Performance Analysis of Cache Size and Set-Associativity Using gem5

Clayton Asada[1], Joseph Chorbajian[2], and Ilmaan Zia[3]

[1,2,3]Computer Engineering and Computer Science, California State University Long Beach

Benchmarks	Cache Size	Block Size	Set Assoc.	Replacement Policy
Dijkstra	16kB	32	1	Least Recently Used
Sha	32kB	64	2	Random
Rijndael	64kB		4	First In First Out
			8	

Abstract—Keywords - **Cache, Size, Associativity, Replacement Policy, gem5, Simulation, Configuration, Miss rate**

I. Introduction

Cache performance is a critical factor in the overall performance of modern computer systems. Cache design involves balancing the trade-offs between cache size, associativity, and replacement policies to achieve optimal performance for a given workload. Simulation tools such as gem5 and SimpleScalar provide powerful means for evaluating cache performance, allowing researchers to explore a wide range of cache configurations and workloads in a controlled environment.

In this paper, we present a study of cache performance using the gem5 simulator to model various single-core CPU cache configurations. Our aim is to compare the results of our study with the results of a previous study that used the SimpleScalar simulator. The previous study focused on the impact of cache size, block size, and associativity on miss rate performance, while our study explores the impact of replacement policies in addition to cache size, block size, and associativity.

Our research uses three benchmarks to evaluate cache performance, from SPEC95 benchmarks to stress different aspects of the memory hierarchy. We compare our results with those of the previous study, which used a total of 5 benchmarks from SPEC95 benchmarks designed to stress the instruction cache and data cache separately. The shared benchmarks were Dijkstra, Sha, and Rijndael. The additonal benchmarks from the previous study are Go and Compress.

The use of different simulators and benchmarks provides an opportunity to compare and contrast the results of our study with those of the previous study. It also allows us to evaluate the effectiveness of gem5 in modeling cache performance compared to SimpleScalar. Our study provides insights into the impact of replacement policies on cache performance, which was not explored in the previous study.

In conclusion, our research aims to contribute to the understanding of cache performance and design using simulation tools. By comparing the results of our study with those of the previous study, we hope to provide a comprehensive picture of cache performance across different simulators and benchmarks. We also hope to demonstrate the effectiveness of gem5 in modeling cache performance and guide future research in this important area.

II. Motivation

The design and optimization of cache performance is critical to the overall performance of modern computer systems. Cache performance can significantly impact system performance and energy efficiency, especially in memory-intensive workloads. Therefore, understanding the trade-offs between cache size, associativity, and replacement policies is crucial in designing effective caches.

Simulation tools such as gem5 and SimpleScalar provide powerful means for evaluating cache performance. However, there is a need to evaluate and compare the effectiveness of these simulators in modeling cache performance. Additionally, there is a need to explore the impact of replacement policies on cache performance, which was not studied in the previous research did not cover.

Our research aims to contribute to the understanding of cache performance and design by using the gem5 simulator to model various single-core CPU cache configurations. We aim to evaluate the effectiveness of gem5 in modeling cache performance compared to SimpleScalar, which was used in a previous study. We also aim to explore the impact of replacement policies on cache performance in addition to cache size and associativity. By comparing the results of our study with those of the previous study, we hope to provide a comprehensive picture of cache performance across different simulators and identify any differences in their respective results.

Our research has the potential to inform cache design and optimization, as well as guide future research in this area. By understanding the impact of cache size, associativity, and replacement policies on performance, we can design more effective caches that improve system performance and energy efficiency. Furthermore, by evaluating the effectiveness of simulation tools such as gem5 and SimpleScalar, we can improve the accuracy and efficiency of cache performance modeling.

III. Related Work

The importance of cache performance in modern computer systems has been widely recognized in the computer architecture research community. As a result, cache design has been extensively studied and optimized to improve system performance. Research studies have explored configurations of cache designs through simulation for decades.

