Project 4 Report: Signature-based Hit Predictor on AI Workloads

Tai Duc Nguyen
ECEC 412: Modern Processor Design

November 25, 2019

Abstract

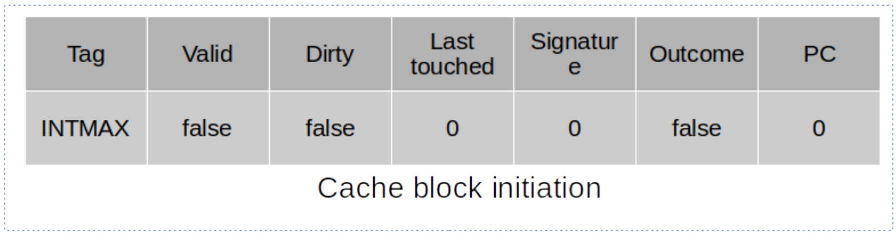
Previously in project 3, different cache replacement policies: Least Recently Used (LRU) and Least Frequently Used (LFU) have been evaluated against AI Workloads. Continue that work, this project takes a look at the paper "SHiP: Signature-based Hit Predictor for High Performance Caching" by Wu et al. by evaluating their cache's architecture against the same 3 workloads as the last project.

1 Introduction

In order to achieve the objective of evaluating the performance of different cache replacement policies, a simulation was written in C by Shihao Song and Adarsha Balaji (available here: `https://github.com/Shihao-Song/Computer-Architecture-Teaching/tree/master/C621/Cache_Policy`). Using the framework established, a mechanism to test all pre-defined hardware configurations at once is written on top. After automating the testing of different configs, the LFU replacement policy is implemented in a similar fashion to the already existed code for the LRU replacement policy. And finally, the algorithm for SHiP cache predictor is written into the code for LRU and LFU.

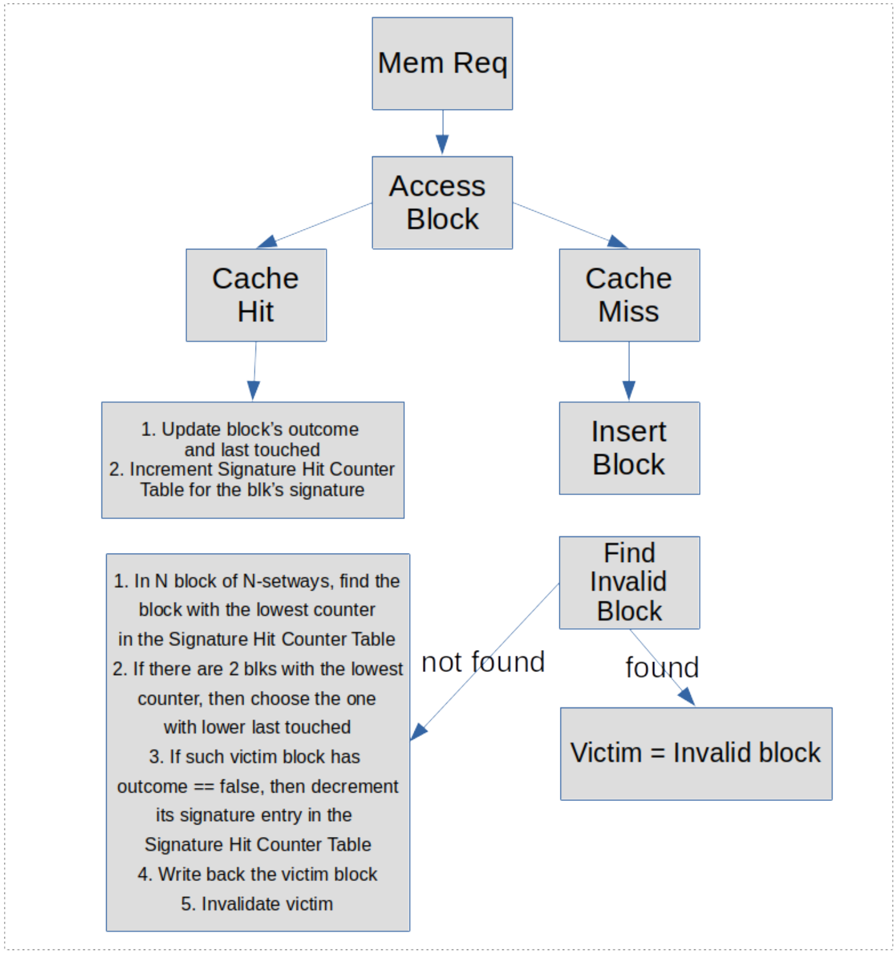
2 Simulation and Results

Using the same setup and simulation tool as Project 3, the algorithm for SHiP cache predictor is written on top. The pseudocode and its diagram is shown in Figure 1 below.



