Cache Persistence Analysis: Finally Exact

40th IEEE Real-Time Systems Symposium (RTSS)

Gregory Stock Sebastian Hahn Jan Reineke

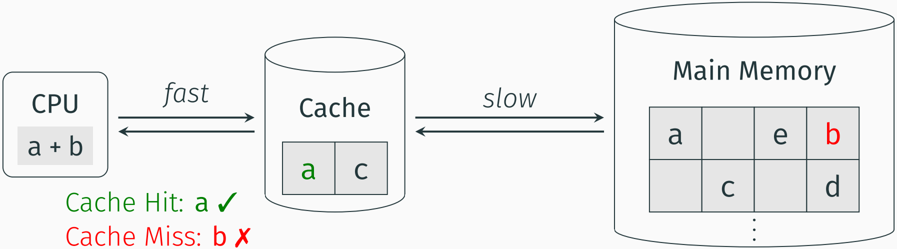
December 3–6, 2019

Saarland University, Germany

Deutsche Forschungsgemeinschaft
German Research Foundation

This presentation is available as a video:
https://dcloud.cs.uni-saarland.de/s/MkpMgBYoFt9q5Tx
https://youtu.be/PI9SauvflTo

Caches



Caches[1]

- contain memory blocks $b \in \mathcal{B}$, stored in cache lines of same size
- organized in sets, size of cache set called associativity k
- cache replacement policy: least recently used (LRU)

[1] W. l. o. g., we assume a fully-associative cache, i. e., with a single cache set. This is no real restriction as set-associative caches with n sets can be treated as n independent fully-associative caches.

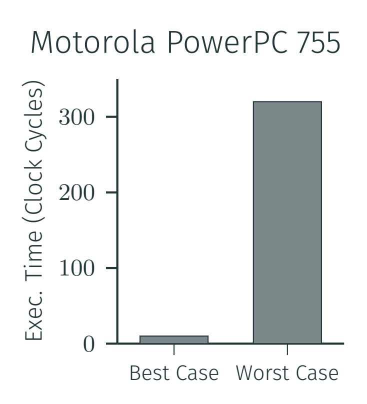
Caches Are Important

```
x = a + b;
```

⇒

```
LOAD r2, _a
LOAD r1, _b
ADD  r3, r2, r1
```

⇒



Control-Flow Graphs

Control-Flow Graphs (CFG)

A control-flow graph $\mathcal{G} = (V, E, i)$ is an abstraction of a program.

Control-Flow Graphs

Control-Flow Graphs (CFG)

A control-flow graph $\mathcal{G} = (V, E, i)$ is an abstraction of a program.

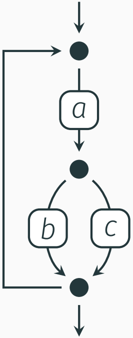
```
int a, b, c;

for (int i = 0; i < 100; i++) {
    if (a % 2) {
        b++;
    } else {
        c--;
    }
}
```

Control-Flow Graphs

Control-Flow Graphs (CFG)

A control-flow graph $\mathcal{G} = (V, E, i)$ is an abstraction of a program.

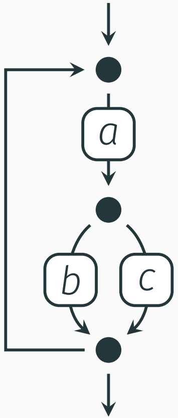
```
int a, b, c;

for (int i = 0; i < 100; i++) {
    if (a % 2) {
        b++;
    } else {
        c--;
    }
}
```



Timing Analysis

Two Approaches

- classify accesses as “hits” or “misses”
- bound the total number of misses



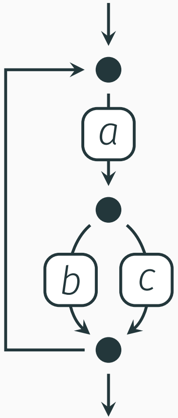
Timing Analysis

Two Approaches

- classify accesses as “hits” or “misses”
- bound the total number of misses

Example

- cache of associativity $k = 4$
- *no* classification possible
  (no block either always misses or always hits the cache)
- at most three cache misses; max. one per block



Persistence Analysis

History of Cache Persistence Analysis

1994
Mueller • Arnold et al.

1997
White et al. • Ferdinand et al.

1999
Ferdinand et al.

2000
Mueller

2008
Ballabriga et al.

2011
Cullmann • Huynh et al.

2012
Nagar • Nagar et al.

2013
Cullmann • Cullmann

Dimensions

- Instruction Cache
- Data Cache
- Direct-Mapped Cache
- Set-Associative Cache
- Analysis Precision

History of Cache Persistence Analysis

1994
Mueller • Arnold et al.

1997
White et al. • Ferdinand et al.

1999

2000

Ballabriga et al.

