Cache Analysis Paper

Introduction:

The purpose of this cache simulator programming assignments is to analyze and explore the findings of how different cache sized associated with different cache designs affect the performance (hit rate).

The cache memory is built into CPU to increase data retrieval speed by reducing the need to access main memory. When it comes to cache designs, there are three categories' caches fall into, direct mapped, n-way set associative, and fully associative. This analysis will also discuss the use of two different replacement strategies, least recently used (LRU) and First in first out (FIFO), which are used in cache management to decide which item to remove when cache is full. The goal of this analysis is to accurately analyze the findings of the cache hit rates from the provided cache_sim.cpp program.

In this project, we are analyzing how the performance of a cache changes with its associativity (design), how the performance of a cache changes with varying cache sizes, and how the performance of a cache changes with different replacement policies. Note: the performance is determined by the cache hit rate by tracking the cache hits and missed over time. Additionally, there is a cache hit if the requested data is found in the cache (there is a match in the tag bits) leading to faster access. There is a cache miss if the requested data is not found in the cache, where slower main memory will be used to find the requested data.

Description of Tests:

When running the provided cache_sim.cpp program, there are a total of 5 parameter to be considered, the size of the cache (bytes), size of block (bytes), the associativity referring to the cache design, the replacement strategy being used if applicable, and the trace file needed to run the simulation (MB). These parameters are chosen as the cache_sim.cpp program depends on various factors related to a specific cache design being simulated. Therefore, these 5 parameters provide flexibility for simulating different caches configurations and replacement strategies, making it easier to analyze different cache designs. Two out of five of these parameters are kept constant, with size of block being 64 bytes long and using the "swim.trace" file of 4.8 MB. These parameters need to be constant throughout the analysis as it allows for a more controlled experiment and fair comparisons between the different cache designs. To look closer, by keeping the block size constant, we can focus more on how the caches size, associativity, and replacement policies affect cache hits, misses, and overall performance.

For each test the parameter for size of cache, entered as an exponent of 2, was 10, 11, 12, 13, and 14 equating to a cache size of 1024, 2048, 4096, 8192, and 19384 bytes respectively. When performing a run, I would first determine the associativity being tested, input all 5-cache sizes to see how a smaller cache size affects the larger one. Coming to the conclusion that large cache

can hold more data, which reduces the likelihood of a cache missing, but also increase the access time as there are more memory locations to search for.

The next input prompted the user for the size of cache line or block entered as an exponent of 2. For this parameter, I kept it constant with value 6 constituting to a block size of 64 bytes for each run.

The associativity is the size of sets, and it influences how cache lines are mapped to sets. Determining the associativity of the cache being either direct-mapped, fully associative, or n-way set associative is another parameter that needs to be considered. Cache associativity refers to how the cache is organized in terms of mapping main memory addresses to cache locations. In a direct-mapped cache, each block of main memory can be placed in exactly one specific cache line. Direct-mapped associativity can lead to more conflicts because it's possible that multiple main memory blocks may map to the same cache line.

In fully associative cache, each block of main memory can be placed in any cache line and there are no restrictions in mapping. This comes at the cost of having longer access time. The hit rate of fully associative tends to be higher since any block can go to any location.

When discussing an n-way set associative cache, it tends to be in a balance between direct-mapped and fully associative designs. For set associative designs, the cache is divided into sets, each containing multiple lines. The number of lines in each set is referred to as the associativity (n, where n == 2, 4, or, 8). The mapping is done in such a way that a block in main memory can be placed in any line within a specific set. Due to this behavior, set associative reduces conflicts when compared to direct-mapped and maintains simplicity when compared to a fully associative cache design.

Another parameter that needed to be determined to better analyze cache performance is weather the associativity uses a LRU or FIFO replacement strategy. Replacement strategies refer to the policy used to determine which cache line to replace when a new block of data needs to be bought into the cache when cache is full. The goal is to remove the cache line that will be hit less frequently. It's important to note that different strategies may be more suitable for different applications and that one size does not fit all.

