Enhancing CPU Performance Through Advanced Cache Design and Optimization Techniques

Sai Kumar Marri[1], E. Sikender[2]

[1]Student Researcher, Department of Electrical Engineering, UTD
[2]Student Researcher, Department of Electrical and Communications Engineering, VCE

Abstract
Optimizing CPU performance has become a critical focus in computing systems due to the increasing demand for speed, efficiency, and energy conservation. Among various performance-enhancing strategies, cache design plays a pivotal role in bridging the gap between high-speed processors and slower memory systems. This paper explores advanced cache design and optimization techniques to enhance CPU performance, emphasizing their applicability in modern embedded systems, mobile devices, and high-performance computing platforms [3]. The study begins with an overview of conventional cache architectures, such as direct-mapped, set-associative, and fully associative caches, highlighting their respective benefits and limitations. Advanced techniques, including variable-way set-associative caches, adaptive cache resizing, and analytical design space exploration, are then discussed. These approaches aim to minimize cache misses, reduce power consumption, and optimize resource utilization by dynamically tailoring cache parameters to specific workloads.
Furthermore, the paper investigates innovative methodologies, such as machine learning-assisted cache tuning, hybrid cache designs combining SRAM and non-volatile memory, and energy-efficient replacement policies [2]. The integration of predictive modeling and simulation tools for rapid design exploration is also emphasized, enabling designers to evaluate cache configurations against a vast design space efficiently. Experimental results from benchmarks reveal that employing these advanced techniques can significantly reduce cache miss rates, improve data access latency, and lower energy consumption without compromising system performance. This research underscores the importance of cache design as a critical enabler for next-generation CPU performance enhancements, particularly in domains constrained by power and thermal budgets. The findings provide valuable insights for researchers and engineers seeking to develop more efficient and adaptable cache systems, paving the way for continued innovation in CPU architecture and overall system design [1].

Keywords: Cache Optimization, CPU Performance Tuning, Memory Hierarchy Design, Adaptive Cache Architectures, Energy-Efficient Computing

1. Introduction

As computing demands grow across diverse applications, from embedded systems and mobile devices to high-performance computing (HPC) platforms, the performance of Central Processing Units (CPUs) is a fundamental driver of system efficiency and user experience. The ever-expanding gap between processor speed and memory latency, known as the memory wall, has underscored the critical role of cache memory in maintaining high CPU performance. Caches, acting as high-speed intermediaries between CPUs and main memory, significantly influence data access times, power consumption, and system throughput. Consequently, advancements in cache design and optimization have emerged as key enablers of CPU performance enhancements [4].

Traditional cache architectures—such as direct-mapped, set-associative, and fully associative caches—have provided effective solutions for improving data locality and reducing memory access latency. However, these conventional designs face challenges in adapting to modern workloads, which are characterized by varying access patterns, dynamic data requirements, and stringent power constraints. To address these challenges, innovative approaches in cache design have been proposed, emphasizing adaptability, efficiency, and workload-specific optimization. This study explores cutting-edge techniques in cache design and optimization, focusing on their impact on CPU performance. Key areas of discussion include variable-way set-associative caches, which adjust associativity dynamically to balance performance and energy efficiency, and adaptive resizing techniques that tailor cache size to workload demands [6]. Additionally, analytical models and machine learning-based methodologies are examined for their ability to expedite design space exploration and predict optimal configurations with high accuracy. Beyond traditional designs, hybrid caches combining static RAM (SRAM) and non-volatile memory are gaining traction for their potential to address power and latency concerns in energy-constrained environments. This research also delves into advanced replacement policies and data placement strategies that further optimize cache utilization [9].

The significance of this work lies in its holistic approach to enhancing CPU performance through cache design, offering practical insights and experimental validations to support its findings. By bridging the gap between theoretical advancements and real-world applications, this study aims to provide a comprehensive resource for researchers, system architects, and engineers striving to optimize CPU performance in an era where efficiency and adaptability are paramount. Ultimately, the insights from this research pave the way for the development of more intelligent and resource-efficient computing systems [8].

2. CPI calculation for the benchmarks

CPU benchmarks are standardized tests designed to evaluate the performance of a processor under specific workloads. They provide quantitative metrics that help compare different CPUs across diverse applications, including gaming, multimedia, scientific computing, and enterprise workloads. Benchmarks play a crucial role in guiding hardware selection, system optimization, and architecture design.

