Maxwell Jung 706531056
June 1, 2025

UCLA ECE 209AS Advanced and Secure Computer Architecture CA3 Report

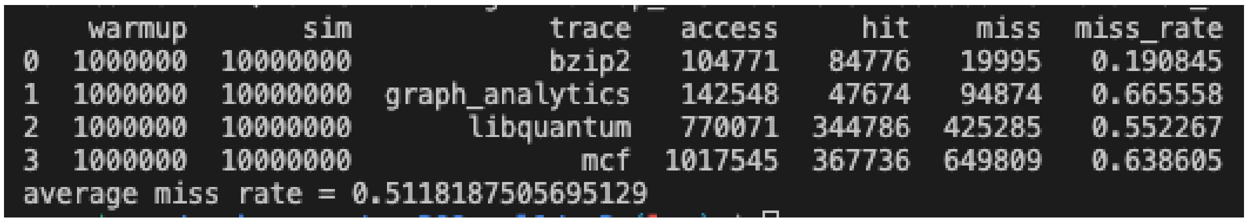
Objective

The goal of this Computer Assignment was to implement a cache replacement policy that achieves a lower average miss rate than LRU. The replacement policy I chose is directly based on **Hawkeye** replacement policy, originally proposed in the paper *Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement*. The algorithm I implemented oversimplifies and skips some portions of the original Hawkeye algorithm in the interest of time and ease of implementation. Thus, I call my algorithm "**Hawkeye?**". Hawkeye? achieves an average miss rate of 0.5068 compared to LRU that achieves 0.5118.

Results

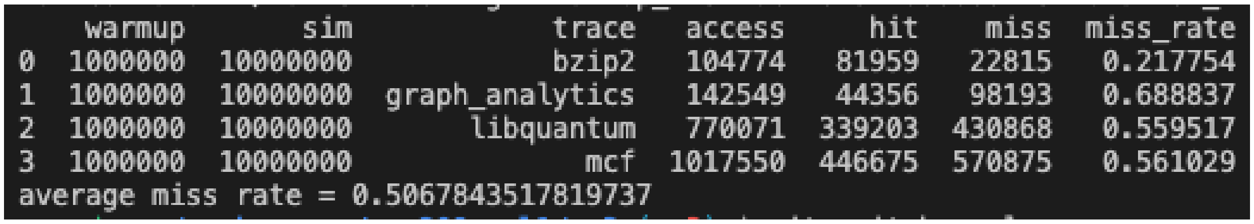
Below screenshots compare benchmark results of LRU and Hawkeye. My implementation of Hawkeye performs marginally worse for the first three benchmarks (bzip2, graph_analytics, libquantum), but performs significantly better on the last mcf benchmark. As a result, the average miss rate across the four benchmarks is slightly lower than LRU by .

LRU results (baseline)



```
    warmup       sim            trace   access     hit    miss  miss_rate
0  1000000  10000000            bzip2   104771   84776   19995   0.190845
1  1000000  10000000  graph_analytics   142548   47674   94874   0.665558
2  1000000  10000000       libquantum   770071  344786  425285   0.552267
3  1000000  10000000              mcf  1017545  367736  649809   0.638605
average miss rate = 0.5118187505695129
```

Hawkeye? results



```
    warmup       sim            trace   access     hit    miss  miss_rate
0  1000000  10000000            bzip2   104774   81959   22815   0.217754
1  1000000  10000000  graph_analytics   142549   44356   98193   0.688837
2  1000000  10000000       libquantum   770071  339203  430868   0.559517
3  1000000  10000000              mcf  1017550  446675  570875   0.561029
average miss rate = 0.5067843517819737
```

How Hawkeye? works

Below diagram shows the components of Hawkeye replacement algorithm. Hawkeye Predictor (middle component) is a binary classifier that determines if the memory instruction (PC) is either cache-friendly or cache-averse. OPTgen (left component) trains Hawkeye Predictor by checking if an instruction would have been cache hit and miss under optimal policy. Hawkeye Predictor is implemented as a table indexed by 16-bit hashed PC where each entry of the table is an 8-bit unsigned integer representing cache friendliness. If the first bit is 1 (value >= 128), the instruction is considered cache-friendly, otherwise the instruction is considered cache-averse.

