Cache Analysis

CDA 3101 - Christian Scaff

Introduction

The following paper will analyze the cache, specifically the effects of a change in cache size and associativity on the hit rate of the cache. In memory architecture, a cache is a piece of hardware such as SRAM or DRAM where data can be temporarily stored for quick but more expensive access. This can improve performance when one piece of memory is accessed more frequently as the slower main memory does not have to be fully accessed when a piece of memory is found in cache memory.

The program created simulates three types of caches: direct mapped, fully associative, and n-way set associative. The overall simulation uses a map data structure where the keys are set indexes and the value stored are vectors of cache lines. A cache line is the smallest component of a cache where an individual piece of main memory known as a block is stored. In a cache, a certain number of sets store a certain number of lines. In this program, each line is represented by a line object that stores the cache tag, a Boolean illustrating if the line is occupied or not, and a count used in replacement strategy.

Direct Mapped

The direct mapped cache simulation involves a cache with a given number of lines such that an inputted address to a block of memory always directly maps to a specific line. The simulation program treats this as a map of sets such that the number of sets equals the number of lines. Each set is mapped to a vector with one index to represent one line. The direct mapped simulation takes in a memory address and decodes it to retrieve the number of the line as well as the tag to be stored. The line number modulo the number of lines in the cache is calculated to find the actual mapped line available in the cache. The program then accesses the set key corresponding to that line number and checks the singular available vector index for that tag. If the tag exists, a hit is returned whereas if the tag differs or the line is empty, that new tag is stored, and a miss is returned. Afterwards, the hit rate is calculated based on the sum of hits divided by the sum of accesses. This effectively simulates storing a tag in a cache of lines with no sets. It should be noted that in this simulation, only one tag can be stored in a line such that any time a new tag is mapped to an already occupied line, it is overwritten.

Fully Associative

The fully associative cache simulation involves a cache with a given number of lines such that any block can be stored in any available line. When a fully associative cache has no more empty lines, replacements are made based on different strategies such as removing the least recently used line or the oldest line. The simulation created uses a map where there is only one set index that maps to one vector of all available lines. The fully associative cache simulation takes in a memory address and decodes it to retrieve the tag of the block being stored. The only lines vector is then traversed until the desired tag is found and a hit is returned, or an empty line is found. Because tags are stored consecutively, an empty line would indicate there are no more occupied lines to check for a matching tag. When the empty line is found, the tag is inserted, and a miss is returned. If the end of the lines vector is reached, it can be known that there are no available lines, and one of the two replacement strategies noted ensues.

N-Way Set Associative

The n-way set associative cache involves multiple sets that store multiple lines. Inputted addresses are mapped to specific sets where any available line in that set can be used to store a block tag. The simulation takes in a memory address and decodes it to retrieve the mapped set number and the tag being stored. Then the map index corresponding to the mapped set is accessed and the corresponding vector of lines is traversed by the same logic in the fully associative cache where any line can be used to store a tag and replacement strategies are used when the set is full.

Replacement Policy

The two replacement policies used in this simulation are “first in, first out” (FIFO) and “least recently used” (LRU). When all available lines are full in a set (where a fully associative cache can be thought of as one major set) and in an n-associative cache, these two policies are used to determine which line should be replaced. FIFO is implemented by updating an access count every time the cache is accessed. Whenever a new tag is entered into the cache, its respective line object’s count variable is updated to match the access count. Thus, the “first in” or oldest stored line is the one that holds the lowest count in the given set. Thus, as lines are traversed in each set, the minimum count is calculated and stored. The vector index with this minimum count in the given set in the cache is then accessed to replace the tag. LRU is implemented by updating the same count when a given tag being accessed in the cache is found to already exist or a “hit” occurs. Thus, lines with higher access counts convey more recent accesses. Then the same traversal and minimum is found for the replacement. This is because the lowest stored count would be the line object with the least stored accesses or the least recently used.

Test Description

Test Parameters

Forty-five tests were performed with the following parameters: cache size, block size, associativity, and replacement strategy. The cache size determines the number of bytes the cache has available. The block size determines the size in bytes of the blocks to be stored in cache lines. Thus, it determines the size of the cache lines. The cache size and block size can be used to derive the number of lines in the cache. Then, associativity indicates the number of lines per set in the cache. One can then derive the number of sets in the cache to create the map data structure. The associativity can then be used to create the lengths of vectors in the map that store the lines for each set. Finally, the two replacement strategies are “FIFO” and “LRU” as demonstrated earlier and dictate how replacements will occur when there are no available cache lines for set associative and fully associative caches.

Parameter Table

Test	Block Size (Bytes)	Associativity (Lines/Set)	Replacement Strategy
1	32	1 (Direct Mapped)	No strategy
2	32	2	FIFO
3	32	2	LRU
4	32	4	FIFO
5	32	4	LRU
6	32	8	FIFO
7	32	8	LRU
8	32	All lines in one set (Fully)	FIFO
9	32	All lines in one set (Fully)	LRU