Benchmarks are generally categorized into synthetic, application-based, and microbenchmarks. Synthetic benchmarks simulate a range of tasks to assess overall performance, often yielding a composite score. Application-based benchmarks measure performance using real-world applications, offering insights into CPU's practical efficiency. Microbenchmarks focus on specific processor features, such as cache performance, memory latency, or floating-point operations, isolating aspects of CPU behavior. Key performance metrics in CPU benchmarking include clock speed, instructions per cycle (IPC), throughput, and power efficiency. Advanced benchmarks also evaluate multi-threading capabilities, thermal performance, and workload-specific optimizations, making them relevant for modern multi-core processors.

$$CPI = 1 + \frac{(IL1.miss\_num + DL1.miss\_num) \times 6 + L2.miss\_num \times 50}{Total\_Inst\_num}$$

Figure 1: CPI calculation

CPI (Cycles Per Instruction) is a key performance metric in computer architecture, indicating the average number of clocks cycles a processor takes to execute a single instruction. It reflects the efficiency of the CPU in handling workloads and depends on factors like instruction complexity, cache performance, and pipeline architecture. A lower CPI signifies higher efficiency, meaning the processor can execute more instructions per cycle. CPI is influenced by the workload and processor design, including multi-core and out-of-order execution capabilities. It is often used alongside clock speed and IPC (Instructions Per Cycle) to evaluate overall CPU performance.

Popular CPU benchmark suites like SPEC (Standard Performance Evaluation Corporation), Cinebench, and Geekbench provide standardized testing environments. These tools help identify performance bottlenecks, validate design improvements, and optimize systems for specific use cases. As computing evolves, benchmarks are adapting to include metrics for artificial intelligence, virtualization, and energy efficiency, ensuring relevance in an era of diverse and specialized workloads.

Computed the cycles per instruction (CPI) for a set of benchmarks with baseline X86 configuration as follows:

- CPU Models – timing
- Cache levels - two (L1 (instruction & data) & L2 (Unified))
- L1 instruction cache size - 128kB
- L1 data cache size - 128kB
- L2 cache size - 1MB
- L1 instruction associativity – 2
- L1 data associativity – 2
- L2 associativity – 2
- cache line size – 64

Computed CPI with a L1 miss penalty of 6 cycles, L2 miss penalty of 50 cycles, and 1 cycle cache hit. CPI is derived from the below equation.

Table 1: CPI for base configuration

Benchmark	CPI
401.bzip2	1.089093568
429.mcf	1.758002656
456.hmmer	1.002037688
458.sjeng	1.938821732
470.lbm	1.800893776

3. Optimize CPI for each benchmark

By exploring the design space using different configurations, we found an optimal configuration of cache design with lower CPI (best performance). Cache design configuration explored different factors such as associativity, block size and size allocation of L1 and L2 cache.

a) Benchmark – I (401.bzip2)

With base configuration of L1 cache & L2 cache size of 128kB and L2 cache size of 1MB, we modified the associativity and cache line size to derive the lowest CPI.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
128kB	128kB	1	8	128	1.066336

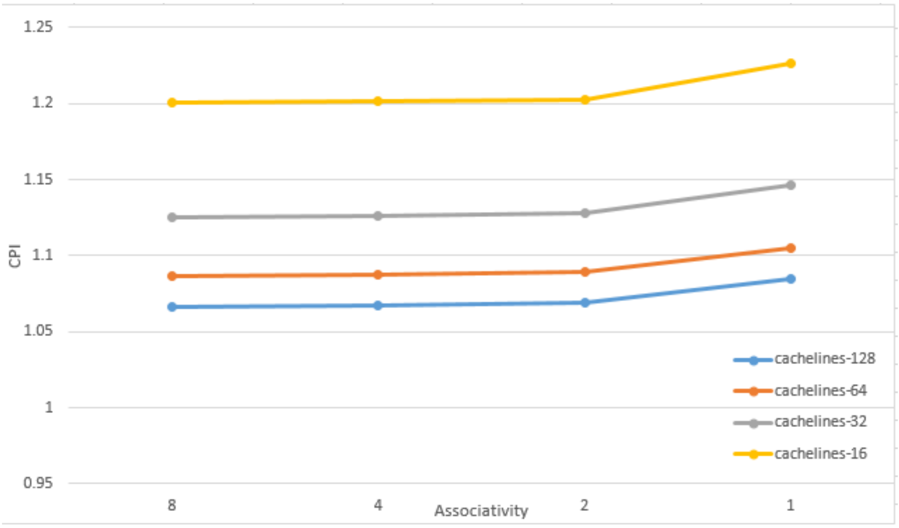

Figure 2: Benchmark – 1: Associativity vs CPI

By fixing Associativity = 8, cache line size = 128, we explored the cache design by varying the L1 and L2 cache sizes.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
256kB	256kB	2	8	128	1.03413
128kB	256kB	2	8	128	1.03413
128kB	128kB	2	4	64	1.056151
256kB	256kB	1	8	128	1.059201
128kB	256kB	1	8	128	1.059201
128kB	128kB	1	8	128	1.066336

