Estimating the Performance of Cache Replacement Policies

Daniel Grund, Jan Reineke

Department of Computer Science
Saarland University

MeMoCode, June 2008
Outline

1. Introduction
   - Locality & Caches
   - Problem & Prior Work

2. Stochastic Model
   - Contributions
   - Model Construction

3. Evaluation
   - Precision & Runtime
   - Related Work

4. Conclusions
Outline

1. Introduction
   - Locality & Caches
   - Problem & Prior Work

2. Stochastic Model
   - Contributions
   - Model Construction

3. Evaluation
   - Precision & Runtime
   - Related Work

4. Conclusions
Locality & Caches

- Typical programs do not behave randomly, they exhibit
  - temporal locality
  - spatial locality
  in their memory access behavior

- Caches try to exploit locality
- Have great influence on system performance
Cache Parameters

Line size \(l\)  Size of blocks stored in cache

Associativity \(k\)  Number of lines one block may be stored in

Number of sets \(s\)  Each set comprises \(k\) lines

Capacity  Determined by \(s \cdot k \cdot l\)

Replacement policy  *next slide*

---

\[
\begin{array}{ccc}
\text{Tag} & \text{Data} & \text{Tag} & \text{Data} & \ldots & \text{Tag} & \text{Data} \\
\hline
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
\end{array}
\]
Replacement Policies

The replacement policy determines the cache line whose contents are replaced upon a cache miss.

Examples:
- **LRU**  Least recently used
- **FIFO**  First-in first-out
- **PLRU**  Approximation on LRU
- **MRU**  Most recently used

...
Problem Statement

- Cache performance depends on (locality of) executed program
- There are lots of design possibilities

Problem

Given a program, determine the best cache configuration.
Stack Histograms (Mattson et al. 1970)

- Concisely capture locality in memory access behavior
- **Age** of cacheable element $e$: number of accesses on different elements since last access on $e$
- **Stack distance** of a memory access: age of the accessed element just before the access
- **Stack histogram**: frequency distribution of stack distances

**Example**

<table>
<thead>
<tr>
<th>Reference string</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>c</th>
<th>b</th>
<th>b</th>
<th>c</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack distance</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>1</td>
<td>$\infty$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Stack distance $d$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>$\geq$</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. frequency $f_d$</td>
<td>$\frac{1}{8}$</td>
<td>$\frac{2}{8}$</td>
<td>$\frac{2}{8}$</td>
<td>0</td>
<td>$\frac{3}{8}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Estimation for LRU Is Easy

Mattson et al. (1970)

- LRU always retains the $k$ most recently used elements
- Exactly the ages $0, \ldots, k - 1$ are cached

**Example**

**Associativity $k = 8$**

$$d \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad \geq$$

$f_d$ \quad always cached \quad $\leftarrow$ \quad never cached \quad $\Rightarrow$ Miss ratio $= 1 - \sum_{i=0}^{7} f_i$
For Other Policies It Is Not

Problem:
- In general, the ages of cached elements vary during execution
- Number and shape of combinations of ages depend on policy

Idea: determine
- all possible combinations (state space)
- probability of each state
- probability of a cache miss in each state
Outline

1 Introduction
   - Locality & Caches
   - Problem & Prior Work

2 Stochastic Model
   - Contributions
   - Model Construction

3 Evaluation
   - Precision & Runtime
   - Related Work

4 Conclusions
Our Approach

- Static method
- Given . . .
  - a concise characterization of the program’s locality
  - parameters of the cache architecture
- ...we construct a Markov chain to estimate the miss ratio

Diagram:

- Locality information
- State space construction
- Markov chain
- Steady state computation
- Miss ratio
- Cache parameters
Contributions

- **Replacement policy** is taken into account
- **Policy tables**
  - uniformly represent replacement policies
  - ease automatic model construction
- **History stack histograms**
  - extend well-known stack histograms
  - increase precision of estimates
Policy Tables

- Uniform representation for a class of replacement policies
- Class contains LRU, PLRU, FIFO, MRU
- Eases automatic model construction

\[
\begin{bmatrix}
\pi_0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_2 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_3 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_4 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_5 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_6 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_7 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_m & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0
\end{bmatrix}
\]