2012
Nagar • Nagar et al.

2013
Cullmann • Cullmann

Approximations

Dimensions

- Instruction Cache
- Data Cache
- Direct-Mapped Cache
- Set-Associative Cache
- Analysis Precision

History of Cache Persistence Analysis

1994
Mueller • Arnold et al.

1997
White et al. • Ferdinand et al.

1999

2000

Ballabriga et al.

al.

2012
Nagar • Nagar et al.

2013
Cullmann • Cullmann

2019
Stock et al.

Approximations

Dimensions

- Instruction Cache
- Data Cache
- Direct-Mapped Cache
- Set-Associative Cache
- Analysis Precision

First Exact Analysis

Persistence on a Trace

$$\tau = f \quad c \quad e \quad b \quad a \quad a \quad b \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss

Persistence on a Trace

$$\tau = f \quad c \quad e \quad \underset{✗}{b} \quad a \quad a \quad b \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss

Persistence on a Trace

$$\tau = f \quad c \quad e \quad b \quad a \quad a \quad b \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

✗ ?

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss

Persistence on a Trace

$$\tau \;=\; f \quad c \quad e \quad \underset{\text{✗}}{b} \quad a \quad a \quad \underset{?}{b} \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$

Persistence on a Trace

$$\tau \;=\; f \quad c \quad e \quad \underset{✗}{b} \quad a \quad a \quad \underset{?}{b} \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- *b*'s conflict set: set of all blocks accessed from last access to block *b* onward

Persistence on a Trace

$$\tau = \boxed{f \quad c \quad \underset{\substack{\uparrow \\ \emptyset}}{e}} \quad \underset{✗}{b} \quad a \quad a \quad \underset{?}{b} \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- *b*'s conflict set: set of all blocks accessed from last access to block *b* onward

Persistence on a Trace

$$\tau = \begin{array}{ccccccccccccccccc} f & c & e & b & a & a & b & d & c & e & f & b & e & c & a & b & f \\ & & \uparrow & \text{✗} & & \uparrow & ? & & & & & & & & & & \\ & & \emptyset & & & \{a, b\} & & & & & & & & & & & \end{array}$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- b's conflict set: set of all blocks accessed from last access to block b onward

$$\tau = \boxed{f \quad c \quad \underset{\emptyset}{\underset{\uparrow}{e}}} \; \boxed{\underset{\text{✗}}{b} \quad a \quad \underset{\{a, b\}}{\underset{\uparrow}{a}}} \quad \underset{\checkmark}{b} \quad d \quad c \quad e \quad f \quad b \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- *b*'s conflict set: set of all blocks accessed from last access to block *b* onward

Persistence on a Trace

$$\tau = \begin{array}{ccc|ccc|ccccccccccc} f & c & e & b & a & a & b & d & c & e & f & b & e & c & a & b & f \\ & & \uparrow & ✗ & & \uparrow & ✓ & & & & & & & & & & \\ & & \emptyset & & & \{a, b\} & & & & & & & & & & & \end{array}$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- b's conflict set: set of all blocks accessed from last access to block b onward

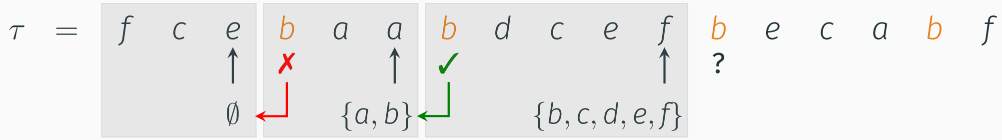
Persistence on a Trace

$$\tau = \boxed{f \quad c \quad \underset{\emptyset}{\overset{e}{\uparrow}}}\ \boxed{\overset{b}{\text{✗}} \quad a \quad \underset{\{a,b\}}{\overset{a}{\uparrow}}}\ \overset{b}{\text{✓}} \quad d \quad c \quad e \quad f \quad \overset{b}{?} \quad e \quad c \quad a \quad b \quad f$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- b's conflict set: set of all blocks accessed from last access to block b onward

Persistence on a Trace



Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- *b*'s conflict set: set of all blocks accessed from last access to block *b* onward

Persistence on a Trace

$$
\tau = \begin{array}{ccc|ccc|ccccc|cccccc}
f & c & e & b & a & a & b & d & c & e & f & b & e & c & a & b & f \\
 & & \uparrow & ✗ & & \uparrow & ✓ & & & & \uparrow & ✗ & & & & & \\
 & & \emptyset & & & \{a,b\} & & & & & \{b,c,d,e,f\} & & & & & &
\end{array}
$$

