CSCE 2303

Computer Organization and Assembly Language Programming

Project 2

Cache Simulator

Instructor: Dr. Mohamed Shalan

Summer 2024

Arwa Abdelkarim
Rana Taher
Yasmina Mahdy

Department of Computer Science and Engineering

Table of contents

Introduction

The purpose of this project is to create a cache simulator that simulates the functions of a Direct Mapped (DM) and a Fully Associative (FA) cache and to measure their hit rate for different cache block sizes, as well as comparing and analyzing the hit rates calculated. A cache is an SRAM block of memory used to counter the fact that the memory (DRAM) is slower in reading and writing data to and from it. The reason the DRAM is used to implement the main memory instead of the SRAM is because it is way cheaper to implement, and allows us to have a large, dense memory, and so the cache is needed to create the illusion that we can access all of this huge memory at a much smaller time (hit time). Everytime data is requested by the CPU, it sends the required address to the cache to check if it is already stored there, in which case the data can be quickly returned to the CPU and the request is thus identified as a cache hit. If the requested address is not in the cache, then the cache has a miss and must retrieve it the requested data from the main memory, which takes a longer time and thus slows down the CPU. As a result, it is desirable to have as many hits and as few misses as possible. Formally, measuring the performance of a cache is achieved by finding the hit rate (HR) or the complementary miss rate. The hit rate (H.R) is calculated using the equation H.R = number of hits / number of addresses requested by the CPU. The way the cache checks its addresses as well as how it stores them identifies what type it is. There are three main types of caches, direct mapped (DM), Fully Associative (FA) and Set Associative (SA), however the first two are actually special cases of the more generic SA. The DM cache uses the concept of indices to store its data, so that each address can only ever be found at its respective index and any two different blocks with the same index will overwrite each other. FA cache, on the other hand, allows its data blocks to be stored anywhere and uses a replacement policy (e.g. random, FIFO, etc.) to overwrite elements if there is no space for the next data block. The SA cache type is a generic form of both two, where each address can only be mapped to just one set, however, each set can contain more than one way, all of which can be used to store the data. When data is read by any type of cache, it is typically not read as single byte, but rather a line of bytes is read at a time to make use of the concept of spatial locality, which dictates that the elements near a recently accessed one is likely to be accessed again in the near future. As a result, different line sizes can result in different numbers of hits depending on how much they make use of spatial locality. Temporal locality, on the other hand, refers to the notion that a requested element is likely to be requested again in the future, and is also directly affected by the line size, but inversely unlike spatial locality. As a result, it proved imperative to study which memory access patterns utilize what type of locality more, and how this reflected in the effect line size variation had on cache performance, which is what we sought to do in this project, as well as studying the difference in performance between the DM and FA cache types.

Methodology

To be able to simulate how the cache works we wrote a C++ program that simulates the Direct Mapped and Fully Associative caches. The code does that by not only simulating how the compulsory misses and hits by spatial locality are generated but also simulating how conflict and capacity misses are handled in DM and FA respectively. To test the functions for DM and FA we used the given MemGen functions **memgen1**, **memgen2**, **memgen3**, **memgen4**, **memgen5**, and **memgen6**. Where **memgen1** is used to generate addresses sequentially from 0 to DRAM size - 1 , **memgen4** and **memgen5** both generate addresses sequentially but the ranges are 0 to 4 KB - 1 and 0 to 64 KB - 1 respectively. Moreover, **memgen2** and **memgen3** generate addresses randomly and range from 0 to 24 KB - 1 and from 0 to

DRAM size - 1 respectively and **memgen6** generates the addresses from 0 to 256 KB - 1 with increments of 32. We used these functions to generate hit rates for one million iterations for each MemGen function and for each cache block size for both the FA and DM caches and we plotted graphs for both of them.