8-way FIFO

- Policy state is virtual order on cache lines [3, 2, 1, 0, 4, 5, 6, 7]
- Updates are specified by permutations
  - \(\pi_i\) for cache-hit on position \(i\)
  - \(\pi_m\) for cache-miss

\[
\begin{bmatrix}
\pi_0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \\
\pi_1 & 0 & 2 & 3 & 4 & 5 & 6 & 7 & 1 \\
\pi_2 & 0 & 1 & 3 & 4 & 5 & 6 & 7 & 2 \\
\pi_3 & 0 & 1 & 2 & 4 & 5 & 6 & 7 & 3 \\
\pi_4 & 0 & 1 & 2 & 3 & 5 & 6 & 7 & 4 \\
\pi_5 & 0 & 1 & 2 & 3 & 4 & 6 & 7 & 5 \\
\pi_6 & 0 & 1 & 2 & 3 & 4 & 5 & 7 & 6 \\
\pi_7 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\pi_m & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0
\end{bmatrix}
\]

8-way LRU

\[ [3, 2, 1, 0, 4, 5, 6, 7] \xrightarrow{\pi_4} [3, 2, 1, 0, 5, 6, 7, 4] \]
History Stack Histograms

Circle plots of stack histograms

\[ \begin{array}{cccccccccccc}
  d & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & d \\
  f_d & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \cdots & \bullet & \cdots & \cdots & \cdots & \geq \\
\end{array} \]

\[ \begin{array}{cccccccccccc}
  d & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & d \\
  f_d & \bullet & \cdots & \cdots & \cdots & \cdots & \bullet & \bullet & \bullet & \bullet & \bullet & \cdots & \cdots & \cdots & \geq \\
\end{array} \]

⇒ increases precision of estimates
History Stack Histograms

Circle plots of stack histograms

History stack histograms consider context of occurring stack distances

$\Rightarrow$ increases precision of estimates

Daniel Grund, Jan Reineke
Concrete cache set state: tuple of elements and status bits
Two FIFO states: \([a, b, c, d]\) and \([d, a, b, c]\)

Only relative order w.r.t. replacement matters
Abstract from physical positions
\[\Rightarrow\] Normalized tuple: \([b, c, d, a]\)

Only ages of elements matter
Abstract from names
\[\Rightarrow\] Normalized tuple of ages, e.g. \([4, 1, 5, 0]\)
Model Construction: States (2)

- Problem: arbitrarily high ages $\Rightarrow$ infinite model
- Introduce cutoff-age, abstract from high ages
  $\Rightarrow \alpha([4, 1, 5, 0], [6, 1, 4, 0]) = [4, 1, 4, 0]$

- Disambiguate states by history to fit history stack histograms
  $\Rightarrow$ Abstract cache set state:

```
[4, 1, 4, 0]
hist: 0, 2
```
Model Construction: Transitions

State space construction:

- Pick unexplored abstract state $s$

[3, 4, 1, 0]

hist: 5
Model Construction: Transitions

State space construction:
- Pick unexplored abstract state $s$
- Enumerate all possible accesses (on ages)

![Diagram](image)

$[3, 4, 1, 0]$

hist: 5

3/hit
Model Construction: Transitions

State space construction:
- Pick unexplored abstract state $s$
- Enumerate all possible accesses (on ages)
- For each successor state $s'$
  - Update ages

![Diagram showing state transitions with [3, 4, 1, 0] history leading to [0, 4, 2, 1] history after 3 hits.]
Model Construction: Transitions

State space construction:

- Pick unexplored abstract state $s$
- Enumerate all possible accesses (on ages)
- For each successor state $s'$
  - Update ages
  - Update order (apply permutation from policy table)

![Diagram showing state transition between [3, 4, 1, 0] with history 5 and [0, 4, 2, 1] with history hit 3]
Model Construction: Transitions

State space construction:
- Pick unexplored abstract state $s$
- Enumerate all possible accesses (on ages)
- For each successor state $s'$
  - Update ages
  - Update order (apply permutation from policy table)
  - Adjust history

```
[3, 4, 1, 0]  [0, 4, 2, 1]
hist: 5        hist: 3
3/hit
```
State space construction:

- Pick unexplored abstract state $s$
- Enumerate all possible accesses (on ages)
- For each successor state $s'$
  - Update ages
  - Update order (apply permutation from policy table)
  - Adjust history
  - Compute probability for transition $s \rightarrow s'$ from history stack histogram
Model Construction: Example