Definitions

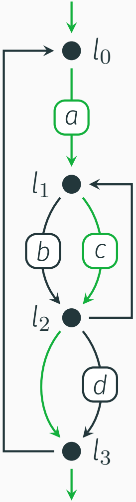
- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- b's conflict set: set of all blocks accessed from last access to block b onward

$$\tau = \begin{array}{ccccccccccccccccc} f & c & e & \color{orange}{b} & a & a & \color{orange}{b} & d & c & e & f & \color{orange}{b} & e & c & a & \color{orange}{b} & f \\ & & \uparrow & \color{red}{\text{✗}} & & \uparrow & \color{green}{\checkmark} & & & & \uparrow & \color{red}{\text{✗}} & & & \uparrow & \color{green}{\checkmark} & \\ & & \emptyset & & & \{a, b\} & & & & & \{b, c, d, e, f\} & & & & \{a, b, c, e\} & & \end{array}$$

Definitions

- persistence: all accesses to memory block result in *at most one* cache miss
- LRU cache of associativity $k = 4$
- *b*'s conflict set: set of all blocks accessed from last access to block *b* onward

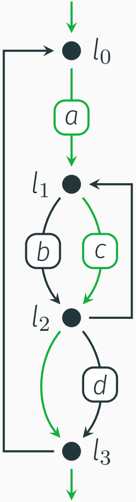
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$

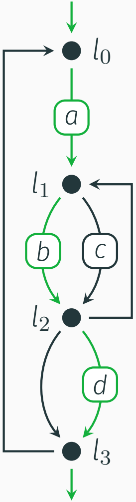
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\qquad b \mapsto \emptyset$

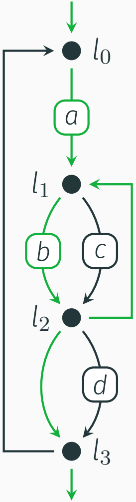
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\qquad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\qquad b \mapsto \{b, d\}$

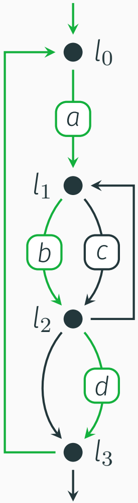
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\qquad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\qquad b \mapsto \{b, d\}$
- $\tau_3 = a\ b\ b\ b\ b\ b$ $\qquad b \mapsto \{b\}$

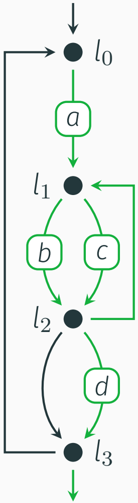
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\quad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\quad b \mapsto \{b, d\}$
- $\tau_3 = a\ b\ b\ b\ b\ b$ $\quad b \mapsto \{b\}$
- $\tau_4 = a\ b\ d\ a\ c\ c\ b\ d$ $\quad b \mapsto \{b, d\}$

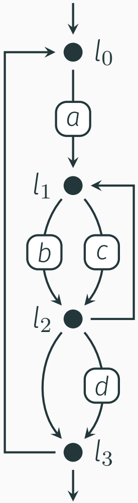
Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\quad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\quad b \mapsto \{b, d\}$
- $\tau_3 = a\ b\ b\ b\ b\ b$ $\quad b \mapsto \{b\}$
- $\tau_4 = a\ b\ d\ a\ c\ c\ b\ d$ $\quad b \mapsto \{b, d\}$

Persistence in a Program



Example Traces

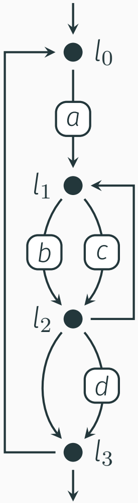
- $\tau_1 = a\ c$ $\quad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\quad b \mapsto \{b, d\}$
- $\tau_3 = a\ b\ b\ b\ b\ b$ $\quad b \mapsto \{b\}$
- $\tau_4 = a\ b\ d\ a\ c\ c\ b\ d$ $\quad b \mapsto \{b, d\}$

Definition: Persistence in a Program

A block b is persistent in a program if:

$$\forall \tau \in \mathit{Traces} : |CS(\tau, b)| \leq \mathit{associativity}\ k$$

Persistence in a Program



Example Traces

- $\tau_1 = a\ c$ $\qquad b \mapsto \emptyset$
- $\tau_2 = a\ b\ d$ $\qquad b \mapsto \{b, d\}$
- $\tau_3 = a\ b\ b\ b\ b\ b$ $\qquad b \mapsto \{b\}$
- $\tau_4 = a\ b\ d\ a\ c\ c\ b\ d$ $\qquad b \mapsto \{b, d\}$
- $\cdots$

