CS204 Project

Optimizing Cache Performance: Strategies for Minimizing Conflict Misses

Tejas Wagh 2022csb1144

Divyansh Verma 2022csb1081

Priyanshu 2022csb1107

Problem Statement:

Given a fixed configuration of last level cache (2MB, 16 ways, 64 byte block size) and for input traces, find a suitable remapping of address to line such that conflict misses could be reduced. A reduction in conflict misses will improve hit rate, therefore, AMAT, and therefore will improve IPC and performance

INTRODUCTION

Here In this project we propose a solution to non uniform access pattern in set accesses which result in more conflict misses where our code proves to be able to increase the hits by transferring the data from hot set to Cold set

Algorithm Outline:

Algorithm Description For Latest Version:

Set Classification: In our approach, cache sets are classified as "hot" or "cold" based on their eviction counts. A "hot" set has a higher likelihood of eviction due to frequent accesses, whereas a "cold" set experiences fewer accesses and evictions.

Maintaining a Sorted Map: We maintain an ordered map of all cache sets sorted by their eviction counts. This map helps identify the coldest sets, which are less frequently accessed and thus have lower eviction counts. The map is dynamically updated as eviction counts change due to cache operations.

Linking Set: Each set can form up to two links with other sets, but only with different sets to prevent circular references. When a set is linked to another, its eviction count is adjusted to reflect this new connection, potentially changing its classification from cold to hot or vice versa.

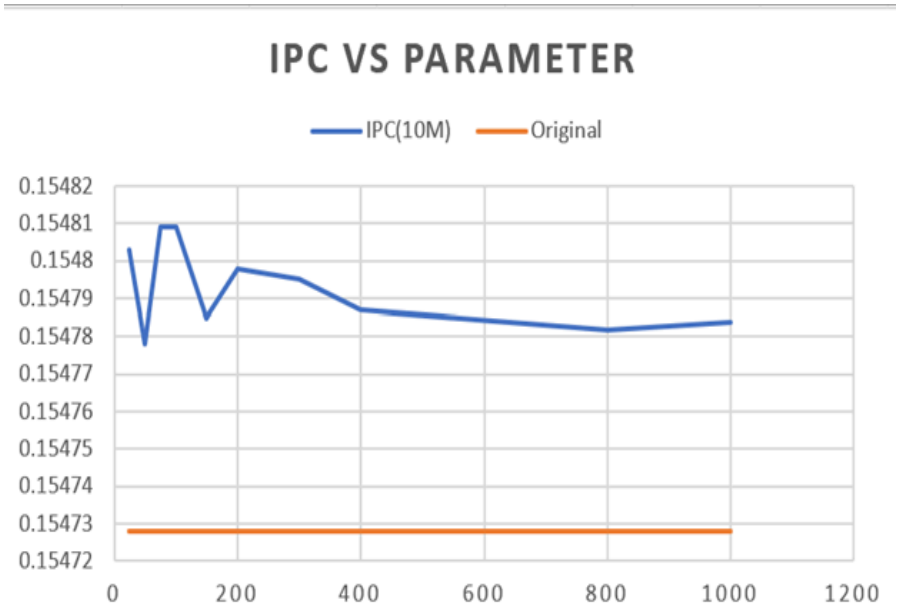
Set Link Management

Eviction Count Impact: Linking a set to another influences the eviction counts, as interactions between sets can lead to changes in their access patterns. This dynamic adjustment is crucial for managing the cache more effectively.

Connection Refresh Mechanism: When a set is accessed and is not within the top L coldest sets (e.g., top 100), its links are reassessed. This mechanism ensures that connections are always between sets that are less likely to compete for cache space, reducing conflict misses. If a set falls out of this category, it is disconnected from its current links, and new links are formed with sets within the top 100 coldest list.

Renew_set Function: The `Renew_set` function is implemented to flush data from the previously linked set and refresh the connections. This function plays a pivotal role in maintaining the relevance of links by ensuring they are always between the coldest available sets.

Results(for 605 Trace File on 10M)



Here we are showing the statistics of variation of IPC vs the Parameter we have selected i.e the top L coldest sets varying Value of L we got the max value for around the L=100.

Performance Metrics: The implemented algorithm was tested under the given 4 traces for cache workloads, comparing the number of conflict misses against a standard caching algorithm without set redirecting. The results showed a significant reduction in conflict misses, particularly in scenarios with repetitive access patterns to a few hot sets.