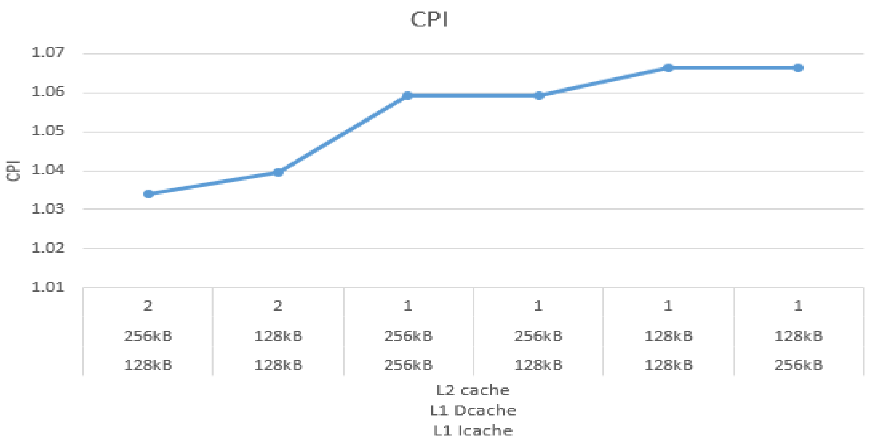

Figure 3: Benchmark – 1: Design choices vs CPI

b) Benchmark – II (429.mcf)

With base configuration of L1 cache & L2 cache size of 128kB and L2 cache size of 1MB, we modified the associativity and cache line size to derive the lowest CPI.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
128kB	128kB	1	8	128	1.495679

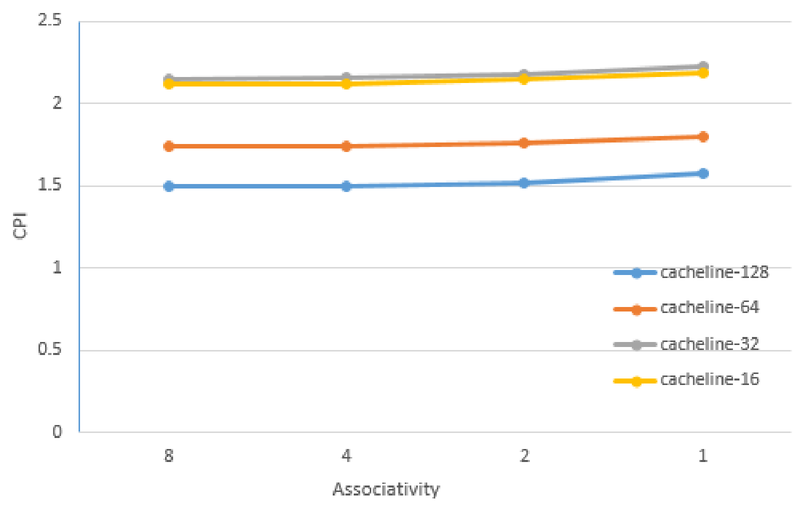

Figure 4: Benchmark – 2: Associativity vs CPI

By fixing Associativity = 8, cache line size = 128, we explored the cache design by varying the L1 and L2 cache sizes.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	2	8	128	1.368246	2622727
128kB	256kB	2	8	128	1.368246	2491671
128kB	128kB	2	4	128	1.410814	2359927
256kB	256kB	1	8	128	1.452938	1574167
128kB	256kB	1	8	128	1.452938	1443111
128kB	128kB	1	8	128	1.495679	1312055