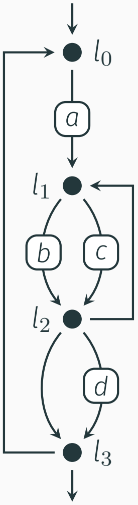
Definition: Persistence in a Program

A block b is persistent in a program if:

$$\forall \tau \in \mathit{Traces} : |CS(\tau, b)| \leq \mathit{associativity}\ k$$

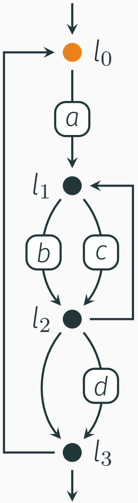
Exact Persistence Analysis

$Exact$-CS: Example



LRU Cache of associativity $k = 3$

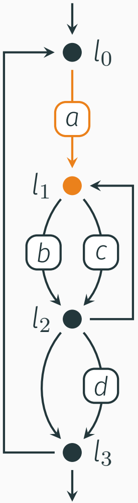
Exact-CS: Example



$a \mapsto \{\emptyset\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example

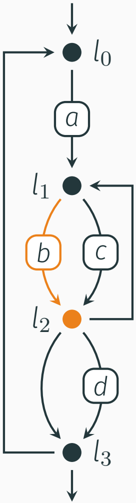


$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



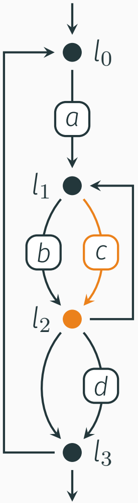
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}\}$

$a \mapsto \{\{a, b\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



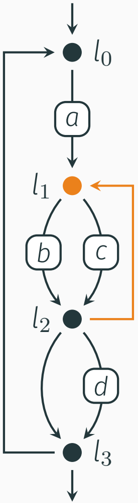
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



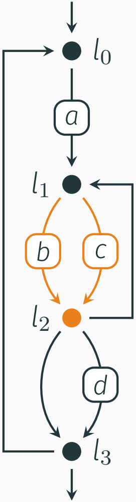
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



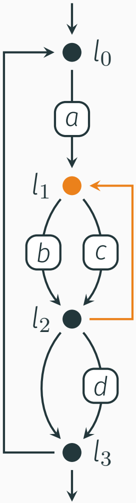
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



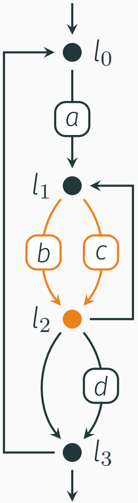
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



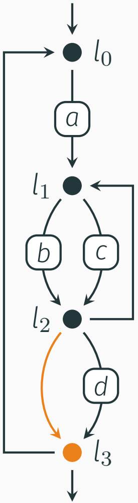
$a \mapsto \{\emptyset\}$

$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$

LRU Cache of associativity $k = 3$

Exact-CS: Example



$a \mapsto \{\emptyset\}$

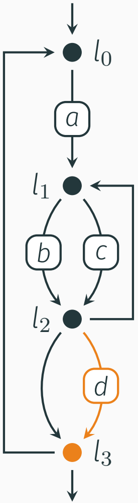
$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$

$$a \mapsto \left\{ \quad \{a, b\}, \{a, c\}, \{a, b, c\} \quad \right\}$$

LRU Cache of associativity $k = 3$

Exact-CS: Example



$a \mapsto \{\emptyset\}$

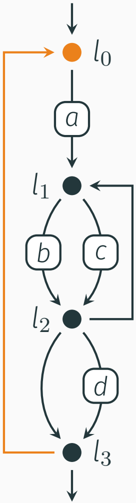
$a \mapsto \{\{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}\}$

$a \mapsto \{\{a, b\}, \{a, c\}, \{a, b, c\}\}$

$$a \mapsto \left\{ \begin{array}{c} \{a, b\}, \{a, c\}, \{a, b, c\}, \\ \{a, b, d\}, \{a, c, d\}, \{a, b, c, d\} \end{array} \right\}$$

LRU Cache of associativity $k = 3$

Exact-CS: Example



$$a \mapsto \left\{ \begin{array}{c} \emptyset, \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

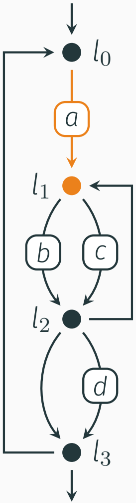
$$a \mapsto \{\{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \{\{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \left\{ \begin{array}{c} \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

