Jose Antonio Mora M

C15114

CI-0120 - Arquitectura de Computadoras

Profesor Oscar Caravaca Mora

SIMICS parte 4

Investigación

La memoria caché es un tipo de memoria de computadora de alta velocidad muy importante en los sistemas computacionales, ya que brinda ciertas ventajas y conveniencias que mejoran el rendimiento y estabilidad del sistema en general. La memoria caché funciona como un "buffer" entre el CPU y la memoria principal RAM, y guarda datos e instrucciones frecuentemente accesados, actuando efectivamente como un intermediario entre componentes, por lo que reduce el tiempo de acceso del CPU a la memoria principal. Esta característica mejora sustancialmente el rendimiento del procesador y ayuda a reducir la latencia en el acceso a los datos, debido a que permite el acceso a los mismos de manera rápida, reduce el trabajo y el tiempo de espera que debe de realizar el procesador, lo que mejora sustancialmente el rendimiento y velocidad del sistema computacional [1].

Existen diferentes niveles de caché en los sistemas computacionales, cada uno con sus respectivas diferencias. El primer nivel de caché, denominado L1, es la más pequeña de todas, con un rango promedio de 2KB a 64KB dependiendo del procesador, pero es la más rápida, ya que se encuentra incorporado directamente en el procesador, lo que le permite operar a la misma velocidad que el CPU. El siguiente nivel de caché, denominado L2, es más grande que el nivel anterior, con un rango de 256KB a 512KB, es levemente más lenta que la caché L1, y puede estar dentro del CPU o fuera, pero siempre está más cerca del CPU que la memoria principal. Dependiendo de la arquitectura del CPU, el caché L2 puede ser compartido entre múltiples núcleos de CPU o ser exclusivo de un único núcleo. Por último, se encuentra la caché L3, que es más grande que los dos niveles anteriores, con un rango de 1MB a 8MB, pero es la más lenta de todas, debido a que se encuentra fuera del CPU, y es compartida por todos los núcleos del procesador. La caché L3 juega un papel importante a la hora de comunicar y compartir datos entre núcleos. Estas diferencias influyen en el rendimiento general del sistema, debido a que se crea una jerarquía eficiente de memoria entre los diferentes niveles para aprovechar sus diferencias, ya que las cachés más rápidas, pero más pequeñas se utilizan para almacenar los datos críticos que se utilizan con mayor frecuencia o se necesitan acceder de manera rápida, mientras que los demás niveles inferiores van almacenando cantidades de datos más grandes, pero con velocidades inferiores. Esto crea una escalera de memoria que mantiene un equilibrio entre velocidad y capacidad de almacenamiento, maximizando el rendimiento del sistema y organizando de mejor manera las cargas de trabajo [1].

Usualmente determinar qué datos serán los más frecuentemente accesados para guardarlos en la memoria caché es una tarea difícil o imposible, por lo que se hace uso del principio de localidad. La mayoría de programas poseen algún grado de localidad, por lo que la caché explota esta característica para determinar qué datos almacenar. La localidad se divide en dos conceptos: el principio de localidad temporal, que define que, si un programa

accede a una dirección de memoria, hay una alta posibilidad de que la vuelva a acceder de nuevo pronto. Un ejemplo de esto es en los ciclos de un programa, ya que existen múltiples instrucciones que serán ejecutadas múltiples veces, o al usar una variable múltiples veces. Y el principio de localidad espacial, que define que, si un programa accede a una dirección de memoria, hay una alta posibilidad de que también accederá a una dirección cercana. Un ejemplo de esto existe en la mayoría de programas, ya que las instrucciones son ejecutadas de manera secuencial, o a la hora de acceder a datos que se guardan de manera continua, como un arreglo. Es importante tener claros estos conceptos a la hora de diseñar algoritmos y programas, debido a que al tener en cuenta el principio de localidad, se puede llegar a mejorar sustancialmente el rendimiento y eficiencia, se optimiza el uso de recursos y se mejora la escalabilidad del programa o algoritmo, ya que aumenta la probabilidad de que los datos necesarios estén cargados en la memoria caché cuando se necesitan, se disminuyen los accesos a la memoria principal, se utilizan las estructuras de datos y se acceden a las mismas de manera eficiente, y brinda mayor estabilidad y orden a los accesos de caché. En general, un uso adecuado del principio de localidad permite aprovechar al máximo el rendimiento y las capacidades de la memoria caché, mejorando el desempeño de los programas [2].