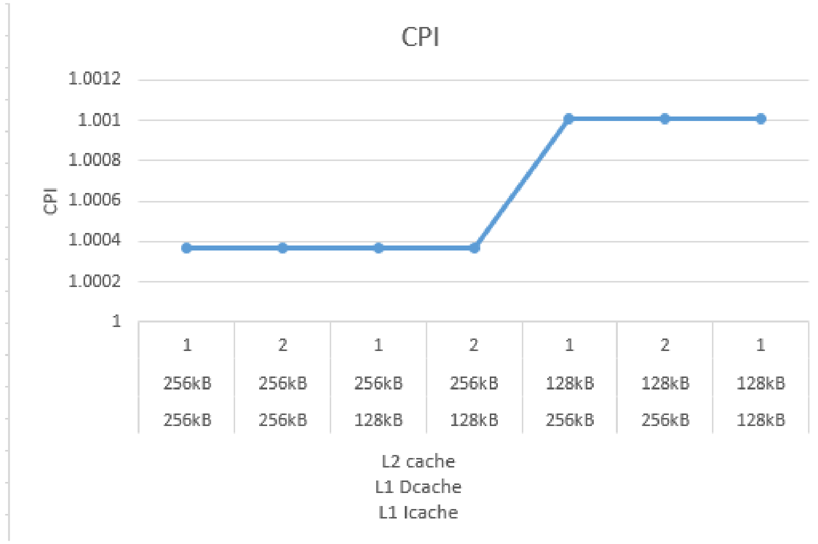

Figure 5: Benchmark – 2: Design choices vs CPI

c) Benchmark – III (456.hmmer)

With base configuration of L1 cache & L2 cache size of 128kB and L2 cache size of 1MB, we modified the associativity and cache line size to derive the lowest CPI.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
128kB	128kB	1	8	128	1.001007

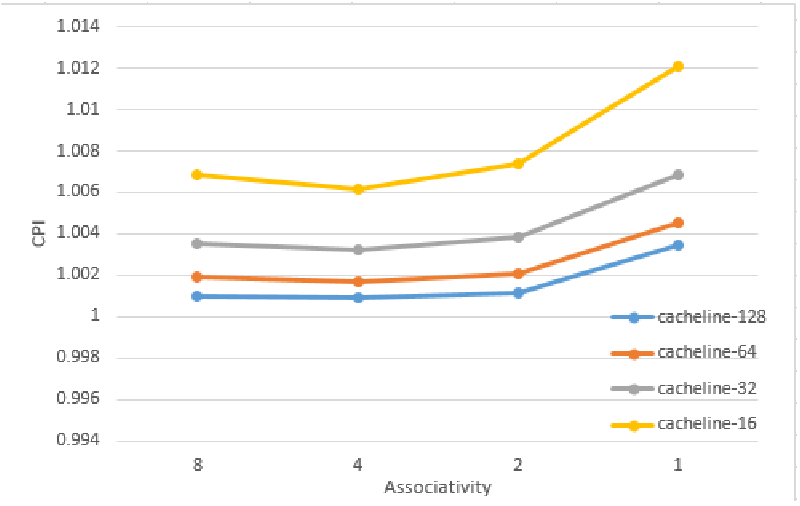

Figure 6: Benchmark – 3: Associativity vs CPI

By fixing Associativity = 8, cache line size = 128, we explored the cache design by varying the L1 and L2 cache sizes.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
256kB	256kB	1	8	128	1.000364
256kB	256kB	2	8	128	1.000364
128kB	256kB	1	8	128	1.000364
128kB	256kB	1	2	128	1.000369
128kB	128kB	1	4	128	1.000947
128kB	128kB	1	8	128	1.001007

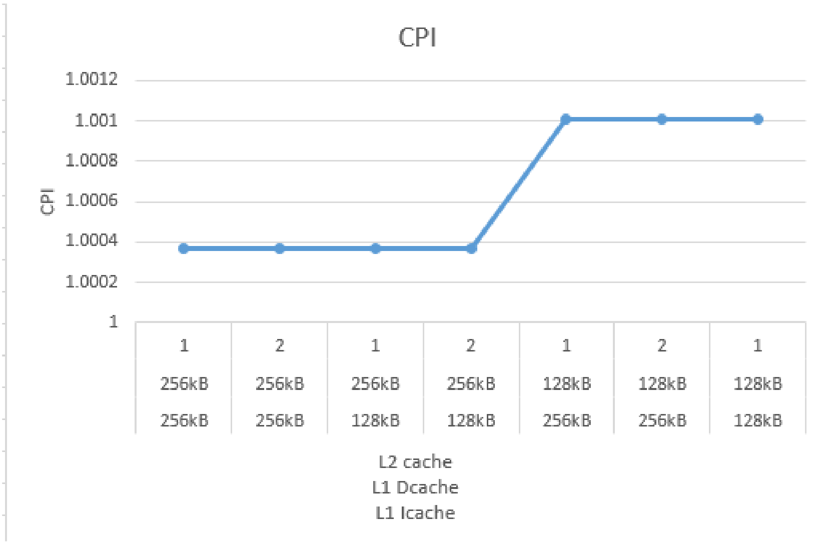

Figure 7: Benchmark – 3: Design choices vs CPI

d) Benchmark- IV (458.sjeng)

With base configuration of L1 cache & L2 cache size of 128kB and L2 cache size of 1MB, we modified the associativity and cache line size to derive the lowest CPI.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
128kB	128kB	1	4	128	1.474073