LRU Cache of associativity $k = 3$

Exact-CS: Example



$$a \mapsto \left\{ \begin{array}{c} \emptyset, \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

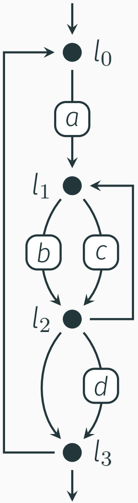
$$a \mapsto \{\{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \{\{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \left\{ \begin{array}{c} \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

LRU Cache of associativity $k = 3$

Exact-CS: Example



$$a \mapsto \left\{\begin{array}{c} \emptyset, \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array}\right\}$$

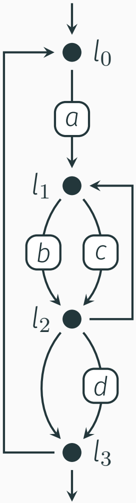
$$a \mapsto \{\{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \{\{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \left\{\begin{array}{c} \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array}\right\}$$

LRU Cache of associativity $k = 3$

$\mathit{Exact}\text{-}\mathit{CS}$: Example (Optimized)



$$a \mapsto \left\{ \begin{array}{c} \emptyset, \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

$$a \mapsto \{\{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}$$

$$a \mapsto \{\{a,b\}, \{a,c\}, \{a,b,c\}\}$$

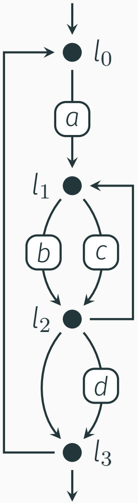
$$a \mapsto \left\{ \begin{array}{c} \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$$

LRU Cache of associativity $k = 3$

Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

Exact-CS: Example (Optimized)



$a \mapsto \left\{ \begin{array}{c} \emptyset, \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$

$a \mapsto \{\{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}$

$a \mapsto \{\{a,b\}, \{a,c\}, \{a,b,c\}\}$

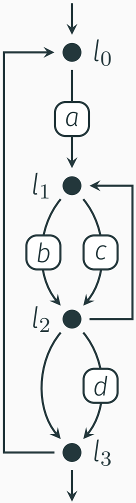
$a \mapsto \left\{ \begin{array}{c} \{a,b\}, \{a,c\}, \{a,b,c\}, \\ \{a,b,d\}, \{a,c,d\}, \{a,b,c,d\} \end{array} \right\}$

Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

LRU Cache of associativity $k = 3$

Exact-CS: Example (Optimized)



$a \mapsto \{\{a, b, c, d\}\}$

$a \mapsto \{\{a, b, c\}\}$

$a \mapsto \{\{a, b, c\}\}$

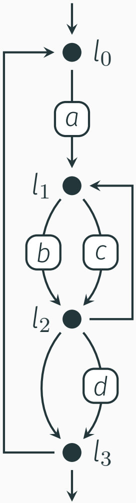
$a \mapsto \{\{a, b, c, d\}\}$

LRU Cache of associativity $k = 3$

Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

Exact-CS: Example (Optimized)



$a \mapsto \{\{a, b, c, d\}\}$

$a \mapsto \{\{a, b, c\}\}$

$a \mapsto \{\{a, b, c\}\}$

$a \mapsto \{\{a, b, c, d\}\}$

LRU Cache of associativity $k = 3$

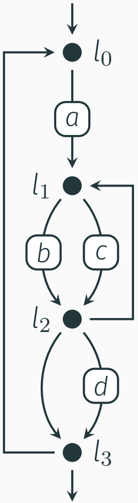
Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

Cutoff at Associativity

- check existence of single conflict set C with $|C| > k$
- collapse all large conflict sets into single representative $\mathcal{B}$

$\mathit{Exact\text{-}CS}$: Example (Optimized)



$a \mapsto \{\{a, b, c, d\}\}$

$a \mapsto \{\{a, b, c\}\}$

$a \mapsto \{\{a, b, c\}\}$

$a \mapsto \{\{a, b, c, d\}\}$

LRU Cache of associativity $k = 3$

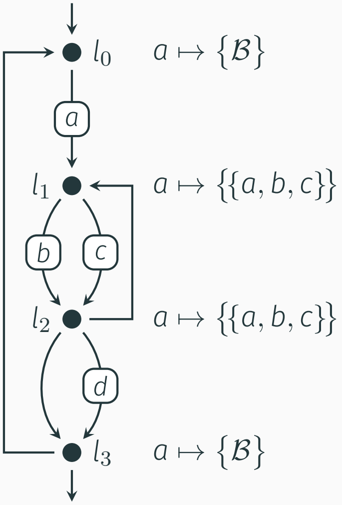
Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