Para mantener un uso optimizado y eficiente de la caché, una vez que la caché se llena, es necesario tener una política u algoritmo de reemplazo que decida qué dato descartar de la caché para tener espacio para un nuevo dato que se necesita. Es necesario que se defina eficientemente qué datos descartar para minimizar los fallos y maximizar el rendimiento de la caché. Tres de las políticas de reemplazo más comunes utilizadas en los sistemas modernos son las siguientes: el algoritmo FIFO o First In-First Out, que trata a los datos como un "buffer" circular, y reemplaza el dato que fue introducido de primero o el más antiguo en el "buffer". El algoritmo LFU o Least Frequently Used, que cuenta qué tanto se han usado los datos de la caché, y el dato que ha sido menos utilizado es el reemplazado. Por último, el algoritmo LRU o Least Recently Used, que descarta el dato que ha sido menos recientemente utilizado de los presentes en la caché. Las variantes del LRU son las más populares entre los algoritmos de reemplazo. La elección de una u otra política de reemplazo puede afectar considerablemente al rendimiento general del sistema, debido a que varias políticas pueden tener diferentes niveles de eficiencia del uso de la caché, además de que diferentes políticas pueden tener diferentes niveles de requisitos de recursos computacionales para funcionar correctamente, lo que puede producir resultados diferentes. Es por esto que no existe una solución única y hay que tomar en cuenta varios factores a la hora de seleccionar una política de reemplazo para una aplicación dada, como la dificultad de implementación del algoritmo, los requerimientos computacionales de la política, los recursos y características de Hardware con los que se cuenta, y la naturaleza de la aplicación. [3].

Referencias

[1] «Cache Memory - Definition and general information - Redis», *Redis*, 13 de julio de 2023. https://redis.io/glossary/cache-memory/.

[2] «Cache introduction - lec16», *UW Homepage*, 9 de mayo de 2010. https://courses.cs.washington.edu/courses/cse378/10sp/lectures/lec16.pdf.

[3] Q. Javaid, A. Zafar, M. Awais, y M. A. Shah, «Cache Memory: An Analysis on Replacement Algorithms and Optimization Techniques», *Archive Ouverte HAL*, 4 de febrero de 2018. https://hal.science/hal-01700364/document.