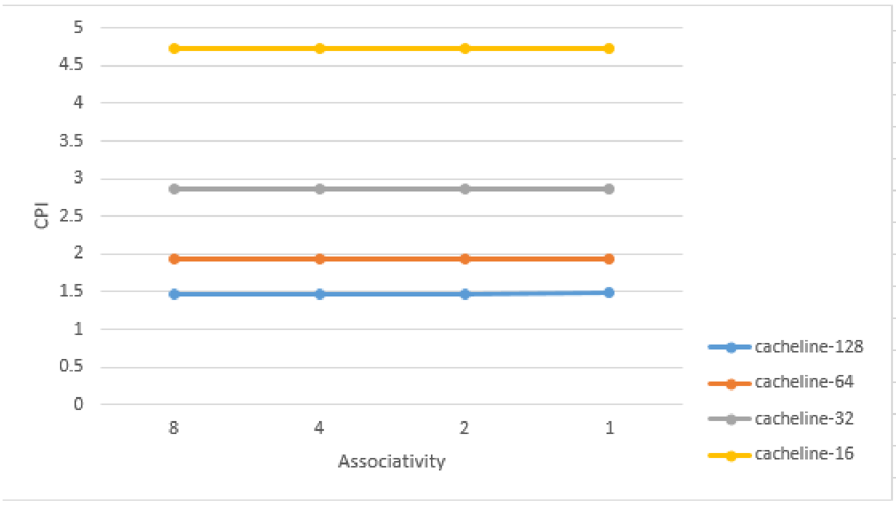

Figure 8: Benchmark – 4: Associativity vs CPI

By fixing Associativity = 8, cache line size = 128, we explored the cache design by varying the L1 and L2 cache sizes.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
256kB	256kB	2	8	128	1.473388
128kB	256kB	2	8	128	1.473388
256kB	256kB	1	8	128	1.473761
128kB	256kB	1	8	128	1.473761
128kB	128kB	1	8	128	1.473952
128kB	128kB	1	4	128	1.474073

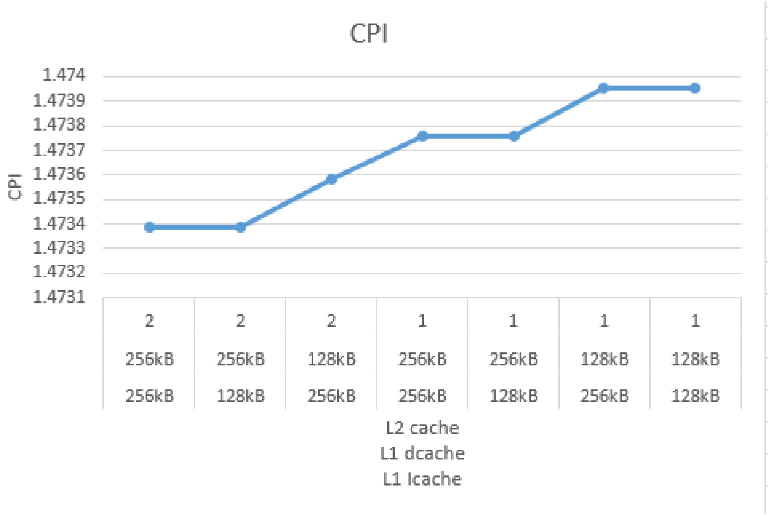

Figure 9: Benchmark – 4: Design choices vs CPI

e) Benchmark-V (470.lbm)

With base configuration of L1 cache & L2 cache size of 128kB and L2 cache size of 1MB, we modified the associativity and cache line size to derive the lowest CPI.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
256kB	256kB	2	8	128	1.425413

