

Precise and Efficient FIFO-Replacement Analysis Based on Static Phase Detection

Daniel Grund¹ Jan Reineke²

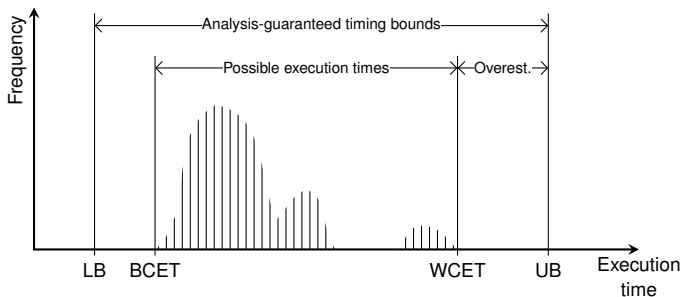
¹Saarland University, Saarbrücken, Germany

²University of California, Berkeley, USA

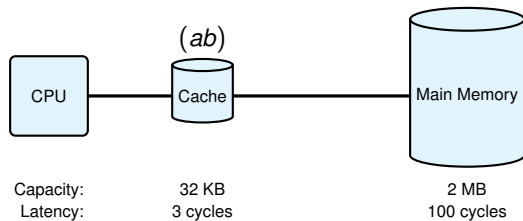
Euromicro Conference on Real-Time Systems 2010



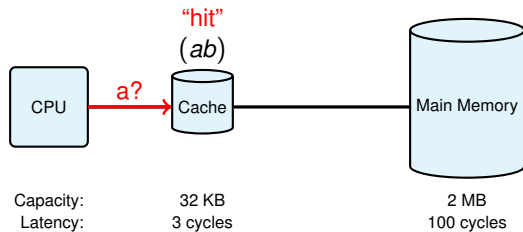
- 1 Introduction and Problem
 - Timing Analysis
 - Cache Analysis
 - Challenge FIFO Replacement
- 2 Predicting Hits for FIFO
 - Idea and Theorem
 - Must Analysis
 - Efficient Implementation
- 3 Paper Contents
- 4 Evaluation
 - Related Work
 - Analysis Precision
- 5 Summary



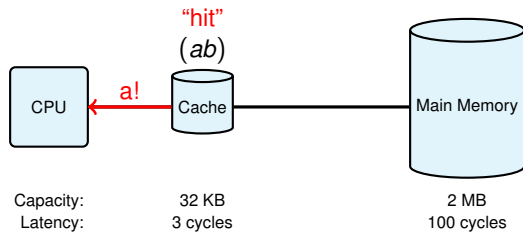
- Need to bound execution time of programs
- Execution time influenced by architectural features
 - ▶ pipelines, caches, branch prediction, ...
- Need to analyze behavior of architectural components



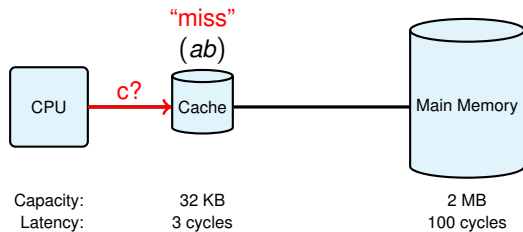
- Caches transparently buffer memory blocks



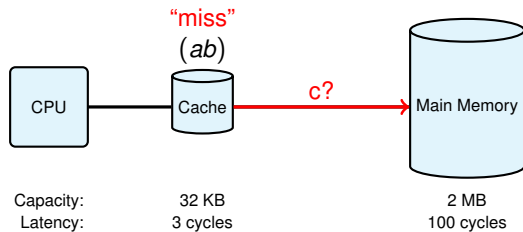
- Caches transparently buffer memory blocks



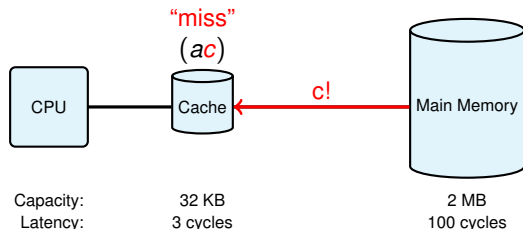
- Caches transparently buffer memory blocks



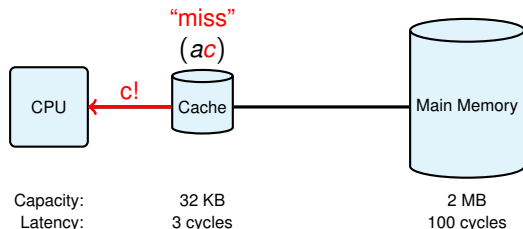
- Caches transparently buffer memory blocks



