Optimizing Cache Performance: RTL-Designed Adaptive Replacement Policies for Enhanced Cache Efficiency

# Jayanthi Abhilash Preetham

*Dept. of Electronics & Communication*

*R.V College of Engineering*

Bangalore, India [jayanthiap.ec20@rvce.edu.in](mailto:jayanthiap.ec20@rvce.edu.in)

# Smruthi Srinivasan

*Dept. of Electronics & Communication*

*R.V College of Engineering*

Bangalore, India [smruthis.ec20@rvce.edu.in](mailto:smruthis.ec20@rvce.edu.in)

# Abhijith.S

*Dept. of Electronics & Communication*

*R.V College of Engineering*

Bangalore, India [abhijiths.ec20@rvce.edu.in](mailto:abhijiths.ec20@rvce.edu.in)

# Dr.N.Ramavenkateswaran

*Dept. of Electronics & Communication*

*R.V College of Engineering*

Bangalore, India [ramavenkatesw](mailto:ramavenkateswarann@rvce.edu.in)[arann@rvce.edu.in](mailto:arann@rvce.edu.in)

***Abstract*—This paper presents a comprehensive study on the implementation of adaptive cache replacement policies in RTL cache controllers. The proposed cache controller incorporates three distinct replacement policies - LRU (Least Recently Used), pseudo LRU, and LFU (Least Frequently Used) - which are dynamically enabled based on input processes. This adaptive approach aims to improve cache hit rates by accommodating changes in access patterns effectively. Through extensive RTL design and simulation, our results demonstrate a significant enhancement in cache hit rates, achieving a remarkable im- provement of 1.7-1.93% compared to conventional LRU-based designs. Furthermore, our design exhibits a notable reduction of over 70% in both area and power requirements compared to ma- chine learning-based cache methods. This research highlights the effectiveness of adaptive cache designs in optimizing cache per- formance while minimizing resource utilization, thereby offering promising prospects for future cache controller implementations.**

***Index Terms*—adaptive cache, replacement, System Verilog**

1. INTRODUCTION

With the widening gap between processor and main memory speeds, the demand for advanced caching mechanisms has become more pressing than ever. Processor performance now heavily relies on the efficiency of on-chip caches, as access- ing RAM incurs substantial latency costs. While computer architects have explored various avenues such as cache size expansion and latency hiding techniques, there has been lim- ited recent focus on enhancing cache replacement algorithms. The prevailing notion appears to be that Least Recently Used (LRU) is sufficient, and little can be done to improve cache hit rates without costly processing overhead.

However, recent advancements in machine learning (ML) have introduced novel approaches to cache replacement, of-

Identify applicable funding agency here. If none, delete this.

fering the promise of improved cache hit rates and system efficiency. ML-based cache replacement algorithms leverage predictive models trained on historical access patterns to dy- namically adjust cache replacement decisions. By harnessing ML techniques, these algorithms aim to anticipate future data accesses more accurately, potentially leading to better cache utilization and enhanced application performance. However, the computational overhead associated with training and de- ploying ML models, coupled with the need for substantial hardware resources, presents challenges in terms of implemen- tation feasibility.

In contrast to ML-based approaches, adaptive cache re- placement policies offer a different paradigm by dynamically adapting replacement strategies based on runtime behavior. These policies seek to strike a balance between performance gains and resource efficiency, adjusting replacement decisions in real-time to optimize cache utilization without incurring excessive hardware overhead. This adaptive approach holds promise for achieving efficient cache management in diverse computing environments.

1. REPLACEMENT POLICIES IN CACHE
2. *Least Recently Used (LRU)*

Least Recently Used (LRU) cache replacement policy is a well-established method employed in cache management, primarily designed to evict the least recently accessed cache line when the cache reaches its capacity threshold. This policy operates by maintaining a timestamp or access counter for each cache line, enabling the identification of the line with the oldest timestamp or the least number of accesses for eviction upon cache overflow. Despite LRU’s effectiveness in enhancing cache performance by prioritizing recently accessed

data, its implementation necessitates considerable hardware resources to track the access history of individual cache lines.

1. *Pseudo Least Recently Used (pLRU)*

PLRU, an advancement over the classic LRU policy, em- ploys a binary tree structure to mimic its behavior efficiently. In this scheme, each node within the binary tree corresponds to a cache set, while the child nodes represent individual cache lines. Upon accessing a cache line, the associated tree node is modified to reflect its usage, effectively emulating LRU princi- ples without the requirement for explicit timestamp or counter management. This approach of PLRU achieves a harmonious blend of LRU’s performance advantages and the streamlined hardware complexity inherent in simpler replacement policies.