Results

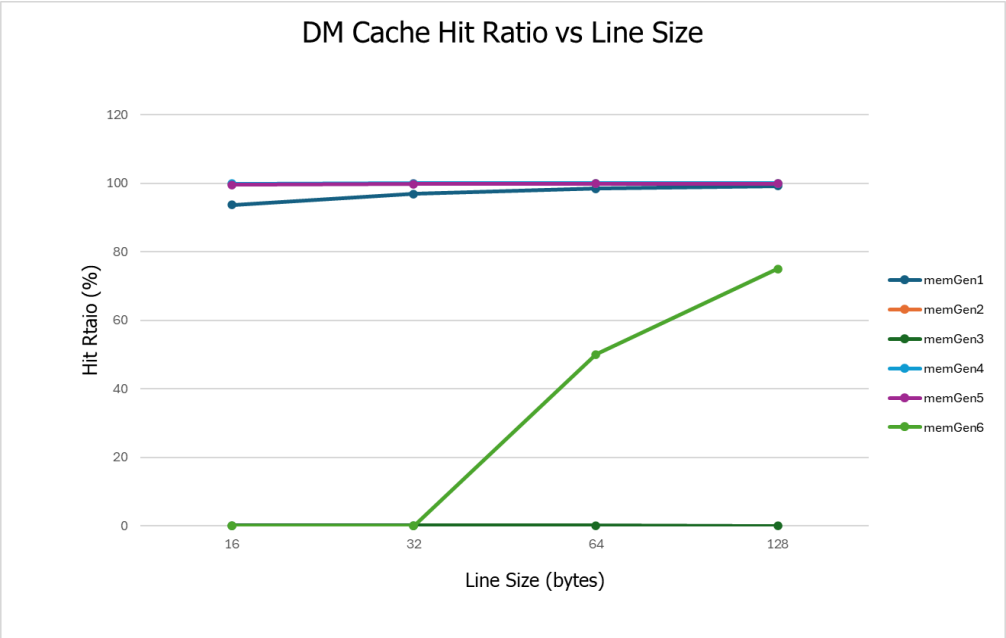

Figure 1 DM Cache Hit Ratio vs Line Size

The graph shows the direct map hit ratio as a function of the cache line size for 6 different memory random generators. For **memGen1**, presented by the blue line, the hit ratio remains close to 100% for all 4 line sizes. Its graph started from 93.75 and increased until it overlapped with the graphs of **memGen2**, **memGen4**, and **memGen5**. Likewise **memGen2**, presented by the orange line, maintains a high hit ratio close to 100% for all 4 line sizes, and it ranks the second closest graph to 100%. Also, **memGen4**, represented by the light blue line, represents the highest values in all memory generators making it the one with the closest hit ratios to 100%. The graph of **memGen5**, represented by purple line, also remains close to 100% in all 4 line sizes ranking the 3rd closest to 100% hit ratio. On the other hand, **memGen3**, represented by the dark green line, remains close to 0% hit ratio in all line sizes which makes it close to the x-axis all the time. For **memGen6**, represented by the light green line, starts by giving 0% ratio in 16-bytes and 32-bytes line size then increases to almost 50% in 64-byte line size and to 75% ratio in the 128-byte line size.

