

Toward Precise PLRU Cache Analysis

Daniel Grund¹ Jan Reineke²

¹Saarland University, Saarbrücken, Germany

²University of California, Berkeley, USA

Workshop on Worst-Case Execution-Time Analysis 2010



1 Introduction and Problem

2 Analysis Challenges

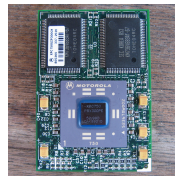
- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

4 Summary

■ Cache replacement policies

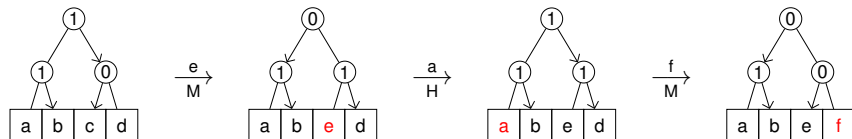
- ▶ Least-recently used
- ▶ Pseudo-LRU
- ▶ First-in First-out
- ▶ Pseudo Round-Robin
- ▶ “Random”



■ PLRU

- ▶ is flying in MPC603E
- ▶ is flying in MPC755
- ▶ will fly in MPC7448
- ▶ will drive in TRICORE 1798





- Tree bits point to next victim
- After access, tree bits on path are set to point away

1 Introduction and Problem

2 Analysis Challenges

- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

4 Summary

1 Introduction and Problem

2 Analysis Challenges

- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

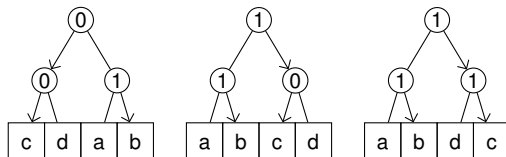
4 Summary

Non-Trivial Logical States

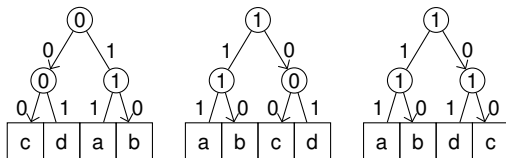
- There are cache states that
 - ▶ differ in physical arrangement of cached blocks
 - ▶ exhibit the same replacement behavior
- Inefficient to distinguish such states

⇒ Abstract from physical cache states to logical ones

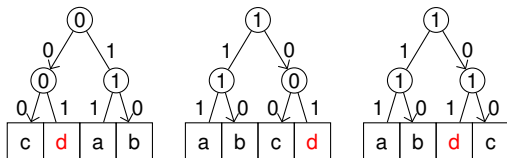
- Easy for LRU $\begin{array}{c} \text{MRU} \\ \downarrow \\ [b_1, \dots, b_k] \end{array}$ $\begin{array}{c} \text{LRU} \\ \downarrow \end{array}$ and for FIFO $\begin{array}{c} \text{last-in} \\ \downarrow \\ [b_1, \dots, b_k] \end{array}$ $\begin{array}{c} \text{first-in} \\ \downarrow \end{array}$
- How to do this for PLRU? How to abstract from the tree bits?



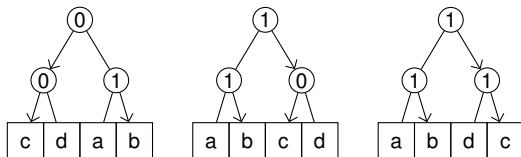
- Only differ in interchanged subtrees
- What matters? Whether tree bit points towards an element or not



- Only differ in interchanged subtrees
- What matters? Whether tree bit points towards an element or not
- **Edge bits** encode this “points towards”



- Only differ in interchanged subtrees
- What matters? Whether tree bit points towards an element or not
- Edge bits encode this “points towards”
- **Access path** consists of edge bits from leaf to root
- Access path of **d** is 10



1. Same replacement behavior \iff coinciding access paths

$$q_1 \sim q_2 \iff \forall b \in \mathcal{B} : ap(q_1, b) = ap(q_2, b)$$

2. Imply order on access paths (*miss-replacement distance*)

\Rightarrow Unique representation for all equivalent physical states

- Logical state $\tilde{q} = [a, d, b, c]^\sim$ represents all physical states from above