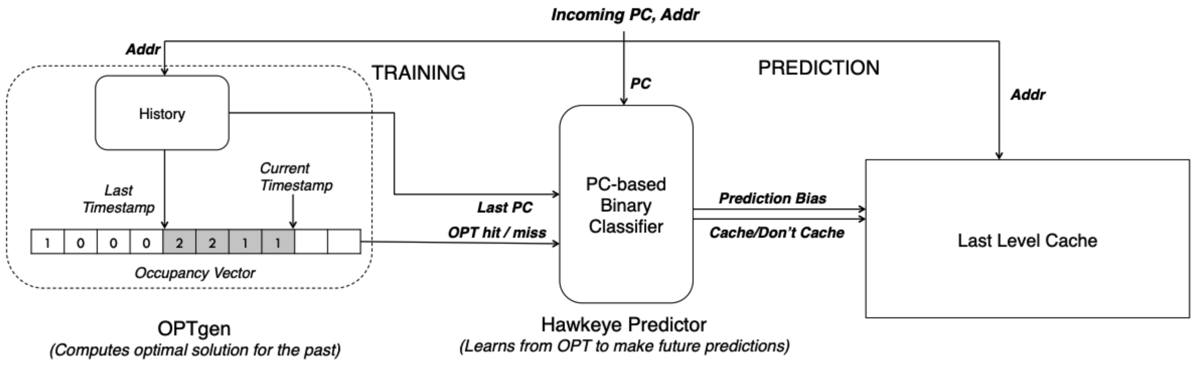


Figure 4: Block diagram of the Hawkeye replacement algorithm.

Insertion/update Policy

Hawkeye? insertion policy is similar to Bimodal Insertion Policy (BIP) in that the new cache is either inserted at the MRU or LRU position. However, unlike BIP where the new position is determined randomly, Hawkeye will insert at MRU if the instruction corresponding to the new cache line is considered “cache-friendly” or insert at LRU if the instruction is considered “cache-averse”. For updating the age of other cache lines, they are incremented if and only if the new cache line is cache-friendly. Below table summarizes the insertion and update policy. This update policy guarantees that the cache line with the maximum possible age (i.e. RRIP of 7) is always cache-averse and evicted under LRU-like replacement policies.

Hit or Miss / Hawkeye Prediction	Cache Hit	Cache Miss
Cache-averse	RRIP = 7	RRIP = 7
Cache-friendly	RRIP = 0	RRIP = 0; Age all lines: if (RRIP < 6) RRIP++;

Table 1: Hawkeye’s Update Policy.

Replacement Policy

Replacement policy is nearly identical to LRU where the oldest cache line is evicted. Any cache line with the maximum possible age (i.e. RRIP of 7) is guaranteed to be cache-averse with the above update policy, so they are evicted as in LRU replacement policy. If no cache line has the maximum possible age, that means all cache lines are somewhat cache-friendly, so the Hawkeye predictor detrains the instruction corresponding to the evicted cache line to be more cache-averse.

Hawkeye Predictor/OPTGen

To determine if the instruction is “cache-friendly” or “cache-averse”, Hawkeye approximates the behavior of an optimal replacement policy (aka Belady’s Algorithm) on past memory access patterns using a structure called “OPTGen”. Belady’s algorithm is optimal in that it looks into the future memory access patterns and replaces the cache line that will be used latest in the future. However this requires knowledge of the future which is impossible. OPTGen works by storing a history of memory access patterns and running optimal policy starting from the oldest memory access. The size of this history is 8 times the cache associativity, which effectively simulates each memory access seeing up to [8*associativity] future memory accesses from the perspective of past memory accesses. The exact mechanism of OPTGen can be found in the original Hawkeye paper.

If OPTGen determines that the instruction would have been a cache hit under optimal policy, the Hawkeye Predictor increments the cache friendliness corresponding to that instruction. Otherwise, the cache friendliness is decremented. OPTGen can also be configured to allow cache bypassing to minimize dead-on-arrival cache lines where the cache line is used only once and never reused in the future.

Storage Overhead

Each OPTGen entry stores the memory address (64 bits), PC (64 bits), and occupancy value (4 bits) totaling to 132 bits/entry. The history length of each OPTGen is 8 x associativity (16 way) = 128 entries/OPTGen. Each cache set has a unique OPTGen entry (1 OPTGen/1 set). Therefore, the OPTGen structure for 16 way 2048 set cache requires 2048 x 128 x 132 = 34603008 bits.

The predictor has 2^16 entries where each entry is 8 bits, so the predictor requires 2^16 x 8 = 524288 bits.

Each LRU entry requires a timestamp (4 bits) and PC (64 bits) to track the age and instruction corresponding to the cache line. For a 16 way 2048 set cache, there are a total of 16 x 2048 = 32768 LRU entries, so 32768 x 68 = 2228224 bits are required.

The total bits required for Hawkeye? is 34603008 (OptGen) + 524288 (Predictor) + 2228224 (LRU) = 4669440 bits = 4.453125 MiB.