Analysis of Zero Inclusion Victim (ZIV) Last Level Cache Architecture

Summer Project Monthly Report - September

Submitted in partial fulfillment of the requirements

of

Summer Project

by

Ankith R

(Roll No. 200070006)

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.

Ankith R
Electrical Engineering
IIT Bombay

Abstract

Cache is a random access memory used by the CPU to reduce the average time taken to access memory. Multi-level caches is one of the technique to improve the cache performance by reducing the miss penalty, where miss penalty refers to the extra time required to bring the data into the cache from main memory whenever there is a miss in the cache

Multi-level caches can be designed in various ways depending on whether the content of one cache is present in other levels of the cache. If all the contents of higher level caches are present in the lower level cache then the lower level cache is called the inclusive cache. The advantage of the inclusive policy is that it simplifies the cache coherence protocol in case of multi-core processors. The major drawback of the inclusiveness is the inclusion victims which can lead to severe performance degradation. An inclusion victim is a block that must be forcefully replaced from the inner levels of the cache hierarchy when the copy of the block is replaced from the inclusive last level cache (LLC)

The Zero Inclusion Victim (ZIV) LLC design guarantees freedom from inclusion victims while retaining all advantages of an Inclusive LLC thereby isolating the core caches from LLC evictions. The ZIV LLC efficiently and minimally enables a global victim selection scheme in the inclusive LLC to avoid the generation of inclusion victims. This report contains the details and analysis of all the simulations done by implementing the zero inclusion victim design in the Champsim simulator. The cache hierarchy aware replacement (CHAR) algorithm for inclusive LLCs is implemented for finding the likely dead state of an LLC block

Contents

List of Figures **2**

1 Introduction **4**

2 Literature survey **6**

3 Proposed Idea **7**

3.1 Zero Inclusion victim 7

4 Conclusion **9**

4.1 Completed work 9

4.2 Future plans for next month 9

4.3 End goal of the project 9

List of Figures

3.1 Overview of functional flow of ZIV LLC (Ref: [1]) 8

Chapter 1

Introduction

Cache memory is a high-speed memory that acts as a buffer between the CPU and RAM. It holds the data and instructions that are frequently used so that they are made available to CPU when needed, so it reduces the average time of access of data and instructions. Since the size of these caches is small as compared to main memory it is very important to store the information that will be reused and replace the information that is less important using better replacement policies. An optimal replacement policy is one that makes its replacement decisions based on perfect knowledge of the reuse pattern of each cache block and replaces the block that will be reused furthest in the future. However, the implementation is not possible since it requires future information so practical cache replacement policies can be viewed as basing their replacement decisions on a prediction of which block will be reused furthest in the future and picking that block for replacement. Applications that exhibit a distant re-reference interval perform badly under the least recently used policy, such applications usually have large working set or have frequent bursts of references to non-temporal data (called scans). To improve the performance of such workloads cache replacement using re-reference interval prediction [2] was proposed

Multilevel caches is one of the techniques to improve cache performance by reducing the miss penalty, where the miss penalty refers to the extra time required to bring the data into cache from the main memory whenever there is a miss in the cache. Three types of Multi-level cache designs based on the contents of one cache level being present in another cache level are inclusive, exclusive and non-inclusive. If all blocks in the higher level cache are also present in the lower level cache, then the lower level cache is said to be inclusive of the higher level cache. If the lower level cache contains only blocks that are not present in the higher level cache, then the lower level cache is said to be exclusive of the higher level cache. If the contents of the lower level cache are neither strictly inclusive nor exclusive of the higher level cache, then it is called non-inclusive cache

The most widely used last level cache (LLC) architecture in microprocessors has been the inclusive LLC design because it offers bandwidth optimization and simplification to cache coherence protocols. Cache coherence refers to the uniformity of the shared data stored in the multiple local caches in a multi-core system. Cache coherence ensures that the changes in the values of shared data are propagated throughout the system in a timely fashion. If there is a cache miss in one of the private caches then other peer caches are checked for that block, in the case of inclusive LLC there is no need to search higher level caches if there is a miss in LLC because inclusive policy ensures that all the blocks in the private caches are present in the last level cache but in case of exclusive LLC or non-inclusive LLC we need to search all the private caches for the block

