Técnicas de Prefetching

Maximizando el rendimiento: Hardware vs Software

Sebastian Alcantar Vicencio

Kevin Osmar Nuñez Manriquez

Fransisco Ricardo Hernandez Astorga

Jose Emilio Angulo Bermudez

Andreas Armando Botello Gomez

¿Qué es el Prefetching?

La analogía del restaurante: Imagina estar comiendo y que el mesero te traiga el siguiente plato justo antes de que lo pidas, porque predijo tu comportamiento.

- **Anticipación inteligente:** Consiste en adivinar qué datos o instrucciones necesitará la CPU en el futuro cercano.
- **Movimiento de Datos:** Trae la información desde la memoria principal a la caché antes de que el procesador la solicite.
- **Objetivo principal:** Evitar que el procesador desperdicie tiempo esperando información.

El Problema: La Latencia de Memoria

100x

Diferencia de Velocidad

¿Por qué lo necesitamos?

El avance en la velocidad de los procesadores ha superado drásticamente a la velocidad de las memorias RAM (El "Muro de la Memoria").

- Buscar un dato en la RAM puede costar más de 300 ciclos de reloj de la CPU.
- Sin prefetching, un procesador potente se vuelve inútil, pasando el 70% del tiempo inactivo.
- Ocultar esta lentitud es el trabajo del prefetching.

El Entorno: Jerarquía de Memoria

¿Hacia dónde se mueven los datos durante el prefetching? La meta es subirlos en esta pirámide.

RAM (Principal)

Masiva pero muy lenta. Aquí residen todos los datos del programa. Desde aquí comienza el proceso de extracción (fetch).

Caché L3 / L2

Memorias intermedias más rápidas y pequeñas. El prefetching suele mover grandes bloques de RAM hacia estos niveles primero.

Caché L1

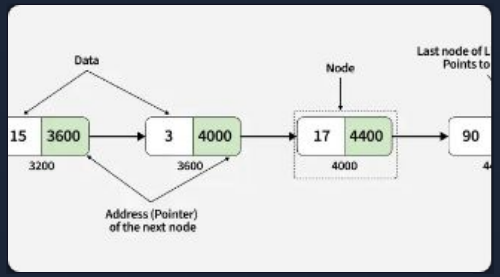
Diminuta pero vuela a la velocidad del procesador. El destino final ideal para los datos que se usarán en el siguiente ciclo.

Prefetching por Hardware

Es un mecanismo físico integrado directamente en el silicio del procesador. Funciona como un observador silencioso.

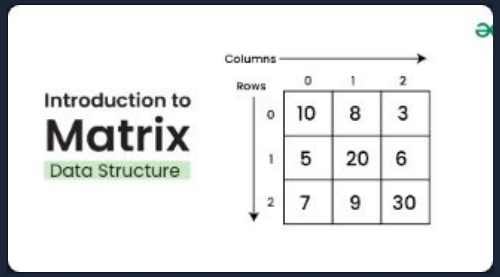
- **Transparencia total:** El programador no tiene que escribir código especial. Todo ocurre automáticamente.
- **Entrenamiento:** Circuitos especializados analizan el historial de memoria en tiempo real para encontrar un patrón repetitivo.
- **Adaptabilidad:** Si la CPU detecta que sus predicciones están fallando, se "apaga" para no saturar el sistema.

Patrones que detecta el Hardware



1. Stream Secuencial

Detecta lecturas consecutivas (Bloque 1, luego 2, luego 3...). Clásico al recorrer un Array de principio a fin.



2. Acceso Strided (Saltos)

Detecta distancias constantes entre lecturas (Ej. lee cada 4to bloque). Común al procesar columnas en matrices 2D.

3. Su Debilidad (Irregular)

El hardware falla y se detiene si los accesos son aleatorios o complejos, como en grafos o árboles desordenados.

Prefetching por Software

Para superar las debilidades del hardware, usamos el software. Consiste en insertar comandos de prefetch directamente en el código del programa.

- **Explícito:** Usando directivas como __builtin_prefetch(), le decimos a la CPU qué dirección exacta cargar.
- **Puede ser automático:** Los compiladores modernos (como GCC o Clang) pueden inyectar estas instrucciones al optimizar el código.
- **Poder Irregular:** Es perfecto para estructuras ligadas (grafos, árboles) porque el software *sabe* exactamente dónde está el siguiente nodo.

El Gran Peligro: Cache Pollution

El prefetching no es gratuito. Hacerlo mal (especialmente por software) puede empeorar dramáticamente el rendimiento del sistema.

Prefetching Exitoso

Traemos datos que **sí** se van a usar pronto.

- La CPU encuentra el dato en L1 inmediatamente.
- Se evitan cientos de ciclos de espera.
- El programa corre fluido y al máximo de los GHz disponibles.

Contaminación de Caché

Traemos datos incorrectos o muy anticipados.

- **Desalojo dañino:** Se borran datos útiles de la caché L1 para meter "basura".
- **Tráfico inútil:** Se satura el ancho de banda de la RAM moviendo datos que no sirven.
- Resulta en una doble penalización de rendimiento.

Resumen: Hardware vs Software

Característica	Hardware Prefetching	Software Prefetching
Ejecución	Dinámica. Ocurre en tiempo real en los circuitos.	Estática. Las instrucciones se ejecutan sí o sí.
Patrones Ideales	Arrays contiguos y saltos matemáticos predecibles.	Árboles, listas enlazadas, punteros y grafos.
Costo Computacional	**Bajo.** No requiere instrucciones extra de CPU.	**Alto.** Añade instrucciones que la CPU debe procesar.
Control de Errores	Si falla mucho, el hardware se auto-desactiva.	Si está mal programado, causará *Cache Pollution* constante.

¿Dónde se utilizan hoy en día?

Streaming (HW)

Al reproducir videos 4K, la memoria se lee de forma puramente secuencial. El hardware detecta este patrón al instante y satura la caché de video fluido.

Bases de Datos (SW)

Las búsquedas en grandes bases de datos (usando Árboles B) saltan aleatoriamente en la RAM. Los programadores inyectan prefetching por software para estos saltos.

Videojuegos (HW + SW)

Usan hardware para cargar las texturas planas, y software para calcular físicas complejas o la Inteligencia Artificial de múltiples enemigos esparcidos.

La Clave del Alto Rendimiento

El prefetching es el arte invisible de predecir el futuro para ocultar la lentitud del pasado (la memoria).

Sinergia Tecnológica

Ni el hardware ni el software son perfectos por sí solos. Los sistemas computacionales más rápidos del mundo combinan ambos inteligentemente para garantizar que la CPU nunca deje de calcular.

Fin de la presentación / ¿Preguntas?