- Caches transparently buffer memory blocks



- Caches transparently buffer memory blocks
- Replacement policy *dynamically* decides which element to replace
 - LRU least recently used
 - PLRU pseudo LRU
 - FIFO first-in first-out



- Caches transparently buffer memory blocks
- Replacement policy *dynamically* decides which element to replace
 - LRU least recently used
 - PLRU pseudo LRU
 - FIFO first-in first-out

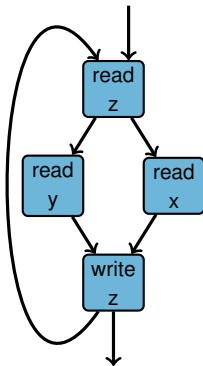
- Derive **approximations to cache contents** at each program point
- in order to **classify memory accesses** as cache hits or cache misses

Must-information

- **Under**approximation of cache contents
- Used to soundly classify cache **hits**

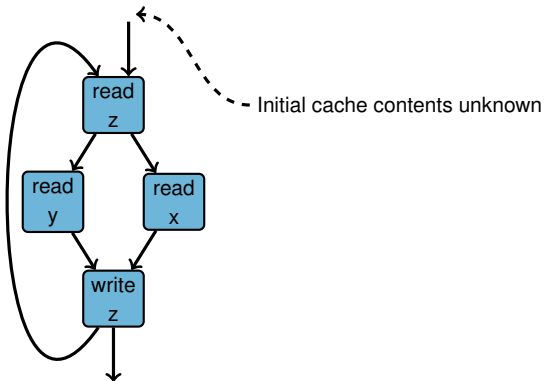
May-information

- **Over**approximation of cache contents
- Used to soundly classify cache **misses**



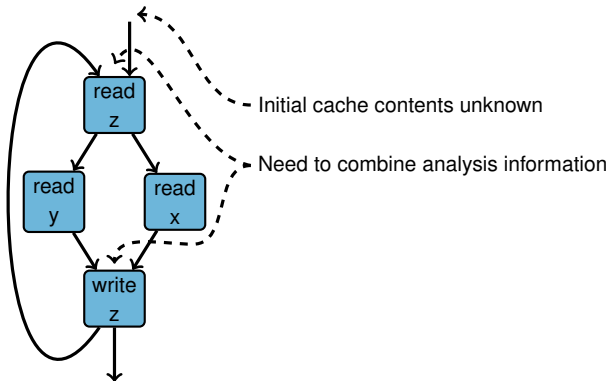
Static Cache Analysis

Challenges



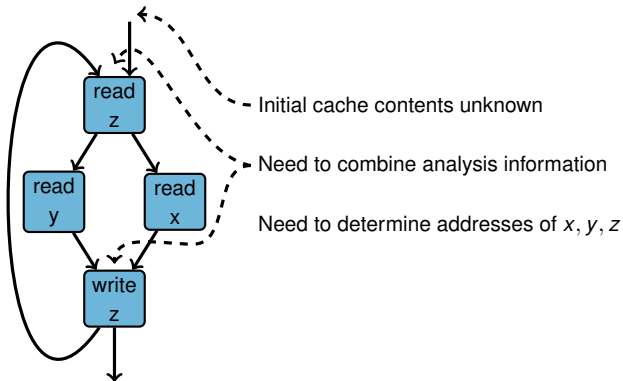
Static Cache Analysis

Challenges



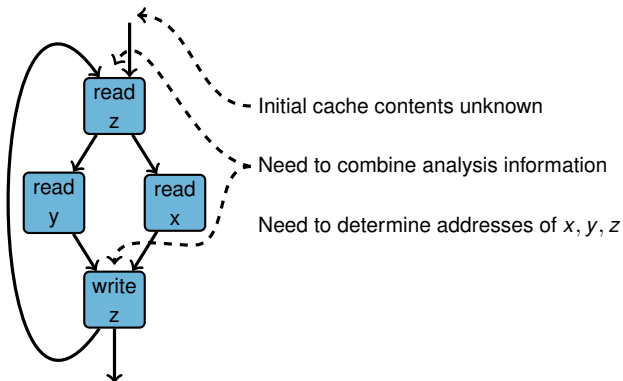
Static Cache Analysis

Challenges



Static Cache Analysis

Challenges



1 Approximate accessed addresses by value analysis (not this talk)

2 Approximate cached contents by replacement analysis

⇒ Cache analysis = value analysis \oplus replacement analysis

- FIFO cache of size k :

$$\begin{array}{ccc} \text{last-in} & & \text{first-in} \\ \downarrow & & \downarrow \\ [b_1, \dots, b_k] \in \mathcal{Q}_k := \mathcal{B}_{\perp}^k \end{array}$$

