# Analysis of LLC replacement policies

**Research Project Report**

Submitted in partial fulfillment of the requirements
for the completion of

**Summer Project**

by
**Rishabh**
**(Roll No. 200260041)**

Under the guidance of
**Prof. Virendra Singh**


**Department of Electrical Engineering**
**Indian Institute of Technology Bombay**
**October 2022**

**Acknowledgement**

I express my gratitude to my guide Prof. Virendra Singh for providing me the opportunity to work on this topic.

Rishabh
Electrical Engineering
IIT Bombay

**Abstract**

Cache replacement policies are crucial for improving the performance of computer systems. The large access time for data and instructions from the main memory led to the introduction of caches. Furthermore, to effectively handle caching and eviction of data blocks in caches' we look for replacement algorithms.

Fundamental algorithms are LRU and MRU that simulate the cache to function exactly like a FIFO and LIFO queue respectively. These are efficient algorithms but there is a scope for a lot of improvement.

RRIP builds upon LRU and maintains counters for each line in a set, that ages on set accesses and resets on line hits. This provides marginal improvement as compared to LRU. Further improvements on this involved DRRIP that dynamically based on set access patterns chose between SRRIP and BRRIP as its algorithm.

A more advanced algorithm was the Hawkeye algorithm that was a ML based predictor. It primarily predicted if the demanded line was cache friendly or not. This decided if it were to be inserted with high priority into the cache or not

Mockingjay an even more advanced algorithm, used multi class prediction to accurately predict the reuse distance and store those line whose predicted reuse was in the near future.

# Contents

# List of Figures

# Chapter 1

# Introduction

The cache memory is a resource that does not need to be explicitly managed by the user. Instead, the cache is managed by a set of cache replacement policies (also called cache algorithms) that determine which data is stored in the cache during the execution of a program. To be both cost-effective and efficient, caches are usually several orders-of-magnitude smaller than main memory (e.g., there are typically a few KB of L1-cache and a few MB of L3-cache versus many GB or even a few TB of main memory). As a consequence, the dataset that we are currently working on (the working set) can easily exceed the cache capacity for many applications. To handle this limitation, cache algorithms are required that address the questions of

1. Which data do we load from main memory and where in the cache do we store it?
2. If the cache is already full, which data do we evict?

If the CPU requests a data item during program execution, it is first determined whether it is already stored in cache. If this is the case, the request can be serviced by reading from the cache without the need for a time-consuming main memory transfer. This is called a cache hit. Otherwise, we have a cache miss. Cache algorithms aim at optimizing the hit ratio; i.e. the percentage of data requests resulting in a cache hit.

# Chapter 2

# Literature Survey

## 2.1 RRIP

Re-Reference Interval Prediction (RRIP) proposes the idea to maintain an array of counters for set in the cache.The counters increase on each miss to the set, and reset the accessed line's counter ona hit. New lines are inserted with the counter value predicting a long (SRRIP) or a distant (BRRIP) reuse interval. It also suggests DRRIP that dynamically shifts between SRRIP and BRRIP based on the access patterns. These favour recency friendly and distant access patterns respectively. DRRIP provides both scan-resistance and thrash-resistance by using set dueling to dynamically select between inserting all missing cache blocks with an intermediate re-reference interval or with a distant re-reference interval.

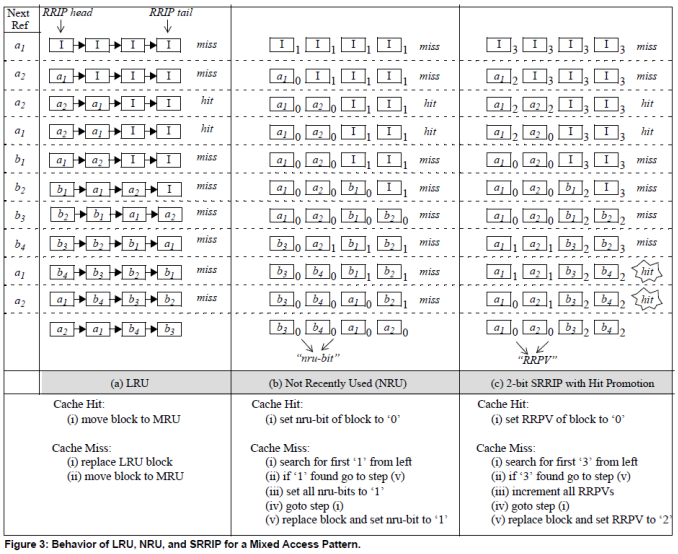


Figure 2.1: Behavior of LRU, NRU, and SRRIP for a Mixed Access Pattern

## 2.2 Hawkeye

Hawkeye[2] is a cache replacement algorithm which learns from Belady's optimal solution (OPT) algorithm by applying it to past cache accesses to predict the future cache replacement decisions. To learn from the past behavior of Belady's algorithm, it observes that if with the OPT solution, a load instruction has historically brought in lines that produce cache hits, then in the future,

