Timing analysis and timing predictability
Caches in WCET Analysis

Reinhard Wilhelm\textsuperscript{1}  Jan Reineke\textsuperscript{2}

\textsuperscript{1}Saarland University, Saarbrücken, Germany
\textsuperscript{2}University of California, Berkeley, USA

January 18, 2011
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Outline

1. Caches

2. Cache Analysis for Least-Recently-Used

3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4. Summary
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

![Diagram of CPU, Cache, and Main Memory with capacity and latency values: 32 KB/3 cycles for Cache, 2 MB/100 cycles for Main Memory.]

- Why caches work: principle of locality
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: principle of locality
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: *principle of locality*
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

![Diagram of CPU, Cache, and Main Memory with capacity and latency details.](image-url)
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: *principle of locality*
- spatial: e.g. in sequential instructions, accessing arrays
- temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

![Diagram of CPU, Cache, and Main Memory]

- Why caches work: principle of locality
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops

- Why caches work: principle of locality
  - spatial: e.g. in sequential instructions, accessing arrays
  - temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: principle of locality

- spatial: e.g. in sequential instructions, accessing arrays
- temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

Why caches work: *principle of locality*

- spatial: e.g. in sequential instructions, accessing arrays
- temporal: e.g. in loops
Caches

- Small but very fast memories that buffer part of the main memory
- Bridge the gap between speed of CPU and main memory

CPU - Cache - Main Memory

Capacity: 32 KB, 2 MB
Latency: 3 cycles, 100 cycles

“hit” [ac]

c

Why caches work: *principle of locality*
- spatial: e.g. in sequential instructions, accessing arrays
- temporal: e.g. in loops
Fully-Associative Caches

Address:

Tag | Block offset
---|---

$log_2(8 \times b)$

MUX

Data

= associativity

Yes: Hit!

No: Miss!

$k = \text{associativity}$

Reinhard Wilhelm

Caches in WCET Analysis

January 18, 2011
Set-Associative Caches

Special cases:
- direct-mapped cache: only one line per cache set
- fully-associative cache: only one cache set
Cache Replacement Policies

- Least-Recently-Used (LRU) used in Intel Pentium I and MIPS 24K/34K
- First-In First-Out (FIFO or Round-Robin) used in Motorola PowerPC 56x, Intel XScale, ARM9, ARM11
- Pseudo-LRU (PLRU) used in Intel Pentium II-IV and PowerPC 75x
- Most-Recently-Used (MRU) as described in literature

Each cache set is treated independently:
→ Set-associative caches are compositions of fully-associative caches.
Outline

1. Caches

2. Cache Analysis for Least-Recently-Used

3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4. Summary
Cache Analysis

Two types of cache analyses:

1. Local guarantees: classification of individual accesses
   - May-Analysis $\rightarrow$ Overapproximates cache contents
   - Must-Analysis $\rightarrow$ Underapproximates cache contents

2. Global guarantees: bounds on cache hits/misses

- Cache analyses almost exclusively for LRU
- In practice: FIFO, PLRU, ...
Abstract Interpretation in Timing Analysis

- Abstract interpretation is always based on the semantics of the analyzed language.
- A semantics of a programming language that talks about time needs to incorporate the execution platform!
- Static timing analysis is thus based on such a semantics.
Galois Connection

- Abstraction function $\alpha$
- Concretization function $\gamma$

$\Rightarrow \forall m' \in M' : \gamma(m') = \gamma(m)$
Abstract Interpretation in Timing Analysis

Determines:

1. invariants about the values of variables (in registers, on the stack)
   - to compute loop bounds
   - to eliminate infeasible paths
   - to determine effective memory addresses

2. invariants on architectural execution state
   - Cache contents ⇒ predict hits and misses
   - Pipeline states ⇒ predict or exclude pipeline stalls
Challenges for Cache Analysis

Always a cache hit/always a miss?

- Initial cache contents unknown.
- Different paths lead to these points.
- Cannot resolve address of $z$. 

Reinhard Wilhelm
Challenges for Cache Analysis

- Always a cache hit/always a miss?

1. Initial cache contents unknown.
2. Different paths lead to these points.
3. Cannot resolve address of $z$. 
Deriving Invariants about Cache States using Abstract Interpretation

**Collecting Semantics =**
set of states at each program point that any execution may encounter there

Two approximations:
- Collecting Semantics \(\subseteq\) Cache Semantics \(\subseteq\) \(\gamma\) (Abstract Cache Sem.) efficiently computable
- Collecting Semantics uncomputable
Deriving Invariants about Cache States using Abstract Interpretation

**Collecting Semantics** = set of states at each program point that any execution may encounter there

Two approximations:

- Collecting Semantics \( \subseteq \) Cache Semantics
- Collecting Semantics \( \subseteq \gamma \)(Abstract Cache Sem.) efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:
- Collecting Semantics uncomputable
- \( \subseteq \) Cache Semantics computable
- \( \subseteq \gamma \) (Abstract Cache Sem.) efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

Collecting Semantics = set of states at each program point that any execution may encounter there