Tag	Valid	Dirty	Last touched	Signatur e	Outcome	PC
INTMAX	false	false	0	0	false	0

Cache block initiation

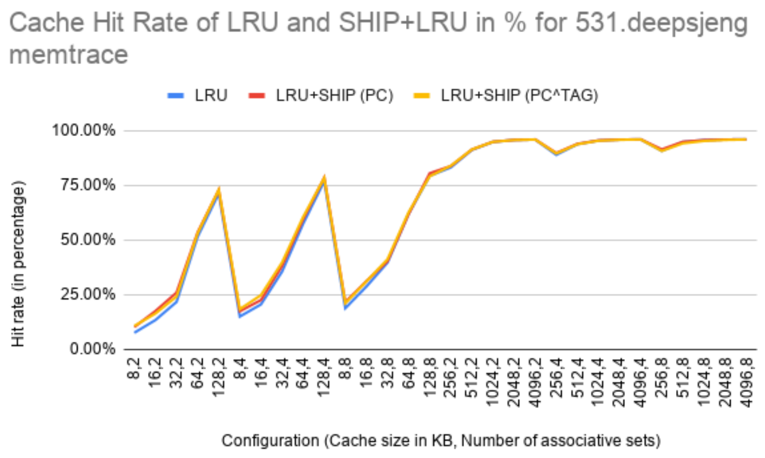


1. Update block's outcome and last touched
2. Increment Signature Hit Counter Table for the blk's signature

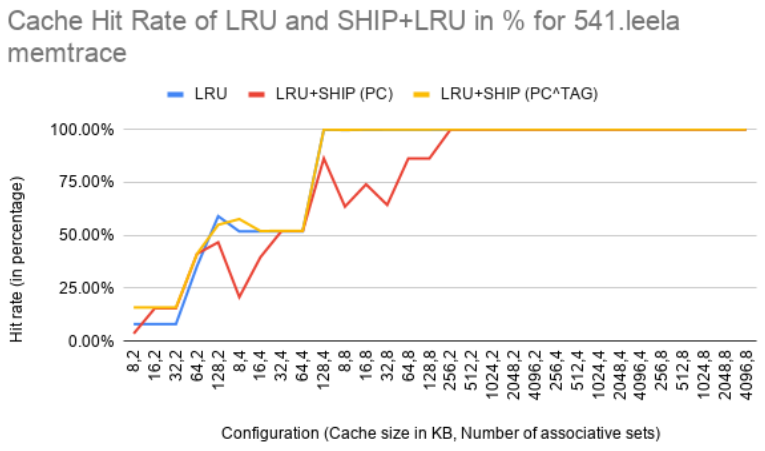
1. In N block of N-setways, find the block with the lowest counter in the Signature Hit Counter Table
2. If there are 2 blks with the lowest counter, then choose the one with lower last touched
3. If such victim block has outcome == false, then decrement its signature entry in the Signature Hit Counter Table
4. Write back the victim block
5. Invalidate victim

Figure 1: Sample output of the simulation

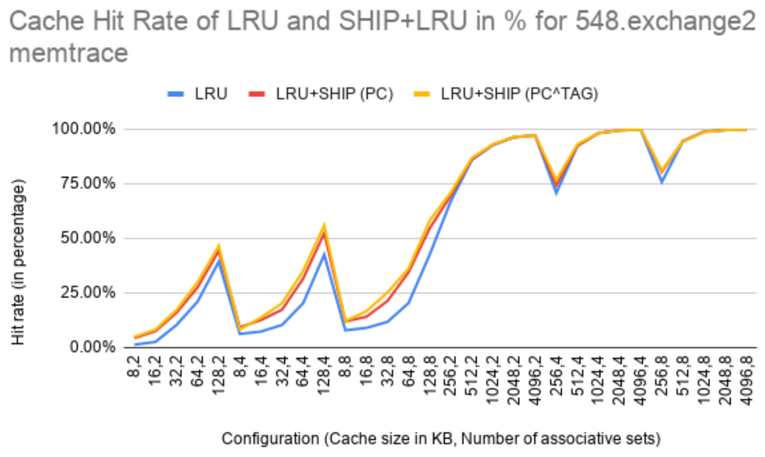
Performing this simulation over all 3 memory traces (each trace includes many Load and Store instructions), the results are shown in graphs (Figure 4a, 4b, and 4c).