the same load instruction is likely to bring in lines that will also produce cache hits. Its main components are the Hawkeye Predictor, which makes eviction decisions, and OPTgen, which simulates OPT's behavior to produce inputs that train the Hawkeye Predictor.

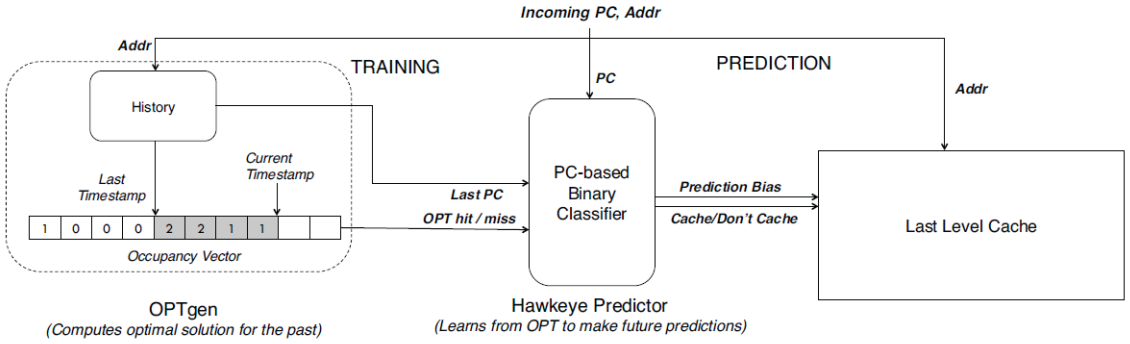


Figure 2.2: Block diagram of the Hawkeye replacement algorithm

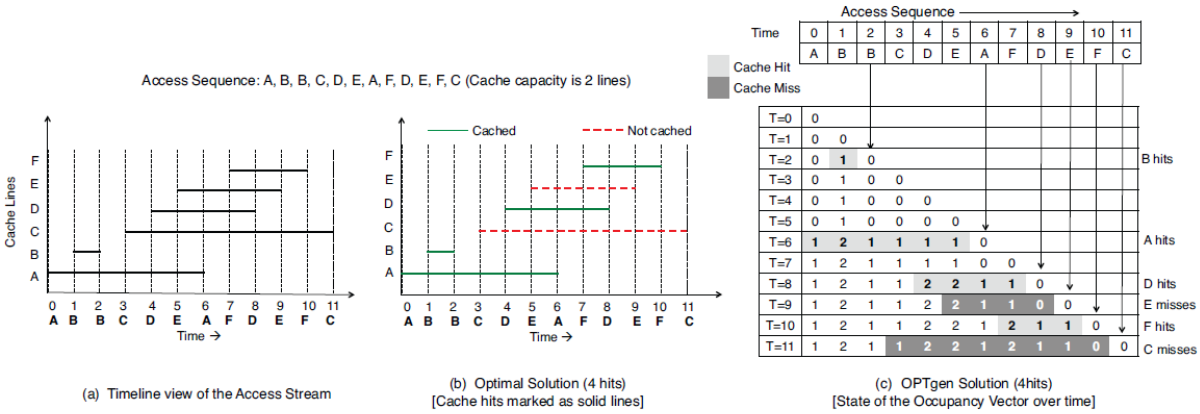


Figure 2.3: Figure to illustrate OPTgen

# Chapter 3

# Proposed Idea

## 3.1 Mockingjay

The past decade has seen the rise of highly successful cache replacement policies that are based on binary prediction. For example, the Hawkeye[2] policy learns whether lines loaded by a given PC are Cache Friendly (likely to remain in the cache if Belady's MIN policy had been used) or Cache Averse (likely to be evicted by Belady's MIN policy). This paper, presents a cache replacement policy that is based on multiclass prediction, which allows it to directly mimic Belady's MIN policy in a surprisingly simple and effective way. The policy uses a PC-based predictor to learn each cache line's reuse distance; it then evicts lines based on their predicted time of reuse. It is shown that use of multiclass prediction is more effective than binary prediction because it allows for a finergrained ordering of cache lines during eviction and because it is more robust to prediction errors. Binary classification leads to two shortfalls: First, a small prediction error will flip a prediction from one class to the other. Second, there can be many ties among lines in the same class, which are typically broken by falling back on the same LRU heuristic that these polices attempt to improve upon. Mockingjay[1] has three main components which are the Sampled Cache, the Reuse Distance Predictor (RDP), and the ETA Counters that reside in the LLC.