Cutoff at Associativity

- check existence of single conflict set C with $|C| > k$
- collapse all large conflict sets into single representative $\mathcal{B}$

Exact-CS: Example (Optimized)



LRU Cache of associativity $k = 3$

Exploiting Monotonicity

- only largest conflict set relevant for classification
- safe to remove conflict sets subsumed by others

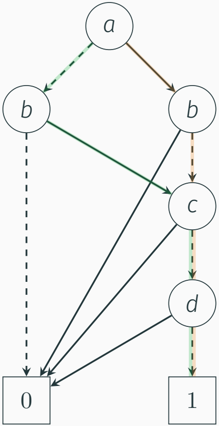
Cutoff at Associativity

- check existence of single conflict set C with $|C| > k$
- collapse all large conflict sets into single representative $\mathcal{B}$

Efficient Implementation

Zero-suppressed BDDs: Sets of Combinations

Binary Decision Diagram

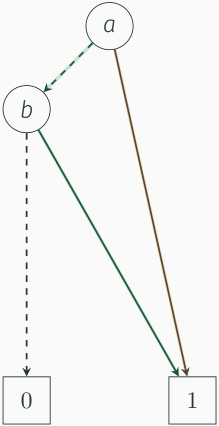


$a \wedge \neg b \wedge \neg c \wedge \neg d$
$\vee \neg a \wedge \quad b \wedge \neg c \wedge \neg d$

Sets of
Conflict Sets

$\{\{a\}, \{b\}\}$
$\subseteq 2^{\{a,b,c,d\}}$

Zero-suppressed BDD



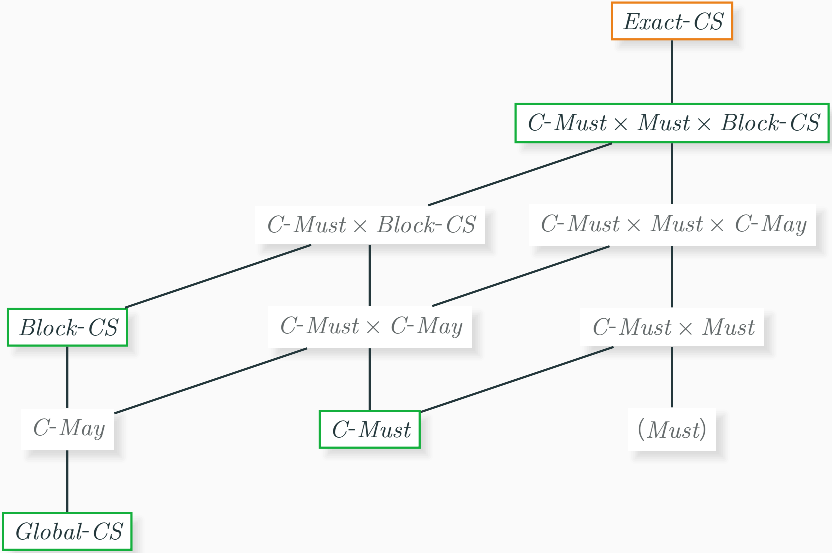
non-occurring variables
automatically suppressed

Evaluation

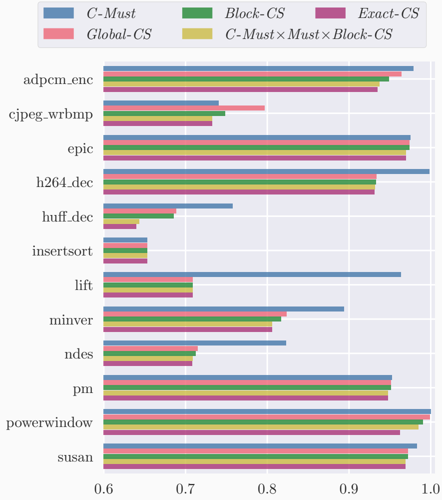
Gain in Analysis Precision?