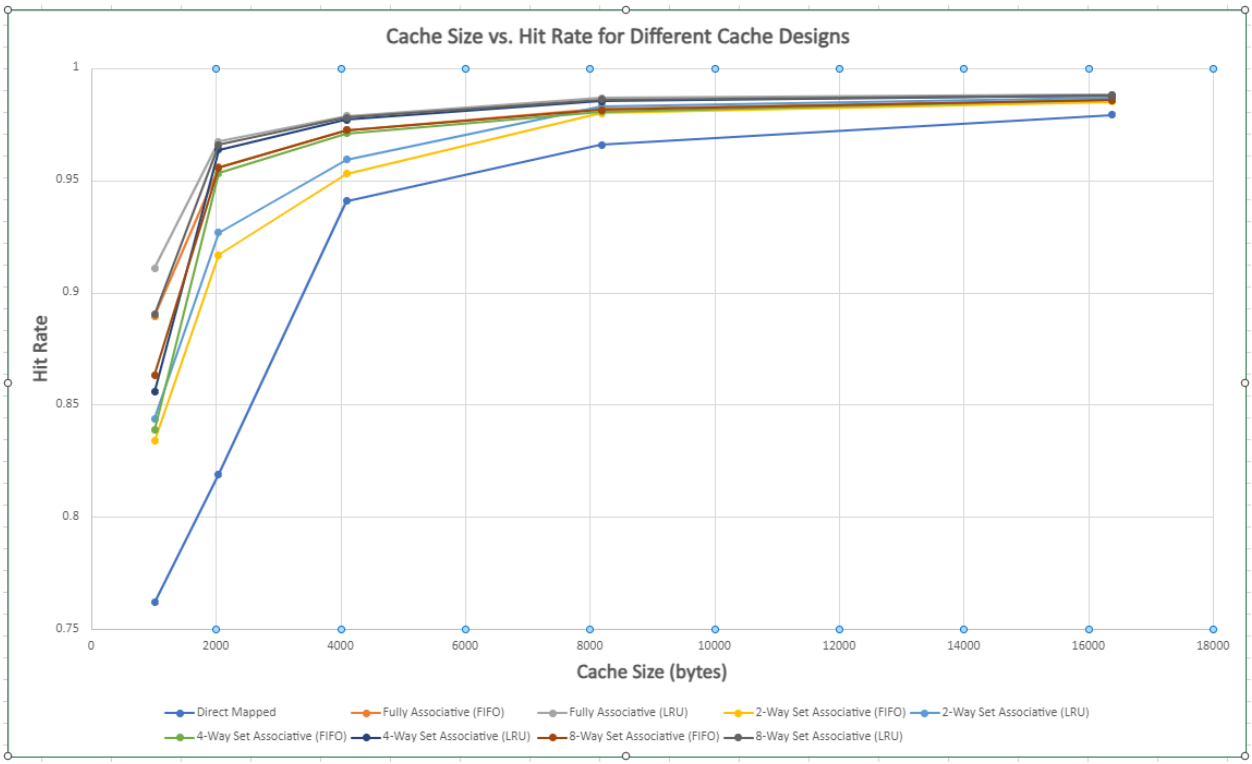
Two common replacement strategies are LRU (Least Recently Used) and FIFO (First-In-First-Out):

LRU is a replacement strategy that removes the cache line that has not been used for the longest period. The idea is to keep the most recently accessed data in the cache, assuming that recently used data is more likely to be used again in the near future. To dive deeper, the LRU strategy is implemented using a counter for each cache line which is updated whenever the line is accessed. When a new block needs to be added to the cache, the cache line associated with the lowest counter will be removed and replaced.

FIFO is a replacement strategy that removes the cache line that has been in the cache the longest, based on the order of insertion. For example, the first block that was loaded into the

cache is the first to be evicted when space is needed for a new block. However, this strategy may not always capture the variations in access patterns.

Results:



	A	B	C	D	E	F
1		1024	2048	4096	8192	16384
2	Direct Mapped	0.761869	0.818901	0.94081	0.965613	0.979135
3	Fully Associative (FIFO)	0.889424	0.955768	0.972246	0.981678	0.985788
4	Fully Associative (LRU)	0.91078	0.967183	0.978661	0.986652	0.987981
5	2-Way Set Associative (FIFO)	0.833859	0.916713	0.952816	0.979904	0.984864
6	2-Way Set Associative (LRU)	0.843866	0.926588	0.959168	0.982994	0.986537
7	4-Way Set Associative (FIFO)	0.838727	0.953149	0.97086	0.980573	0.985583
8	4-Way Set Associative (LRU)	0.855799	0.963462	0.976843	0.985052	0.987651
9	8-Way Set Associative (FIFO)	0.863259	0.955435	0.972272	0.981197	0.98564
10	8-Way Set Associative (LRU)	0.890113	0.96583	0.977892	0.985969	0.987767

Conclusions:

- How does cache design (direct mapped/set associative/fully associative) affect hit rate?