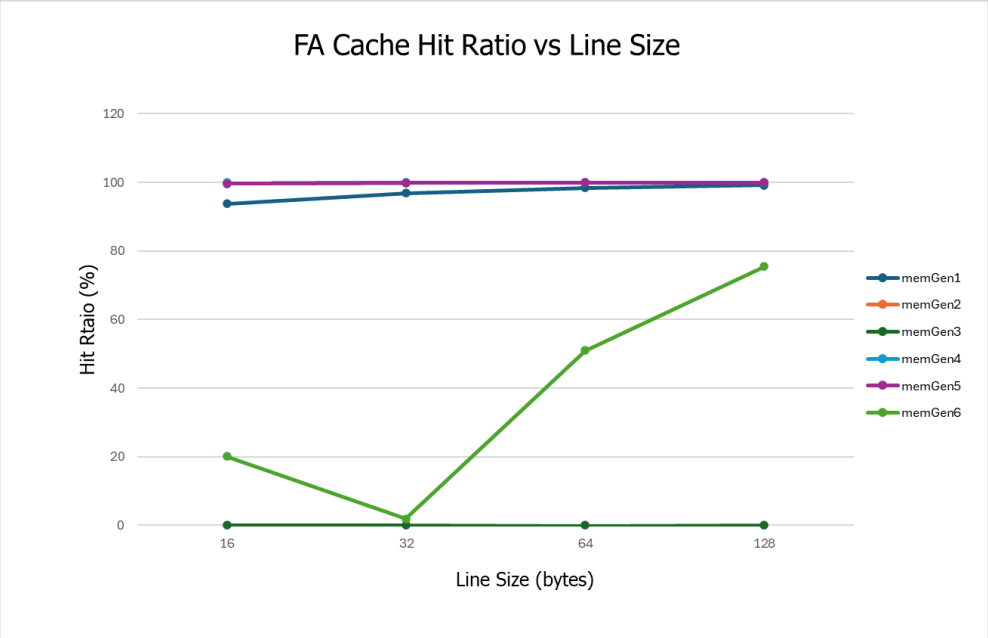

Figure 2 FA Cache Hit Ratio vs Line Size

The graph shows the fully associative hit rate as a function of the cache line size for 6 different memory random generators. For **memGen1**, presented by the blue line, the hit ratio remains close to 100% for all 4 line sizes. Its graph started from 93.75 and increased until it overlapped with the graphs of **memGen2**, **memGen4**, and **memGen5**. Likewise **memGen2**, presented by the orange line, maintains a high hit ratio close to 100% for all 4 line sizes, and it ranks the second closest graph to 100%. Also, **memGen4**, represented by the light blue line, represents the highest values in all memory generators making it the one with the closest hit ratios to 100%. The graph of **memGen5**, represented by purple line, also remains close to 100% in all 4 line sizes ranking the 3rd closest to 100% hit ratio. Similarly, **memGen1**, **memGen2**, **memGen4**, and **memGen5** have the same values as the DM cache graph explained above. Thus, these 4 lines overlap each other and are all close to 100%. On the other hand, similar to the DM cache graph, **memGen3**, represented by the dark green line, remains close to 0% hit ratio in all line sizes which makes it close to the x-axis all the time. For **memGen6**, represented by the light green line, it starts with around 20% for the 16-byte line and then drops to around 2% in the 32-byte line, the the hit ratio goes up again and reaches 50% in the 64-byte line and 75% in the 128-byte line, note that the values for the 64-byte and 128-byte lines are the same as the DM graph.

Analysis

Analyzing **memGen1**, with its range of 1 million accesses (almost 15 times the cache size), we found that the calculated high hit rate is primarily due to the sequential access pattern which effectively utilizes spatial locality. It ensures that there is only 1 miss for every (block_size - 1) hit. However, since the number of addresses generated exceeds the number of cache lines, conflict misses occur, leading to a certain miss rate. The trend shows a noticeable increase in the hit ratio as the line size increases, indicating more hits per miss as the block size grows. This is because the number of hits for every miss approximately doubles with each increase in line size, although not exactly doubling. This results in a slightly significant difference between line sizes. The hit rate remains high because, as the line size

increases, the number of cold start misses decreases due to fewer lines in the cache. There is no difference between the hit rates in Direct-Mapped (DM) and Fully-Associative (FA) caches for **memGen1** because the access pattern does not revisit any addresses, making the replacement strategy irrelevant. Consequently, both DM and FA caches exhibit the same miss-to-hit ratio. The miss rate is approximately equal to 1 divided by the block size. The reason why **memGen1** changes noticeably at the beginning compared to minor change in **memGen4**, is because the contribution of cold start misses, which is affected as the line size increases, is way more significant in **memGen1** than in **memGen4** due to the difference in the range of addresses.