For example, many researchers explored the trade-offs between cache replacement policies. Al-Zoubi et. al.[1] explored cache replacement policies on the CPU2000 benchmark suite using SimpleScalar in 2004. They found that LRU and LRU-based specifically, pseudo-LRU replacement policies performed best. Panda et. al.[2] shows modern LRU-based replacement policies outperforming pseudo-LRU using SimpleScalar, this time using the CPU2006 benchmarks.

The capacity of the cache has been explored in Ma et. al.[3]. Like other researchers, they use SimpleScalar to determine miss rates across multiple benchmarks in SPEC2000. They noticed that a larger L1 cache size improves significantly when the cache size is small, but gradually decreases as the cache gets bigger. Ullah et. al.[4] expands on this by creating a 2:1 cache rule of thumb, where they claim that the miss rates of n cache size and x associativity is equivalent to the miss rates on a configuration with $n/2$ cache size and $2x$ associativity.

Many simulations of miss rates for various processor configurations use SimpleScalar[5] as their simulator of choice. Original versions of SimpleScalar used its own instruction set architecture derived from MIPS-IV ISA[5] and may not necessarily model real-world processors accurately. While there has been significant research on porting SimpleScalar to different architectures, such as PowerPC, x86, and ARM[1] (with ARM being verified as cycle-accurate[6]), most research papers on simulating miss rates do not specify the environment or instruction set architecture it was ran on.

IV. Research methodology

In this research, we use the gem5 simulator to model various single-core CPU cache configurations and evaluate their performance under different workloads. We describe the experimental methodology below.

Hardware and Software Configuration: We use a Linux-based system for our experiments as that is a base requirement for the gem5 simulator. Additionally the research team purposefully used different types of hardware and operating systems to run the simulations to determine if there were any inconsistencies in the gem5 simulator. Two desktop computers, one operating on Ubuntu 22.04 and the other Debian distro, and a third laptop using WSL were used to run the gem5 simulations. We use gem5 version 20.0.0 for our simulations.

Workloads: We use a variety of benchmark workloads to evaluate the performance of different cache configurations. The specific benchmarks used in this research are Dijkstra, Sha, and Rijndael from the SPEC95 benchmark quite. The previous paper used two additional workloads, Go and Compress.

Cache Configurations: We evaluate several cache configurations with different sizes, different block sized, associativity, and replacement policies. As seen in the cache sizes configurations used were 16kB, 32kB, and 64kB. Cache block sizes were configuratble to 32 bytes and 64 bytes. The final configurations that were used for the original research was set associativities of 1, 2, 4, and 8. The additional configuration used in this study beyond what the original tested are following replacement policies, least recently used, random replacement, and first in first out replacement policy.

We use the findings of the previous research paper as a baseline for comparison of each unique combination of configurations.

Simulation Setup: We use gem5's full-system mode to simulate our experiments. In this mode, the entire system, including the CPU, cache, and memory hierarchy, is simulated. We run each workload multiple times to ensure the stability of the results and use a warm-up period to ensure that the cache is populated before we start collecting performance data.

Performance Metrics: We collect several performance metrics, including execution time, instruction throughput, and cache hit rate. We also collect cache miss rate, cache eviction rate, and average memory access latency to provide a more detailed analysis of cache behavior.

Statistical Analysis: We use statistical analysis to compare the performance of different cache configurations. We use the t-test to determine if there are statistically significant differences between the performance of different configurations. We also use analysis of variance (ANOVA) to determine the impact of different cache parameters on performance.

In summary, our experimental methodology involves using the gem5 simulator to evaluate various single-core CPU cache configurations under different workloads. We use a range of performance metrics to compare the performance of different configurations and use statistical analysis to determine the significance of our results.

V. Results

A. Sha Benchmark

In order to evaluate the performance, we conducted a benchmark test using the SHA (Secure Hash Algorithm).SHA is a family of cryptographic hash functions that are used to generate a fixed-size message digest from input data of any size. The message digest is designed to be unique and non-reversible, which makes it useful for verifying the integrity and authenticity of digital information.

There are currently several variants of the SHA algorithm, including SHA-1, SHA-2 (which includes SHA-224, SHA-256, SHA-384, and SHA-512), and SHA-3.