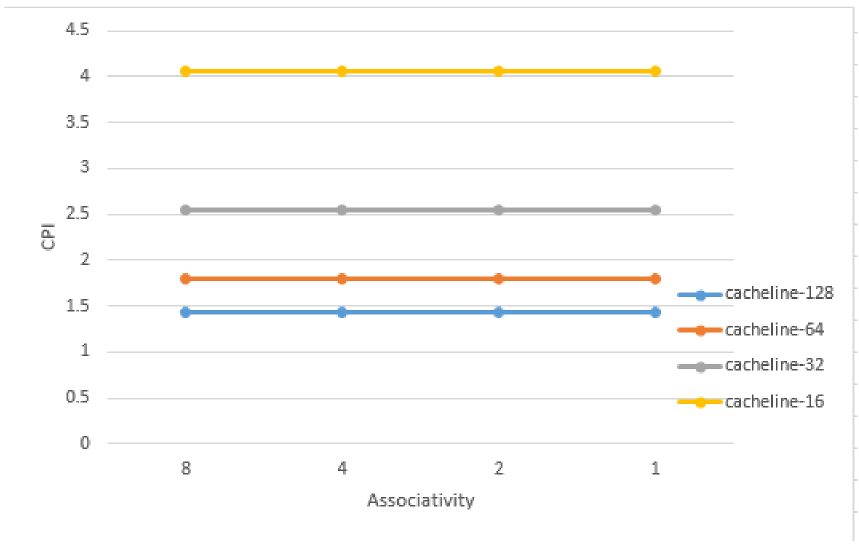

Figure 10: Benchmark – 5: Associativity vs CPI

By fixing Associativity = 8, cache line size = 128, we explored the cache design by varying the L1 and L2 cache sizes.

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI
256kB	256kB	2	8	128	1.425413
128kB	256kB	2	8	128	1.425413
128kB	128kB	1	8	128	1.42542
256kB	128kB	1	8	128	1.42542
128kB	128kB	1	4	128	1.425446
128kB	128kB	1	2	128	1.425598

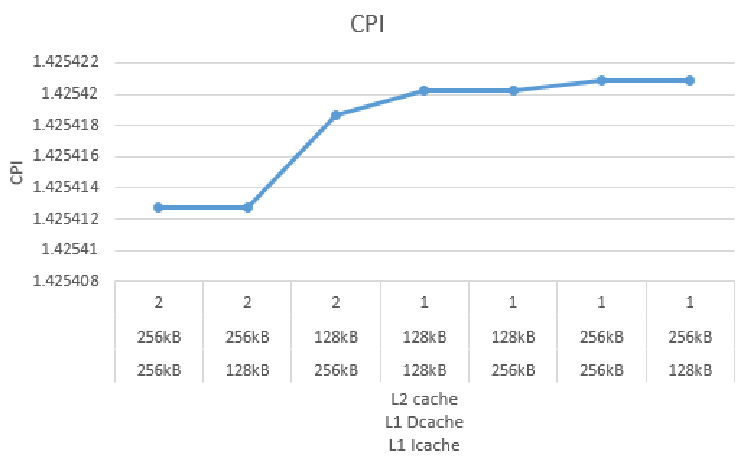

Figure 11: Benchmark – 5: Design choices vs CPI

4. Defining cost function

We defined a cost function for the caches in terms of area overhead. We defined the cost based on the number of transistors of the logic circuits. 'x' is an arbitrary cost unit.

Logic Gate	Cost approximation
2 input AND gate	x
2 input OR gate	x
2 input one bit MUX	2x
1 bit comparator	2x
L1 cache 1Byte	x
L2 cache 1Byte	X

We assumed L1 & L2 cache cost is same as per sizes. L2 cache is slower than L1 cache as the SRAM latency increases with the size. For associativity, we assumed additional mux logic and comparator logic required. For Multiple cache lines, we assumed additional comparator logic required. For larger cache sizes, as the number of bytes increases the size of the die increases and the cost of design increases.
With the base configuration discussed in section-3, we computed the cost for each benchmark.

- L1 instruction cache size – 128kB
- L1 data cache size – 128kB
- L2 cache size – 1MB
- L1 instruction associativity – 2
- L1 data associativity – 2
- L2 associativity – 2
- cache line size – 64

Cost of the cache design as per the above configuration = 1310975x

5. Optimize cache design based on performance/cost

Cache design configuration optimized for high performance (lower CPI) and lower cost.

a) Benchmark – I (401.bzip2)

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	2	8	128	1.03413	2622727
128kB	256kB	2	8	128	1.03413	2491671
128kB	128kB	2	4	64	1.056151	2359831
256kB	256kB	1	8	128	1.059201	1574167
128kB	256kB	1	8	128	1.059201	1443111
128kB	128kB	1	8	128	1.066336	1312055

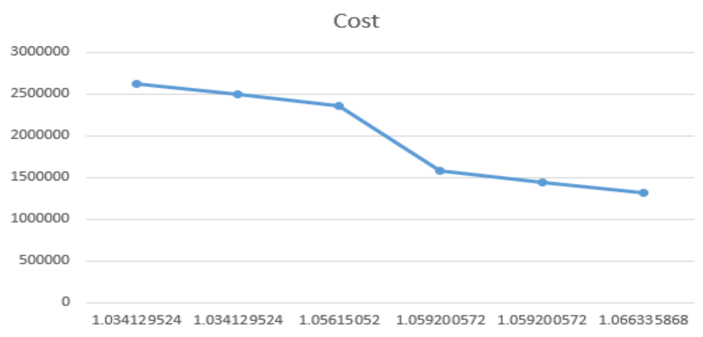

Figure 12: Benchmark – 1: CPI vs Cost

b) Benchmark – II (429.mcf)

