MAXIMIZING CACHE PERFORMANCE UNDER UNCERTAINTY

Nathan Beckmann
CMU

Daniel Sanchez
MIT

HPCA-23 in Austin TX, February 2017
The problem

• Caches are a critical for overall system performance
  • DRAM access = ~1000x instruction time & energy

• Cache space is scarce

• With perfect information (ie, of future accesses), a simple metric is optimal
  • Belady’s MIN: Evict candidate with largest time until next reference

• In practice, policies must cope with uncertainty, never knowing when candidates will next be referenced
WHAT’S THE RIGHT REPLACEMENT METRIC UNDER UNCERTAINTY?
PRIOR WORK HAS TRIED MANY APPROACHES

**Practice**
- Traditional: LRU, LFU, random
- Statistical cost functions [Takagi ICS’04]
- Bypassing [Qureshi ISCA’07]
- Likelihood of reuse [Khan MICRO’10]
- Reuse interval prediction [Jaleel ISCA’10] [Wu MICRO’11]
- Protect lines from eviction [Duong MICRO’12]
- Data mining [Jimenez MICRO’13]
- Emulating MIN [Jain ISCA’16]

**Theory**
- MIN—optimal! [Belady, IBM’66][Mattson, IBM’70]
  - But needs perfect future information
- LFU—Independent reference model [Aho, J. ACM’71]
  - But assumes reference probabilities are static
- Modeling many other reference patterns [Garetto’16, Beckmann HPCA’16, …]

Without a foundation in theory, are any “doing the right thing”?```
GOAL: A PRACTICAL REPLACEMENT METRIC WITH FOUNDATION IN THEORY
Fundamental challenges

- **Goal:** Maximize cache hit rate

- **Constraint:** Limited cache space

- **Uncertainty:** In practice, don’t know what is accessed when
Key quantities

- **Age** is how long since a line was referenced
- Divide cache space into *lifetimes* at hit/eviction boundaries
- Use *probability* to describe distribution of *lifetime* and *hit age*
  - $P[L = a] \rightarrow$ probability a randomly chosen access lives $a$ accesses in the cache
  - $P[H = a] \rightarrow$ probability a randomly chosen access hits at age $a$
Fundamental challenges

- **Goal:** Maximize cache hit rate

\[ P[\text{hit}] = \sum_{a=1}^{\infty} P[H = a] \]

Every hit occurs at some age < \( \infty \)

- **Constraint:** Limited cache space

\[ S = E[L] = \sum_{a=1}^{\infty} a \times P[L = a] \]

Little’s Law

**Observations:**
Hits beneficial irrespective of age
Cost (in space) increases in proportion to age
Insights & Intuition

- Replacement metric must balance benefits and cost

**Observations:**
- Hits beneficial irrespective of age
- Cost (in space) increases in proportion to age

**Conclusion:**
- Replacement metric $\propto$ hit probability
- Replacement metric $\propto$ −expected lifetime
Simpler ideas don’t work

• MIN evicts the candidate with largest time until next reference
• Common generalization \(\Rightarrow\) largest predicted time until next reference
Simpler ideas don’t work

- MIN evicts the candidate with largest time until next reference
- Common generalization $\Rightarrow$ largest predicted time until next reference

Q: Would you rather have A or B?

We would rather have A, because we can gamble that it will hit in 1 access and evict it otherwise

...But A’s expected time until next reference is larger than B’s.
THE KEY IDEA: REPLACEMENT BY ECONOMIC VALUE ADDED
Our metric: Economic value added (EVA)

• EVA reconciles hit probability and expected lifetime by measuring time in cache as forgone hits

• Thought experiment: how long does a hit need to take before it isn’t worth it?

• Answer: As long as it would take to net another hit from elsewhere.
  • On average, each access yields hits = \frac{\text{Hit rate}}{\text{Cache size}}
  • \Rightarrow \text{Time spent in the cache costs this many forgone hits}

\[
\text{EVA} = \text{Candidate's expected hits} - \frac{\text{Hit rate}}{\text{Cache size}} \times \text{Candidate's expected time}
\]
Our metric: Economic value added (EVA)

- EVA reconciles hit probability and expected lifetime by measuring time in cache as forgone hits

\[
EVA = \text{Candidate's expected hits} - \frac{\text{Hit rate}}{\text{Cache size}} \times \text{Candidate's expected time}
\]

- EVA measures how many hits a candidate nets vs. the average candidate
- EVA is essentially a cost-benefit analysis: is this candidate worth keeping around?
- Replacement policy evicts candidate with lowest EVA

Efficient implementation!
Estimate EVA using informative features

- EVA uses **conditional probability**

- Condition upon informative features, e.g.,
  - **Recency**: how long since this candidate was referenced? (candidate’s age)
  - **Frequency**: how often is this candidate referenced?