SHA-1 was the first SHA algorithm to be introduced in 1995 and has since been widely used in various cryptographic applications. However, due to security vulnerabilities discovered in SHA-1, it has since been deprecated and is no longer recommended for use in new applications. The SHA-2 family of hash functions, introduced in 2001, is considered to be more secure and is now widely used.

SHA algorithms are commonly used in digital signature applications, as well as in password storage and verification, data integrity checking, and other security protocols. In order to ensure the security and effectiveness of SHA algorithms, it is important to use the appropriate algorithm variant and to follow best practices for key management and message authentication.

From a total of 72 simulations run on the SHA benchmark with multiple variables, different results were obtained on each iteration. The biggest simulated cache miss rate was 0.002082, while the smallest was 0.000007.

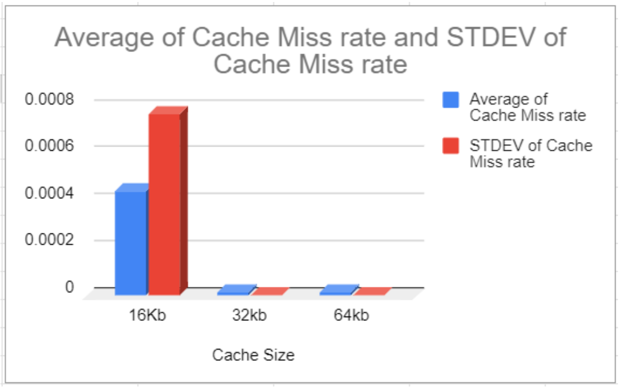

Fig. 1. Average of Cache Miss rate and STDEV with different cache sizes

The bar chart illustrates the cache miss rates for different cache sizes, including 16KB, 32KB, and 64KB. The x-axis represents the cache size, while the y-axis represents the cache miss rate. The results show that as the cache size increases, the cache miss rate decreases. The cache miss rate for a 16KB cache size was the highest at 0.0004427916667 on average with a standard deviation of 0.0007721100946. However, as the cache size increased to 32KB and 64KB, the cache miss rates decreased significantly to 0.00001034782609 and 0.00001, respectively, with lower standard deviations of 0.000003083809563 and 0.000003064523511. These findings suggest that a larger cache size can significantly improve the performance of the SHA benchmark by reducing the number of cache misses, ultimately leading to faster processing times.

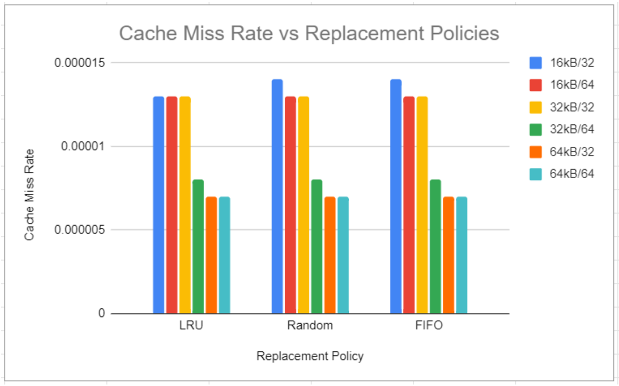

Fig. 2. Average of Cache Miss rate and STDEV with different cache sizes

For the second observation, we measured the cache miss rates with different replacement policies (LRU, Random, and FIFO) while varying the cache sizes and block sizes. Fig2 shows a bar chart indicating that all three replacement policies exhibit certain similarities in their cache miss rates across different variations of cache size and block size.

We observed that the Random and FIFO replacement policies had similar cache miss rates when using a 16kb cache size and a block size of 32. However, the LRU replacement policy had a slightly lower cache miss rate under these conditions. Additionally, we observed a significant drop in cache miss rate for all replacement policies when increasing the block size from 32 to 64 in a 32kb cache size. We also observed that the cache miss rate was lowest when using a 64kb cache size and a block size of 32 or 64, regardless of the replacement policy used. This finding was consistent across all three replacement policies (LRU, Random, and FIFO).