1 Introduction and Problem

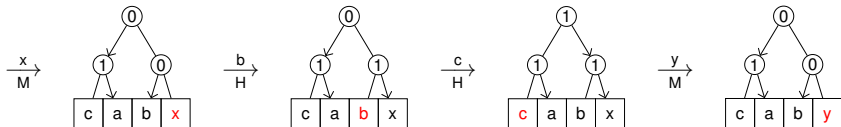
2 Analysis Challenges

- Non-Trivial Logical States
- **Logarithmic-Time Eviction**
- Arbitrary Survival

3 Proof of Concept

4 Summary

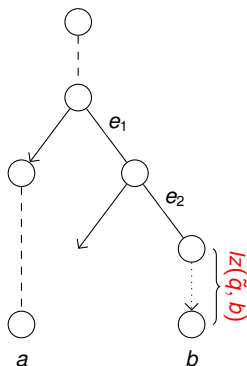
Logarithmic-Time Eviction



- x inserted by miss
- x evicted after $\log_2(4) + 1 = 3$ accesses
- although size $k = 4$

\Rightarrow Must-analysis easy up to $\log_2(k) + 1$ blocks
 ... But hard beyond that

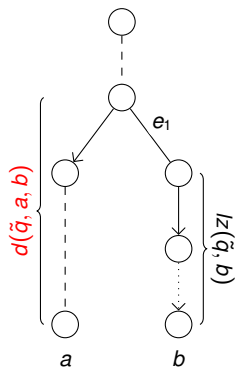
Leading Zeros



- To evict b , its access path must be $0 \dots 0$
- Edge-bits need to be flipped bottom-up, i.e. e_2 before e_1

$$ap(\tilde{q}, b) : 111 \rightarrow 011 \rightarrow 001 \rightarrow 000$$

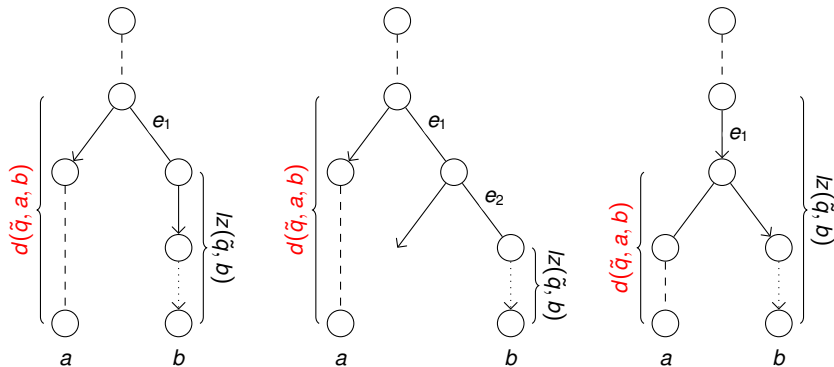
⇒ **Leading zeros** in access paths are interesting



■ How does $ap(\tilde{q}, b)$ change when accessing a ?

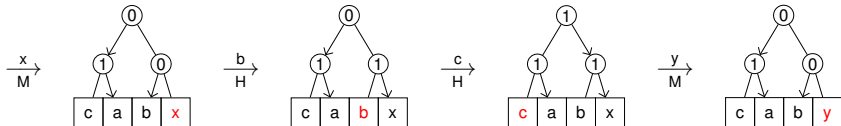
■ Depends on relative position of a to b

⇒ **Subtree distance** between blocks are interesting



- How does $ap(\tilde{q}, b)$ change when accessing a ?
 - Depends on relative position of a to b
- ⇒ **Subtree distance** between blocks are interesting

Excluding Logarithmic-Time Eviction



- For log-time eviction, need to access blocks in increasing subtree distance
- Knowledge about subtree distances can exclude log-time eviction

- Potentially leading zeros

$$PLZ_k := \mathcal{B} \rightarrow \{0, \dots, \log_2(k), \top\}$$

- Approximated Distance

$$AD_k := \mathcal{B} \times \mathcal{B} \rightarrow \{ \{0\}, [1, \log_2(k)), \{\log_2(k)\}, \top \}$$

