché y optimizar su diseño.

### Bloques Calientes (Hot Blocks)

Suelen ser datos críticos o instrucciones en un bucle. Representan un pequeño porcentaje del espacio total, pero concentran la mayoría de los accesos. Ejemplo: Punteros.

### Bloques Fríos (Cold Blocks)

Ocupan la mayor parte de la memoria, pero tienen pocos accesos. Ejemplo: Datos inicializados una vez y luego olvidados, variables globales no usada.

### Efecto en la Caché

Bloques calientes tienden a permanecer en caché (porque se accede a ellos repetidamente). Bloques fríos son reemplazados rápidamente (poco reuso).

# Análisis del código
## Estructuras de datos

Clase 1
### CacheSet

- *Define cuántas vías (bloques) caben en el conjunto.*
- *Entonces, intenta acceder a un bloque dentro de este conjunto.*
- *Si devuelve miss, elimina el bloque menos recientemente usado, que está al final de la lista.*
- *Si devuelve hit, Mueve el bloque accedido al frente de la lista para marcarlo como el más recientemente usado.*

Clase 2
### Cache

- *Calcula cuántos conjuntos necesita.*
- *Simula un acceso a una dirección de memoria específica.*
- *Devuelve hit o miss al intentar acceder.*
- *Además, usa prefetching para traer bloques cercanos y comprueba si está en la caché.*
- *La idea es precargar bloques contiguos.*

Clase 3
### ExternalMemory

- *Recibe el número total de bloques y crea el vector STXXL.*
- *Inicializa cada bloque con un valor basado en su posición.*
- *Luego, simula la lectura de un bloque desde la memoria externa y devuelve una referencia al bloque solicitado.*

Clase 4
### Block

- *Definimos cómo es un bloque de datos.*
- *Su tamaño lo da MyBLOCK_SIZE*
- *Inicializa todos los elementos del bloque a 0.*
- *Luego, Inicializa los datos del bloque con valores secuenciales a partir de 'initial_value'.*

Clase 5
### RealisticAccessPattern

- *Distribuye bloques calientes y frios en un cuarto del total de bloques.*
- *Luego, genera la siguiente dirección de memoria a acceder.*
- *Decide aleatoriamente si acceder a un bloque caliente o frío.*
- *Calcula la dirección y la retorna.*

Clase 6
### CacheSimulador

- *Define un disco virtual en un archivo de configuración para que la librería stxxl la cree.*
- *Prepara los patrones de acceso de los bloques calientes y fríos.*
- *Crea un puntero a la memoria externa.*
- *Llama a las demás clases para realizar las operaciones de caché.*
- *Además, mide el tiempo de ejecución y muestra los resultados en pantalla.*

# Resultados de la ejecución

| Configuración | Tasa de aciertos | Tasa efectiva con prefetching | 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 |

# Conclusión

## Análisis de resultados

### Impacto de la asociatividad

- 2-vías tendría una tasa de aciertos inferior, pero también sería menos compleja
- 4-vías ofrece el mejor equilibrio entre tasa de aciertos y complejidad.
- 8-vías mejora ligeramente el rendimiento, pero con mayor sobrecarga.

### Efectividad del prefetching

- Aumenta la tasa efectiva de aciertos en —4-5%.
- Más útil en configuraciones con menor localidad (50% accesos calientes).
- Mientras más baja sea la localidad, aumenta hasta un 13% la efectividad del prefetching.

### Tiempos de ejecución

- Todos los casos se ejecutan en menos de 1 milisegundo.
- Diferencias mínimas entre configuraciones de equipos modernos.