Informe Practica de Laboratorio 4

Cesar Rojas, #31406902

Octubre 2025

Memorias cache, sus tipos y como funcionan

Composicion

Una memoria cache forma parte de la jerarquia de memoria. Esta se ubica entre el procesador y la memoria principal. Aunque pueden existir varios niveles de cache, en este texto se tratara un único nivel.

Una memoria cache se compone de varios elementos, pero principalmente de *Bloques*. Cuando hablamos de bloques nos referimos a bloques de memoria, los cuales pueden contener una cantidad arbitraria de *Palabras*. Una palabra siendo una medida de información generalizada cuyo tamaño también es arbitrario.

Una memoria (no necesariamente una memoria cache) puede estar compuesta de bloques que contengan 4 palabras cada una. Estas palabras pueden contener cierta cantidad de información medida en bits. Por ejemplo, se puede tener una memoria con bloques de 4 palabras, donde cada palabra sean 8 bits de información. Por lo tanto, cada bloque contiene en total 32 bits.

Las memorias cache también están compuestas de estos bloques y palabras. La principal diferencia siendo que las memorias caches se ubican de forma mas cercana a el procesador que la memoria principal, por lo que es mas veloz acceder a esta que a la memoria principal. Sin embargo esto viene con un coste de *tamaño*. Las memorias cache son mas pequeñas que la memoria principal, por lo que solo pueden contener una parte del total de información.

Aciertos y fallos

Cuando el procesador envía una dirección de memoria, el cache la recibe y revisa si esta dirección se encuentra dentro de si. Si este es el caso, no hace falta acceder a la memoria principal y puede retornar la información deseada de forma mas rápida que si se haya accedido a la memoria. Esto se denomina un *acierto*.

Pero si no se encuentra, el cache debe acceder a la memoria, guardar la información dentro de si, y retornar a el procesador la información deseada. Esto se denomina un *fallo*.

Correspondencia directa, asociatividad completa, y asociatividad por conjuntos

Existen tres modelos de organización de bloques para caches. El primero y mas simple de estos se denomina *correspondencia directa*.

Un cache por correspondencia directa le asigna a cada bloque una posición especifica en la cache. Usualmente usando operaciones matemáticas como *mod*. Así, se simplifica el posicionamiento de los bloques para que cada uno se pueda ubicar y asignar inmediatamente.

Un cache por asociatividad completa por su parte, puede asignar un bloque en cualquier posición de la cache. Uno de los métodos para lograr esto es *LRU* (Least Recently Used), que significa 'Menos Recientemente Usado'. Como su nombre implica, al guardar un bloque en la memoria cache este se coloca en

la posición que lleva mas tiempo sin usarse. Pero a diferencia de un cache de correspondencia directa, es necesario recorrer todo el cache para encontrar esta posición cada vez que se desee reemplazar un bloque.

Finalmente, un cache por asociatividad por conjuntos es muy similar al de asociatividad completa, pero los bloques se agrupan en *conjuntos* de bloques. Cada bloque se le asigna un conjunto, de la misma manera que con la correspondencia directa, pero también puede ubicarse en cualquier posición dentro de este conjunto como con la asociatividad completa.

De esta forma, los caches asociativos por conjuntos son un tipo de mezcla entre la correspondencia directa y la asociatividad completa.

Prefetching

Prefetching (Pre-carga) consiste en cargar bloques a la memoria de cache antes de que estos sean llamados para así reducir la tasa de fallos y el tiempo que se tarda en obtener la información desde la memoria al iniciar el acceso de antemano.

Existen muchos métodos de prefetching, pero los que se tratan en este proyecto son el método *Fall-Through* (secuencial) y el método *Target* (salto).

Estos dos métodos tambien se pueden usar en conjunto para reducir la tasa de fallos y tiempos de acceso de forma mas significativa..

Prefetching secuencial

El método secuencial consiste en pre-cargar los bloques siguientes a el ultimo bloque que se obtuvo de la memoria. El numero de bloques a pre-cargar puede depender, con la cantidad mas optima siendo algo compleja de obtener y afectada por varios factores como el tipo de cache, patrones de acceso, y de mas. Sin embargo, este método es tan simple como es efectivo, asiendo uso del principio de la localidad espacial para reducir la cantidad de fallos y el tiempo de acceso cuando se accede a la memoria de manera secuencial.