- Example updates:

$$[d, c, b, a] \xrightarrow[\text{hit}]{c} [d, c, b, a]$$

$$[d, c, b, a] \xrightarrow[\text{miss}]{e} [e, d, c, b]$$

FIFO Replacement Analysis

Why Predicting Hits is Difficult

- Take a set of blocks B that **does fit into** a cache q
- For example, $B = \{a, b, e\}$ and $k = 4$. $|B| \leq k$.
- Access all blocks in B :

$$q \xrightarrow{\langle a, b, e \rangle} q'$$

- Must all accessed blocks be cached? $\forall q : B \subseteq q'$?

FIFO Replacement Analysis

Why Predicting Hits is Difficult

- Take a set of blocks B that **does fit into** a cache q
- For example, $B = \{a, b, e\}$ and $k = 4$. $|B| \leq k$.
- Access all blocks in B :

$$q \xrightarrow{\langle a, b, e \rangle} q'$$

- Must all accessed blocks be cached? $\forall q : B \subseteq q'$? **No.**

$$[d, c, b, a] \xrightarrow[\text{hit}]{a} [d, c, b, a] \xrightarrow[\text{hit}]{b} [d, c, b, a] \xrightarrow[\text{miss}]{e} [e, d, c, b] \not\supseteq a$$

Observation

After accessing a set of “fitting” blocks,
 not all of them must be cached.

FIFO Replacement Analysis

Why Predicting Misses is Difficult

- Take a set of blocks B that **does not fit into** a cache q
- For example, $B = \{a, b, c, d, e, f\}$ and $k = 4$. $|B| \geq k$.
- Access all blocks in B :

$$q \xrightarrow{\langle a, b, c, d, e, f \rangle} q'$$

- Must all non-accessed blocks be evicted? $\forall q : q' \subseteq B?$

FIFO Replacement Analysis

Why Predicting Misses is Difficult

- Take a set of blocks B that **does not fit into** a cache q
- For example, $B = \{a, b, c, d, e, f\}$ and $k = 4$. $|B| \geq k$.
- Access all blocks in B :

$$q \xrightarrow{\langle a, b, c, d, e, f \rangle} q'$$

- Must all non-accessed blocks be evicted? $\forall q : q' \subseteq B$? **No.**

$$[x, c, b, a] \xrightarrow[\text{hits}]{\langle a, b, c \rangle} [x, c, b, a] \xrightarrow[\text{misses}]{\langle d, e, f \rangle} [f, e, d, x] \ni x$$

Observation

After accessing a set of “non-fitting” blocks, other blocks may still be cached.

- 1 Introduction and Problem
 - Timing Analysis
 - Cache Analysis
 - Challenge FIFO Replacement
- 2 Predicting Hits for FIFO
 - Idea and Theorem
 - Must Analysis
 - Efficient Implementation
- 3 Paper Contents
- 4 Evaluation
 - Related Work
 - Analysis Precision
- 5 Summary

To the point: Anticipation & Idea

- Considering repeated accesses to “fitting” blocks B helps:

$$B = \{a, b, c\}$$

$$s = \langle a, b, b, c, b, b, a, c, c, a, b, \dots \rangle$$

Eventually, all blocks in B must be cached.

- Need to detect repetitions
- Partition access sequence s into phases

Definition (Phase)

A **B -phase** is an access sequence s such that the set of accessed blocks $A(s) = B$.

$$\langle \underbrace{a, b, b, c}_{\{a,b,c\}\text{-phase}}, \underbrace{b, b, a, c}_{\{a,b,c\}\text{-phase}}, \dots \rangle$$

Lemma (Single Phase)

Let s be a B -phase and $|B| \leq k$.

$$\forall q \in Q_k, q \xrightarrow{s} q' : \quad \vee$$

Lemma (Single Phase)

Let s be a B -phase and $|B| \leq k$.

$$\forall q \in \mathcal{Q}_k, q \xrightarrow{s} q' : B \subseteq q' \vee$$

1 Either all blocks already cached:

- ▶ $B \subseteq q \Rightarrow$ only hits in $s \Rightarrow B \subseteq q'$

Predicting Hits by Detecting Phases

Lemma (Single Phase)