Two approximations:
- \( \text{Collecting Semantics} \) uncomputable
- \( \subseteq \text{Cache Semantics} \) computable
- \( \subseteq \gamma(\text{Abstract Cache Sem.}) \) efficiently computable
Deriving Invariants about Cache States using Abstract Interpretation

**Collecting Semantics** = set of states at each program point that any execution may encounter there

Two approximations:
- Collecting Semantics uncomputable
- $\subseteq$ Cache Semantics computable
- $\subseteq \gamma$(Abstract Cache Sem.) efficiently computable
Least-Recently-Used (LRU): Concrete Behavior

“Cache Miss”:

```
  z  
  y  
  x  
  t  
```

LRU has notion of age

```
  s  
  z  
  y  
  x  
```

“Cache Hit”:

```
  z  
  y  
  s  
  t  
```

```
  s  
  z  
  y  
  t  
```
LRU: Must-Analysis: Abstract Domain

- Used to predict cache hits.
- Maintains upper bounds on ages of memory blocks.
- Upper bound ≤ associativity → memory block definitely cached.

Example

<table>
<thead>
<tr>
<th>Abstract state:</th>
<th>... and its interpretation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>{x}</td>
<td>Describes the set of all concrete cache states in which x, s, and t occur,</td>
</tr>
<tr>
<td>{}</td>
<td>■ x with an age of 0,</td>
</tr>
<tr>
<td>{s,t}</td>
<td>■ s and t with an age not older than 2.</td>
</tr>
<tr>
<td>{}</td>
<td>age 3</td>
</tr>
<tr>
<td>γ([[{x}, {}], {s, t}, {}]) =</td>
<td></td>
</tr>
<tr>
<td>{[[x, s, t, a], [x, t, s, a], [x, s, t, b], ...}</td>
<td></td>
</tr>
</tbody>
</table>
Sound Update – Local Consistency

\( (\text{must}) \quad \text{Abstract Update} \quad \rightarrow \quad (\text{must}') \)

\( \gamma \)

\( \gamma \)

\[ \text{concrete cache states} \]

\[ \text{Lifted Concrete Update} \]

\[ \text{concrete cache states} \]
LRU: Must-Analysis: Update

“Potential Cache Miss”:

“Definite Cache Hit”:

Why does \( t \) not age in the second case?
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:
- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:
- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

“Intersection + Maximal Age”
Must-Analysis for LRU: Join

Need to combine information where control-flow merges.

Join should be conservative:

- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

“Intersection + Maximal Age”

How many memory blocks can be in the must-cache?
Example: Must-Analysis

\[
\text{entry} \quad [\{\}, \{\}, \{\}, \{\}]
\]

\[
\text{exit} \quad \perp
\]
Example: Must-Analysis

\[
\text{entry } [{}, {}, {}, {}, {}] \\
\downarrow \\
\downarrow \sqcup [{}, {}, {}, {}, {}] = [{}, {}, {}, {}, {}] \\
\downarrow \\
B \perp C \perp D \perp \text{exit} \perp 
\]
Example: Must-Analysis

\[
\text{entry } [\{\}, \{\}, \{\}, \{\}]
\]

\[
\bot \sqcup [\{\}, \{\}, \{\}, \{\}] = [\{\}, \{\}, \{\}, \{\}]
\]

\[
[\{A\}, \{\}, \{\}, \{\}] \quad [\{A\}, \{\}, \{\}, \{\}]
\]

\[
\bot
\]

exit \quad \bot
Example: Must-Analysis

\[
\begin{align*}
\text{entry: } & [\{\}, \{\}, \{\}, \{\}, \{\}] \\
\downarrow \\
\downarrow \\
\downarrow \\
\downarrow \\
\text{exit: } & \bot
\end{align*}
\]

\[
\begin{align*}
A & \cup [\{\}, \{\}, \{\}, \{\}, \{\}] = [\{\}, \{\}, \{\}, \{\}, \{\}] \\
B & \cup [\{A\}, \{\}, \{\}, \{\}, \{\}] = [\{B\}, \{A\}, \{\}, \{\}, \{\}] \\
C & \cup [\{C\}, \{A\}, \{\}, \{\}, \{\}] = [\{\}, \{A\}, \{\}, \{\}, \{\}] \\
D & \cup [\{C\}, \{A\}, \{\}, \{\}, \{\}] = [\{\}, \{A\}, \{\}, \{\}, \{\}]
\end{align*}
\]
Example: Must-Analysis

No cache hits can be predicted :-(

Reinhard Wilhelm

Caches in WCET Analysis
Context-Sensitive Analysis/Virtual Loop-Unrolling

Problem:
- The first iteration of a loop will always result in cache misses.
- Similarly for the first execution of a function.

Solution:
- Virtually Unroll Loops: Distinguish the first iteration from others
- Distinguish function calls by calling context.

Virtually unrolling the loop once:
- Accesses to \( A \) and \( D \) are provably hits after the first iteration
- Accesses to \( B \) and \( C \) can still not be classified. Within each execution of the loop, they may only miss once.

\[ \rightarrow \text{Persistence Analysis} \]
LRU: May-Analysis: Abstract Domain

- Used to predict cache misses.
- Maintains lower bounds on ages of memory blocks.
- Lower bound $\geq$ associativity

$\rightarrow$ memory block definitely not cached.