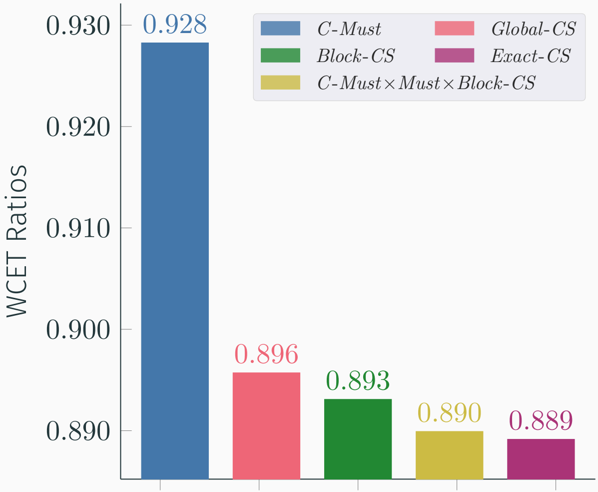
Effect on Run Time and Memory?

Landscape of Cache Persistence Analyses



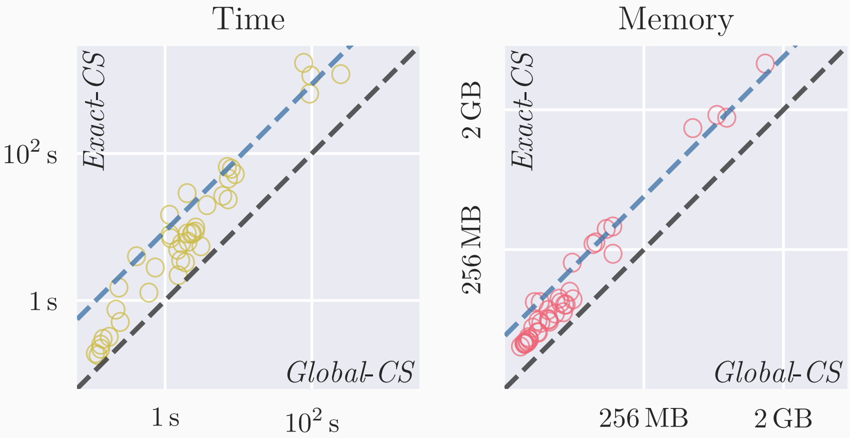
Analysis Precision: WCET Ratios





Cache Configuration: 32 Cache Sets, 8 Ways, 16B Line Size, with Compiler Optimizations

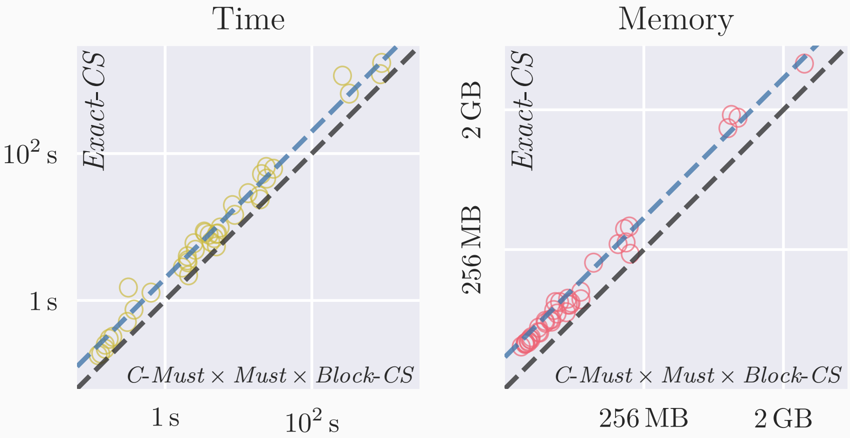
Analysis Cost: *Global-CS* vs. *Exact-CS*



Cache Configuration: 32 Cache Sets, 8 Ways, 16B Line Size, with Compiler Optimizations

Running time on average 8.8 times higher; Memory 2.2 times higher (blue line)

Analysis Cost: *C-Must* × *Must* × *Block-CS* vs. *Exact-CS*



Cache Configuration: 32 Cache Sets, 8 Ways, 16B Line Size, with Compiler Optimizations

Running time on average 2 times higher; Memory 1.6 times higher (blue line)

In Our Paper

Cache Persistence Analysis: Finally Exact

Gregory Stock, Sebastian Hahn, and Jan Reineke
Saarland University
Saarland Informatics Campus
Saarbrücken, Germany
{g.stock, sebastian.hahn, reineke}@cs.uni-saarland.de

***Abstract*—Cache persistence analysis is an important part of worst-case execution time (WCET) analysis. It has been extensively studied in the past twenty years. Despite these efforts, all existing persistence analyses are approximative in the sense that they are not guaranteed to find all persistent memory blocks.**

In this paper, we close this gap by introducing the first exact persistence analysis for caches with least-recently-used (LRU) replacement. To this end, we first introduce an exact abstraction that exploits monotonicity properties of LRU to significantly reduce the information the analysis needs to maintain for exact persistence classifications. We show how to efficiently implement this abstraction using zero-suppressed binary decision diagrams (ZDDs) and introduce novel techniques to deal with uncertainty that arises during the analysis of data caches.