Let s be a B -phase and $|B| \leq k$.

$$\forall q \in \mathcal{Q}_k, q \xrightarrow{s} q' : B \subseteq q' \vee C_1(q') \subseteq B$$

1 Either all blocks already cached:

▶ $B \subseteq q \Rightarrow$ only hits in $s \Rightarrow B \subseteq q'$

2 Or not:

▶ $B \not\subseteq q \Rightarrow$ at least one miss $s \Rightarrow C_1(q') \subseteq B$

▶ $[d, c, b, a] \xrightarrow{\langle a, b, e \rangle} [\underbrace{e}_{C_1(q') = \{e\} \subseteq B}, d, c, b]$

Predicting Hits by Detecting Phases

Theorem (Multiple Phases)

Let s_i be B -phases and $|B| \leq k$ and $s = s_1 \circ \dots \circ s_j$

$$\forall q \in \mathcal{Q}_k, q \xrightarrow{s} q' : \quad B \subseteq q' \vee C_j(q') \subseteq B$$

- 1 For each individual phase the lemma applies
- 2 Misses, if any, accumulate in last-in positions $C_j(q')$

$$[d, c, b, a] \xrightarrow{\langle a, b, e \rangle} [\underbrace{e}_{C_1 \subseteq B}, d, c, b] \xrightarrow{\langle b, a, e \rangle} [\underbrace{a, e}_{C_2 \subseteq B}, d, c] \xrightarrow{\langle a, b, e \rangle} [\underbrace{b, a, e, d}_{C_3 \subseteq B}]$$

Predicting Hits by Detecting Phases

Theorem (Multiple Phases)

Let s_i be B -phases and $|B| \leq k$ and $s = s_1 \circ \dots \circ s_j$

$$\forall q \in \mathcal{Q}_k, q \xrightarrow{s} q' : \quad B \subseteq q' \vee C_j(q') \subseteq B$$

- 1 For each individual phase the lemma applies
- 2 Misses, if any, accumulate in last-in positions $C_j(q')$

$$[d, c, b, a] \xrightarrow{\langle a, b, e \rangle} [\underbrace{e}_{C_1 \subseteq B}, d, c, b] \xrightarrow{\langle b, a, e \rangle} [\underbrace{a, e}_{C_2 \subseteq B}, d, c] \xrightarrow{\langle a, b, e \rangle} [\underbrace{b, a, e}_{C_3 \subseteq B}, d]$$

- Corollary: After $|B|$ B -phases, all blocks in B must be cached

The Must Analysis

How to Count Phases

- For phase blocks B , the analysis maintains:
 - P phase progress, blocks already accessed in current phase
 - pc phase counter, number of detected B -phases
- Predicts hits for blocks in B if $pc = |B|$

Example for $B = \{a, b\}$

s	a	b	b	b	a	b			
P	\emptyset	$\{a\}$	$\{a, b\}$	\emptyset	$\{b\}$	$\{b\}$	$\{a, b\}$	\emptyset	$\{b\}$
pc	0	0	0	1	1	1	1	2	2
Hit									Hit

The Must Analysis

Dependency on Future Accesses

- Need $|B|$ B -phases to predict hits for blocks in B
- How to choose B ?
- After observing $\langle a, b, c \rangle$ it makes sense trying to detect
 - ▶ 2 further $\{a, b, c\}$ -phases
 - ▶ 1 further $\{b, c\}$ -phase
 - ▶ 0 further $\{c\}$ -phases
- Optimal B depends on future accesses
 - ▶ $\langle a, b, c, \quad a, b, c, a, b, c, a, b, c \rangle$
 - ▶ $\langle a, b, c, \quad b, c, b, c, b, c, b, c \rangle$

The Must Analysis

Resolving the Dependency

- Perform multiple analyses for different B sets
- For which?
- $|B|$ already determines sensible contents of B
- For $|B| = 2$, after $\langle a, b, c \rangle$
 - ▶ already detected 1 $\{b, c\}$ -phase
 - ▶ no advantage in trying to detect 2 $\{x, y\}$ -phases

⇒ Perform k analyses for different B sets

- ▶ for each phase size $n = 1 \dots k$
- ▶ B_n consists of the n most-recently-used blocks