1. *Least Frequently Used (LFU)*

LFU represents a cache replacement strategy focused on removing the cache line with the least number of accesses accumulated over time. This policy monitors the access fre- quency of individual cache lines and opts to evict the one with the lowest count when the cache reaches its capacity limit. Despite LFU’s objective of preserving less frequently accessed data, it may encounter challenges in adjusting to dynamic access patterns and demands additional hardware resources to track access counts for each line. Nevertheless, LFU exhibits effectiveness in scenarios characterized by stable and predictable access behaviors.

1. *Adaptive Replacement Algorithm*

This represents a case where LRU, pLRU and LFU policies can be chosen depending on the workload and application. Though extremely efficient ARA invlove the usage of real time prediction of policies and require hardware overhead and real-time adaptation capabilities, the one implemented in the paper is user centric and the choice is left to the user.

1. METHODOLOGY

The major shortcomings of the replacement techniques are that: LRU requires sophisticated hardware support and uses a rigorous amount of age bits and is difficult to manage for higher set associative caches. pLRU reduces the number of age bits used but the design logic is more intensive than LRU and might prove futile if the cache is direct mapping based, not requiring extra logic. LFU is a good choice if the memory requirement leans more towards spatial locality.

Therefore, the idea of adaptive cache replacement proposed in [8] is modified and presented in this paper. Adaptive replacement policies enable the user to select the cache replacement policies depending on the type of workload to be accessed and the intensity of data retrieval. This is a better choice than the use of Machine Learning (ML) as ML is more computationally intensive and requires the use of more hardware than the simple case based replacement policy switching.

Our cache controller design leverages a state machine architecture to efficiently manage dynamic cache operations,

ensuring optimal data access and consistency. The design features key states such as Idle, Write, Read, Invalidate, Replacement, and Penalty Read. Transitions between these states are governed by specific operational signals. Initially,

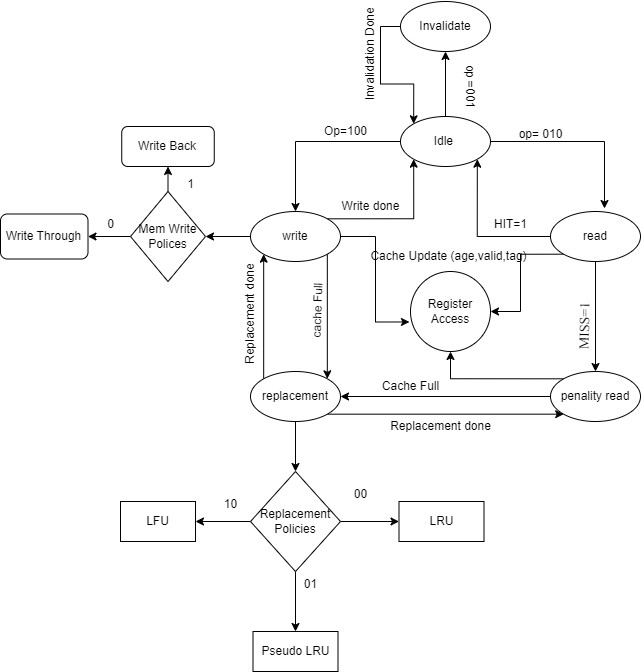


Fig. 1. Design of cache controller

the controller is in the Idle state, awaiting operations. For a write operation (op = 100), it transitions to the Write state, where it performs the write and updates cache metadata (age, valid bit, tag). If the cache is full, it moves to the Replacement state to free up space. After completion (Write done), it returns to Idle.1 shows the design flow of the RTL implementation

During a read operation (op = 010), the controller transitions to the Read state. If a cache hit occurs (HIT = 1), the data is accessed directly. On a cache miss (MISS = 1), a penalty read fetches the data from the main memory, followed by a cache update.

For invalidate operations (op = 001), the controller transi- tions to the Invalidate state to invalidate specific cache lines and then returns to Idle. In this state, the snoopy bus protocol is implemented to maintain cache coherence.

In the Replacement state, the controller selects a cache line to replace based on the active replacement policy, which can be Least Recently Used (LRU), Least Frequently Used (LFU), or Pseudo-LRU, determined by control signals (00 for LRU, 01 for Pseudo-LRU, 10 for LFU). To choose between these policies, a case structure maps to the specified policy implementation.

Our design incorporates a 256kB cache and a 3MB main memory, with provisions for parameterized cache and memory configurations. The cache contains 1024 sets, with set asso- ciativity varying as 2-way, 4-way, and 8-way. This flexible

design supports both write-back and write-through techniques, accommodating a variety of memory operations.

To evaluate performance, we conducted constrained random generation of addresses for 2-way, 4-way, and 8-way set associative configurations. We calculated hit and miss rates for LRU, pLRU, and LFU replacement policies, as well as adaptive replacement techniques. The read and write penalties are used when the address is not found in the cache, necessi- tating data fetch or write operations to the main memory.

