Simulador de memoria Caché en C++

José López C.I.: 30.077.008
Edgar Gutiérrez C.I.: 28.505.513

Mayo 2023

1 ¿Qué es la memoria caché?

La memoria caché es una memoria de alta velocidad que se utiliza para almacenar temporalmente los datos que se utilizan con mayor frecuencia. Esta memoria se encuentra en la CPU y se utiliza para acelerar el acceso a los datos, reduciendo el tiempo que tarda la CPU en acceder a la memoria principal.

La memoria caché funciona de manera similar a una biblioteca. Cuando un usuario necesita un libro, primero busca en la biblioteca local. Si el libro no está disponible, el usuario busca en una biblioteca cercana. Si el libro aún no está disponible, el usuario busca en una biblioteca más grande y así sucesivamente hasta encontrar el libro deseado.

De manera similar, la CPU busca los datos en la memoria caché. Si los datos están disponibles, la CPU los utiliza directamente desde la memoria caché, lo que reduce el tiempo de acceso. Si los datos no están disponibles en la memoria caché, la CPU busca en la memoria principal y los almacena en la memoria caché para su uso posterior.

2 Funcionamiento del programa

El funcionamiento del programa consiste en el uso de una clase Cache, la cual se encarga de almacenar los respectivos datos de la cache, como lo es su bit de validez, su etiqueta, la dirección que la llamo, así como también respectivos contadores para tipos de reemplazamiento como lo es LRU y LFU.

El programa inicia realizándole preguntas al usuario al respecto de que parámetros desea para su cache. Asimismo, una vez obtenga todos los datos se generará una matriz dinámica de clase Cache, de forma tal que las filas de la cache serán los conjuntos de la misma, mientras que las columnas representaran las vías de la cache.

Este simulador obtendrá las direcciones de un archivo entrada.in, y una vez

obtenido este será procesado de tal forma que según los parámetros ingresados como lo pueden ser "cantidad de palabras por bloque" o el "tipo de correspondencia" darán lugar a que el programa separe la dirección en "bits de bloque, "bits de etiqueta" y finalmente "bits de desplazamiento", todo ello será utilizado para determinar en qué parte de la memoria cache deberá ser almacenado dicha dirección.

Asimismo, durante el funcionamiento del simulador este procede a verificar si la dirección obtenida da un acierto. Es decir que se encuentra en la cache, sino se encuentra se tomara como fallo, dando lugar a que el programa ejecute un tipo de reemplazamiento, seleccionado por el usuario ya sea "LRU", "LFU", o "Random". De esta forma, el programa decidirá a que elemento reemplazar, según el caso solicitado. Cabe destacar, que si el usuario decidió mantener activo la opción de "Prefetching" entonces podría cambiar el resultado de la cache, ya que si la cache no tiene dicho elemento, pero el Prefetching si entonces, será un acierto.

De igual forma, es importante mencionar que el prefetching utilizado se basa en buscar predecir mediante la localidad espacial, puesto que buscara anticipar una dirección cercana a la última obtenida. De igual manera, el prefetching funciona en conjunto con un buffer de prefetching, este último almacena temporalmente los datos anticipados por el prefetching, y antes de ser enviado a la cache es verificado si es un acierto o fallo, esto ayuda a que la cache no se ensucie con direcciones anticipadas que resultaron erróneas evitando así la posibilidad de que ocurra una mayor tasa de fallos por malas predicciones.

Finalmente, se tiene que todos estos datos serán mostrados en un archivo siempre y cuando el usuario seleccione la opción correspondiente para imprimir el archivo, este archivo representa la caché de forma tal que por cada dirección leída mostrará si fue un acierto o fallo. Así como también, una representación gráfica de la caché mostrando su bit de validez, etiqueta y dirección leída, junto con sus respectivas vías. Una vez que todas las direcciones hayan sido leídas, al final del archivo se encontrará el respectivo porcentaje de tasa de aciertos, así como también de fallos. Esto permite tener una mejor comprensión del rendimiento de la caché y cómo se están manejando los datos en ella.

