University of South Florida

CDA 4205L Computer Architecture Lab

Lab Report

Semester: Fall 2023		
Experiment	*Number:*	Lab 12
	Date:	11/15/2023
Lab	*Section:*	003
	Lab TA:	Gil Olenscki Neto
Report	*Due Date:*	11/27/2023
Group (Lab3 onwards)	*Member #1 name:*	Aidan Khalil
	Member #2 name:	Sean Finch

RUBRIC SUMMARY
[10%] For attendance
[90%] Report

ABSTRACT:

[10%] State the purpose and objective of the lab. Give a brief introduction and summary of what you did, what you learned and your conclusion (1 paragraph).

The purpose of this lab was to investigate the functionality and impact of different cache configurations on system performance. By implementing a matrix multiplication program in RISC-V assembly and utilizing the Data Cache Simulator tool provided in RARs, we explored direct mapped, n-way set associative, and fully associative caches. Through experimentation with various cache sizes and block sizes, we observed how these configurations influenced hit rates and miss rates. The lab aimed to deepen our understanding of cache design principles and their trade-offs, providing insights into the optimal configurations for mitigating cache misses and enhancing overall system efficiency. In conclusion, the exercise provided valuable experience in cache simulation and analysis, contributing to a comprehensive understanding of memory hierarchy in computer systems.

INTRODUCTION:

[5%] Write an introduction making sure to explain the lab/experiment (1-2 paragraphs).

In this lab, we learn more about cache configurations and go into the intricacies of data caches and their role in computer system performance. The experiment focuses on understanding the nuances of different cache configurations - direct mapped, n-way set associative, and fully associative. By implementing a matrix multiplication program in RISC-V assembly and utilizing the Data Cache Simulator, we explore how varying cache sizes and block sizes influence hit rates and miss rates. The lab aims to show the complexities of cache design and highlight the trade-offs involved in achieving optimal system efficiency. Through practical simulations and analysis, we aim to gain insights into the impact of cache configurations on memory access and overall computational speed.

METHOD:

Software Tools Used:

[5%] List all the software tools you need to implement the lab.

- RARS – we used the data cache simulator on RARS for this lab alongside the matmul.asm file.

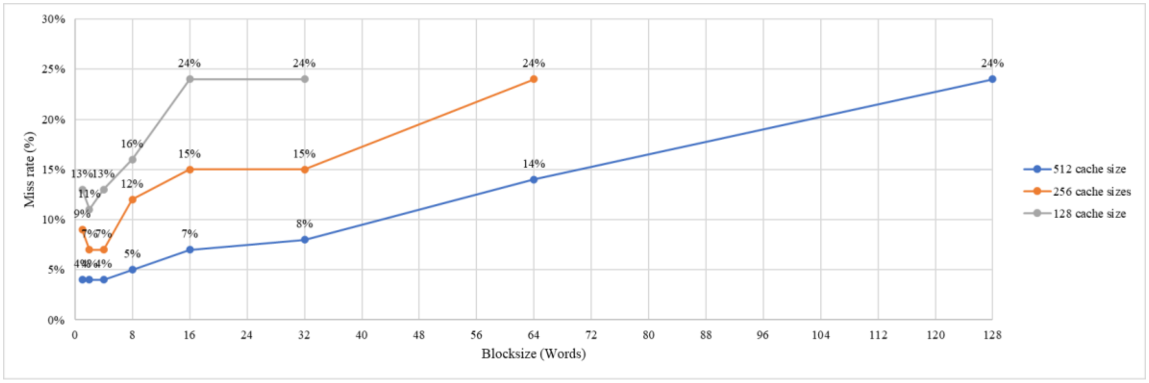
Files:

[5%] List all files included in your submission. Include a brief description (1 sentence) of each script. Only include files you/your group wrote or modified.

- CDA4205L Lab12.pdf – pdf report with screenshots and tasks (code ran w/ different memory configurations)

RESULTS:
[40%] Describe your observations of your completed experiment. Include screenshots, plots, tables, or answers to questions, as required, for each lab task (T). Describe any issues experienced in the implementation. If your measured results differ from the expected results, mention how far off you were and provide an explanation. This should be 1 paragraph per task (T). Your answers must be concise, coherent, complete, and correct.

T1: Generate a plot showing the miss rate for each configuration, with one line representing a cache size (bytes). The y-axis should be the miss rate, the x-axis should be the block size (words). All axes should be labeled, include miss rates values on each datapoint, and there should be a legend showing which line corresponds to which cache size.