Example

Abstract state:

<table>
<thead>
<tr>
<th>{x,y}</th>
<th>age 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td></td>
</tr>
<tr>
<td>{s,t}</td>
<td></td>
</tr>
<tr>
<td>{u}</td>
<td>age 3</td>
</tr>
</tbody>
</table>

... and its interpretation:

Describes the set of all concrete cache states in which no memory blocks except $x$, $y$, $s$, $t$, and $u$ occur,

- $x$ and $y$ with an age of at least 0,
- $s$ and $t$ with an age of at least 2,
- $u$ with an age of at least 3.

$$\gamma([[\{x, y\}, \{\}, \{s, t\}, \{u\}]] = \{[x, y, s, t], [y, x, s, t], [x, y, s, u], \ldots\}$$
LRU: May-Analysis: Update

“Definite Cache Miss”:

“Potential Cache Hit”:

Why does \( t \) age in the second case?
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- \( \gamma(A) \subseteq \gamma(A \sqcup B) \)
- \( \gamma(B) \subseteq \gamma(A \sqcup B) \)

“Union + Minimal Age”
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \cup B)$
- $\gamma(B) \subseteq \gamma(A \cup B)$

“Union + Minimal Age”
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:
- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

"Union + Minimal Age"
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:
- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

“Union + Minimal Age”
LRU: May-Analysiss: Join

Need to combine information where control-flow merges.

Join should be conservative:

- $\gamma(A) \subseteq \gamma(A \sqcup B)$
- $\gamma(B) \subseteq \gamma(A \sqcup B)$

“Union + Minimal Age”
LRU: May-Analysis: Join

Need to combine information where control-flow merges.

Join should be conservative:
- \( \gamma(A) \subseteq \gamma(A \cup B) \)
- \( \gamma(B) \subseteq \gamma(A \cup B) \)

“Union + Minimal Age”
Outline

1. Caches

2. Cache Analysis for Least-Recently-Used

3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4. Summary
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Uncertainty in WCET Analysis

- Amount of uncertainty determines precision of WCET analysis
- Uncertainty in cache analysis depends on replacement policy

Variation due to inputs and initial hardware state × penalty = uncertainty

BCET  ACET  WCET (upper bound)  execution time
Uncertainty in Cache Analysis

1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of $z$.

$\Rightarrow$

Amount of uncertainty determined by ability to recover information.
Uncertainty in Cache Analysis

1. Initial cache contents unknown.
Uncertainty in Cache Analysis

1. Initial cache contents unknown.
2. Need to combine information.
Uncertainty in Cache Analysis

1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of z.

Amount of uncertainty determined by ability to recover information.
Uncertainty in Cache Analysis

1. Initial cache contents unknown.
2. Need to combine information.
3. Cannot resolve address of z.

Amount of uncertainty determined by ability to recover information
Predictability Metrics

Evict

Sequence: \langle a, \ldots, e, f, g, h \rangle
Meaning of Metrics

- **Evict**
  - Number of accesses to obtain *any* *may*-information.
  - I.e. when can an analysis predict any cache misses?

- **Fill**
  - Number of accesses to complete *may*- and *must*-information.
  - I.e. when can an analysis predict each access?

Evict and Fill bound the precision of *any* static cache analysis. Can thus serve as a benchmark for analyses.
Evaluation of Least-Recently-Used

- LRU “forgets” about past quickly:
  - cares about most-recent access to each block only
  - order of previous accesses irrelevant

In the example: Evict = Fill = 4

In general: Evict(k) = Fill(k) = k, where k is the associativity of the cache
Evaluation of First-In First-Out (sketch)

- Like LRU in the miss-case
- But: “Ignores” hits

In the worst-case $k - 1$ hits and $k$ misses: $(k = \text{associativity})$

$\rightarrow \text{Evict}(k) = 2k - 1$

Another $k$ accesses to obtain complete knowledge:

$\rightarrow \text{Fill}(k) = 3k - 1$
Evaluation of Pseudo-LRU (sketch)

- Tree-bits point to block to be replaced

```
1 1
  a b c d
```

```
0 1
  a b c d
```

```
0 1
  a e c d
```

- Accesses “rejuvenate” neighborhood
  - Active blocks keep their (inactive) neighborhood in the cache

- Analysis yields:
  - Evict\((k) = \frac{k}{2} \log_2 k + 1\)
  - Fill\((k) = \frac{k}{2} \log_2 k + k - 1\)
Evaluation of Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>Evict((k))</th>
<th>Fill((k))</th>
<th>Evict((8))</th>
<th>Fill((8))</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>(k)</td>
<td>(k)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>FIFO</td>
<td>(2k - 1)</td>
<td>(3k - 1)</td>
<td>15</td>
<td>23</td>
</tr>
<tr>
<td>MRU</td>
<td>(2k - 2)</td>
<td>(\infty/3k - 4)</td>
<td>14</td>
<td>(\infty/20)</td>
</tr>
<tr>
<td>PLRU</td>
<td>(\frac{k}{2} \log_2 k + 1)</td>
<td>(\frac{k}{2} \log_2 k + k - 1)</td>
<td>13</td>
<td>19</td>
</tr>
</tbody>
</table>

- LRU is optimal w.r.t. metrics.
- Other policies are much less predictable.