Los casos de pruebas fueron realizados mediante el uso de un archivo externo denominado tiempos.cpp el cual daba uso a la librería "chrono" todo ello con el fin de saber el tiempo de ejecución del programa por máquina, en diversas condiciones. Desde distintos tamaños de cache, distintos tipos de correspondencia, así como también el tiempo de ejecución al mostrar la representación grafica de la cache

Especificaciones PC1
Procesador: i5-3470 (4 nucleos, 4 hilos)
RAM: 8gb Ddr3 1333Mhz
Almacenamiento: SSD (240gb)
Version Linux: Ubuntu 22.04.1 LTS
Version Kernel: 5.15.90.1-microsoft-standard-WSL2 basada en la serie kernel 5.15 de Linux

	1024	2048	4096	8192	16384
Directa SP	8.09s	11.92s	16.20s	26.42s	47.35s
Directa CP	8.02s	11.82s	16.28s	25.66s	46.20s
4 Vias SP	8.30s	11.63s	16.35s	22.93s	41.73s
4 Vias CP	8.94s	11.50s	16.18s	24.22s	42.84s
Totalmente SP	8.43s	14.55s	24.05s	42.28s	81.17s
Totalmente SP	8.00s	15.11s	24.16s	40.68s	81.70s

Table 1: **Cuando el programa imprime**

	1024	2048	4096	8192	16384
Directa SP	1.72s	1.72s	1.76s	1.78s	1.79s
Directa CP	1.71s	1.77s	1.77s	1.72s	1.78s
4 Vias SP	1.65s	1.75s	1.69s	1.74s	1.77s
4 Vias CP	1.70s	1.70s	1.68s	1.74s	1.72s
Totalmente SP	1.71s	1.71s	1,77s	1.71s	1.75s
Totalmente SP	1.79s	1.74s	1.69s	1.78s	1.73s

Table 2: **Cuando el programa no imprime**

Especificaciones PC2
Procesador: Intel Celeron (2 nucleos, 2 hilos)
RAM: 2gb Ddr3 1333Mhz
Almacenamiento: HDD (320gb)
Version Linux: LMDE 5 (elsie)
Version Kernel: 5.10.0-12-amd64

	1024	2048	4096	8192	16384
Directa SP	18.66s	26.70s	24.02s	43.80s	54.92s
Directa CP	22.70s	28.06s	27.70s	38.54s	51.25s
4 Vias SP	27.44s	19.83s	30.57s	34.52s	48.02s
4 Vias CP	24.64s	20.88s	22.12s	34.97s	54.07s
Totalmente SP	26.80s	23.07s	36.42s	46.56s	85.92s
Totalmente SP	22.56s	28.02s	36.43s	63.39s	81.58s

Table 3: **Cuando el programa imprime**

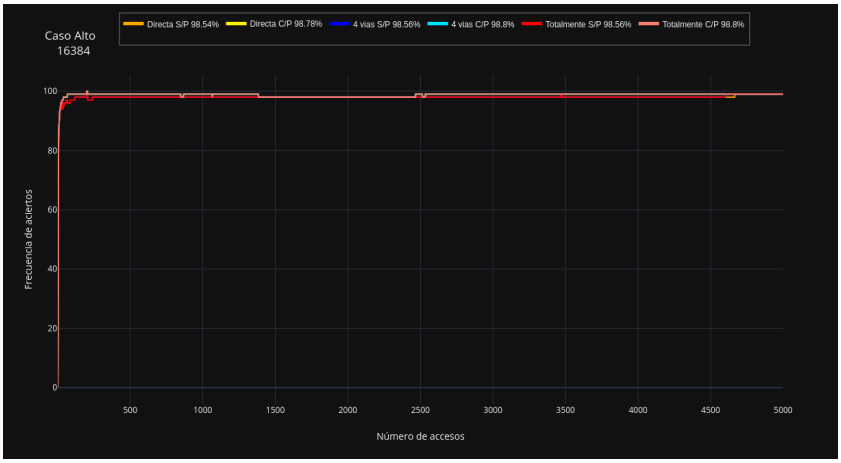
	1024	2048	4096	8192	16384
Directa SP	2.66s	4.79s	9.40s	16.92s	33.85s
Directa CP	2.77s	4.82s	9.43s	17.51s	33.73s
4 Vias SP	2.45s	3.90s	7.64s	14.50s	29.70s
4 Vias CP	2.40s	3.89s	7.45s	14.73s	29.87s
Totalmente SP	4.39s	8.67s	17.33s	33.47s	67.27s
Totalmente SP	4.82	8.81s	17.26s	34.31s	66.30s