The design of a cache, specifically the choice between direct-mapped, set-associative, and fully associative cache architectures, can significantly impact the cache hit rate. The hit rate is a measure of how often a requested piece of data is found in the cache without needing to access

the slower main memory. These associativity differ in the how the cache is organized in terms of mapping main memory addresses to cache locations.

In direct mapped, the hit rate is the lowest because of all the conflict misses that occur when mapping multiple main memory blocks to one specific cache line causing the block at that line to be replaced. When looking at the results for direct-mapped and cache size of 4096 bytes, we see the hit rate be 0.9408, which if fairly low when compared to the others.

In a set associative cache design, the cache is divided into sets, and each set contains multiple lines. A memory block can be placed in any line within a specific set. This design helps reduce conflict misses compared to direct-mapped caches because there are multiple options within a set. Therefore, the hit rate in a set-associative cache is higher than a direct-mapped cache. Associativity (number of lines per set, n== 2, 4, or 8) plays an important role in this; higher the associativity, results in fewer conflicts, which leads to a higher hit rate. Taking a closer look at the results for set associativity with cache size of 4096 bytes, we see that the hit rate increases as the number of lines per set increases. With n = 2, 4, and 8, we have hit rates of 0.952, 0.970, and 0.972 respectively.

When it comes to a Fully Associative cache, any block of main memory can be placed in any cache line, which in turn significantly reduces conflict misses. Fully associative generally has the highest hit rate because these designs generally go well with many different access patterns. Additionally, there are very little chances of conflict missing in fully associative as any block can be placed anywhere in the cache. From the results we obtained through running the program, we see the hit rate, at a cache size of 4069 bytes, to be 0.972 which is on the higher end compared with all the different associativity.

- How does hit rate change with the two replacement policies?

To begin, direct-mapped cache designs doesn't use a replacement policy at is a one-to-one mapping. When a new block of data needs to be stored in the cache, it replaces the existing in the cache line that corresponds to its mapped location.

In my cache simulation analysis, I used the swim.trace file. From this file we observe from the analysis that the cache designs that used the LRU (Least Recently Used) replacement strategy, had a higher hit rate than cache designs that used the FIFO (First-In-First-Out) replacement strategy. This tells me that the "swim.trace" file compliments the idea of replacing the lease recently used addresses in the cache and storing the new address that was recently used.

Examining the graph and chart respectively, I analyzed LRU and FIFO replacement strategies for a fully associative cache, 2-way set associative, 4-way set associative, and 8-way set associative caches. Through this analysis, I found that for reach run (keeping the size of cache, associativity, size of block, and trace file used all constant), the cache design that used LRU had a greater hit rate than that of the FIFO replacement strategy, with the hit rate varying by an average of about 0.0035.

- How does cache size affect hit rate?

The size of a cache has a large impact on the hit rate. Cache is a subset of main memory, we want to put things in the cache that are likely to be accessed, and we do this by using Temporal Locality and Spatial Locality. Temporal locality means that recently accessed items from a program may be accessed again. So, having a large cache will retain more data that was recently used, increasing the hit rate. On the flip side, having a smaller cache will decrease result is a smaller hit rate ratio as there are less lines in the cache to put recently accessed data.

Spatial locality refers to the tendency of a program to access memory locations that are close to each other. A larger cache size will allow more data to be stored, which will increase the likelihood that nearby memory will be a hit because it is present in the cache. The vice versa is also true, a smaller cache will store less blocks leading to a lower hit rate.

Taking a look at the results, we observe that a cache size of 1024 bytes has an average hit rate of all cache designs is 0.85422 and the average hit rate of a cache size of 4096 bytes is 0.96633. With that, we can confirm that a smaller cache, results is a lower hit rate and a larger cache results in a higher hit rate.