→ Use LRU if predictability is a concern.

- How to obtain *may*- and *must*-information within the given limits for other policies?
Outline

1. Caches

2. Cache Analysis for Least-Recently-Used

3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4. Summary
Relative Competitiveness

- **Competitiveness** (Sleator and Tarjan, 1985): worst-case performance of an online policy *relative to the optimal offline policy*
  - used to evaluate online policies

- **Relative competitiveness** (Reineke and Grund, 2008): worst-case performance of an online policy *relative to another online policy*
  - used to derive local and global cache analyses
Definition – Relative Miss-Competitiveness

Notation

\[ m_p(p, s) = \text{number of misses that policy } P \text{ incurs on} \]
\[ \text{access sequence } s \in M^* \text{ starting in state } p \in C_p \]
Definition – Relative Miss-Competitiveness

Notation

\[ m_P(p, s) = \text{number of misses that policy } P \text{ incurs on access sequence } s \in M^* \text{ starting in state } p \in C_P \]

Definition (Relative miss competitiveness)

Policy \( P \) is \( (k, c) \)-miss-competitive relative to policy \( Q \) if

\[ m_P(p, s) \leq k \cdot m_Q(q, s) + c \]

for all access sequences \( s \in M^* \) and cache-set states \( p \in C_P, q \in C_Q \) that are compatible \( p \sim q \).
Definition – Relative Miss-Competitiveness

Notation

\[ m_P(p, s) = \text{number of misses that policy } P \text{ incurs on access sequence } s \in M^* \text{ starting in state } p \in C_P \]

Definition (Relative miss competitiveness)

Policy \( P \) is \((k, c)\)-miss-competitive relative to policy \( Q \) if

\[ m_P(p, s) \leq k \cdot m_Q(q, s) + c \]

for all access sequences \( s \in M^* \) and cache-set states \( p \in C_P \), \( q \in C_Q \) that are compatible \( p \sim q \).

Definition (Competitive miss ratio of \( P \) relative to \( Q \))

The smallest \( k \), s.t. \( P \) is \((k, c)\)-miss-competitive rel. to \( Q \) for some \( c \).
Example – Relative Miss-Competitiveness

\( P \) is \((3, 4)\)-miss-competitive relative to \( Q \).
If \( Q \) incurs \( x \) misses, then \( P \) incurs at most \( 3 \cdot x + 4 \) misses.
Example – Relative Miss-Competitiveness

\( P \) is \((3, 4)\)-miss-competitive relative to \( Q \).
If \( Q \) incurs \( x \) misses, then \( P \) incurs at most \( 3 \cdot x + 4 \) misses.

**Best:** \( P \) is \((1, 0)\)-miss-competitive relative to \( Q \).
Example – Relative Miss-Competitiveness

\( P \) is \((3, 4)\)-miss-competitive relative to \( Q \).

If \( Q \) incurs \( x \) misses, then \( P \) incurs at most \( 3 \cdot x + 4 \) misses.

**Best:** \( P \) is \((1, 0)\)-miss-competitive relative to \( Q \).

**Worst:** \( P \) is not-miss-competitive (or \( \infty \)-miss-competitive) relative to \( Q \).
**Example – Relative Hit-Competitiveness**

\[ P \text{ is } \left( \frac{2}{3}, 3 \right) \text{-hit-competitive relative to } Q. \]

If \( Q \) has \( x \) hits, then \( P \) has at least \( \frac{2}{3} \cdot x - 3 \) hits.
Example – Relative Hit-Competitiveness

\( \mathbf{P} \) is \((\frac{2}{3}, 3)\)-hit-competitive relative to \( \mathbf{Q} \).
If \( \mathbf{Q} \) has \( x \) hits, then \( \mathbf{P} \) has at least \( \frac{2}{3} \cdot x - 3 \) hits.

**Best:** \( \mathbf{P} \) is \((1, 0)\)-hit-competitive relative to \( \mathbf{Q} \).
Equivalent to \((1, 0)\)-miss-competitiveness.
Example – Relative Hit-Competitiveness

$P$ is $(\frac{2}{3}, 3)$-hit-competitive relative to $Q$. If $Q$ has $x$ hits, then $P$ has at least $\frac{2}{3} \cdot x - 3$ hits.

**Best:** $P$ is $(1, 0)$-hit-competitive relative to $Q$. Equivalent to $(1, 0)$-miss-competitiveness.

**Worst:** $P$ is $(0, 0)$-hit-competitive relative to $Q$. Analogue to $\infty$-miss-competitiveness.
Local Guarantees: $(1, 0)$-Competitiveness

Let $P$ be $(1, 0)$-competitive relative to $Q$:

$$m_P(p, s) \leq 1 \cdot m_Q(q, s) + 0$$

$$\Leftrightarrow m_P(p, s) \leq m_Q(q, s)$$

If $Q$ "hits", so does $P$, and if $P$ "misses", so does $Q$.