As for **memGen2**, with a memory size of 24 Kbytes (smaller than the cache size), the access pattern is non-sequential and random. This configuration results in a very high hit rate because the only misses encountered are the initial cold start misses, which are negligible compared to the 1 million total accesses. The trend shows a very small increase in the hit ratio as the line size increases. This is due to the reduction in the number of cold start misses, although the difference remains minimal because these misses constitute a very small fraction of the total accesses. The worst-case miss rate is given by 24K divided by the line size. Since all the data fits into the cache, there is no need to replace any lines; thus, the replacement strategy is irrelevant. Therefore, the hit rates for both Direct-Mapped (DM) and Fully-Associative (FA) caches are identical. **MemGen2** shows a higher hit rate compared to **memGen1** because it benefits from temporal locality as well as spatial locality. However, due to the randomness of the access pattern, the impact of temporal locality diminishes with larger line sizes, explaining the significant difference in hit ratios for the first two line sizes but not for the last two. Briefly, the high hit rate in **memGen2** is due to the effective use of both spatial and temporal locality, with cold start misses being the only source of misses. Since all the data fits into the cache, there is no need for overwriting, ensuring full utilization of the cache's capacity and resulting in identical performance for both DM and FA caches.

Next, **MemGen3** function randomly generates addresses that are within the DRAM (64*64*1024) to be stored on the cache and since the cache is 64 times smaller than the DRAM that causes an extremely low hit ratio in both the DM and FA. The hit ratio is almost zero for all the cache block sizes for the following reason. First, since the function generates a very large range of addresses randomly then the probability of being able to make use of the spatial or temporal locality is almost zero. This is due to the fact that the probability of returning to an address and not finding it overwritten is extremely small. Therefore, the addresses generated by this function cause mostly misses and barely any hits. On the other hand, **MemGen2** as mentioned above generates random addresses and yet in both caches and every cache block size it produces an extremely high hit rate.This is because the range for **MemGen2** is relatively small compared to that of **MemGen3**. Therefore, we don't need to overwrite in **MemGen2** as its range is smaller than the cache size , making use of spatial and temp locality. Moreover, the hit rates produced for different cache block sizes are all almost the same with minor differences which we assume is due to the random generation happening in the **MemGen3**. The hit rates generated for both the DM and FA caches are almost the same as we do not take into the account the effects of locality due to its insignificance in this function. Both caches have almost the same number of compulsory misses and the conflict misses in the DM are almost equal to the capacity misses in the FA.

Then **MemGen4** function produces addresses sequentially in the range from 0 to 4KB - 1, covering only 1/16 of the cache and the first 4KB addresses that are generated from the 1 million addresses will be the only iteration over the cache that will have misses and they are only cold start misses. This is because of the spatial locality where for every compulsory miss generated , (block size - 1) hits will be generated

with the help of the spatial locality. After that the same range of addresses will be generated for about 243 times and that will make use of the temporal locality as the exact same addresses that are generated are already stored inside the cache. Moreover, There is a slight difference between the hit rates for each cache block size as mentioned above with every miss in the first iteration , (block size - 1) hits will be generated and as the block size increases the number of hits increases. For both the FA and DM caches the hit rate is exactly the same as not only there are no conflict misses generated by DM or capacity misses generated for FA as only 1/16 of the cache had data stored in them but also since the compulsory misses are only caused when are first storing the data and that is the same for both caches. Furthermore, when comparing the hit rates of **MemGen1** with **MemGen4** it could be noticed that the hit rate of **MemGen4** is greater than that of **MemGen1** even though both make use of the sequential method of address generation ,yet for **MemGen1** it only has spatial locality unlike **MemGen4** which makes use of both spatial and temporal locality.

Evidently, **MemGen5** shows almost the exact same behavior as **MemGen4**. We believe this to be due to the fact that the range of addresses produced by each memGen completely fits within the cache size, meaning that there will be no conflict or capacity misses for either of them. As a result, the only difference between the two arises due to the different number of compulsory misses, which is due to the fact that **MemGen5** covers a wider range of addresses (precisely 16x) than **MemGen4**, and thus occupies more lines. Those, however, are very few, regardless of the line size, compared to the 1 million overall calls, which is why the difference between both hit rates is very small (<1%) and the two lines appear to almost overlap.

