Precise and Efficient Analysis of Context-Sensitive Cache Conflict Sets

Florian Brandner
LTCI, Télécom Paris, Institut Polytechnique de Paris

What is this paper about?

Cache analysis – analyzing the state of the instruction cache:

- Which instructions are present in the cache?
- Will a given access be a cache hit or miss?

What is this paper about?

Cache analysis – analyzing the state of the instruction cache:

- Which instructions are present in the cache?
- Will a given access be a cache hit or miss?

- Considering:
  - Cache design.
  - The replacement policy.
  - The cache size, block size, and associativity.

What is this paper about?

Cache analysis – analyzing the state of the instruction cache:

- Which instructions are present in the cache?
- Will a given access be a cache hit or miss?

- Considering:
  - Cache design. (here: method cache and set-associative caches)
  - The replacement policy. (here: least-recently used aka. LRU)
  - The cache size, block size, and associativity.

Access Classification

Classify each instruction fetch operation as [34]:

- Always Hit:
  The requested instructions are always in the cache.
- Always Miss:
  The requested instructions are never in the cache.
- Not Classified:
  The requested instructions might or might not be in the cache.

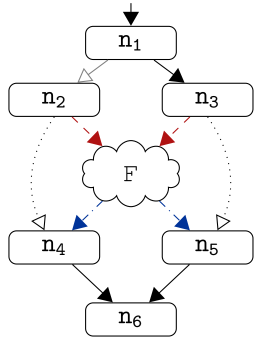
Access Classification

Classify each instruction fetch operation as [34]:

- Always Hit:
  The requested instructions are always in the cache.
- Always Miss:
  The requested instructions are never in the cache.
- Not Classified:
  The requested instructions might or might not be in the cache.
- Persistent / First Miss:
  The instructions stay in the cache once loaded within a scope.

Program Representation

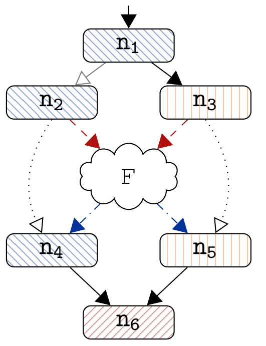
Inter-procedural Control-Flow Graphs (ICFG)



Edge kinds		
FLOW	▷	
FILL	▶	(cache fill)
CALL	▶	(cache fill)
RET	▶	(cache fill)
LINK	▷	

- Nodes represent code (the code itself is irrelevant here).
- Edges represent potential successor instructions at runtime.

Inter-procedural Control-Flow Graphs (ICFG)



Edge kinds

FLOW	
FILL	(cache fill)
CALL	(cache fill)
RET	(cache fill)
LINK	

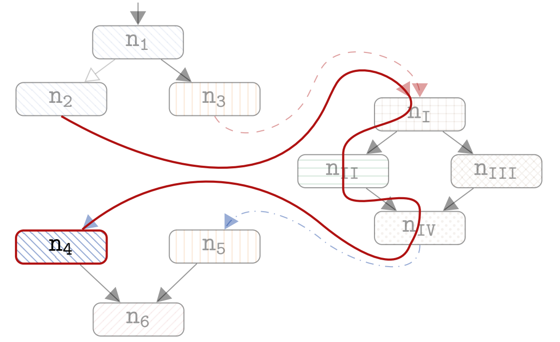
Memory blocks

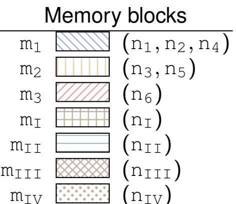
m_1	(n_1, n_2, n_4)
m_2	(n_3, n_5)
m_3	(n_6)

- Nodes represent code (the code itself is irrelevant here).
- Edges represent potential successor instructions at runtime.
- Nodes are associated with **memory blocks**
  (i.e., address ranges potentially fetched to the cache).

Example: Cache States and Traces

Assume a cache with 4 blocks and an actual control-flow graph for function F:





Cache state before executing node n_4:

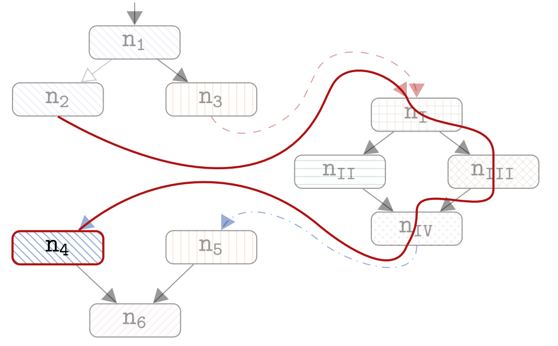
m_{IV}	m_{II}	m_I	m_1

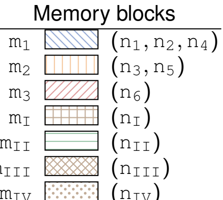
MRU LRU

hit

Example: Cache States and Traces

Assume a cache with 4 blocks and an actual control-flow graph for function `F`:





Cache state before executing node n_4:

m_{IV}	m_{III}	m_I	m_1
MRU			LRU

hit

Cache States and Conflict Sets [33,34]

- Computing these cache states on all traces is too expensive.