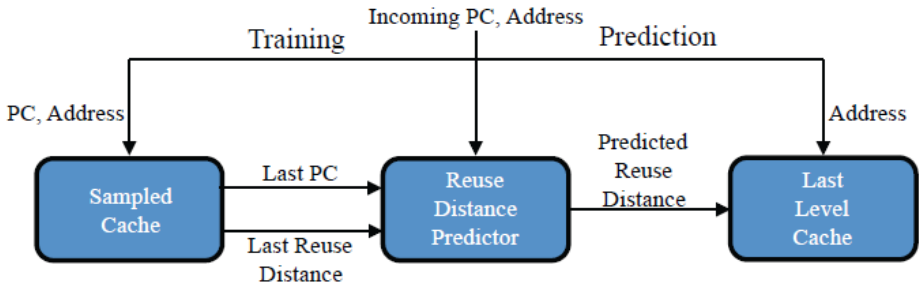


Figure 3.1: Overview of the Mockingjay[1] cache replacement policy

Sampled Cache: The goal of the Sampled Cache is to track past reuse distances to train the RDP, which predicts future reuse distances
Reuse Distance Predictor (RDP): The RDP is a PCbased predictor that learns reuse distances for loads initiated by a given program counter (PC). The RDP is organized as a direct-mapped cache and is indexed by a PC signature. Mockingjay[1] uses separate predictor entries—and therefore learns separate reuse distances—for loads by a PC that hit in the cache and for loads by that same PC that miss in the cache
ETA Counters: Finally, the cache itself maintains the ETA for each line. Upon insertion into the cache, a line's predicted reuse distance is converted to an ETA, and these ETA values are used to make eviction decisions for future cache misses. Cache insertions that are predicted to have ETA values larger than any existing line in the set are bypassed, which means that they are not inserted in the cache.

# Chapter 4

# Conclusion

## 4.1 Motivation

The introduction of memory hierarchy to tackle the long latencies involved with direct memory access lead to the search for effective replacement policies. The LLC is usually where these policies are implemented as they are relatively large compared to other caches. The LLC's primary objective is to cater to the misses that occur in higher cache levels and due to the large size of main memory compared to cache, the search for efficient replacement policies began. The goal is decrease the number of misses on the LLC so that the requirements of higher cache levels are met immediately thereby decreasing delays associated with access to main memory.

## 4.2 Completed Tasks

1. Covered literature on the architecture of Graphic Processing Unit, and parallel programming
2. Covered literature on Instruction Level Parallelism, introduced to concepts of VLIW and Superscalar architecture
3. Learnt memory architecture in computers, introducing concepts of virtual memory
4. Extensively analyzed existing literature on Last Level Cache (LLC) optimizations
5. Covered literature on more efficient replacement policies such as RRIP[3], Hawkeye[2] and Mockingjay[1]
6. Read SNIPER implementation of LRU, SRRIP[3], DRRIP[3], MRU, PLRU, NMRU, DAAIP[4] and NRU
7. Read the ChampSim Simulator implementation of the same replacement policies

## 4.3 End Goals

1. Explore new policies to optimally replace cache lines in the LLC
2. Successfully implement the Mockingjay replacement policy on SniperSim

## 4.4 Future Plans

1. Intend to complete simulation of the Mockingjay LLC replacement policy on SNIPERsim
2. Present the Mockingjay paper
3. Explore more on effective repacement policies

# References


[1] A. J. Calvin Lin, Ishan Shah, "Effective mimicry of belady's min policy," *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022.

[2] A. J. Calvin Lin, "Back to the future: Leveraging belady's algorithm for improved cache replacement," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture*, 2022.

[3] G. Jia, X. Li, C. Wang, X. Zhou, and Z. Zhu, "Cache promotion policy using re-reference interval prediction," in *2012 IEEE International Conference on Cluster Computing*, pp. 534–537, 2012.

[4] Newton, S. K. Mahto, S. Pai, and V. Singh, "Daaip: Deadblock aware adaptive insertion policy for high performance caching," in *2017 IEEE International Conference on Computer Design (ICCD)*, pp. 345–352, 2017.