Table 4: **Cuando el programa no imprime**

Tras analizar los resultados en ambas computadoras, se tiene que a medida que mayor tamaño es la cache esta tarda más tiempo, esto es debido a que son más sitios a los que debe acceder, por ende, más tiempo ocurrirá en cada uno de esos accesos. Asimismo, cabe destacar que la utilización de correspondencias como la directa, o asociativa por conjuntos, estas mantienen una similitud en el rango de tiempo. Sin embargo, en la PC2 en algunas ocasiones la asociativa por conjuntos muestra mejores resultados en tiempo. Esto es debido a que, si bien la correspondencia directa es más ágil, esta es más propensa a tener errores que la asociativa por conjuntos, y es allí donde esa penalización de fallo, afecta su rendimiento en el tiempo, llevando a cabo de esta manera que la asociativa por conjuntos sea más ágil en algunas ocasiones. De igual forma, se puede realizar observaciones en cuanto a la correspondencia totalmente asociativa, debido a que esta es la que muestra un peor desempeño en cuanto a velocidad, esto se debe principalmente a que cada bloque de memoria puede ser almacenado en cualquier ubicación de la memoria cache, lo que provoca que el procesador deba comparar la dirección de memoria solicitada con todas las etiquetas de la cache almacenadas en la memoria cache con el fin de determinar si el bloque de memoria se encuentra en la memoria cache. Este proceso de búsqueda es mucho más lento que la correspondencia directa o asociativa por conjuntos, donde la búsqueda se limita a una ubicación especifica o un conjunto limitado de ubicaciones en la memoria cache.

CASO ALTO

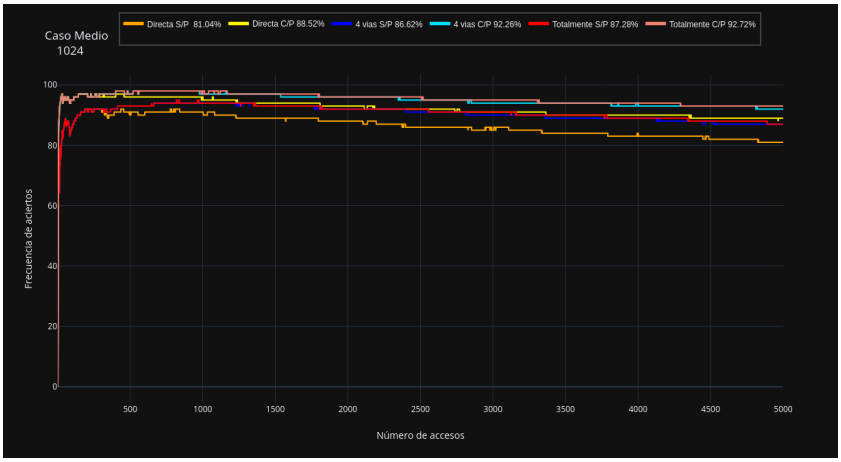
En cuanto al caso alto se tomó una serie de direcciones, que fomentaban un caso ideal para la memoria caché, en donde estas poseían alta posibilidad de encontrarse con una dirección ya utilizada, ya sea para aprovechar la localidad temporal, así como también en la localidad espacial, obteniendo así también direcciones cercanas a las últimas obtenidas. En cuanto al caso alto, se tuvo como límite más alto 98.8%, se tiene que todos los casos de prueba realizados fueron capaces de llegar a este valor, donde las memorias caché de 1024 y 2048 bytes fueron capaces de llegar gracias a la correspondencia totalmente asociativa, mientras que el resto alcanzaban llegar no solo con la correspondencia totalmente asociativa. Sino que también, con la asociatividad por conjuntos de 4 vías. Se considera que la asociativa por conjuntos llegó a alcanzar el mismo valor de porcentaje que la totalmente asociativa debido a que a partir de los 4096 bytes en la memoria caché era suficiente sustento para almacenar los casos y evitar posibles fallos. No obstante, el motivo por el cual quedará hasta 98.8% es ocasionado por aquellos fallos que fueron por direcciones accedidas por primera vez las cuales el prefetching no pudo predecir.