As a consequence, 1-a-analysis for $Q$ is also a-m-analysis for $P$, and 2-a-analysis for $P$ is also a-m-analysis for $Q$. 
Local Guarantees: (1, 0)-Competitiveness

Let \( P \) be \((1, 0)\)-competitive relative to \( Q \):

\[
m_P(p, s) \leq 1 \cdot m_Q(q, s) + 0
\]

\( \iff m_P(p, s) \leq m_Q(q, s) \)

1. If \( Q \) “hits”, so does \( P \), and
2. if \( P \) “misses”, so does \( Q \).
Local Guarantees: (1, 0)-Competitiveness

Let $P$ be $(1, 0)$-competitive relative to $Q$:

$$m_P(p, s) \leq 1 \cdot m_Q(q, s) + 0$$

$$\Leftrightarrow m_P(p, s) \leq m_Q(q, s)$$

1. If $Q$ “hits”, so does $P$, and
2. if $P$ “misses”, so does $Q$.

As a consequence,

1. a *must*-analysis for $Q$ is also a *must*-analysis for $P$, and
2. a *may*-analysis for $P$ is also a *may*-analysis for $Q$. 
Global Guarantees: \((k, c)\)-Competitiveness

**Given:** Global guarantees for policy \(Q\).

**Wanted:** Global guarantees for policy \(P\).

1. Determine competitiveness of policy \(P\) relative to policy \(Q\).

   \[ m_P \leq k \cdot m_Q + c m_Q(T) = m_P(T) \]

2. Compute global guarantee for task \(T\) under policy \(Q\).

3. Calculate global guarantee on the number of misses for \(P\) using the global guarantee for \(Q\) and the competitiveness results of \(P\) relative to \(Q\).
Global Guarantees: \((k, c)\)-Competitiveness

Given: Global guarantees for policy \(Q\).
Wanted: Global guarantees for policy \(P\).

1. Determine competitiveness of policy \(P\) relative to policy \(Q\).

\[ m_P \leq k \cdot m_Q + c \]
Global Guarantees: \((k, c)\)-Competitiveness

Given: Global guarantees for policy \(Q\).
Wanted: Global guarantees for policy \(P\).

1. Determine competitiveness of policy \(P\) relative to policy \(Q\).
   \[ m_P \leq k \cdot m_Q + c \]

2. Compute global guarantee for task \(T\) under policy \(Q\).
   \[ m_Q(T) \]
Global Guarantees: \((k, c)\)-Competitiveness

Given: Global guarantees for policy \(Q\).

Wanted: Global guarantees for policy \(P\).

1. Determine competitiveness of policy \(P\) relative to policy \(Q\).
   \[
   m_P \leq k \cdot m_Q + c
   \]

2. Compute global guarantee for task \(T\) under policy \(Q\).
   \[
   m_{Q}(T)
   \]

3. Calculate global guarantee on the number of misses for \(P\) using the global guarantee for \(Q\) and the competitiveness results of \(P\) relative to \(Q\).
   \[
   m_{Q}(T) = m_{P}(T)
   \]
Relative Competitiveness: Automatic Computation

\( P \) and \( Q \) (here: FIFO and LRU) induce transition system:

\[
\begin{align*}
\text{Legend} & \\
[abcd]_{\text{FIFO}}, [abcd]_{\text{LRU}} & \overset{\text{first-in}}{\longrightarrow} [abcd]_{\text{FIFO}}, [abcd]_{\text{LRU}} \\
\text{Cache-set state} & \\
[abcd]_{\text{FIFO}}, [abcd]_{\text{LRU}} & \overset{\text{last-in}}{\longrightarrow} [abcd]_{\text{FIFO}}, [abcd]_{\text{LRU}} \\
\text{Memory access} & \\
\] (h, h) & \text{(m, m)}
\]

Competitive miss ratio = maximum ratio of misses in policy \( P \) to misses in policy \( Q \) in transition system
Problem: The induced transition system is $\infty$ large.

Observation: Only the relative positions of elements matter:

\[
\begin{align*}
\{abc\}_{\text{LRU}}, \{bde\}_{\text{FIFO}} & \approx \{fgl\}_{\text{LRU}}, \{ghm\}_{\text{FIFO}} \\
\{cab\}_{\text{LRU}}, \{cbd\}_{\text{FIFO}} & \approx \{lfg\}_{\text{LRU}}, \{lgh\}_{\text{FIFO}}
\end{align*}
\]

Solution: Construct \textit{finite} quotient transition system.
~ - Equivalent States in Running Example

\begin{itemize}
\item \([eabc]_{FIFO}, [eabc]_{LRU}\)
\item \([abcd]_{FIFO}, [abcd]_{LRU}\)
\item \([eabc]_{FIFO}, [ceab]_{LRU}\)
\item \([abcd]_{FIFO}, [dabc]_{LRU}\)
\item \([eabc]_{FIFO}, [ceda]_{LRU}\)
\item \([eabc]_{FIFO}, [edab]_{LRU}\)
\item \([deab]_{FIFO}, [deab]_{LRU}\)
\end{itemize}

