ZIV LLC Design

Summer Project Stage 1 Report

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

XX.

Contents

List of Figures

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 are 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 the one that makes its replacement decisions based on perfect knowledge of 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 pick 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 [1] was proposed.

The most widely used last level cache (LLC) architecture in the microprocessors has been the inclusive LLC design because it offers bandwidth optimization and simplification to cache coherence protocols. One of the major problem that is 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. Second, 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 is managed. To overcome this Zero inclusion victim (ZIV) [2] proposes an 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.

Zero inclusion victim paper uses the cache hierarchy aware replacement (CHAR) [3] proposal to infer likely death of a block in from LLC's viewpoint when it is evicted from the private cache of a core when allocating a relocation set. Cache hierarchy aware replacement is an algorithm that classifies the 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 few classes based on some properties and generate dead hints if the reuse probability is less than a dynamically set threshold and send that dead hint to LLC then based on the replacement policy for example SRRIP the block is given a RRPV value of $RRPV_{max}$-1. In case of an exclusive design block will be bypassed if it is inferred dead and not dirty.

1.1 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
- Read the papers on RRIP, ZIV and CHAR. Gave presentations on cache hierarchy, re-reference interval prediction and cache hierarchy aware replacement

1.2 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

1.3 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

Chapter 2

Literature survey

2.1 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 this zero inclusion victim paper is to design an inclusive LLC design that is free from inclusion victims

2.1.1 Proposal

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 2.1. 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.

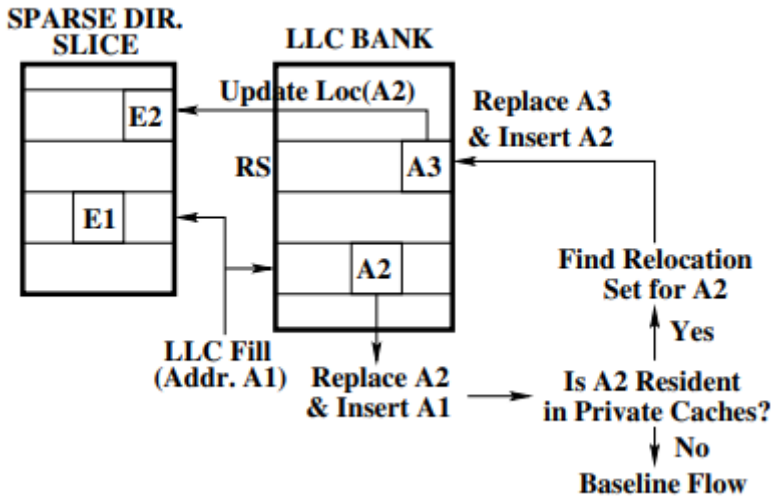

Figure 2.1: Overview of functional flow of ZIV LLC

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, 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.

2.1.2 Key results

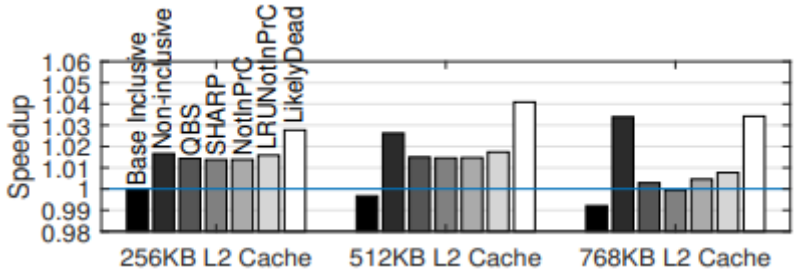

Figure 2.2: Performance of multi-programmed workloads with LRU as the baseline

The above results 2.2 are normalized to the configuration having 256KB L2 cache and an inclusive LLC with LRU policy. . With a 256 KB L2 cache, QBS and SHARP deliver performance close to the non-inclusive LLC, but fail to scale their performance up as the L2 cache capacity increases because with increasing L2 cache size the number of blocks that are not residing in the private cache drops rapidly as a result it becomes increasingly important to select the appropriate LLC victims from this small set of LLC blocks that are not resident in the private caches. The QBS and SHARP donot offer any guarantee of freedom from inclusion victim, while the ZIV LLC design implementing the NotInPrC and LRUNotInPrC properties guarantee freedom from inclusion victims. The ZIV LLC design implementing the LikelyDead property performs the best across the board

References

[1] 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.

[2] 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.

[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.