MemGen6, which implements stride access, shows the most noticeable difference between the DM and FA cache types. This occurs precisely in the first two sizes (16 & 32), which appear to have a zero hit rate for the DM, and an initially confusing decline in the FA, both of which followed by a comparable rise for the remaining two sizes (64 & 128). This divergence necessitates that each type be evaluated on its own, unlike in the previous memGens where both types were evaluated together and any minor differences subsequently explained. Beginning with the DM type, it is easy to notice that the line sizes with the zero hit rates are those not big enough to fit more than one stride within their lines, and as a result, fail to make use of any spatial locality. Furthermore, since the number of calls (1 million) far exceeds the number of lines available for both sizes (4096 and 2048), by the time the addresses wrap around and start being repeated, the indices would have long been overwritten, resulting in no temporal locality and an overall zero miss rate. As we move to the next two sizes (64 & 128), both are large enough to accommodate more than one stride within their lines, allowing the cache to make use of spatial locality. As a result, each miss is followed by a hit in the 64-line-size cache, and three hits in the 128 cache, perfectly explaining the respective 50% and 75% hit rates. No temporal locality is observed for the 64 and 128 sizes similar to the 16 and 32. As for the FA, the reason why size 16 has a high rate (almost 20%) compared to a zero in DM is due to the fact that not only does the FA utilize all of the cache lines compared to only half being used in the DM on account of mapping to the correct indices, but more importantly due to the random replacement policy of FA, which makes it so that even though we still cannot make use of spatial locality, there is still a chance to achieve temporal locality if an address is not replaced before the next time it is requested. The reason why this number falls as we move up to 32-byte lines is because the number of lines decreases without any added benefit from spatial locality, which increases the number of replacements needed before the memGen cycles back to each address, greatly reducing the probability that a number would not be replaced by the time it is re-requested. As we move on to bigger sizes, just like in the DM case we are able to use the spatial locality to greatly increase the hit

rate. A very small increase is noted compared to the DM, which is due to the probability of achieving temporal locality that was observed in the 16-byte and 32-byte cases, however the smaller line sizes have made it so that the contribution of the latter is very small. Thus, the initially surprising drop at the 32-byte becomes easily explicable by noting that increasing the line size increases the probability of achieving spatial locality but decreases that of temporal locality, and vice versa, and that the 32-byte lines, which are exactly the stride size, is at the critical point that has the lowest of both localities, giving it the overall minimum hit rate.

Conclusions

After collecting the results and analyzing them, we were able to come to many conclusions regarding cache behavior, the most important of which are as follows:

- By comparing the hit rates of MemGen1, MemGen4 and MemGen5, we concluded that, when it comes to sequential access, larger line sizes are preferred to make more use of spatial locality. Moreover, the larger the range of requested values is, the greater the contribution of cold starts to the hit rate and, consequently, the greater the difference between the line sizes.
- Comparing the same 3 MemGens we were also able to conclude that in the case of sequential access, not repeating addresses (i.e. eliminating the need to retrieve a previously fetched address) makes it so that DM and FA caches always have the same hit rate, as the primary difference between their hit rates arises due to their different ways of replacing elements, which becomes irrelevant in this scenario. Nevertheless, DM and FA may still perform identically in cases of repeated access if the full range of addresses fits within the cache, as no overwrites will take place. In other words, DM and FA caches will perform differently only if replacements occur and there is a need to retrieve a previously fetched address.
- By observing random access MemGens (2 & 3), we were able to note that, so long as the range of requested values fits entirely within the cache, the cache may still achieve very high hit rates due to spatial/temporal locality, but as the range grows bigger and starts to surpass the size of the cache, the probability to make use of either locality decreases.
- In the case of stride access (MemGen6), we observed that caches whose line size is the same as the stride value serves as a critical point, after which a consistent increase in hit rates due to spatial locality can be seen but before which no spatial locality may be achieved. In the case of DM, if the range of addresses surpasses the cache size by one or more times, this will additionally result in no temporal locality, achieving an overall zero hit rate. In FA, however, even if the range exceeds the cache size, a random replacement policy could make it so that there is still probability to achieve temporal locality, however this probability decreases the larger the line size (i.e. the fewer the lines) and the range get. This trend of the decreasing temporal locality as the cache size increases, coupled with a concurrent increasing trend in spatial locality past the stride size is exactly what causes the critical point to have the least hit rate compared to any other possible line sizes.