Controlador de caché

Logs:

```
elmics> run-python-file modules/cache-controller/cache_demo.py
Address 0x2 requested
Addresses in cache: []
Searching cache...
Cache miss for address 0x2
Searching memory...
Data read from main memory: 0x6d7bae908eccbef3
Block of address 0x2 loaded in cache
Data read from address 0x2: 0x6d7bae908eccbef3
------------------------------------------------------------
Address 0xe requested
Addresses in cache: ['0x2']
Searching cache...
Cache miss for address 0xe
Searching memory...
Data read from main memory: 0xe45f17b82a60ec19
Block of address 0xe loaded in cache
Data read from address 0xe: 0xe45f17b82a60ec19
------------------------------------------------------------
Address 0x1 requested
Addresses in cache: ['0x2', '0xe']
Searching cache...
Cache miss for address 0x1
Searching memory...
Data read from main memory: 0x3e984fdba64c8179
Block of address 0x1 loaded in cache
Data read from address 0x1: 0x3e984fdba64c8179
------------------------------------------------------------
Address 0x2 requested
Addresses in cache: ['0x2', '0xe', '0x1']
Searching cache...
Cache hit for address 0x2
Frequency updated for block: 2
Returning data from cache: 0x6d7bae908eccbef3
Data read from address 0x2: 0x6d7bae908eccbef3
------------------------------------------------------------
```

```
------------------------------------------------------------
Address 0x7 requested
Addresses in cache: ['0x2', '0xe', '0x1']
Searching cache...
Cache miss for address 0x7
Searching memory...
Data read from main memory: 0x81b6fa3ed597ac0f
Block of address 0x7 loaded in cache
Data read from address 0x7: 0x81b6fa3ed597ac0f
------------------------------------------------------------
Address 0x10 requested
Addresses in cache: ['0x2', '0xe', '0x1', '0x7']
Searching cache...
Cache miss for address 0x10
Searching memory...
Data read from main memory: 0xc5aedb1f987abd62
Block of address 0x10 loaded in cache
Data read from address 0x10: 0xc5aedb1f987abd62
------------------------------------------------------------
Address 0xb requested
Addresses in cache: ['0x2', '0xe', '0x1', '0x7', '0x10']
Searching cache...
Cache miss for address 0xb
Searching memory...
Data read from main memory: 0x74d1a9ec59e6fca8
Replacing block in cache with address 0xe (frequency: 1)
Block of address 0xb loaded in the cache
Data read from address 0xb: 0x74d1a9ec59e6fca8
------------------------------------------------------------
Address 0x2 requested
Addresses in cache: ['0x2', '0x1', '0x7', '0x10', '0xb']
Searching cache...
Cache hit for address 0x2
Frequency updated for block: 3
Returning data from cache: 0x6d7bae908eccbef3
Data read from address 0x2: 0x6d7bae908eccbef3
------------------------------------------------------------
```

```
------------------------------------------------------------
Address 0xc requested
Addresses in cache: ['0x2', '0x1', '0x7', '0x10', '0xb']
Searching cache...
Cache miss for address 0xc
Searching memory...
Data read from main memory: 0xd08c416a3b759f21
Replacing block in cache with address 0x1 (frequency: 1)
Block of address 0xc loaded in the cache
Data read from address 0xc: 0xd08c416a3b759f21
------------------------------------------------------------
Address 0xe requested
Addresses in cache: ['0x2', '0x7', '0x10', '0xb', '0xc']
Searching cache...
Cache miss for address 0xe
Searching memory...
Data read from main memory: 0xe45f17b82a60ec19
Replacing block in cache with address 0x7 (frequency: 1)
Block of address 0xe loaded in the cache
Data read from address 0xe: 0xe45f17b82a60ec19
------------------------------------------------------------
Address 0xd requested
Addresses in cache: ['0x2', '0x10', '0xb', '0xc', '0xe']
Searching cache...
Cache miss for address 0xd
Searching memory...
Data read from main memory: 0x672e8a94b1cd2f7e
Replacing block in cache with address 0x10 (frequency: 1)
Block of address 0xd loaded in the cache
Data read from address 0xd: 0x672e8a94b1cd2f7e
------------------------------------------------------------
Address 0xb requested
Addresses in cache: ['0x2', '0xb', '0xc', '0xe', '0xd']
Searching cache...
Cache hit for address 0xb
Frequency updated for block: 2
Returning data from cache: 0x74d1a9ec59e6fca8
Data read from address 0xb: 0x74d1a9ec59e6fca8
------------------------------------------------------------
```

```
------------------------------------------------------------
Address 0x12 requested
Addresses in cache: ['0x2', '0xb', '0xc', '0xe', '0xd']
Searching cache...
Cache miss for address 0x12
Searching memory...
Data read from main memory: 0x5392f8dfa0ce647b
Replacing block in cache with address 0xc (frequency: 1)
Block of address 0x12 loaded in the cache
Data read from address 0x12: 0x5392f8dfa0ce647b
------------------------------------------------------------
Address 0x12 requested
Addresses in cache: ['0x2', '0xb', '0xe', '0xd', '0x12']
Searching cache...
Cache hit for address 0x12
Frequency updated for block: 2
Returning data from cache: 0x5392f8dfa0ce647b
Data read from address 0x12: 0x5392f8dfa0ce647b
------------------------------------------------------------
Address 0xd requested
Addresses in cache: ['0x2', '0xb', '0xe', '0xd', '0x12']
Searching cache...
Cache hit for address 0xd
Frequency updated for block: 2
Returning data from cache: 0x672e8a94b1cd2f7e
Data read from address 0xd: 0x672e8a94b1cd2f7e
------------------------------------------------------------
simics>
```