The Must Analysis

Subanalyses for $n = 1 \dots 3$

a b c c b c a a c a b a

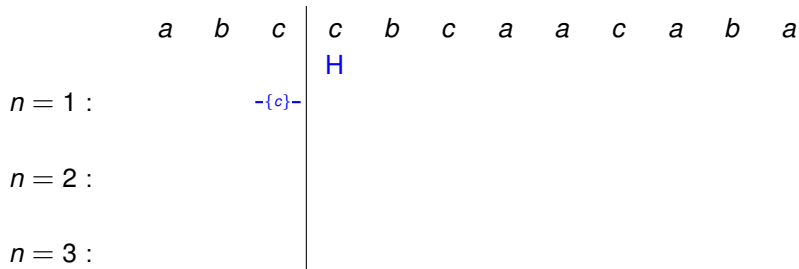
$n = 1 :$

$n = 2 :$

$n = 3 :$

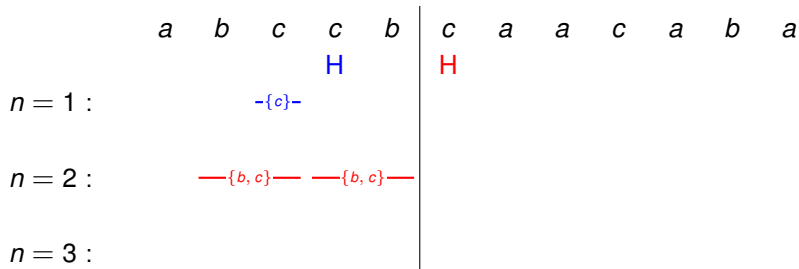
The Must Analysis

Subanalyses for $n = 1 \dots 3$



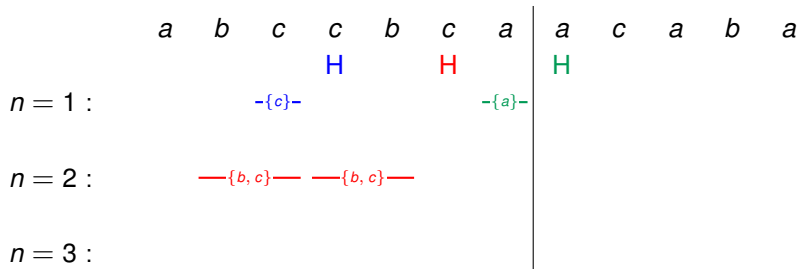
The Must Analysis

Subanalyses for $n = 1 \dots 3$



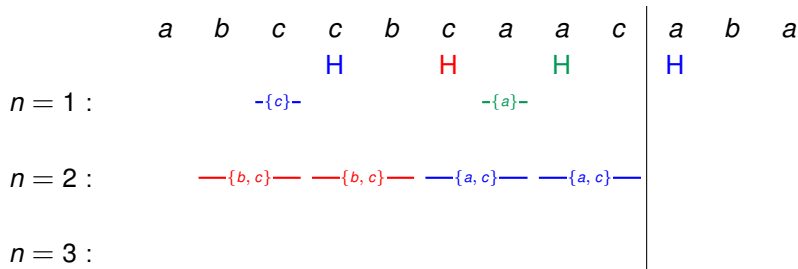
The Must Analysis

Subanalyses for $n = 1 \dots 3$



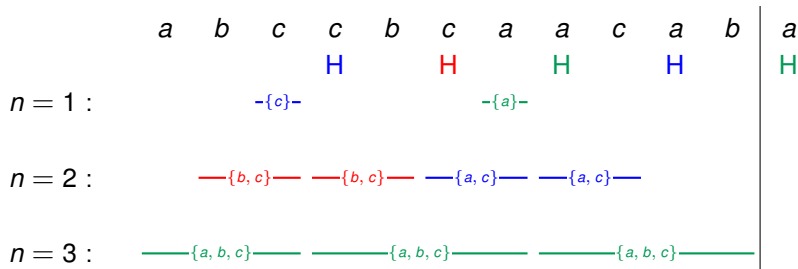
The Must Analysis

Subanalyses for $n = 1 \dots 3$



The Must Analysis

Subanalyses for $n = 1 \dots 3$



Efficient Implementation

Observation

- For $n = 1 \dots k$, analysis needs to maintain:
 - ▶ phase blocks $B_n \in 2^{\mathcal{B}}$
 - ▶ phase progress $P_n \in 2^{\mathcal{B}}$
 - ▶ phase counter $pc_n \in \mathbb{N}$

- Phase blocks B_n are the n most-recently-used blocks
- ⇒ For $i < j : B_i \subseteq B_j$
- ⇒ Encode all B_n in a single LRU-stack

- For all $i : P_i \subseteq B_i$
- ⇒ Encode all P_n as “pointers” into the stack