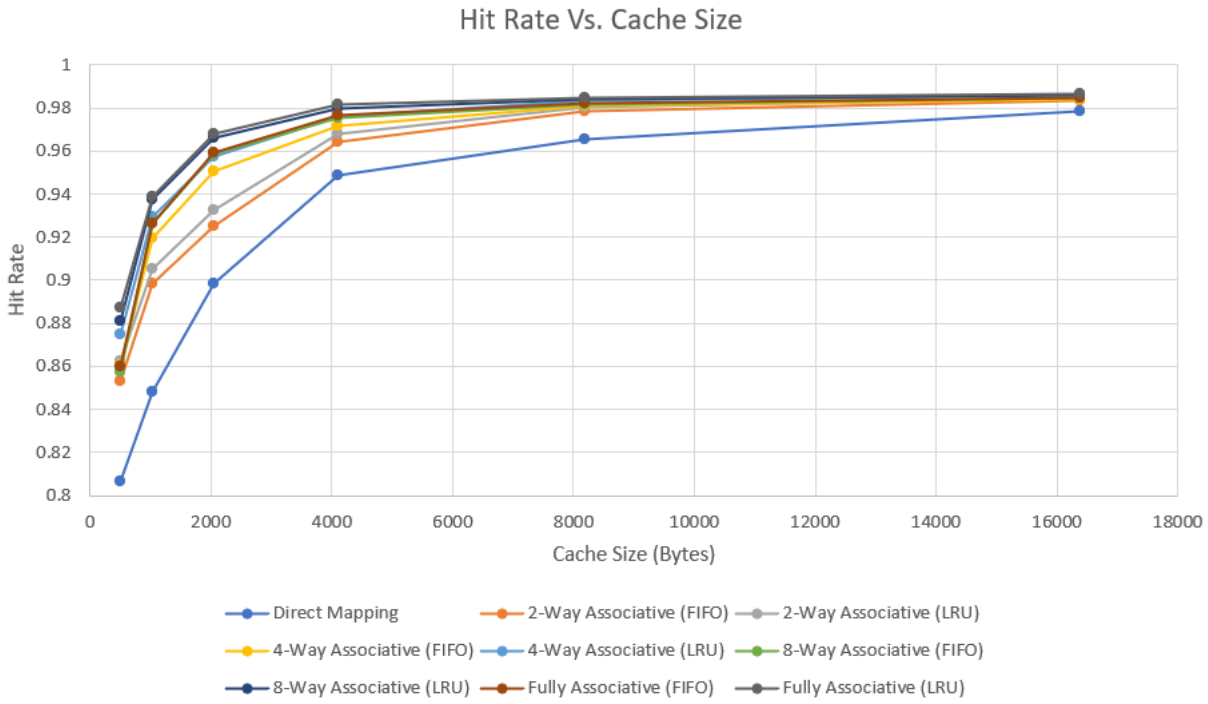
The last parameter, cache size, was not included in the table to avoid a lengthy redundant table. The nine tests illustrated were ran for each cache size in the table below.

Cache Size (Bytes):	512	1024	2048	4096	8192	16384

These parameters were chosen with a focus on determining how a variable cache size, associativity, and replacement strategy would affect a cache's hit rate or how well a cache is at accessing more frequented memory. Thus, block size was left constant for all tests as to ignore its effects on hit rate and focus only on associativity, replacement strategy, and cache size. Associativity was varied between the three major cache types: direct mapped, set-associative, and fully associative. Three variations of set associative were investigated to understand how a higher associativity affects hit rate. For each associativity, two different replacement strategies were tested. This allows one to understand the effect of replacement strategy on hit rate at different associativity's. By testing different replacement strategies on the same associativity and block size, one can ignore these constant values and see the direct relation between a variable replacement strategy and hit rate. This also applies to understanding associativity's effect on hit rate as one can look at variable associativity's for one replacement strategy to ignore its effect and see the direct relation between associativity and hit rate.

Results

The following graph demonstrates the change in hit rate in response to a variable cache size across the nine tests illustrated before and as depicted in the table key.



Conclusion

Cache Design

Looking at the effect of cache design on hit rate, it can be noted that higher associativity increases hit rate. This can be seen when comparing the different associativity's on the graph for one type of replacement strategy. It is important to compare the associativity's within one type of replacement strategy as to look at the direct effect of associativity on hit rate without the interference of different replacement strategies. Looking at the graph for LRU and then for FIFO, it is seen that fully associative has the highest hit rate, followed by 8-way set associative, 4-way set associative, 2-way set associative, and then direct mapping in last. This is most likely due to the increased likelihood of conflicts as associativity decreases. As associativity decreases, the number of sets in the cache increases whereas the lines in each set decreases. Thus, there is a heavier focus on lines being mapped to specific areas of the cache. As a result, certain lines could be replaced even when there are empty lines elsewhere in the cache, even if the replaced lines are useful and potentially frequented such that they contribute more to a higher hit rate. This leads to increased misses and overall lower hit rate as associativity decreases.

Replacement Policy

Based on the results of the simulation, it can be said that the “least recently used” or LRU policy provides an overall better hit rate than the “first in, first out” or FIFO policy. When comparing the hit rate between replacement strategies for each individual cache design, LRU continuously offers a higher hit rate. This is likely due to the LRU policy removing less used pieces of memory which make up lines that are accessed less and contribute less to the overall hit rate. On the other hand, FIFO removes the oldest available line even if this line is the most frequently accessed and offers the most to the hit rate. Therefore, LRU is likely to prioritize higher hit rate over FIFO as demonstrated in the simulation.

Cache Size

As cache size increases, hit rate increases on an asymptotic curve, eventually reaching a horizontal tangent line where hit rate levels out and becomes roughly equivalent across cache designs. Hit rate increases with cache size initially as smaller caches have less space to store lines of memory and have conflicts or replacements more often. When a cache is small enough for conflicts or replacements to be occurring throughout the cache, it is less likely for frequented pieces of memory to consistently exist in the cache as they are constantly replaced due to the lack of space for new lines. This causes more misses. As the cache size increases, there is more space for new memory to be stored and the risk of more frequented lines of memory being replaced continuously goes down. Thus, the hit rate increases. However, eventually there is a point where the cache has become so large, there are minimal conflicts and replacements occurring as most blocks are able to be stored in the available lines in the cache, causing a relatively high and similar hit rate across designs and replacement strategies.