(a) Results for 531.deepsjeng memory trace



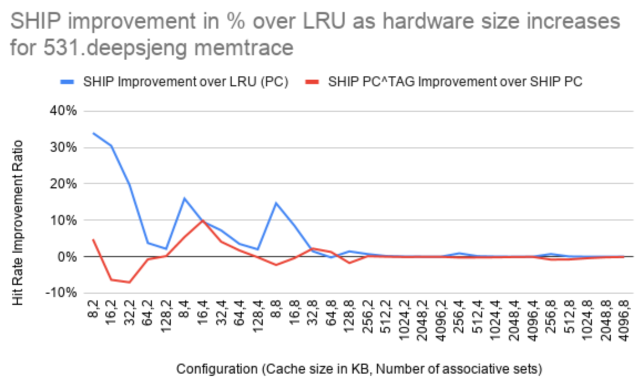
(b) Results for 541.leela memory trace



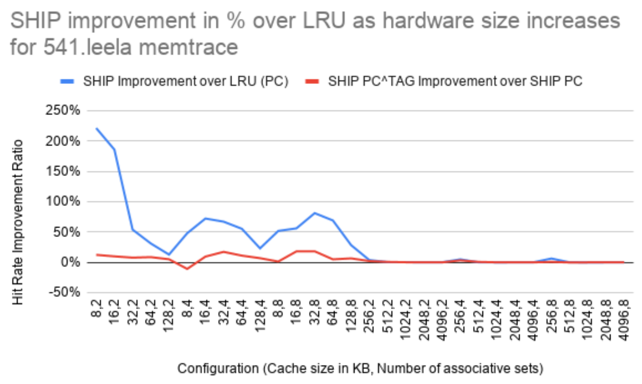
(c) Results for 548.exchange2 memory trace

Figure 2: Simulation results. The x-axis have different hardware configurations with the first number being the cache size in KB and the second number being the number of associative sets. The y-axis is the cache hit rate in percentage.

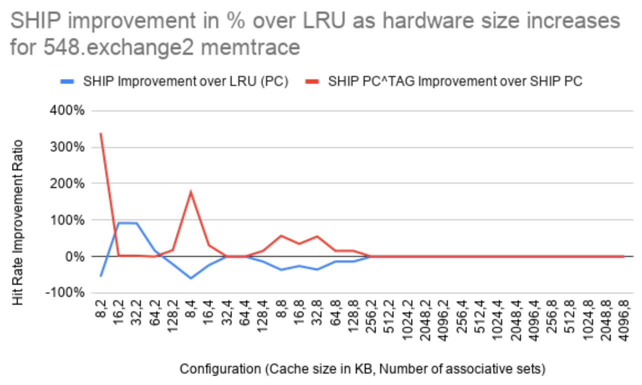
To better see the differences in improvement of the SHiP cache predictor, Figure 3a, 3b, and 3c take the improvement ratios of SHiP+LRU using only the Program Counter (PC) to index the Signature Hit Counter Table (SHCT) over only LRU; and of SHiP+LRU using both PC and Memory Address over only using PC for SHCT indexing.



(a) Improvement ratios for 531.deepsjeng memory trace



(b) Improvement ratios for 541.leela memory trace



(c) Improvement ratios for 548.exchange2 memory trace

Figure 3: Simulation results. The x-axis have different hardware configurations with the first number being the cache size in KB and the second number being the number of associative sets. The y-axis is the improvement ratio in percentage

From the 3 figures above, SHiP+LRU improves cache hit rate by a very large margin at low cache sizes (¡ 128KB) and low number of associative cache stores (¡ 4). However, when the cache size is greater than