- \((h, h)\) to \((m, m)\) via \(e\)
- \((h, h)\) to \((m, m)\) via \(c\) and \(d\)
- \((h, m)\) to \((m, h)\) via \(c\) and \(d\)

Reinhard Wilhelm
Caches in WCET Analysis
January 18, 2011
Finite Quotient Transition System

Merging $\approx$-equivalent states yields a finite quotient transition system:

\[
\begin{align*}
&[abcd]_{\text{FIFO}}; [abcd]_{\text{LRU}} \\
&[abcd]_{\text{FIFO}}; [dabc]_{\text{LRU}} \\
&[eabc]_{\text{FIFO}}; [ceda]_{\text{LRU}} \\
&[eabc]_{\text{FIFO}}; [edab]_{\text{LRU}}
\end{align*}
\]

\[(h, h), (m, m), (h, h), (m, h), (m, h), (h, m)\]
Competitive Ratio = Maximum Cycle Ratio

Competitive miss ratio =

maximum ratio of misses in policy \( P \) to misses in policy \( Q \)

\[
\begin{align*}
(0, 0) & \quad \circlearrowleft
\end{align*}
\]

\[
\begin{align*}
(1, 1) & \quad \structure
(0, 0) & \quad \downarrow
(1, 0)
\end{align*}
\]

\[
\begin{align*}
(1, 1) & \quad \downarrow
(0, 1)
\end{align*}
\]
Competitive Ratio = Maximum Cycle Ratio

Competitive miss ratio =
maximum ratio of misses in policy $P$ to misses in policy $Q$

Maximum cycle ratio $= \frac{0+1+1}{0+1+0} = 2$
Tool Implementation

- Implemented in Java, called Relacs
- Interface for replacement policies
- Fully automatic
- Provides example sequences for competitive ratio and constant
- Analysis usually practically feasible up to associativity 8
  - limited by memory consumption
  - depends on similarity of replacement policies

Online version:
http://rw4.cs.uni-sb.de/~reineke/relacs
Generalizations

Identified patterns and proved generalizations by hand. Aided by example sequences generated by tool.
Generalizations

Identified patterns and proved generalizations by hand. Aided by example sequences generated by tool.

Previously unknown facts:

\[
\text{PLRU}(k) \text{ is } (1, 0) \text{ comp. rel. to } \text{LRU}(1 + \log_2 k),
\]

\[\rightarrow \text{LRU-}must\text{-analysis can be used for PLRU}\]
Generalizations

Identified patterns and proved generalizations by hand. Aided by example sequences generated by tool.

Previously unknown facts:

$$\text{PLRU}(k) \text{ is } (1, 0) \text{ comp. rel. to } \text{LRU}(1 + \log_2 k),$$

$$\text{LRU}-\text{must-}\text{analysis can be used for PLRU}$$

$$\text{FIFO}(k) \text{ is } \left(\frac{1}{2}, \frac{k-1}{2}\right) \text{ hit-comp. rel. to } \text{LRU}(k), \text{ whereas}$$

$$\text{LRU}(k) \text{ is } (0, 0) \text{ hit-comp. rel. to } \text{FIFO}(k), \text{ but}$$
Generalizations

Identified patterns and proved generalizations by hand. Aided by example sequences generated by tool.

Previously unknown facts:

\[ \text{PLRU}(k) \text{ is } (1, 0) \text{ comp. rel. to } \text{LRU}(1 + \log_2 k), \]
\[ \rightarrow \text{LRU-}must\text{-analysis can be used for PLRU} \]

\[ \text{FIFO}(k) \text{ is } \left( \frac{1}{2}, \frac{k-1}{2} \right) \text{ hit-comp. rel. to } \text{LRU}(k), \text{ whereas} \]
\[ \text{LRU}(k) \text{ is } (0, 0) \text{ hit-comp. rel. to FIFO}(k), \text{ but} \]

\[ \text{LRU}(2k - 1) \text{ is } (1, 0) \text{ comp. rel. to FIFO}(k), \text{ and} \]
\[ \text{LRU}(2k - 2) \text{ is } (1, 0) \text{ comp. rel. to MRU}(k). \]
\[ \rightarrow \text{LRU-}may\text{-analysis can be used for FIFO and MRU} \]
\[ \rightarrow \text{optimal with respect to predictability metric Evict} \]
Generalizations

Identified patterns and proved generalizations by hand. Aided by example sequences generated by tool.