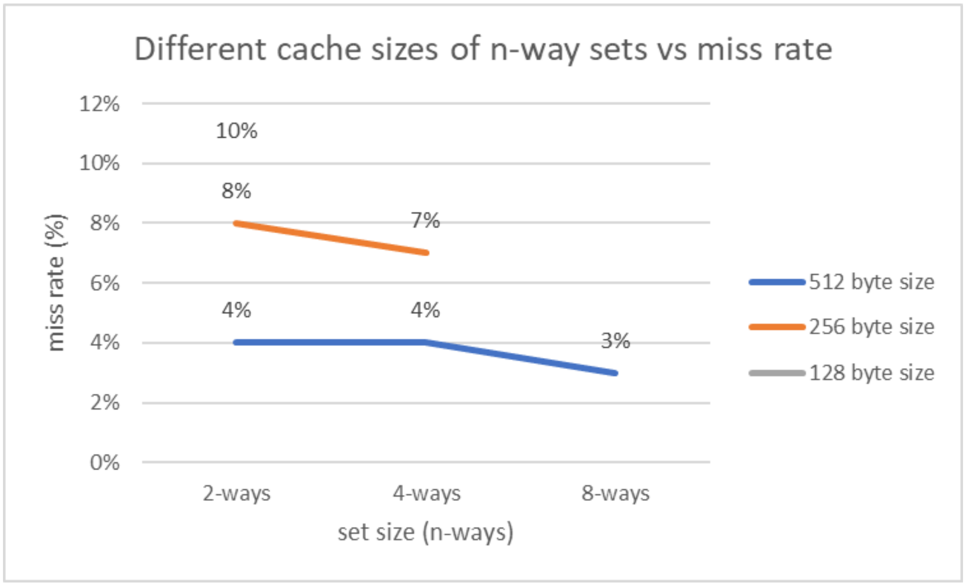


T2: Explain your observations. How does the block size impact the miss rate for different sizes of cache? In your answer, consider the total size of data that must be stored and the type of locality that is exploited.

The observed trend indicates that, as the block size increases, there is a consistent rise in miss rates across different cache sizes. This phenomenon can be attributed to the intricacies of memory access patterns and the interplay between block size and cache efficiency. Larger block sizes, while potentially advantageous in capturing more contiguous data during cache misses, seem to lead to an increase in overall miss rates. This result suggests that the benefits of larger blocks in exploiting spatial locality might be counteracted by a higher probability of conflict misses. As the block size increases, there is a greater chance that multiple memory locations map to the same cache block, resulting in more frequent evictions of relevant data. The impact of block size on miss rates underscores the importance of carefully considering the trade-offs between spatial locality and efficient cache utilization. The optimal block size is program-specific and hinges on striking a balance between leveraging spatial locality and minimizing contention for cache space.

T3: Generate a plot showing the miss rate for each configuration, with one line representing a cache size (bytes). The y-axis should be the miss rate, the x-axis should be the set size n (n-ways). All axes should be

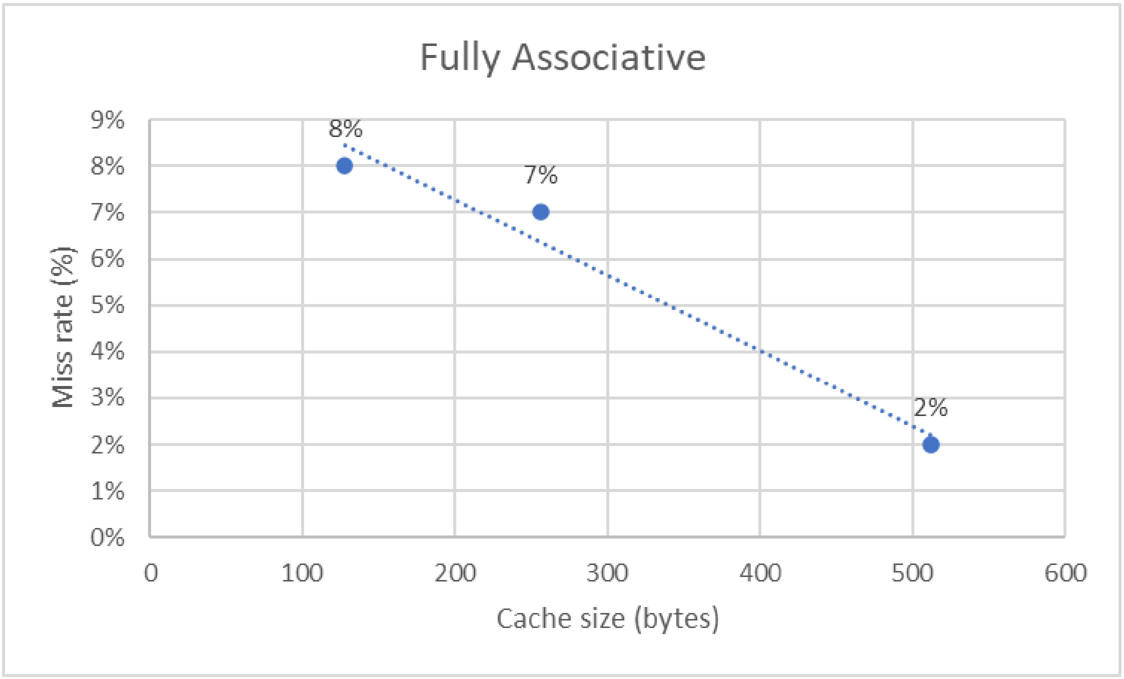
labeled, include miss rates values on each datapoint, and there should be a legend showing which line corresponds to which cache size.