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	2	8	128	1.368246	2622727
128kB	256kB	2	8	128	1.368246	2491671
128kB	128kB	2	4	128	1.410814	2359927
256kB	256kB	1	8	128	1.452938	1574167
128kB	256kB	1	8	128	1.452938	1443111
128kB	128kB	1	8	128	1.495679	1312055

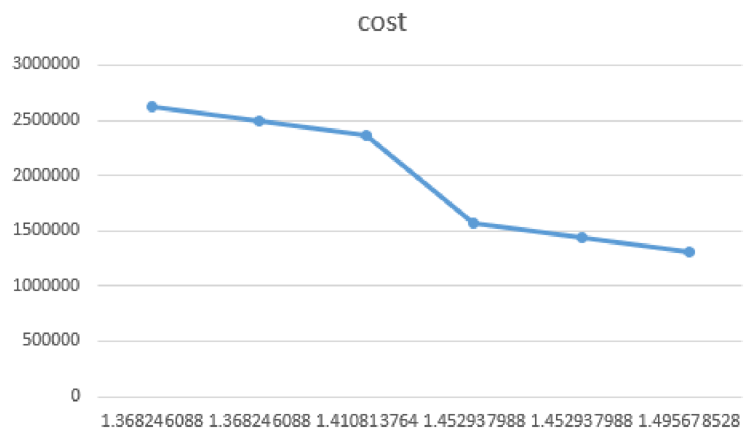

Figure 13: Benchmark – 2: CPI vs Cost

c) Benchmark – III (456.hmmer)

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	1	8	128	1.000364	1574167
256kB	256kB	2	8	128	1.000364	2622727
128kB	256kB	1	8	128	1.000364	1443111
128kB	256kB	1	2	128	1.000369	1442091
128kB	128kB	1	4	128	1.000947	1311359
128kB	128kB	1	8	128	1.001007	1312055

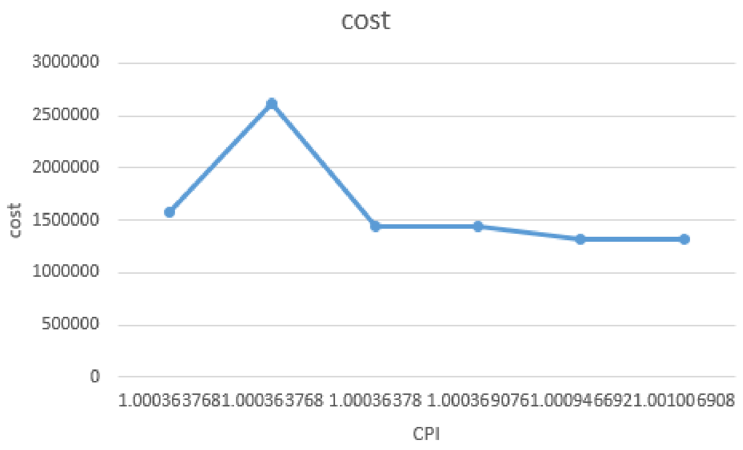

Figure 14: Benchmark – 3: CPI vs Cost

d) Benchmark- IV (458.sjeng)

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	2	8	128	1.473388	2622727
128kB	256kB	2	8	128	1.473388	2491671
256kB	256kB	1	8	128	1.473761	1574167
128kB	256kB	1	8	128	1.473761	1443111
128kB	128kB	1	8	128	1.473952	1312055
128kB	128kB	1	4	128	1.474073	1311359

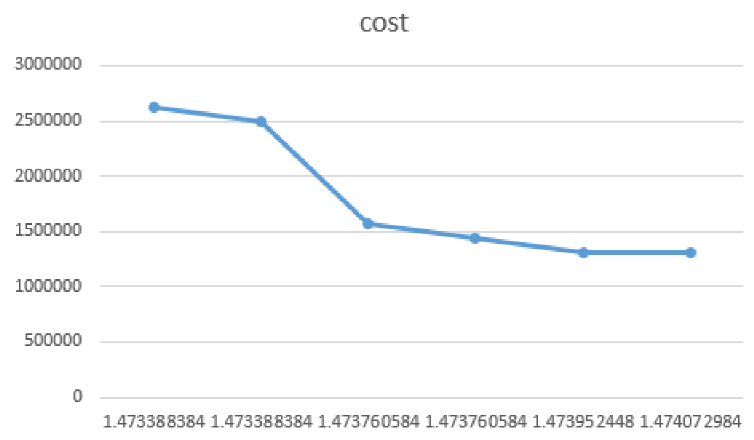

Figure 15: Benchmark – 4: CPI vs Cost

e) Benchmark-V (470.lbm)