Previously unknown facts:

PLRU\((k)\) is \((1, 0)\) comp. rel. to \(\text{LRU}(1 + \log_2 k)\),

\[\rightarrow \text{LRU-}must\text{-analysis can be used for PLRU}\]

FIFO\((k)\) is \(\left(\frac{1}{2}, \frac{k-1}{2}\right)\) hit-comp. rel. to \(\text{LRU}(k)\), whereas

\(\text{LRU}(k)\) is \((0, 0)\) hit-comp. rel. to \(\text{FIFO}(k)\), but

\(\text{LRU}(2k - 1)\) is \((1, 0)\) comp. rel. to \(\text{FIFO}(k)\), and

\(\text{LRU}(2k - 2)\) is \((1, 0)\) comp. rel. to \(\text{MRU}(k)\).

\[\rightarrow \text{LRU-}may\text{-analysis can be used for FIFO and MRU}\]

\[\rightarrow \text{optimal with respect to predictability metric Evict}\]

FIFO-\(may\text{-analysis used in the analysis of the branch target buffer of the MOTOROLA PowerPC 56x.}\)
Outline

1. Caches

2. Cache Analysis for Least-Recently-Used

3. Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4. Summary
Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.
Measurement-Based Timing Analysis

- Run program on a number of inputs and initial states.
- Combine measurements for basic blocks to obtain WCET estimation.
- Sensitivity Analysis demonstrates this approach may be dramatically wrong.
Influence of Initial Cache State

Definition (Miss sensitivity)
Policy $\mathbf{P}$ is $(k, c)$-miss-sensitive if

$$m_{\mathbf{P}}(q, s) \leq k \cdot m_{\mathbf{P}}(q', s) + c$$

for all access sequences $s \in M^*$ and cache-set states $q, q' \in C^\mathbf{P}$. 

variation due to
initial cache state

BCET \hspace{2cm} WCET \hspace{2cm} upper bound \hspace{2cm} execution time
Sensitivity Results

<table>
<thead>
<tr>
<th>Policy</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>1,2</td>
<td>1,3</td>
<td>1,4</td>
<td>1,5</td>
<td>1,6</td>
<td>1,7</td>
<td>1,8</td>
</tr>
<tr>
<td>FIFO</td>
<td>2,2</td>
<td>3,3</td>
<td>4,4</td>
<td>5,5</td>
<td>6,6</td>
<td>7,7</td>
<td>8,8</td>
</tr>
<tr>
<td>PLRU</td>
<td>1,2</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td>∞</td>
</tr>
<tr>
<td>MRU</td>
<td>1,2</td>
<td>3,4</td>
<td>5,6</td>
<td>7,8</td>
<td>MEM</td>
<td>MEM</td>
<td>MEM</td>
</tr>
</tbody>
</table>

- LRU is optimal. Performance varies in the least possible way.
- For FIFO, PLRU, and MRU the number of misses may vary strongly.
- Case study based on simple model of execution time by Hennessy and Patterson (2003): WCET may be 3 times higher than a measured execution time for 4-way FIFO.
Outline

1 Caches

2 Cache Analysis for Least-Recently-Used

3 Beyond Least-Recently-Used
   - Predictability Metrics
   - Relative Competitiveness
   - Sensitivity – Caches and Measurement-Based Timing Analysis

4 Summary
Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.
...requires context-sensitivity for precision.
Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

→ LRU is the most predictable policy.
Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

→ LRU is the most predictable policy.

Relative Competitiveness

...allows to derive guarantees on cache performance,

...yields first may-analyses for FIFO and MRU.
Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

→ LRU is the most predictable policy.

Relative Competitiveness

...allows to derive guarantees on cache performance,

...yields first may-analyses for FIFO and MRU.

Sensitivity Analysis

...determines the influence of initial state on cache performance.
Summary

Cache Analysis for Least-Recently-Used

...efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

...requires context-sensitivity for precision.

Predictability Metrics

...quantify the predictability of replacement policies.

→ LRU is the most predictable policy.

Relative Competitiveness

...allows to derive guarantees on cache performance,

...yields first \textit{may}-analyses for FIFO and MRU.

Sensitivity Analysis

...determines the influence of initial state on cache performance.

Thank you for your attention!
Summary

Cache Analysis for Least-Recently-Used

... efficiently represents sets of cache states by bounding the age of memory blocks from above and below.

... requires context-sensitivity for precision.

Predictability Metrics

... quantify the predictability of replacement policies.

→ LRU is the most predictable policy.

Relative Competitiveness

... allows to derive guarantees on cache performance,

... yields first may-analyses for FIFO and MRU.

Sensitivity Analysis

... determines the influence of initial state on cache performance.

Thank you for your attention!


Most-Recently-Used – MRU

MRU-bits record whether line was recently used

\[
\begin{align*}
&[abcd]_{0101} \Rightarrow b,d \\
&e \\
&[ebcd]_{1101} \Rightarrow e,b,d \\
&c \\
&[ebcd]_{0010} \Rightarrow c \\
&\longrightarrow \text{Never converges}
\end{align*}
\]
Pseudo-LRU – PLRU

Initial cache-set state: \([a, b, c, d]_{110}\).

After a miss on \(e\). State: \([a, b, e, d]_{011}\).

After a hit on \(a\). State: \([a, b, e, d]_{111}\).

After a miss on \(f\). State: \([a, b, e, f]_{010}\).