One of the major problem attached with the inclusive LLC design is inclusion victims. The inclusive victim is the block that is forcefully replaced from the inner level cache of the cache hierarchy when the copy of the same is replaced from the inclusive LLC. This tight coupling leads to three major drawbacks

1. Live inclusion victim leads to severe performance degradation depending on LLC replacement policy
2. A process can victimize the blocks of another process in an LLC shared by multiple cores and this can be exploited to leak information through well-known eviction-based timing side-channels
3. To reduce the above mentioned drawbacks the inner-level caches particularly the mid-level cache in a three-level hierarchy must be kept small

These inclusion victims are not fundamental to the inclusion property but arise due to the way the contents of the inclusive LLC are managed. To overcome this Zero inclusion victim (ZIV) [1] proposes a design that guarantees freedom from inclusion victims while retaining the advantages of an LLC design. This design completely isolates the core caches from LLC evictions. ZIV relaxes the constraint that an LLC victim must be chosen from the set pointed to by the set indexing function by efficiently and minimally enabling a global victim selection scheme in the inclusive LLC to avoid the generation of inclusion victims

Zero inclusion victim paper uses the cache hierarchy aware replacement (CHAR) [3] algorithm for inclusive LLCs to infer the likely death of a block in LLC when it is evicted from the private cache of a core. Cache hierarchy aware replacement is an algorithm that classifies the blocks residing in the L2 cache based on their reuse patterns and dynamically estimates the reuse probabilities to generate selective replacement hints to the LLC. CHAR classifies the evicting block from L2 into a few classes based on some properties and generate a dead hint if the reuse probability is less than a dynamically set threshold then send this dead hint to LLC. If the dead hint is received by the LLC then it is set as the block that has furthest reuse, in case of SRRIP replacement policy it is given maximum re-reference interval prediction (RRPV) value. In case of an CHAR exclusive design block will be bypassed if it is inferred dead and not dirty.

Chapter 2

Literature survey

Cache replacement policies attempt to emulate optimal replacement by predicting the re-reference interval of a cache block. The most common replacment policy is Least Recently Used (LRU) which always predicts a near immediate re-reference interval. The LRU performs badly when the application have a working set larger than the cache or have frequent bursts of references to non-temporal data. To improve performances of such workloads Re-Reference Interval Prediction (RRIP) [2] was proposed. RRIP policies require only 2-bit per cache block and easily integrate into existing LRU approximations

These replacement policies for the last level caches (LLCs) uses the access information available locally at the LLC. These are inherently sub-optimal because of lack of information about the activities in the inner levels of the hierarchy. To overcome this the Cache Hierachy Aware Replacement (CHAR) algorithm [3] was proposed which mine the L2 cache eviction stream and decide if a block evicted from the L2 cache should be made a victim candidate in the LLC based on the access pattern of the evicted block

The inclusive LLC design offers bandwidth optimization and simplification in implementing the cache coherence protocols. However, these inclusive LLCs are associated with curse of inclusion victims which can lead to severe performance degradation. So to counter this a new inclusive LLC design named Zero Inclusion Victim (ZIV) [1] was proposed which guarantees freedom from inclusion victims while retaining all advantages of an inclusive LL

Chapter 3

Proposed Idea

3.1 Zero Inclusion victim

The Last level cache (LLC) is said to be inclusive when the contents of all the private caches are the subset of the contents of shared LLC cache. To maintain the inclusion property LLC needs to implement the following two actions

1. A block fetched from the off-chip memory system on an LLC miss must be allocated in the LLC in addition to allocating it in the private caches of the requesting core
2. When a block is evicted from the LLC, all copies of the block resident in the private caches must be forcefully invalidated. The extra interconnect messages sent by the LLC to the private caches for this purpose are usually referred to as the back-invalidations. The block that gets invalidated is known as inclusion victim