L1 Icache	l1 Dcache	L2 Cache	Assoc	Cachelines	CPI	Cost
256kB	256kB	2	8	128	1.425413	2622727
128kB	256kB	2	8	128	1.425413	2491671
128kB	128kB	1	8	128	1.42542	1312055
256kB	128kB	1	8	128	1.42542	1443111
128kB	128kB	1	4	128	1.425446	1311359
128kB	128kB	1	2	128	1.425598	1311023

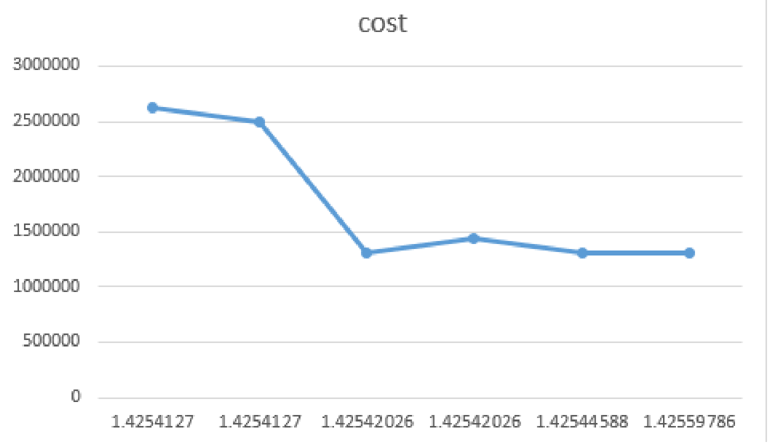

Figure 16: Benchmark – 5: CPI vs Cost

6. Conclusion

Advanced cache design and optimization techniques play a pivotal role in enhancing CPU performance, addressing challenges posed by modern workloads and energy constraints. This study has explored innovative approaches, such as variable-way set-associative caches, adaptive resizing techniques, and hybrid memory solutions, to optimize cache utilization and reduce latency. These techniques enable CPUs to handle dynamic workloads more efficiently, lowering cache miss rates and improving the overall system throughput. The integration of predictive modeling, machine learning-based cache tuning, and analytical design space exploration further highlights the potential to automate and accelerate cache optimization. By tailoring cache configurations to specific applications and hardware environments, these methods ensure a fine balance between performance and power efficiency, crucial for embedded systems, mobile devices, and high-performance computing platforms [5].

The findings underscore the importance of adaptability in cache architecture to cater to diverse and evolving computational demands. By addressing key bottlenecks such as cache conflicts and memory access delays, these advancements pave the way for significant gains in CPU efficiency and energy savings. Future research should focus on extending these techniques to multi-core and heterogeneous computing environments, where cache behavior becomes increasingly complex. These efforts will continue to drive innovation, making CPUs faster, more efficient, and better suited to meet the demands of modern computing systems [7].

7. References

1. Kenneth O'Neal and Philip Brisk, "Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey," IEEE Computer Society Annual Symposium on VLSI, 2018.
2. Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi, "Mobile CPU's Rise to Power: Quantifying the Impact of Generational Mobile CPU Design Trends on Performance, Energy, and User Satisfaction," The University of Texas at Austin, 2016.
3. Priati Assiroj, April Lia Hananto, Ahmad Fauzi, and Harco Leslie Hendric Spits Warnars, "High Performance Computing (HPC) Implementation: A Survey," INAPR International Conference, IEEE, 2018.
4. Xiancheng Xu, "Flow Cache Design for Improving Traffic Collection in NP-based Network Monitor System," International Conference on E-Business and E-Government, IEEE, 2010.
5. Ramy E. Aly, Bharat R. Nallamilli, and Magdy A. Bayoumi, "Variable-way Set Associative Cache Design for Embedded System Applications," IEEE, 2004.
6. Yan Li, Kenneth Chang, Oceane Bel, Ethan L. Miller, and Darrell D. E. Long, "CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning," SC17, Denver, CO, 2017.
7. Michał Getka and Michał Karpowicz, "Fixed-Point Self-Tuning CPU Performance Controller for Linux Kernel," IEEE, 2019.
8. Xiaoan Ding, Yi Liu, and Depei Qian, "JellyFish: Online Performance Tuning with Adaptive Configuration and Elastic Container in Hadoop YARN," IEEE International Conference on Parallel and Distributed Systems, 2015.
9. Arijit Ghosh and Tony Givargis, "Analytical Design Space Exploration of Caches for Embedded Systems," Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), IEEE, 2003.