Hit on \(a\) “rejuvenates” neighborhood; “saves” \(b\) from eviction.
May- and Must-Information

\[ May^P(s) := \bigcup_{p \in C^P} CC_p(update_P(p, s)) \]

\[ Must^P(s) := \bigcap_{p \in C^P} CC_p(update_P(p, s)) \]

\[ may^P(n) := \left| May^P(s) \right|, \text{ where } s \in S \neq \emptyset \subset M^*, |s| = n \]

\[ must^P(n) := \left| Must^P(s) \right|, \text{ where } s \in S \neq \emptyset \subset M^*, |s| = n \]

\[ S \neq : \text{ set of finite access sequences with pairwise different accesses} \]
Definitions of Metrics

\[
\begin{align*}
\text{Evict}^P & := \min \left\{ n \mid \text{may}^P(n) \leq n \right\}, \\
\text{Fill}^P & := \min \left\{ n \mid \text{must}^P(n) = k \right\},
\end{align*}
\]

where \( k \) is \( P \)'s associativity.
Let $P(k)$ be $(1, 0)$-miss-competitive relative to policy $Q(l)$, then

(i) $Evict^P(k) \geq Evict^Q(l)$,

(ii) $mls^P(k) \geq mls^Q(l)$. 

Let $l$ be the smallest associativity, such that $LRU(l)$ is $(1, 0)$-miss-competitive relative to $P(k)$. Then

$$\text{Alt-Evict}^P(k) = l.$$ 

Let $l$ be the greatest associativity, such that $P(k)$ is $(1, 0)$-miss-competitive relative to $LRU(l)$. Then

$$\text{Alt-mls}^P(k) = l.$$
Size of Transition System

\[ 2^{l+l'} \cdot \sum_{i=0}^{k} \binom{k}{i} \cdot \sum_{i'=0}^{k'} \binom{k'}{i'} \cdot \sum_{j=0}^{\min\{i,i'\}} \left( \binom{i}{j} \binom{i'}{j} j! \right) \]

status bits of P and Q

non-empty lines in P

non-empty lines in Q

number of overlappings in non-empty lines

\[ \leq k! \cdot k'! \sum_{j=0}^{\min\{k,k'\}} \frac{1}{(k-j)!j!(k'-j)!} \]

\[ \leq k! \cdot k'! \sum_{j=0}^{\infty} \frac{1}{j!} = e \cdot k! \cdot k'! \]

This can be bounded by

\[ 2^{l+l'+k+k'} \leq \|(C_k^l \times C_{k'}^{l'})\| \approx | \leq 2^{l+l'+k+k'} \cdot \frac{e \cdot k! \cdot k'}{\min\{k,k'\}} \]

bound on number of overlappings
Compatible States

\[ i^P = [⊥⊥⊥⊥]_P \approx i^Q = [⊥⊥⊥⊥]_Q \]

update\(_P\)(i^P, s) \approx update\(_Q\)(i^Q, s)

\( p \approx q \)
Let $P$ be $(1, 0)$-competitive relative to $Q$, then

$$m_P(p, \langle x \rangle) = 1 \implies m_Q(q, \langle x \rangle) = 1$$
(1, 0)-Competitiveness and May/Must-Analyses

\[
\forall p \in P : m_p(p, \langle x \rangle) = 1 \quad \Rightarrow \quad \forall q \in Q : m_q(q, \langle x \rangle) = 1
\]
Case Study: Impact of Sensitivity

- Simple model of execution time from Hennessy & Patterson (2003)
- $CPI_{hit} = $ Cycles per instruction assuming cache hits only
- $\frac{\text{Memory accesses}}{\text{Instruction}}$ including instruction and data fetches

\[
\frac{T_{wc}}{T_{meas}} = \frac{CPI_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{wc} \times \text{Miss penalty}}{CPI_{hit} + \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate}_{meas} \times \text{Miss penalty}}
\]

\[
= \frac{1.5 + 1.2 \times 0.20 \times 50}{1.5 + 1.2 \times 0.05 \times 50} = \frac{13.5}{4.5} = 3
\]
Evolution of May- and Must-Information for LRU

must/may information

\[ mls \]
\[ evict_{HM} \]
\[ fill_{HM} \]

\[ k \]

\[ \#accesses \]
Evolution of May- and Must-Information for FIFO

must/may information

\[ \text{mls} \quad \text{evict}_{HM} \quad \text{fill}_{HM} \quad \text{# accesses} \]

\[ 1 \quad 5 \quad 10 \quad 15 \quad 20 \quad 23 \]

\[ 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \]

Reinhard Wilhelm

Caches in WCET Analysis

January 18, 2011
Evolution of May- and Must-Information for PLRU

must/may information

\[\text{must/may information}\]

\[c\]

\[k\]

\[\text{#accesses}\]

\[m_{ls}\]

\[\text{evict}_{\text{HM}}\]

\[\text{fill}_{\text{HM}}\]
Evolution of May- and Must-Information for MRU

Evolution of may/must-information 8-way MRU:

- \( c \)
- \( k-1 \)
- \( 2k-2 \)

Reinhard Wilhelm

Caches in WCET Analysis

January 18, 2011