For the Given trace files we have calculated the Overhead value and are as follows:

	Overhead
605.mcf_s-1536B.champsimtrace.xz	*15*
444.namd-120B.champsimtrace.xz	*11*
445.gobmk-17B.champsimtrace.xz	*11*
473.astar-153B.champsimtrace.xz	*9*

Algorithm Effectiveness: The reduction in conflict misses suggests that dynamically managing set connections based on set "temperature" (hot or cold) and adjusting links based on eviction counts can substantially enhance cache utilization and efficiency.

Scalability and Limitations: While the results are promising, the scalability of this algorithm across different cache architectures and sizes remains to be rigorously tested. Additionally, the computational overhead introduced by managing the sorted list and updating links could offset some of the performance gains in high-frequency access scenarios.

Initial Stages of Progress

Stage 1: Implementation of Static Naïve Mapping

In the initial stage of our exploration into cache management enhancements, we implemented a straightforward static redirecting algorithm. This method was designed to manage access conflicts by redirecting traffic to an adjacent cache set under specific conditions.

Description of Static Naïve Mapping Strategy:

The core principle of the static naïve mapping strategy is to provide a simple and direct approach to handle cases where the initially targeted cache set is full. The process implemented is as follows:

Initial Cache Set Full:

Upon an access request, if the primary cache set targeted (e.g., Set 1) is determined to be full, the system then evaluates the next sequential set (Set 2) for available space.

Redirection Decision:

If Set 2 has available space, the incoming access is redirected to this set. This redirection aims to alleviate the load on the initially targeted set, potentially reducing the occurrence of conflict misses.

Fallback to Original Set:

In cases where Set 2 is also full, the access remains directed at the original set (Set 1). At this point, the existing replacement policy of the cache is invoked to manage the space within the set.

This stage served as a foundational approach, setting the groundwork for more complex strategies by introducing the concept of set redirection to manage cache fill states. The simplicity of the static naïve mapping provided valuable insights into the behavior of cache accesses under a slightly modified set allocation protocol, laying the basis for subsequent enhancements in our cache management strategies.

Stage 2: Enhancing Static Mapping with Expanded Set Selection Options

In this phase of our strategy development, we extended the static mapping approach by increasing the number of alternative sets considered for redirection when handling cache set

conflicts. This adjustment aimed to provide greater flexibility and reduce the likelihood of conflict misses by offering additional redirection options.

Enhanced Redirection Strategy:

When a cache access request encounters a full set, the system now evaluates additional sets sequentially to find available space. Specifically, the following procedure is implemented:

Initial Set Full:
Upon determining that the initially targeted set (e.g., Set 1) is full, the system proceeds to check the next sequential set (Set 2) for available space.

Sequential Checking:
If Set 2 is also found to be full, the system then checks the subsequent set (Set 3). If no space is available in Set 3, the check continues to Set 4.

Final Decision:
If all examined sets (Sets 2, 3, and 4) are full, the access is directed back to the original set (Set 1), where the standard replacement policy is applied to manage the space conflict.

This enhanced strategy of checking three additional sets allows for a more flexible response to full set situations, potentially dispersing load more effectively across the cache and reducing the frequency of conflict misses. If no alternative sets with available space are found, the system defaults to the original set, relying on established replacement policies to handle the conflict. This approach optimizes cache utilization while maintaining system performance.

Conclusion

This project demonstrates a novel approach to reducing conflict misses through dynamic set linking based on eviction counts. The ability to dynamically re-link sets according to their temperature significantly enhances cache performance and could be beneficial in systems where cache efficiency is critical.

How to run the program

1> Clone the repository:
git clone
:https://github.com/bakasingh/Optimizing-Cache-Performance-Strategies-for-Minimizing-Conflict-Misses

2> Open in UNIX based system and navigate to the directory containing the codebase.

3>For granting permissions, run for both build_champsim.sh and run_champsim.sh

chmod +x script.sh example : *chmod +x build_champsim.sh*
sed -i -e 's/\r$/\n/' script.sh example: *sed -i -e 's/\r$/\n/' build_champsim.sh*

4>Build the ChampSim simulator:
./build_champsim.sh bimodal no no no no srrip 1

5> Run the ChampSim simulator with the desired configuration:
./run_champsim.sh bimodal-no-no-no-no-srrip-1core 1 10 <trace_file_name>

Replace <*trace_file_name*> with the name of the trace file you want to simulate.

You can also replace *10* by how many instructions in Million you want to run the code for.

6> You can check the new file made in the same folder by name of trace and number of instructions.

Trances_Link:
https://drive.google.com/drive/folders/1lxbyLJ30uXWaxK7uy_84BHyr5WjlC8Qu

Bibliography

-> GitHub repository ***GitHub***

-> Computer Organization and Embedded Systems by Carl Hamacher

->Research Paper - *https://www.cse.iitk.ac.in/users/mainakc/manuscripts/hotset.pdf*