Cache Replacement Championship

The 3P and 4P cache replacement policies

Pierre Michaud
INRIA

June 20, 2010
Optimal replacement?

- **Offline** (we know the future) $\rightarrow$ Belady

- **Online** (we don’t know the future) $\rightarrow$ problem without a solution
  - On random address sequences, all the online replacement policies perform equally on average

The best online replacement policy does not exist
In practice…

- We search a policy that performs well on as many applications as possible

- We hope that our benchmarks are representative

- But there is no guarantee that a replacement policy will always perform well
The DIP replacement policy

• Qureshi et al., ISCA 2007

• Key idea #1: bimodal insertion (BIP)
  – LRU behaves badly on cyclic accesses → try to correct this
  – On a miss, insert block in MRU position only with probability $E=1/32$, otherwise leave it in LRU position

• Key idea #2: set sampling
  – 32 LRU sets, 32 BIP sets, use best policy in the other sets

• Beauty of DIP: just one counter!
Proposed policy

• Incrementally derived from DIP
  – Start from a carefully tuned DIP

• Based on CLOCK instead of LRU
  – needs less storage than LRU

• Combines more than 2 different insertion policies
  – (new ?) method for multi-policy selection
Carefully tuned DIP

- Cache levels use unique line size? ➔ OK
  - Otherwise a (small) filter would have been needed

- Don’t update replacement info on writes
  - The fact that a block is evicted from a cache level does not mean that the block is likely to be accessed soon
    - If it is the case, it is chance, not a manifestation of temporal locality

- 28 SPEC 2006, CRC simulator, 16-way 1M L3

- Speedup DIP / LRU ➔ avg: +2% ; max: +20% ; min: -4%
CLOCK DIP

• CLOCK policy
  – one use bit per block, one clock hand per cache set
    • 16-way cache → 16+4 = 20 bits per set
  – On access to a block (hit or insertion), set the use bit
  – On a miss,
    • hand points to potential victim
    • If use bit is set, reset it and increment the hand (mod 16), repeat till victim is found

• CLOCK BIP
  – On insertion, set the use bit with probability E=1/32

• CLOCK DIP / DIP → avg: +0.2% ; max: +1.2% ; min: -0.5%
Multi-policy selection mechanism

• DIP uses a single PSEL counter
  – Miss in LRU-dedicated set $\Rightarrow$ decrement PSEL
  – Miss in BIP-dedicated set $\Rightarrow$ increment PSEL

• Generalization: $N$ policies, $N$ counters $P_1, \ldots, P_N$
  – Miss in set dedicated to policy $j$ $\Rightarrow$ add $N-1$ to $P_j$, subtract 1 to all the other counters
  – Keep $P_1 + P_2 + \ldots + P_N = 0$ $\Rightarrow$ if a counter saturates, all counters stay unchanged
  – Best policy is the one with the smallest counter value
The 3P policy

• For a few benchmarks, neither LRU nor BIP perform well
  – For example, 473.astar exhibits access patterns that are approximately cyclic, but drifting relatively quickly

• We found that, on a few benchmarks, BIP with E=1/2 can outperform both LRU and BIP with E=1/32 → 3 policies
  – All policies use the same hardware

• For E=1/2, it is possible to improve MLP
  – Instead of setting the use bit every other insertions, set the use bit for 64 consecutive insertions every 128 misses

• 3P / CLOCK DIP: avg: +0.5% ; max: +5.7% ; min: -2.1%
Shared-cache replacement

- Thread-unaware policies like DIP or 3P may be unfair
  - OK when threads have equal force, i.e., equal miss rates (in misses per cycle)
  - But **fragile** threads (low miss rate) are penalized when they share the cache with **aggressive** threads (high miss rate)

- BIP is good for containing aggressive threads

- **Thread-aware bimodal insertion (TABIP)**: use normal insertion for fragile threads and bimodal insertion for aggressive threads
TABIP: identifying fragile threads

- Heuristic

- One TMISS counter per running thread

- Update TMISS counters the same way as policy-selection counters
  - E.g., 4 running threads
  - Thread k miss → add 3 to TMISS [k], subtract 1 to TMISS of the other threads (keep sum of TMISS [i] null)

- Define fragile threads as threads whose TMISS is negative
The 4P policy

- $4P = 3P + \text{CLOCK TABIP}$
  - Use 4 policy-selection counters instead of 3

- 28 SPEC 2006, CRC simulator, 16-way 4MB L3

- 100 fixed random 4-thread mixes $\Rightarrow$ perf for an app = arithmetic mean of CPIs for that app among the 400 CPIs

- Speedup $4P / \text{LRU}$: avg: +3% ; max: +18% ; min: -4.5%
- Speedup $4P / 3P$: avg: +1% ; max: +7% ; min: -3%
- 4P is fairer than 3P
Questions ?