Even though the inclusive design is coupled with drawbacks as mentioned in introduction it is still a popular design because it simplifies the cache coherence protocol. So the central contribution of the zero inclusion victim (ZIV) paper is to design an inclusive LLC design that is free from inclusion victims
The inclusion victims are generated at the time of LLC replacements due to the way set-associative caches are architected and managed. When filling a block B in the LLC, a replacement candidate must be picked from the LLC set B maps to. This restricts the choice of replacement candidates. Since the aggregate private cache capacity must be less than the total capacity of shared LLC capacity so there will be a block in LLC such that it is not present in any of the private cache so the proposal says to enable a global replacement policy in LLC such that it evicts the block such that it is not present in private cache to prevent inclusion victims. Such policy must be invoked only when the baseline replacement policy selects such a block that is a resident of private cache. The ZIV design leaves the coherence protocol and the basic architecture of the LLC unaltered.

The ZIV LLC design on a high level can be understood from the functinal flow shown in figure 3.1. An LLC fill to address A1 looks up the home LLC bank and the home sparse directory slice. A new directory entry E1 is allocated in the target sparse directory set. The LLC block with address A2 is replaced from the LLC set to make room for A1. Next, the sparse directory entry of A2 is looked up, as is done in the baseline, to determine if copies of the evicted block are resident in the private caches.

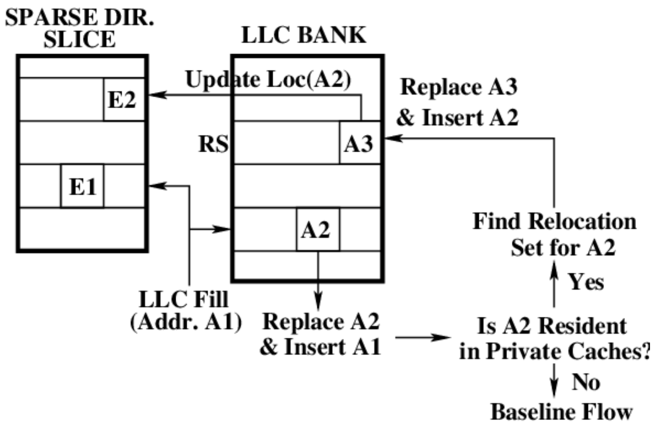

Figure 3.1: Overview of functional flow of ZIV LLC (Ref: [1])

If there is no copies of A2 in any of the private caches then nothing more needs to be done. If we found a copy of A2 in private cache then it will generate a back invalidation, to prevent this the ZIV design relocates A2 to a different LLC set. To do this we should find a relocation set that has atleast one block such that it is not present in the private cache or a block with invalid bit set, in this case A3 is such a block.The ZIV LLC leverages the sparse directory entry E2 of A2 to record the new location of A2. So the three main components of ZIV LLC design is finding a relocation set, relocating the block to relocation set and replacing a appropriate block from the relocation set.

Chapter 4

Conclusion

4.1 Completed work

- Configured the Champsim to manually select the L2 cache replacement policy
- Implemented and tested the Not in private cache property that is used for relocating sets and dead block detection for LLC in Champsim
- Implemented Likely dead block property by using the cache hierarchy aware replacement (CHAR) algorithm and also implemented the CHAR replacement
- Read the papers on RRIP, ZIV and CHAR. Gave presentations on cache hierarchy, re-reference interval prediction and cache hierarchy aware replacement

4.2 Future plans for next month

- Deriving and implementing a plan for finding the relocation set and maintaining the contents of sparse directory that stores the location of the relocated blocks
- Analyzing the results of already implemented properties by running large number of simulations with different cache configurations, applications and properties

4.3 End goal of the project

- The end goal of the project is to implement zero inclusion victim (ZIV) LLC design in champsim simulator along with all the analysis of the results and simulations

References

[1] M. Chaudhuri, "Zero inclusion victim: isolating core caches from inclusive last-level cache evictions," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 71–84, IEEE, 2021.

[2] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *ACM SIGARCH computer architecture news*, vol. 38, no. 3, pp. 60–71, 2010.

[3] M. Chaudhuri, J. Gaur, N. Bashyam, S. Subramoney, and J. Nuzman, "Introducing hierarchy-awareness in replacement and bypass algorithms for last-level caches," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 293–304, 2012.