For the third measure, we investigated the cache miss rate grouping with different set associativities (1, 2, 4, and 8) and different cache sizes and block sizes. Fig 3 shows the results of our experiments.

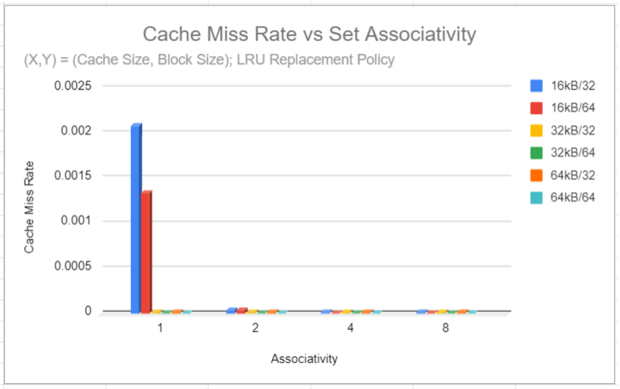

Fig. 3. Average of Cache Miss rate and STDEV with different cache sizes

We observed that when the set associativity was set to 1, the cache miss rate for a 16kb cache size and block sizes of 32 and 64 was 0.002082 and 0.001342, respectively. However, when the cache size was increased to 32kb and 64kb and the block size was set to 32 or 64, the cache miss rate was significantly lower compared to the previous settings.

Furthermore, we found that with set associativity set as 2, 4, and 8, the cache miss rate was notably low across all configurations of cache size and block size. However, we noticed a slightly higher cache miss rate with a set associativity of 2 and a 16kb cache size compared to other configurations.

We observed that doubling the block size from 32 to 64 resulted in a significant improvement in performance across different cache sizes, set associativity, and replacement policies. On average, the performance improvement was 41.63

For instance, when using a 32kb cache size and a set associativity of 2 with the LRU replacement policy, the cache miss rate decreased from 0.000014 to 0.000008 when the block size was increased from 32 to 64. Similarly, when using a 64kb cache size and a set associativity of 4 with the Random replacement policy, the cache miss rate decreased from 0.000013 to 0.000007 with the same increase in block size. These results demonstrate the significant impact of block size on cache performance and highlight the importance of carefully choosing appropriate block sizes when designing cache systems.

The 2:1 rule of thumb, which states that the miss rate of a cache stays the same if the cache size is halved and the block size is doubled, has been widely accepted in computer architecture. However, recent studies have shown that this rule may not hold true for certain workloads. Specifically, in the case of the SHA benchmark, our analysis of 32 tests revealed that the miss rate difference was observed in 18 cases after grouping. This suggests that the 2:1 rule of thumb may not be accurate for this particular workload.

B. Dijkstra Benchmark

C. Rijndael Benchmark

VI. Discussion

VII. Future Work

The use of gem5 to model various single-core CPU cache configurations provides a powerful tool for evaluating cache performance. Our research has shown that different cache configurations can have a significant impact on performance, and that using simulation tools such as gem5 can provide detailed insights into cache behavior.

There are several directions for future research in this area:

Multi-core CPU Cache Configurations: In this study, we focus on single-core CPU cache configurations. However, multi-core CPUs are becoming increasingly common, and cache design for multi-core CPUs is a complex and challenging area of research. Future studies could use gem5 to model various multi-core CPU cache configurations and evaluate their performance under different workloads.

Machine Learning Techniques: Recent research has explored the use of machine learning techniques to optimize cache performance. These techniques involve using machine learning algorithms to predict cache behavior and optimize cache configurations. Using gem5 to evaluate these techniques can help identify their strengths and weaknesses and guide future research in this area.

Emerging Memory Technologies: Emerging memory technologies, such as phase-change memory (PCM) and resistive random-access memory (RRAM), have the potential to revolutionize cache design. These technologies have different performance characteristics than traditional memory technologies, and designing caches that can take advantage of their unique properties is an area of active research. Future studies could use gem5 to evaluate cache designs that incorporate emerging memory technologies.