T4: Consider the hardware required to support the different n-way set associative caches. Based on your results, is there an optimal configuration that balances area/power with the hit rate? Explain your reasoning.

In our n-way set associative cache experiments, increasing the set size consistently reduces the miss rate. While this improves performance, it comes at the cost of increased hardware complexity, larger chip area, and potentially higher power consumption. The optimal configuration strikes a balance between achieving a lower miss rate and managing hardware-related costs. The diminishing returns in hit rate improvement with higher associativity need to be weighed against the increased complexity and resource requirements, and we determined an 8-block 2-way set associative cache would be the optimal because it is in the middle (in terms of cache blocks) and 2-ways is only 1% worse than 4-ways for this size cache.

T5: Generate a plot showing the miss rate for each configuration. Since there is only 1 line, the y-axis should be the miss rate, the x-axis should be the cache size (bytes). All axes should be labeled and include miss rates values on each datapoint.



T6: In this configuration (fully associative), why does the set size equal the number of blocks?

In a fully associative cache, the set size equals the number of blocks because each set consists of only one way. This means that any block can be placed in any cache location without restrictions, offering maximum flexibility. As a result, the set size is essentially the total number of blocks in the cache, simplifying the block placement process.

T7: What is the goal of using a memory hierarchy in a computer system? Why is a memory hierarchy necessary?

A memory hierarchy in a computer system is used to maximize memory performance while decreasing and balancing the cost and area of CPU and system memory. A memory hierarchy is necessary because fast memory (cache) on the CPU is expensive, while slower system memory (DRAM) and storage (HDD/SSD) is cheaper and offers a larger memory size for less cost.

T8: For the optimal set-associative cache configuration identified in T4, compute the storage percent overhead. Assume a 32-bit address space for the main memory and byte-level addressing (as in the RISC-V ISA). In your answer, note the byte offset, block (word) offset, index bits, and size of the tag. Include valid bit.

Considering our optimal cache configuration was an 8-block cache with 2-ways, the storage percent overhead (assuming a 32-bit address space for main memory and byte level addressing) would be:

Block Size: Since we're dealing with bytes, the block size is 32 bits (4 bytes) * 8 blocks = 32 bytes.

Byte Offset: log2(Block Size) = log2(32) = 5 bits

Block (Word) Offset: log2(Number of Words per Block) = log2(4) = 2 bits (assuming 4 words / block)

Index Bits: log2(Number of Sets) = log2(8) = 3 bits

Tag: 32 - (Byte Offset + Block Offset + Index Bits) = 32 - (5 + 2 + 3) = 22 bits

Valid Bit: 1 bit (since it's a set-associative cache)

Percent overhead:

(10) / 32-bit address space * 100 = 31.25% overhead

Therefore, for the optimal 8-block 2-way set associative cache configuration, the storage percent overhead is approximately 31%. This indicates that, compared to the original 32-bit address space, additional bits are required for addressing and control in the cache. This means that roughly 3.125% more bits are required, which is not a bad overhead.

DISCUSSION:

[15%] Discuss the implications of your work and results. Propose potential future work and any shortcomings of your design (2 paragraphs).

Our findings show that different ways of setting up computer memory caches have a big impact on how fast the computer can retrieve information. We tried out three configurations – direct mapped, n-way set associative, and fully associative caches – and noticed that each has its pros and cons. Direct mapped caches are simpler but might run into problems, while n-way set associative ones strike a balance, and fully associative ones are flexible but need more resources.

Looking ahead, we could explore more complex ways of managing memory in the n-way set associative setup or see how these configurations perform with more complex tasks, and not just the provided multiplication matrix file. However, it's important to note that our lab experience had some limitations, like using a basic program and not considering power usage. In the future, we can dig deeper to better understand how to set up memory caches for different kinds of computer tasks.

CONCLUSION:

[10%] Conclude the lab report, stating what the lab was about. Mention any major implementation issues and summarize your observations and discussion (1 paragraph).

In conclusion, this lab explored the realm of computer memory caches, exploring how different configurations impact system performance. We implemented a matrix multiplication program and simulated three cache setups – direct mapped, n-way set associative, and fully associative. The observations highlighted trade-offs between simplicity and efficiency. While direct mapped caches are straightforward but prone to issues, n-way set associative caches strike a balance as stated previously, and fully associative caches offer flexibility at the cost of more resources. Despite some complexities, this exploration deepened our understanding of cache design, paving the way for future investigations into optimizing memory systems for other computing tasks.