CASO MEDIO

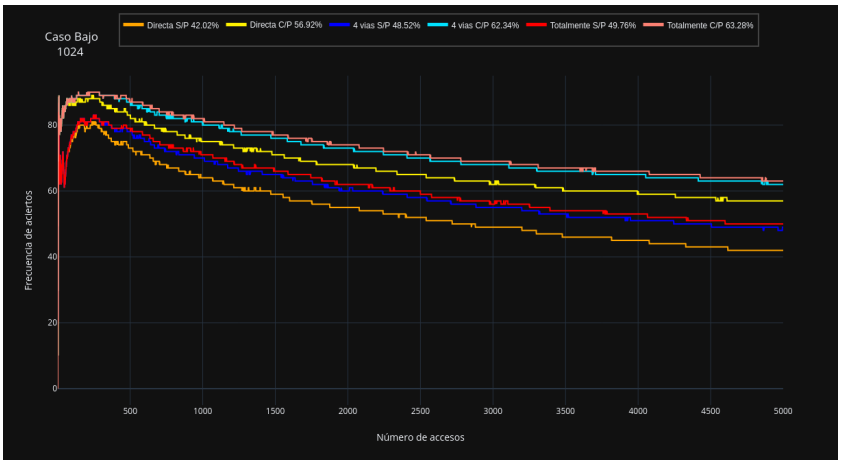
Se tiene que, en el caso medio, se utilizó una serie de direcciones que simulaban un caso ni muy idealizado, ni muy crítico. En donde el cache mostro un buen desempeño en todas las dimensiones y tipos de correspondencia, siendo el valor más bajo 88.52% con prefetching activo. Esto fue en un caso de prueba realizado en la memoria cache de 1024 bytes con el uso de la correspondencia directa. Mientras que el más alto fue de 96.26% con prefetching en una prueba realizada con una memoria cache con capacidad de 16384 bytes y con un tipo

de correspondencia totalmente asociativa



CASO BAJO

Se tiene que el caso más bajo de todos fue el de la memoria cache con 1024 bytes esto debido a que, si bien la cache es de poca capacidad, también esta tiene un bajo porcentaje de aciertos debido a la naturaleza de las direcciones recibidas, puesto que estas si bien tienen direcciones donde se aproveche la localidad espacial y temporal, es en baja cantidad su aparición, lo que provoca que la tasa de aciertos disminuya. Cabe destacar que en el grafico se observa como al inicio comienza alto, esto es debido a la naturaleza del generador, que al inicio tiende a generar valores utilizados recientemente (para aprovechar la localidad temporal). Sin embargo, a medida que el programa avanza, se va observando el desempeño de la cache de 1024 bytes, y como a medida del transcurso de más direcciones leídas, va bajando. Se tiene que en cuanto a los tipos de correspondencia la que dio un peor resultado utilizando prefetching fue la correspondencia directa, con un 59.92%, mientras la mayor fue 63.28% siendo esta la correspondencia totalmente asociativa, esto debido a que la correspondencia totalmente asociativa es más eficiente que la directa, ya que hay flexibilidad para que cualquier bloque sea remplazado, al momento de sobrescribir uno ya existente en la cache, así como también el uso de un método de remplazo como lo es LRU, buscara siempre intercambiar el elemento que ha sido menos utilizado recientemente.



El proyecto se subió a un repositorio de GitHub que contiene los casos de entrada utilizados en la caché, así como también almacena imágenes y archivos .csv que representan dichos casos en diversas pruebas realizadas en la caché, como lo fueron su tipo de correspondencia o tamaño. Este repositorio, a su vez, tiene los archivos que fueron utilizados en las pruebas de tiempo, como es entradaAux.in y tiempos.cpp. Además, consta de un archivo realizado en markdown donde da una breve introducción del programa y por supuesto también abarca el codigo principal, con su respectivo makefile.