* 1. *Algorithm of cache controller:*

1. Start in Idle state
2. Upon receiving an operation (op):
   * If op=100 (Write):
     + Write state: write data, update metadata.
     + If cache full: Transition to Replacement state; otherwise, transition back to Idle state.
   * If op=010 (Read):
     + Read state: if HIT, access data; else, perform Penalty Read and update cache.
     + Transition back to Idle state.
   * If op=001 (Invalidate):
     + Invalidate state: invalidate cache lines.
     + Transition back to Idle state.
3. In Replacement state:
   * Apply replacement policy (LRU, LFU, Pseudo- LRU) based on priority and thresholds.
   * If current policy threshold exceeded:
     + Select new policy based on process input and priority.
     + Transition to Write state.
4. Repeat the process.

This streamlined algorithm outlines the functional flow of the cache controller, ensuring efficient handling of memory requests and maintaining cache coherence.

1. RESULTS

The presented figures illustrate the hit and miss rates corre- sponding to varying degrees of set associativity (2-way, 4-way, and 8-way) across cache replacement policies, namely LRU, PLRU, LFU, and Adaptive replacement techniques.

Figure 2 presents hit rates utilizing the LRU cache replace- ment policy.

Figure 3 displays hit rates employing the pLRU policy. In Figure 4, hit rates are depicted under the LFU cache replacement policy, while Figure 5 represents hit rates using the Adaptive replacement technique.

The ”Hit Max” metric signifies the hit rate when the most frequently accessed tag by the application results in a hit. Similarly, ”Hit Med” denotes the hit rate when a tag accessed moderately by the application leads to a hit, and ”Hit Low” represents the hit rate when a tag accessed least frequently by the application results in a hit. The average hit rates of different replacement policies for 2-way, 4-way and 8-way set associativities are shown in Figure 6.

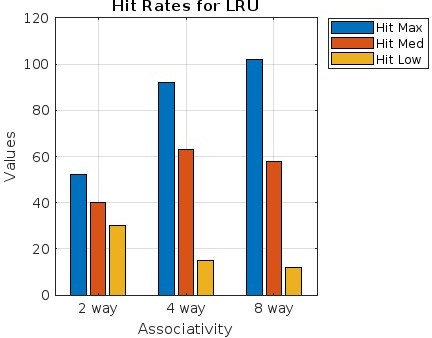


Fig. 2. Hit Rates using LRU

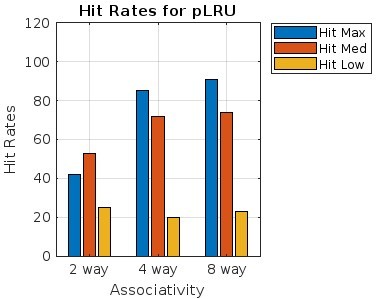


Fig. 3. Hit Rates using PLRU

1. CONCLUSION

The adoption of an adaptive replacement policy for cache management presents a better hit rate compared to traditional policies such as LRU, PLRU, or LFU, for the same set of accesses. The adaptive nature of these policies allows for dynamic adjustment to varying workload conditions, opti- mizing cache utilization and performance. For a 2-way set associative cache, the Adaptive replacement policy achieves a hit rate of 48%, which outperforms both LRU and pLRU by approximately 20%, though it slightly underperforms com- pared to LFU, which registers a hit rate of 50%. In the 4- way associative setting, the Adaptive policy records a hit rate of %, demonstrating a notable improvement of approximately 3.4% over LRU at 58% and 11.1% over pLRU at 54%, while significantly outpacing LFU by 25%. However, in the 8-way associative configuration, while the Adaptive policy sustains a hit rate of 58%, it falls short of LRU’s 62% by approximately

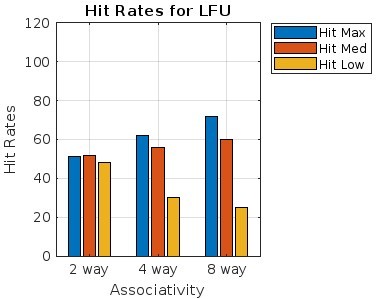
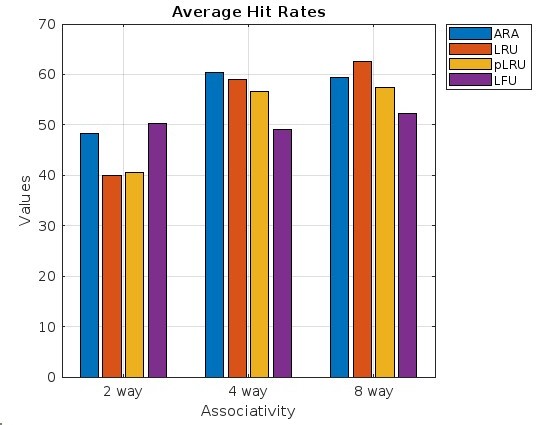
 