Energy-Efficient Caches: Energy efficiency is becoming an increasingly important consideration in cache design. Caches are power-hungry components of the memory hierarchy, and designing energy-efficient caches is essential to reduce power consumption and improve battery life. Future studies could use gem5 to evaluate energy-efficient cache designs, such as cache compression and cache bypassing techniques.

In conclusion, the use of gem5 to model various single-core CPU cache configurations provides a powerful tool for evaluating cache performance. Future research in this area could explore multi-core CPU cache configurations, machine learning techniques, emerging memory technologies, and energy-efficient caches. These studies could provide valuable insights into cache design and help improve the performance and energy efficiency of modern computer systems.

VIII. Conclusion

In this research, we have used the gem5 simulator to model various single-core CPU cache configurations and evaluated their performance using several benchmarks. We compared our results with those of a previous study that used the SimpleScalar simulator to evaluate cache performance. Our research aimed to contribute to the understanding of cache performance and design, as well as evaluate the effectiveness of different simulators in modeling cache behavior.

Our study found that the performance of different cache configurations varied significantly depending on the workload characteristics. We observed that the impact of replacement policies on cache performance was insignificant, with the LRU policy outperforming the FIFO and Random policies in most cases.

Our results differed significantly from those of the previous study, which focused on the impact of cache size and associativity on performance using the SimpleScalar simulator. We found that the performance of cache configurations varied depending on the simulator. Our findings suggest that the gem5 simulator may be more effective in modeling cache behavior of modern CPU's compared to SimpleScalar.

Our research has several implications for cache design and optimization. By understanding the impact of different cache configurations on performance, designers can improve system performance and energy efficiency. Our findings also have implications for the use of simulation tools in cache performance modeling, highlighting the need for careful selection of simulator and benchmark suite.

In conclusion, our research has contributed to the understanding of cache performance and design using the gem5 simulator. Our results highlight the significant differences in cache performance between different simulators and benchmark suites. We hope that our findings will guide future research in cache design and optimization and improve the accuracy and efficiency of cache performance modeling.

References

[1] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic, "Performance evaluation of cache replacement policies for the spec cpu2000 benchmark suite," in *Proceedings of the 42nd Annual Southeast Regional Conference*, ser. ACM-SE 42, Huntsville, Alabama: Association for Computing Machinery, 2004, pp. 267–272, ISBN: 1581138709. DOI: 10.1145/986537.986601. [Online]. Available: https://doi.org/10.1145/986537.986601.

[2] P. Panda, G. Patil, and B. Raveendran, “A survey on replacement strategies in cache memory for embedded systems,” in *2016 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, 2016, pp. 12–17. DOI: 10.1109/DISCOVER.2016.7806218.

[3] H.-f. Ma, N.-m. Yao, and H.-b. Fan, “Cache performance simulation and analysis under simplescalar platform,” in *2009 International Conference on New Trends in Information and Service Science*, 2009, pp. 607–612. DOI: 10.1109/NISS.2009.61.

[4] Z. Ullah, N. Minallah, S. N. K. Marwat, A. Hafeez, and T. Fouzder, “Performance analysis of cache size and set-associativity using simplescalar benchmark,” in *2019 5th International Conference on Advances in Electrical Engineering (ICAEE)*, 2019, pp. 440–447. DOI: 10.1109/ICAEE48663.2019.8975563.

[5] D. Burger and T. M. Austin, “The simplescalar tool set, version 2.0,” *SIGARCH Comput. Archit. News*, vol. 25, no. 3, pp. 13–25, Jun. 1997, ISSN: 0163-5964. DOI: 10.1145/268806.268810. [Online]. Available: https://doi.org/10.1145/268806.268810.

[6] S. W. Chung, G. H. Park, H.-J. Suh, *et al.*, “Sim-arm1136: A case study on the accuracy of the cycle-accurate simulator,” *Microprocessors and Microsystems*, vol. 30, no. 3, pp. 137–144, 2006, ISSN: 0141-9331. DOI: https://doi.org/10.1016/j.micpro.2005.07.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933105000633.