The experimental evaluation demonstrates that the new exact analysis is competitive with state-of-the-art inexact analyses in terms of both memory consumption and analysis run time, which is somewhat surprising as we show that persistence analysis is NP-complete. We also observe that while prior analyses are not exact in theory they come close to being exact in practice.

I. Introduction

Modern processors can perform several arithmetic and logic operations in a single cycle. On the other hand, a single access to main memory can take hundreds of cycles. To bridge this performance gap, modern processors include one or multiple levels of caches. Caches are small but fast memories that store parts of main memory to quickly serve accesses to commonly used instructions and data. Memory accesses that "hit" the cache are served from the cache at a low latency, while accesses that "miss" the cache are served from main memory at a much higher latency. The execution time of a program thus heavily depends on how effective the processor's caches are in hiding the high latency of main memory.

Real-time systems are systems that, in order to function correctly, have to perform their computations with limited amounts of wall-clock time. To verify a system's real-time behavior, a major task is to bound each software component's worst-case execution time (WCET). In the presence of caches, WCET analysis [1] has to account for the software's cache behavior. Simply assuming that each memory access could result in a cache miss would yield extremely pessimistic WCET bounds. Thus, static cache analyses [2] have been developed to soundly and precisely characterize a program's cache behavior on a particular cache architecture. These can broadly be categorized into two groups:

1) *Classifying cache analyses* aim to classify individual memory accesses as cache hits or cache misses.

Fig. 1: Simple motivating example for persistence analysis.

2) *Quantitative cache analyses* aim to determine the number of cache misses resulting from a set of memory accesses.

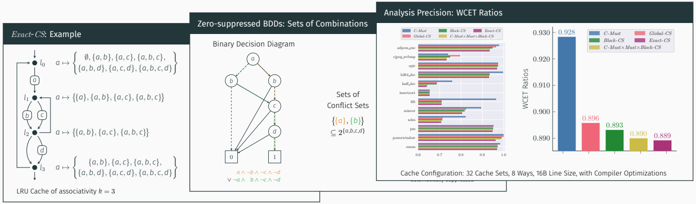
In this paper we study *persistence analysis*, an instance of quantitative cache analysis. Persistence analysis considers all memory accesses in a program, or a fragment of a program such as a loop, that access the *same* memory block. A memory block is persistent if all memory accesses referring to this memory block may cumulatively result in at most one cache miss during any possible program execution.

For a motivating example, consider Figure 1, which contains the control-flow graph of a simple program. The program consists of a loop, in which, in each loop iteration either memory block x or memory block y is accessed. As neither block x nor block y is guaranteed to have been accessed in any loop iteration, it is impossible for a classifying cache analysis to classify any of the memory accesses in the program as a guaranteed cache hit, and so a WCET analysis would have to pessimistically account for misses upon all memory accesses. However, provided the cache is large enough to hold blocks x and y simultaneously, among all memory accesses to x (and similarly to y) only the very first may result in a cache miss. Both x and y are *persistent* and WCET analysis can safely account for at most two misses in total.

Given a program, the goal of persistence analysis is to determine which of the memory blocks accessed in the program are persistent. Persistence analysis has been extensively studied for caches with least-recently-used (LRU) replacement, starting with Mueller's [3]–[6] and Ferdinand's [7], [8] work in the 1990s up until today [9]–[17]. Notably, all prior persistence analyses are *approximative*, in the sense that they are not guaranteed to find all persistent memory blocks of a program.

In this paper, we close this gap by introducing the first *exact* persistence analysis. We develop this analysis via a sequence of three consecutive exact abstractions. The first abstraction is based on the observation that the persistence of a memory block can be determined by examining its possible *conflict sets*, i. e., the sets of blocks that may have been accessed since the last access to the block itself. The two following abstractions exploit a monotonicity property of LRU

- ▶ Formal Definition of Exact Abstraction
- ▶ Extension to Data Caches
- ▶ Proofs and Evaluation (→ Technical Report)
- ▶ Persistence Analysis is NP-complete



References

Gregory Stock, Sebastian Hahn, and Jan Reineke. “Cache Persistence Analysis: Finally Exact.” In: *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*. IEEE, 2019, pp. 481–494. DOI: `10.1109/RTSS46320.2019.00049`.

Gregory Stock, Sebastian Hahn, and Jan Reineke. “Cache Persistence Analysis: Finally Exact.” In: *CoRR* abs/1909.04374 (2019). Technical Report. arXiv: `1909.04374`.