Fig. 4. Hit Rates using LFU

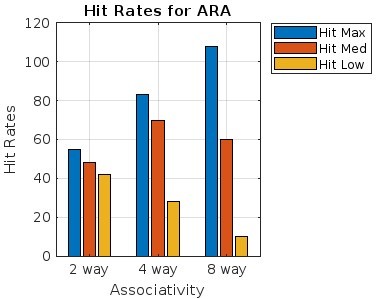


Fig. 5. Hit Rates using Adaptive Cache Replacement Technique

6.5%, yet still outstrips pLRU and LFU by 7.4% and 11.5%, respectively.

REFERENCES

1. Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt, “Accelerating dependent cache misses with an enhanced memory controller,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 444–455, 2016. Publisher: ACM New York, NY, USA.
2. E. Keshav, K. Manas Reddi, A. Pritam, and S. Muthukumar, “Cache Performance Optimization Using Logical Cache Segmentation,” in *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pp. 1–6, 2021. Organization: IEEE..
3. Zhe Jiang, Kecheng Yang, Nathan Fisher, Nan Guan, Neil C. Audsley, and Zheng Dong, “Hopscotch: A Hardware-Software Co-Design for Efficient Cache Resizing on Multi-Core SoCs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 1, pp. 89–104, 2023. Publisher: IEEE.
4. B. Calder, Dirk Grunwald, and Joel Emer, “Predictive sequential as- sociative cache,” in *1996 IEEE/ACM International Symposium on High Performance Computer Architecture (HPCA)*, pp. 244–253, March 1996. ISBN: 0-8186-7237-4. DOI: 10.1109/HPCA.1996.501190.

Fig. 6. Average Hit rates using different replacement policies

1. Fei Guo and Yan Solihin, “An analytical model for cache replacement policy performance,” in *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pp. 228–239, 2006.
2. Armin Vakil-Ghahani, Sara Mahdizadeh-Shahri, Mohammad-Reza Lotfi- Namin, Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad, “Cache replacement policy based on expected hit count,” *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 64–67, 2017. Publisher: IEEE.
3. Debiprasanna Sahoo, Swaraj Sha, Manoranjan Satpathy, Madhu Mutyam, S. Ramesh, and Partha Roop, “Formal modeling and verifi- cation of controllers for a family of DRAM caches,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2485–2496, 2018. Publisher: IEEE.
4. Ranjith Subramanian, Yannis Smaragdakis, and Gabriel Loh, “Adaptive Caches: Effective Shaping of Cache Behavior to Workloads,” in *Proceed- ings of the 2006 IEEE/ACM International Symposium on Microarchi- tecture (MICRO)*, Dec. 2006, pp. 385-396. doi: 10.1109/MICRO.2006.7
5. Daniel A. Jime´nez, Elvira Teran, and Paul V. Gratz, “Last-level cache insertion and promotion policy in the presence of aggressive prefetch- ing,” *IEEE Computer Architecture Letters*, vol. 22, no. 1, pp. 17–20, 2023. Publisher: IEEE.
6. Kousik Kumar Dutta, Prathamesh Nitin Tanksale, and Shirshendu Das, “A fairness conscious cache replacement policy for last level cache,” in *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 695–700. Publisher: IEEE.
7. Yuri A. Nedbailo, Alexander V. Surchenko, and Ignat Nikolaevich Bychkov, “Reducing miss rate in a non-inclusive cache with inclusive directory of a chip multiprocessor,”
8. Kapania, Ashish and Ravish Aradhya, H. V.”, title=”Design Space Exploration of Power Efficient Cache Design Techniques”, bookti- tle=”Advances in Networks and Communications”,2011,Berlin, Heidel- berg, pp. 362–371, Publisher: Springer
9. J. Ye, Z. Li, Z. Wang, Z. Zheng, H. Hu, and W. Zhu, ”Joint cache size scaling and replacement adaptation for small content providers,” in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1-10, 2021.
10. B. S. Gill and D. S. Modha, ”SARC: Sequential Prefetching in Adaptive Replacement Cache,” in *USENIX Annual Technical Conference, General Track*, pp. 293-308, 2005.
11. M. E. Consuegra, W. A. Martinez, G. Narasimhan, R. Rangaswami, L. Shao, and G. Vietri, ”Analyzing adaptive cache replacement strategies,” *arXiv preprint arXiv:1503.07624*, 2015.