Multi-Core Memory Hierarchies

Lecture 3: LLC Replacement and Bypassing

Tong Dong

http://mprc.pku.edu.cn/~tongdong/MMH
Model of cache performance

Average Access Time in $i$-level cache:

$$t_{Li} = (1 - m_{Li})t_{Li}^{hit} + m_{Li}t_{Li-1}$$

For $N$ cores and $T$ thread per core, the miss ratio of the $i$-level cache:

- This model assumes that all threads are independent of each other, and that thread context is saved in the cache (or in other dedicated on-chip storage) when threads swap out.

$$m_{Li} = \left( \frac{C_{Li}}{\beta_{Li}I_{Li}(NT)} \right)^{1-\alpha_{Li}}$$

- $\alpha, \beta$: locality function constant, $\alpha > 1$, $\beta >> 1$
- $I(NT)$: interference function
- $C$: total cache size in level $i$
Basic Replacement Policies

- LRU: least recently used

- LFU: least frequently used (requires small saturating counters)

- pseudo-LRU: organize ways as a tree and track which sub-tree was last accessed

- NRU: every block has a bit; the bit is reset to 0 upon touch; when evicting, pick a block with its bit set to 1; if no block has a 1, make every bit 1
Access types that pollute the cache without yielding too many hits: streaming (no reuse), thrashing (distant reuse)

Current hit rates are far short of those with an oracular replacement policy (Belady): evict the block whose next access is most distant

A large fraction of the cache is useless – blocks that have serviced their last hit and are on the slow walk from MRU to LRU
Cache Replacement Policy

- Inclusive vs Exclusive LLC

- Primary Replacement policies
  - Victim Selection
  - Block Insertion
  - Block Promotion

- Bypassing
  - Non-Insertion
Cost-based Replacement
- Low-cost miss if high memory-level parallelism (MLP)
- MHSR counters per entry
- Linear combination of recency and MLP-cost

LIN(Linear) Policy

Set-Sampling: Sampling Based Adaptive (SBAR)
- Tournament Selection (TSEL)
More than half cache lines is Zero Reuse Lines

Replacement policy
- Victim selection policy
- Insertion policy
### Common Cache Access Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recency-Friendly</td>
<td>$(a_1, ..., a_{k-1}, a_k, a_k, a_{k-1}, ..., a_1)^N$</td>
</tr>
<tr>
<td>Thrashing $(k &gt; C)$</td>
<td>$(a_1, ..., a_{k-1}, a_k)^N$</td>
</tr>
<tr>
<td>Streaming $(k = \infty)$</td>
<td>$(a_1, ..., a_{k-1}, a_k)$</td>
</tr>
<tr>
<td>Mixed $(k &lt; C, m &gt; C)$</td>
<td>$[(a_1, ..., a_k)^A P_\epsilon(b_1, ..., b_m)]^N$</td>
</tr>
</tbody>
</table>

- $C$ represents the cache set associativity.
- $a_i$ represents a cache line access.
- $(a_1, a_2, ..., a_{k-1}, a_k)$ is a temporal sequence of $k$ unique addresses to a cache set.
- $(a_1, a_2, ..., a_{k-1}, a_k)^N$ represents a temporal sequence that repeats $N$ times.
- $P_\epsilon(a_1, a_2, ..., a_{k-1}, a_k)$ is a temporal sequence that occurs with some probability $P_\epsilon$. 


Optimal Replacement Policy?

[Belady, IBM Systems Journal, 1966]

- Evict block with longest re-reference distance
  - i.e. next reference to block is farthest in future
  - Requires knowledge of the future!

- Can’t build it, but can model it with trace
  - Process trace in reverse
  - LRU stack algorithm: How to do this in one pass over the trace with some lookahead (Cheetah simulator)

- Useful, since it reveals opportunity
  - (X,A,B,C,D,X): LRU 4-way SA $, 2\text{nd} X will miss
DIP: Dynamic Insertion Policy

- LRU Insertion Policy (LIP)
  - Thrashing
  - Places coming line in LRU position instead of MRU
  - Bypassing for not inclusive cache

- Bimodal Insertion Policy (BIP)
  - Infrequently places some incoming lines into MRU

- Dynamic Insertion Policy (DIP)
  - Dynamically estimates the number of misses incurred by the two competing insertion policies and selects the policy that incurs the fewest misses.

- Reduces MPKI 21% and 2/3 between LRU/OPT

Set Dueling

Figure 9: DIP-Global

Figure 10: DIP via Set Dueling

**PeLIFO, Chaudburi, MICRO’09**

- **Observation**: a large class of applications that exhibit a significant number of cache blocks with small, yet more than one, uses.

- **Fill Stack**: all blocks in the order in arrival time.
  - Most cache hits incurs at the top of the fill stack.

- **Probabilistic escape LIFO (PeLIFO)**:
  - Blocks at the top of the fill stack should be prioritized for victimization.
Pollute Buffer, Soares et al. MICRO’08

- Software-based page coloring scheme
- Isolate pages with high miss rates to a small region of the LLC.
Re-Reference Interval Prediction: in essence, insert blocks near the end of the list than at the very end

Implement with a multi-bit version of NRU: zero counter on touch, evict block with max counter, else increment every counter by one

RRIP can be easily implemented by setting the initial counter value to max-1 (does not require list management)
RRIP, Jaleel et al., ISCA’10

- Re-use distance vs. Re-reference distance
- Re-Reference Interval Prediction (RRIP)
- Not Recently Used (NRU) in modern processors
- SRRIP: the use of multiple bits per block to track priorities at a finer granularity.
- Incoming blocks are being given more time to determine their re-use potential.
- DRRIP (BRRIP+SRRIP) using Set-Dueling
- DRRIP outperform LRU 10% with 2X less hardware overhead.