4-way FIFO, history length 1, cutoff age 5

\[ [3, 4, 1, 0] \quad \text{hist: 5} \]
\[ p_{5,3} \quad 3/\text{hit} \]
\[ p_{\text{miss}} \quad 5/\text{miss} \]

\[ [5, 2, 1, 0] \quad \text{hist: 5} \]
\[ p_{\text{miss}} \quad 5/\text{miss} \]

\[ [0, 4, 2, 1] \quad \text{hist: 3} \]
\[ p_{3,3} \quad 3/\text{miss} \]

\[ [4, 3, 2, 0] \quad \text{hist: 3} \]
\[ p_{3,3} \quad 3/\text{miss} \]

\[ [0, 3, 2, 1] \quad \text{hist: 5} \]
\[ p_{\text{hit}} \quad 5/\text{hit} \]

\[ p_{\text{miss}} \quad 5/\text{miss} \]

\[ [3, 2, 1, 0] \quad \text{hist: 5} \]
Computing the Miss-Ratio Estimate

1. Compute steady state probabilities $p(s_i)$ of the Markov chain
   ▶ e.g. with the Gauss-Seidel algorithm

2. Compute miss probabilities $p(m_i)$
   ▶ simple sum of values in history stack histogram

⇒ Miss ratio $= \sum_i p(s_i) \cdot p(m_i)$
Outline

1. Introduction
   - Locality & Caches
   - Problem & Prior Work

2. Stochastic Model
   - Contributions
   - Model Construction

3. Evaluation
   - Precision & Runtime
   - Related Work

4. Conclusions
Evaluation: Precision

- Shared 2nd level caches with different parameter combinations
- C programs of SPEC CPU2000
- Evaluated against full simulation runs (SimpleScalar)

- Average absolute error between 0.18% and 2.92%
- Better to spend resources on history than on higher cutoff age

Example: 8-way associative, 1024 sets, 32 bytes line size

<table>
<thead>
<tr>
<th>Replacement Policy</th>
<th>Average Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>0</td>
</tr>
<tr>
<td>PLRU</td>
<td>0.23</td>
</tr>
<tr>
<td>FIFO</td>
<td>0.58</td>
</tr>
<tr>
<td>MRU</td>
<td>2.92</td>
</tr>
<tr>
<td>RAND</td>
<td>1.61</td>
</tr>
</tbody>
</table>
Evaluation: Runtime

- $k=4$ is handled easily
- $k=8$ often requires cheaper model parameters, i.e., stronger abstraction / less precision

Example: Most expensive model parameters for $k = 8$ and 2 GB main memory

<table>
<thead>
<tr>
<th>Replacement Policy</th>
<th>PLRU</th>
<th>FIFO</th>
<th>MRU</th>
<th>RAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>#states</td>
<td>4,166,048</td>
<td>8,216,745</td>
<td>4,129,362</td>
<td>2,687,856</td>
</tr>
<tr>
<td>◇ runtime [s]</td>
<td>386</td>
<td>440</td>
<td>263</td>
<td>234</td>
</tr>
</tbody>
</table>

Comparison: ◇ simulation time was 18 hours
Limitations / Future Work

- Sometimes precision is limited by resources/model parameters, e.g. 8-FIFO or 8-RAND with history

- History length is currently limited to 1. \(\geq 2\) requires \(> 2\)GB main memory or stronger abstractions

- Implementation limited to policies in policy table class. Others (e.g. 2Q, EE-LRU, ARC) conceivable but most likely require different abstractions
Related Work

Mattson, Gecsei, Slutz, and Traiger
Evaluation techniques for storage hierarchies

Guo and Solihin
An analytical model for cache replacement policy performance

- Mattson et al. limited to LRU, but seminal work and still useful
- Guo and Solihin
  - policies directly specified in terms of probability functions
  - hence method more efficient
  - but real-world policies (PLRU, FIFO, MRU) impossible
Conclusions

- Static method for miss-ratio estimation
  - stochastic model yields precise estimates
  - allows for considering real-world policies

- History stack histograms
  - extend well-known stack histograms
  - increase precision of estimates

- Policy tables
  - provide uniform representation of policies