Efficient Implementation

Encoding

- For phase blocks B_n :
 - ▶ pc_n complete B_n -phases were detected
 - ▶ current phase progress is B_{pp_n}

B_1	pc_1, pp_1
$B_2 \setminus B_1$	pc_2, pp_2
$B_3 \setminus B_2$	pc_3, pp_3
$B_4 \setminus B_3$	pc_4, pp_4

B_1	$\{b\}$		=	$\{b\}$	1, 0
P_1	\emptyset			$\{c\}$	2, 1
pc_1	1			$\{a\}$	1, 2
B_2	$\{b, c\}$			\emptyset	0, 3
P_2	$\{b\}$				
pc_2	2				
B_3	$\{a, b, c\}$				
P_3	$\{b, c\}$				
pc_3	1				

- 1 Introduction and Problem
 - Timing Analysis
 - Cache Analysis
 - Challenge FIFO Replacement
- 2 Predicting Hits for FIFO
 - Idea and Theorem
 - Must Analysis
 - Efficient Implementation
- 3 Paper Contents
- 4 Evaluation
 - Related Work
 - Analysis Precision
- 5 Summary

- So far, we have seen parts of the must-analysis
- The paper contains, for must- and may-analysis,
 - ▶ basic theorem
 - ▶ generalization to arbitrary control-flow
 - ▶ formalization as abstract interpretation

- 1 Introduction and Problem
 - Timing Analysis
 - Cache Analysis
 - Challenge FIFO Replacement
- 2 Predicting Hits for FIFO
 - Idea and Theorem
 - Must Analysis
 - Efficient Implementation
- 3 Paper Contents
- 4 Evaluation
 - Related Work
 - Analysis Precision
- 5 Summary

Before '97 LRU analyses

LCTRTS'97 Precise and efficient must- and may-analysis for LRU [1]

LCTES'08 Generic analyses for FIFO and PLRU [2]

SAS'09 Cache analysis framework and FIFO analysis [3]

WCET'10 Toward precise analysis for PLRU [4]

ECRTS'10 Precise and efficient must- and may-analysis for FIFO

- Analyses:

- HAM Must-analysis of SAS'09

- RC Generic analyses of LCTES'08

- PD Phase detecting analyses

- Collecting semantics:

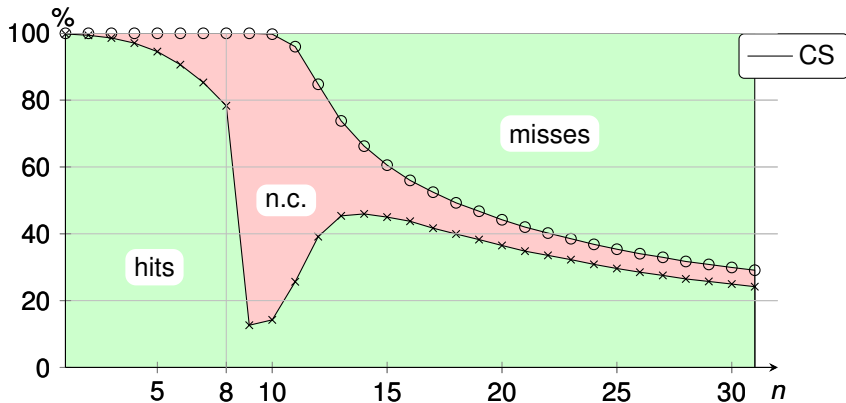
- CS Limit for any static analysis

- Spectrum of synthetic benchmarks:

- ▶ Random access sequences and program fragments
 - ▶ With varying locality

Evaluation Results

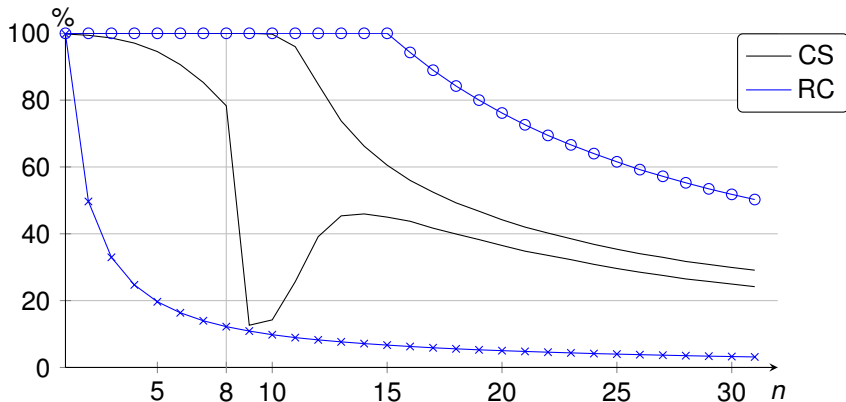
$k=8$, random sequences



- n is number of distinct elements that get accessed
- Average guaranteed hit- and miss-rates