- Many other possibilities: requesting PC, thread id, ...
Estimating EVA from recent accesses

• Compute EVA using conditional probability

• A candidate of age $a$ by definition hasn’t hit or evicted at ages $\leq a$

• $\Rightarrow$ Can only hit at ages $> a$ and lifetime must be $> a$

• Hit probability $= P[\text{hit} \mid \text{age } a] = \frac{\sum_{x=a}^{\infty} P[H=a]}{\sum_{x=a}^{\infty} P[L=x]}$

• Expected remaining lifetime $= E[L - a \mid \text{age } a] = \frac{\sum_{x=a}^{\infty} (x-a) P[L=a]}{\sum_{x=a}^{\infty} P[L=x]}$
EVA by example

- Program scans alternating over two arrays: ‘big’ and ‘small’

Best policy:
Cache small array + as much of big array as fits
EVA by example

- Program scans alternating over two arrays: ‘big’ and ‘small’
At age zero, the replacement policy has learned nothing about the candidate. Therefore, its EVA is zero – i.e., no difference from the average candidate.
EVA policy on example (2/4)

Until size of small array, EVA doesn’t know which array is being accessed.

But expected remaining lifetime decreases ➔ EVA increases.

EVA evicts MRU here, protecting candidates.
If candidate doesn’t hit at size of small array, it must be an access to the big array.

So expected remaining lifetime is large, and EVA is negative.

EVA prefers to evict these candidates.
Candidates that survive further are guaranteed to hit, but it takes a long time.

As remaining lifetime decreases, EVA increases to maximum of $\approx 1$ at size of big array.
EVA implements the optimal policy given uncertainty: Cache small array + as much of big array as fits.
WHY IS EVA THE RIGHT METRIC?
Markov decision processes

• Markov decision processes (MDPs) model decision-making under uncertainty
• MDP theory gives provably optimal decision-making metrics

• We can model cache replacement as an MDP
• EVA corresponds to a decomposition of the appropriate MDP policy

• (Paper gives high-level discussion & intuition; my PhD thesis gives details)
  Happy to discuss in depth offline!
TRANSLATING THEORY TO PRACTICE
Simple hardware, smart software

OS runtime (or HW microcontroller) periodically computes EVA and assigns ranks.

Global timestamp

Hit/eviction event counters

Ranking

Ages

1

2

... 

4

6

Cache bank

Tag

Data

Address... (~45b)

Timestamp (8b)
Updating EVA ranks

• Assign ranks to order (age, reused?) by EVA

• Simple implementation in three passes over ages + sorting:
  1. Compute miss probabilities
  2. Compute unclassified EVA
  3. Add classification term

• Low complexity in software
  • 123 lines of C++

• ...or a HW controller (0.05mm^2 @ 65nm)

Algorithm 1. Algorithm to compute EVA and update ranks.

Inputs: hitCts, evictionCts — event counters, A — age granularity
Returns: rank — eviction priorities for all ages and classes

1. Function UPDATE
2. for a ← 2 to 1:  
3. for c ∈ {nonReused, reused}:  
4. hits_c ← hitCts[c, a]  
5. misses_c ← evictionCts[c, a]  
6. miss_a[c] ← misses_a/(hits_a + misses_a)  
7. miss_a[c] ← misses_a/(hits_a + misses_a)  
8. m ← (hits_c + hits_a)/(misses_c + misses_a)  
9. perAccessCost ← (1 − m) × A/S
10. for c ∈ {nonReused, reused}:  
11. expLifetime, hits, events ← 0  
12. for a ← 2 to 1:  
13. expectedLifetime ← events  
14. eva[c, a] ← (hits − perAccessCost × expectedLifetime)/events  
15. hits ← hitCts[c, a]  
16. events ← hitCts[c, a] + evictionCts[c, a]  
17. evaReused ← eva[reused, 1]/miss[0]  
18. for c ∈ {nonReused, reused}:  
19. for a ← 2 to 1:  
20. eva[c, a] ← (m − m_c[a]) × evaReused  
21. order ← AρSort(eva)  
22. for i ← 1 to 2^{a+1}:  
23. rank[order[i]] ← 2^{i+1} − i  
24. return rank
Overheads

- Software updates
  - 43K cycles / 256K accesses
  - Average 0.1% overhead

- Hardware structures
  - 1% area overhead (mostly tags)
  - 7mW with frequent accesses

*Easy to reduce further with little performance loss.*
EVALUATION
Methodology

• Simulation using zsim
• Workloads: SPEC CPU2006 (multithreaded in paper)
• System: 4GHz OOO, 32KB L1s & 256KB L2

• Study replacement policy in L3 from 1MB → 8MB
  • EVA vs random, LRU, SHiP [Wu MICRO’11], PDP [Duong MICRO’12]

• Compare performance vs. total cache area
  • Including replacement, ≈1% of total area
EVA performs consistently well

See paper for more apps
EVA closes gap to optimal replacement

- “How much worse is X than optimal?”
- Averaged over SPECCPU2006

- EVA closes 57\% random-MIN gap
  - vs. 47\% SHiP, 42\% PDP

- EVA improves execution time by 8.5\%
  - vs 6.8\% for SHiP, 4.5\% for PDP
EVA makes good use of add’l state

- Adding bits improves EVA’s perf.
  - Not true of SHiP, PDP, DRRIP

- ➔ Even with larger tags, EVA saves 8% area vs SHiP

- Open question: how much space should we spend on replacement?
  - Traditionally: as little as possible
  - But is this the best tradeoff?
EVA is easy to apply to new problems

Just change cost/benefit terms in EVA to adapt to...

- Objects of different size (eg, compressed caches)
- Different optimization metrics (eg, byte-hit-rate)
- QoS or application priorities
- ...and so on
THANK YOU!