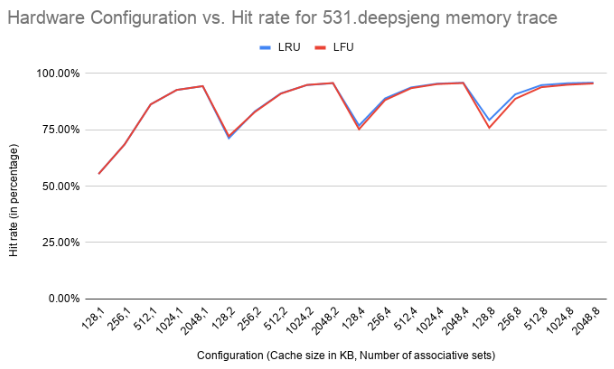
128KB, with more than 2 associative cache sets, zero to very little improvement observed. In addition, the (PC XOR TAG) hashing algorithm in general does better than only using PC. This is probably due to the fact that the memtrace have many of instances where the same PC is requesting different memory locations.

The data for LFU is not shown in this experiment because it is determined from the last experiment that LFU does not provide a large cache hit rate improvement over LRU (best case: 4%). The figures and data for LRU vs. LFU cache hit rate is shown in the Appendix Section.

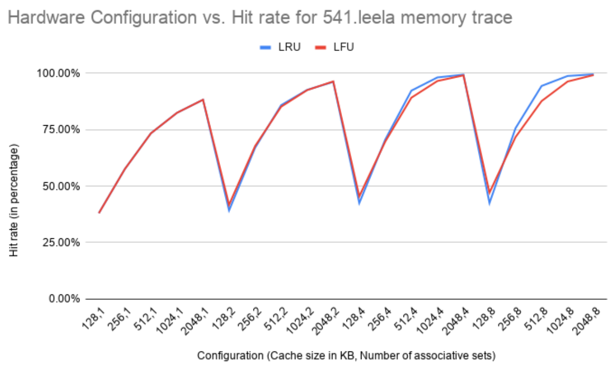
3 Discussion

SHiP cache prediction algorithm provides a way to better control the replacement of cache blocks, hence, allow useful blocks to retain in the cache more often and useless blocks to be removed from the cache more accurately. It is concluded from this experiment that: with the 3 AI Workloads tested, SHiP improves cache hit rate substantially when the cache size is smaller than 128KB or the number of associative cache stores is less than 4. At larger cache size or higher number of stores, SHiP increases the hit rate very slightly.

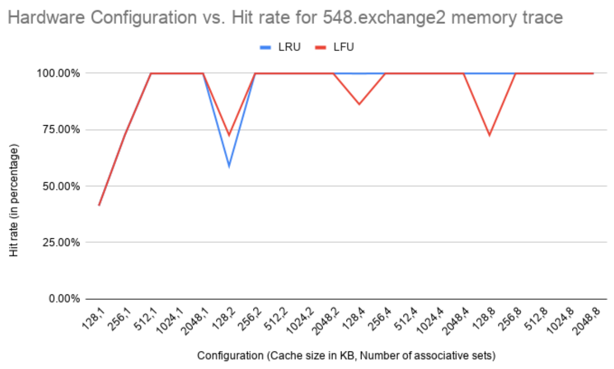
4 Appendix



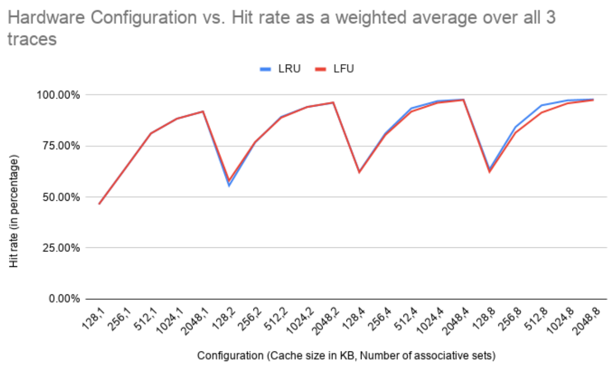
(a) Results for 531.deepsjeng memory trace



(b) Results for 541.leela memory trace



(c) Results for 548.exchange2 memory trace



(d) Overall results as weighted averages of 3 traces

Figure 4: Simulation results for LRU vs. LFU. The x-axis have different hardware configurations with the first number being the cache size in KB and the second number being the number of associative sets. The y-axis is the cache hit rate in percentage.