Behavior of LRU, NRU and SRRIP

<table>
<thead>
<tr>
<th>Next Ref</th>
<th>RRIP head</th>
<th>RRIP tail</th>
<th>RRIP head</th>
<th>RRIP tail</th>
<th>RRIP head</th>
<th>RRIP tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a_1)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(a_2)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(a_2)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(a_1)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(b_1)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(b_2)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(b_3)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(b_4)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(a_1)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
<tr>
<td>(a_2)</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
<td>1 1 1 1</td>
<td>miss</td>
</tr>
</tbody>
</table>

(a) LRU

(b) Not Recently Used (NRU)

(c) 2-bit SRRIP with Hit Promotion

Optimal Belady and Belady+Bypassing

- **Belady Replacement**
  - Optimal longest-forward-distance replacement

- **Belady-B:**
  - Belady extended with bypass which drops an incoming block if its next forward use distance is larger than all the blocks in the target LLC set

- **Base-B: NRU-bypassing**

- **Bypassing is effective.**

- **Dead Block Prediction**
**SHiP.** C. Wu, A. Jaleel et al. MICRO’11

- **Signature-based Hit Predictor**
  - Memory Region
  - Program Counter
  - Instruction History

- **Insertion Policy**
  - If (SHCT[sig] == 0) 3;
  - Always 2 in SRRIP.

```plaintext
if hit then
    cache_line.outcome = true;
    Increment SHCT[signature_m];
else
    if evicted_cache_line.outcome != true
        Decrement SHCT[signature_m];
    cache_line.outcome = false;
    cache_line.signature_m = signature;
    if SHCT[signature] == 0
        Predict distant re-reference;
    else
        Predict intermediate re-reference;
end if
```
Policy: Prevents replacing a cache line until a certain number of accesses to its cache set, called Protecting Distance (PD)

Bypassing when protecting

Reuse Distance (RD)
- The number of accesses to a cache set between two accesses to the same cache line
- Not to evict lines too early
- Not to keep lines in the cache for too long
- Reuse Distance distribution
Dead Blocks

- Access Count Predictor, IEEE TOC 2008 (196)
  - Event counter very block in L2
  - When counter exceeds a threshold, the block dead.
  - Bypassing

- Cache Burst, MICRO 2008 (174)

- Virtual Victim Cache, PACT 2010

- Sampling Dead Block Prediction, MICRO 2010
Fig. 2: Retrace dead block predictor (a), and new dead block predictor with sampler tag array (b). The sampler and dead block predictor table are updated for 1.6% of the accesses to the LLC, while the retrace predictor is updated on every access.

Fig. 3: Block diagrams for dead block predictors
**OPT-Violation, PACT’12**

- Optimal Bypass Monitor
  - Insert-evict pair
- Evicted-Address Filter (EAF)
- Bloom Filter

---

**Figure 2:** Hardware implementation of a Bloom filter using a bit vector of size 16 and two hash functions. The insert operations are performed before the test operations.

**Figure 3:** EAF implementation using a Bloom filter and a counter. The figure shows the three events that trigger operations on the EAF.

---


V. Seshadri, et al. The evicted-address filter: A unified mechanism to address both cache pollution and thrashing. PACT-21. 2012 (74/15)
Reuse Cache MICRO’13

- State of the art replacement policies only achieve average within 5% improvement of NRU
- Decoupled tag/data SLLC which only store the data of lines that reused.
- With Tag Size of 4MB $\text{Data Size of 1MB}$ perform as well as 8MB cache with 16.7% storage

Observation: if with the OPT solution a load instruction has historically brought in lines that produce cache hits, then in the future, the same load instruction is likely to bring in lines that will also produce a cache hit.

Usage interval: from demand to next reference

Liveness intervals: the time period during which that line resides in the cache under the OPT policy.

One would be a cache miss if at any point in its usage interval the number of overlapping liveness intervals matches the cache’s capacity.
Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement.

**Hawkeye Predictor**

(Learns from OPT to make future predictions)

**OPTgen**

(Computes optimal solution for the past)

### Access Sequence

Access Sequence: A, B, B, C, D, E, A, F, D, E, F, C (Cache capacity is 2 lines)

(a) Timeline view of the Access Stream

(b) Optimal Solution (4 hits)
[Cache hits marked as solid lines]

(c) OPTgen Solution (4 hits)
[State of the Occupancy Vector over time]
Machine Learning for locality prediction

- The accuracy of the reuse distance prediction mechanisms limits the scope of optimization.

- Perception Prediction
Summary

MIP: MRU insertion policy (traditional approach)
DIP: Selects the best
LIP: LRU insertion policy
TADIP: Selects the best for each thread
BIP: Bimodal insertion policy
Few insertions at head, most at tail
RRIP: Probabilistic insertion near tail
PIPP: Inserts each thread at different positions + probabilistic promotion
AGGRESSOR-VT: Victimizes the aggressor thread with a high probability
UCP: Partitions ways across threads based on marginal utility

Highest priority ← Priority stack of blocks in a set → Lowest priority