- The complete domain

$$Plru_k^\Delta := AD_k \hookrightarrow PLZ_k$$

- Tradeoff possible by plugging in different AD_k
- Formalization and details in the paper

1 Introduction and Problem

2 Analysis Challenges

- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

4 Summary

- There are access sequences that
 - ▶ access arbitrarily many distinct blocks
 - ▶ do not access x
 - ▶ but x is still cached

⇒ Hard to prove eviction of blocks

- No viable May-Analysis, yet

1 Introduction and Problem

2 Analysis Challenges

- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

4 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

ECRTS'10 Precise and efficient must- and may-analysis for FIFO [4]

→ 4pm session on Thursday

- Analyses:
 - RC Analysis based on relative competitiveness [2]
 - △ Analysis based on subtree distances
- Collecting semantics:
 - CS Limit for any static analysis
- Spectrum of synthetic benchmarks:
 - ▶ Random access sequences
 - ▶ Loops

		Associativity $k = 8$						
n		2	3	4	5	6	7	8
Loop	$Plru_k^{RCH}$	93.8	93.8	93.8	0.0	0.0	0.0	0.0
	$Plru_k^{\Delta}$	93.8	93.8	93.8	92.5	90.6	0.0	0.0
	$Plru_k^{CS}$	93.8	93.8	93.8	92.5	91.7	90.2	86.7
Rand	$Plru_k^{RCH}$	98.0	97.0	96.0	77.7	64.4	55.7	48.1
	$Plru_k^{\Delta}$	98.0	97.0	95.8	93.0	84.3	63.5	52.0
	$Plru_k^{CS}$	98.0	97.0	96.0	93.9	91.0	84.0	68.4

- Hit rates [%] guaranteed by the analyses and the collecting semantics
- n is number of distinct elements that get accessed

1 Introduction and Problem

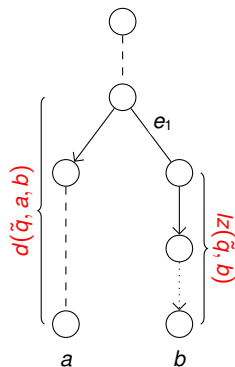
2 Analysis Challenges

- Non-Trivial Logical States
- Logarithmic-Time Eviction
- Arbitrary Survival

3 Proof of Concept

4 Summary

- Logical PLRU-states
 - ▶ abstract from tree-bits
 - ▶ coarsest complete abstraction
- Must-analysis / log-time eviction
 - ▶ leading zeros
 - ▶ subtree distance
- May-analysis / arbitrary survival
 - ▶ unsolved





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

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

ECRTS 2010

- Both!
- Cache analysis = value analysis \oplus replacement analysis
- Value analysis approximates accessed addresses
- Replacement analysis approximates cache contents
- Imprecise value analysis results?
- Replacement analysis can always join over all possibilities

Do you Need a Functional Language?

- No!
 - Let $f \in PLZ_k := \mathcal{B} \rightarrow \{0, \dots, \log_2(k), \top\}$
 - Let $f(a) = 0$, $f(b) = 1$, and $f(x) = \top$ otherwise
 - No need to explicitly represent the default value \top for all $x \in \mathcal{B}$
- \Rightarrow Can represent the function f as $\{(a, 0), (b, 1)\}$

Comparing $Plru_k^\Delta$ and $Plru_k^{RC_H}$

- $Plru_k^\Delta$ is not always better than $Plru_k^{RC_H}$

	a	b	b	c	d	a
$Plru_k^{RC_H}$	T	T	H	T	T	H
$Plru_k^\Delta$	T	T	H	T	T	T

- If $d(\tilde{q}, a, b) = 1$, $Plru_k^\Delta$ cannot remember this (for $k \geq 8$)
- In that case, it only knows $d(\tilde{q}, a, b) \in [1, 2]$
- First access to b : $d(\tilde{q}_1, a, b) = 1$ possible
- Second access to b : $d(\tilde{q}_1, a, b) = 2$ possible