Prefetching de salto

El método de salto consiste en pre-cargar bloques según el patrón de los accesos mediante una tabla de predicciones. Esta tabla se construye usando los accesos anteriores, y usándola se predice cuales bloques se van a solicitar para precargarlos. Este método es mas complejo que el secuencial, pero es capar de predecir patrones de acceso no secuenciales a memoria usando el principio de la localidad temporal.

Análisis de Resultados

Con el proyecto se proveen dos archivos de entrada: entradaSecuencial y entradaSalto. Como sus nombres indican cada una esta diseñada con patrones de

acceso que cada tipo de prefetching puede manejar de manera mas efectiva.

Resultado secuencial

El primer archivo (entradaSecuencial) contiene una memoria cache de correspondencia directa debido a que tiene 8 conjuntos cada uno poseyendo un único bloque; los bloques por su parte conteniendo 4 palabras. Las entradas son una secuencia con saltos aleatorios de hasta 3 bloques, por lo que el numero de bloques por prefetch es 3.

Como podemos ver en el resultado, el cache con prefetching secuencial posee solamente un fallo, mientras que los otros caches terminaron con 17 fallos. Esto es gracias a que nuestro cache pre-cargo los siguientes 3 bloques por adelantado después de cada acceso a memoria, logrando reducir la tasa de fallos a solamente un fallo.

```
Prueba de Cache sin prefetching
Aciertos: 23
Fallos: 17
Prueba de Cache con prefetching secuencial
Aciertos: 39
Fallos: 1
Prueba de Cache con prefetching de salto
Aciertos: 23
Fallos: 17
```

Resultado salto

El segundo archivo (entradaSalto) utiliza una cache completamente asociativa, con 1 solo conjunto compuesto de 8 bloques. Estos bloques también conteniendo 4 palabras cada uno. La entrada consiste de accesos de memoria aleatorios, pero organizados en patrones que aunque no son secuenciales se repiten entre si.

El resultado nos muestra que tanto el cache sin prefetching como el cache con prefetching secuencial poseen una tasa de fallo del 100%, mientras que el cache con prefetching de salto supera el 50% de aciertos. Ya que algunos de los accesos poseen patrones, usando la tabla de predicciones estos se pueden pre-cargar con éxito, permitiendo que el cache no posea una tasa de fallos del 100% si es demasiado pequeño para guardar suficientes bloques.

```
Prueba de Cache sin prefetching
Aciertos: 0
Fallos: 41
Prueba de Cache con prefetching secuencial
Aciertos: 0
Fallos: 41
Prueba de Cache con prefetching de salto
Aciertos: 21
Fallos: 20
```

Instrucciones de uso

Para ejecutar el proyecto usando archivos de acceso personalizados, simplemente se necesita leer el archivo 'formato de entrada.txt' el cual posee el formato de los archivos de entrada. Una vez creado, se requiere compilar el proyecto.

Este viene incluido con una makefile, por lo que solamente ejecutar el comando 'make' debería ser suficiente para compilar. Una vez hecho, simplemente se usa la entrada estándar para abrir el archivo usando un archivo de entrada de su selección.

Conclusión

Usando prefetching se puede reducir de manera muy significativa la tasa de fallos, y aunque en este proyecto no se visualiza, el pre-cargar bloques reduce el tiempo de acceso al solicitarse una dirección de memoria. Estos métodos de prefetching tambien se pueden implementar juntos para obtener los beneficios de ambos. Si se realiza prefetching secuencial, y luego prefetching de salto, ambos métodos pueden funcionar sin interrumpirse el uno al otro gracias a que la tabla de predicción estará en acuerdo con el prefetching secuencial en caso de que se repita cualquier patrón de acceso secuencial.

Juntos, los métodos utilizan los principios de la localidad espacial y temporal para mejorar la eficiencia de las memorias cache. Ademas, estos no son los únicos métodos de prefetching. Existen otros algoritmos, ademas de el prefetching de hardware, que pueden incrementar la eficiencia de una memoria cache aun mas.