Evaluation Results

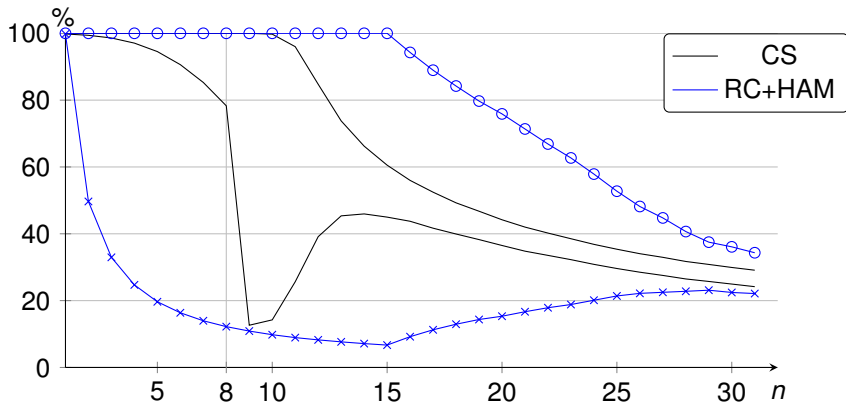
$k=8$, random sequences



- n is number of distinct elements that get accessed
- Average guaranteed hit- and miss-rates

Evaluation Results

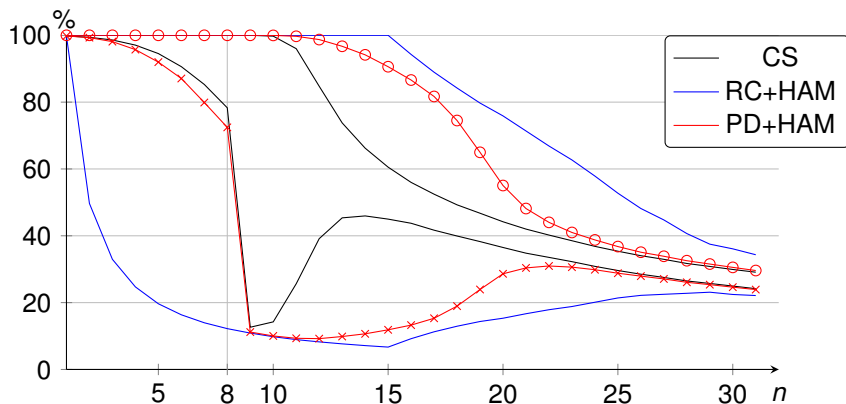
$k=8$, random sequences



- n is number of distinct elements that get accessed
- Average guaranteed hit- and miss-rates

Evaluation Results

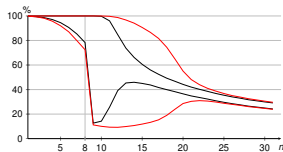
$k=8$, random sequences



- n is number of distinct elements that get accessed
- Average guaranteed hit- and miss-rates

- 1 Introduction and Problem
 - Timing Analysis
 - Cache Analysis
 - Challenge FIFO Replacement
- 2 Predicting Hits for FIFO
 - Idea and Theorem
 - Must Analysis
 - Efficient Implementation
- 3 Paper Contents
- 4 Evaluation
 - Related Work
 - Analysis Precision
- 5 Summary

Precise and Efficient FIFO-Replacement Analysis based on Static Phase Detection



$$\begin{array}{l|l}
 B_1 & \{b\} \\
 P_1 & \emptyset \\
 \rho_{C_1} & 1 \\
 \hline
 B_2 & \{b, c\} \\
 P_2 & \{b\} \\
 \rho_{C_2} & 2 \\
 \hline
 B_3 & \{a, b, c\} \\
 P_3 & \{b, c\} \\
 \rho_{C_3} & 1
 \end{array}
 =
 \begin{array}{l|l}
 \{b\} & 1,0 \\
 \{c\} & 2,1 \\
 \{a\} & 1,2 \\
 \emptyset & 0,3
 \end{array}$$

Precise and Efficient FIFO-Replacement Analysis based on Static Phase Detection

$$\underbrace{\{a, b, c\}}_{\{b, c\}\text{-phase}}\text{-phase}, \underbrace{\{c, b, b, a, \dots\}}_{\{b, c\}\text{-phase}}\text{-phase}$$

■ Two theorems on FIFO-contents

- ▶ bound on number of phases
- ▶ must be cached / evicted

■ Must- and may-analysis

- ▶ static phase detection
- ▶ multiple sub-analyses



C. Ferdinand

Cache Behaviour Prediction for Real-Time Systems

PhD Thesis, Saarland University, 1997



J. Reineke and D. Grund

Relative competitive analysis of cache replacement policies

LCTES 2008



D. Grund and J. Reineke

Abstract Interpretation of FIFO Replacement

SAS 2009



D. Grund and J. Reineke

Toward Precise PLRU Cache Analysis

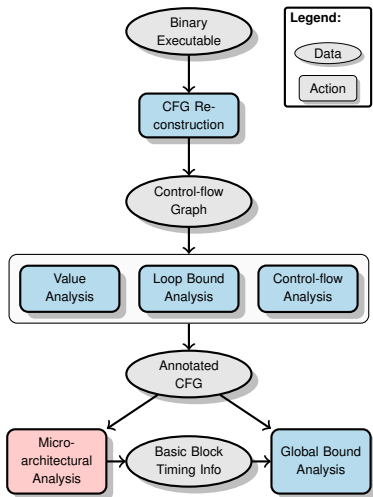
WCET 2010

- Analyses directed at worst-case execution-time analysis
 - Mueller By “static cache simulation”
 - Li By integer linear programming
 - Ferdinand By abstract interpretation

- Other than that
 - Ghosh Cache Miss Equations, loop nests
 - Chatterjee Exact model of cache behavior for loop nests

- All for LRU caches only

Static Timing-Analysis Framework



Micro-architectural analysis

- models pipeline, caches, buses, etc.
- derives bounds on BB exec. times
- is an abstract interpretation with a huge domain
- is the computationally most expensive module

- Any buffer with transparent FIFO replacement:

- ▶ Individual cache sets of instruction or data caches (I\$, D\$)
- ▶ Branch target buffers (BTB, BTIC)
- ▶ Translation lookaside buffers (TLB)

- Instances:

I\$ D\$ ARM 1136, 1156, 1176, 920T, 922T, 926EJ-S ($k \in \{2, 4, 64\}$)

I\$ D\$ Marvell (Intel) XScale(s) ($k = 32$)

BTB Freescale (Motorola) MPC 56x, 7450-Family ($k \in \{4, 8\}$)

...

Must Analysis

Full Example for $k = 3$

- For $1 \leq n \leq k$ maintain $B_n, P_n, \rho C_n$

Example

s		a	b	c	c	b	c	a
B_1	\emptyset	$\{a\}$	$\{b\}$	$\{c\}$	$\{c\}$	$\{b\}$	$\{c\}$	$\{a\}$
P_1	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
ρC_1	0	1	1	1	1	1	1	1
B_2	\emptyset	$\{a\}$	$\{a, b\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{b, c\}$	$\{a, c\}$
P_2	\emptyset	$\{a\}$	\emptyset	\emptyset	$\{c\}$	\emptyset	$\{c\}$	\emptyset
ρC_2	0	0	1	1	1	2	2	1
B_3	\emptyset	$\{a\}$	$\{a, b\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$	$\{a, b, c\}$
P_3	\emptyset	$\{a\}$	$\{a, b\}$	\emptyset	$\{c\}$	$\{b, c\}$	$\{b, c\}$	\emptyset
ρC_3	0	0	0	1	1	1	1	2
Hit					Hit		Hit	

- Analysis domain is $Lru_k^{\leq} \times PC_k \times PP_k \subset (2^B)^k \times \mathbb{N}^k \times \mathbb{N}^k$
 - Lru_k^{\leq} LRU must-analysis, under-approximates accessed blocks
 - PC_k lower bounds on number of phases
 - PP_k lower bounds on phase progress
- Reuse abstract transformer and join of Lru_k^{\leq}
- Define appropriately for PC_k and PP_k

- Similar to must-analysis
- Difference: Phases may be of **different lengths and contents**

Theorem (Multiple Phases)

$s = s_1 \circ \dots \circ s_j, \forall i : |A(s_i)| = n_i \geq k:$

$$\forall q \in \mathcal{Q}_k, q \xrightarrow{s} q' : C_{\sum_{i=1}^j (n_i - k + 1)}(q') \subseteq A(s) = \bigcup_i A(s_i)$$

- More simultaneous sub-analyses
- Similar implementation employing LRU may-analysis