Cache States and Conflict Sets [33,34]

- Computing these cache states on all traces is too expensive.
- **Conflict sets** provide an efficient abstraction:
  - Abstract a cache state by a set of memory blocks.
  - Tracked for each memory block m separately.
  - A conflict set contains m and memory blocks that are younger than m.

Cache States and Conflict Sets [33,34]

- Computing these cache states on all traces is too expensive.
- **Conflict sets** provide an efficient abstraction:
  - Abstract a cache state by a set of memory blocks.
  - Tracked for each memory block m separately.
  - A conflict set contains m and memory blocks that are younger than m.
  - Classification derived from set size.
    (classify has hit when conflict set fits into the cache)

Example: Conflict Sets [33,34]

Path	Cache State	Conflict Set
$(n_1, n_2, n_I, n_{II}, n_{IV})$	m_{IV} | m_{II} | m_I | m_1	$\{m_1, m_I, m_{II}, m_{IV}\}$
$(n_1, n_2, n_I, n_{III}, n_{IV})$	m_{IV} | m_{III} | m_I | m_1	$\{m_1, m_I, m_{III}, m_{IV}\}$

Meet-Over-all-Paths Solution:[1] $\{m_1, m_I, m_{II}, m_{III}, m_{IV}\}$

[1] Compute union over all conflict sets.

Example: Conflict Sets [33,34]

Path	Cache State				Conflict Set
$(n_1, n_2, n_I, n_{II}, n_{IV})$	m_{IV}	m_{II}	m_I	m_1	$\{m_1, m_I, m_{II}, m_{IV}\}$
$(n_1, n_2, n_I, n_{III}, n_{IV})$	m_{IV}	m_{III}	m_I	m_1	$\{m_1, m_I, m_{III}, m_{IV}\}$

Meet-Over-all-Paths Solution:[1] $\{m_1, m_I, m_{II}, m_{III}, m_{IV}\}$

A conflict-set based analysis classifies the access as Miss!

(assuming a cache with 4 cache blocks)

[1] Compute union over all conflict sets.

Precise Conflict Sets

Recent work [36] allows to compute precise conflict sets:

- Represent conflict sets as families of memory blocks.
  (sets of sets of memory blocks)

Precise Conflict Sets

Recent work [36] allows to compute precise conflict sets:

- Represent conflict sets as families of memory blocks.
  (sets of sets of memory blocks)

- Provides Hit/Miss classification only.
  (based on sets sizes within families)

Precise Conflict Sets

Recent work [36] allows to compute precise conflict sets:

- Represent conflict sets as families of memory blocks.
  (sets of sets of memory blocks)
- Provides Hit/Miss classification only.
  (based on sets sizes within families)
- Optimizations for scalability:
  - Replace large conflict sets using a special symbol.
  - Retain only minimum/maximum sets.
  - Use an efficient data structure based on ZDDs.
    (Zero-Suppressed Decision Diagrams)

Objectives of this Work

Objectives of this Work

Four main improvements with regard to the state of the art:

- Support for Call-Context Sensitivity:
  Exploiting **cache summaries** of (nested) function calls.

Objectives of this Work

Four main improvements with regard to the state of the art:

- Support for Call-Context Sensitivity:
  Exploiting **cache summaries** of (nested) function calls.
- Scalability:
  Improve analysis performance using **cache summaries**.

Objectives of this Work

Four main improvements with regard to the state of the art:

- Support for Call-Context Sensitivity:
  Exploiting **cache summaries** of (nested) function calls.

- Scalability:
  Improve analysis performance using **cache summaries**.

- Support for Persistence:
  Provide **First Miss** classification with regard to (nested) function calls.

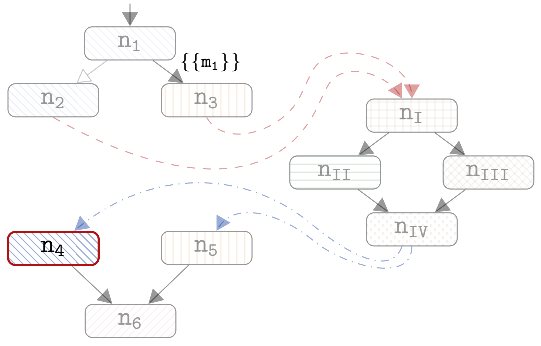
Objectives of this Work

Four main improvements with regard to the state of the art:

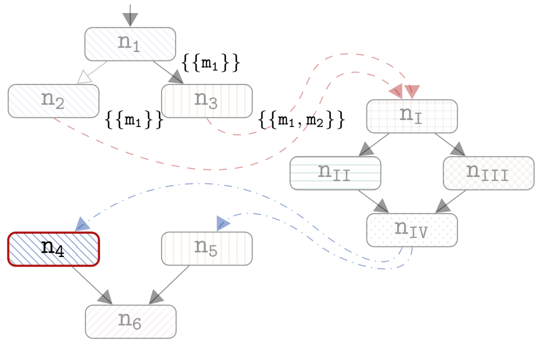
- Support for Call-Context Sensitivity:
  Exploiting **cache summaries** of (nested) function calls.
- Scalability:
  Improve analysis performance using **cache summaries**.
- Support for Persistence:
  Provide **First Miss** classification with regard to (nested) function calls.
- Support for the Method Cache:
  Notably **variable-sized cache blocks** [9] used by the time-predictable processor Patmos [30].

Call-Context Sensitivity

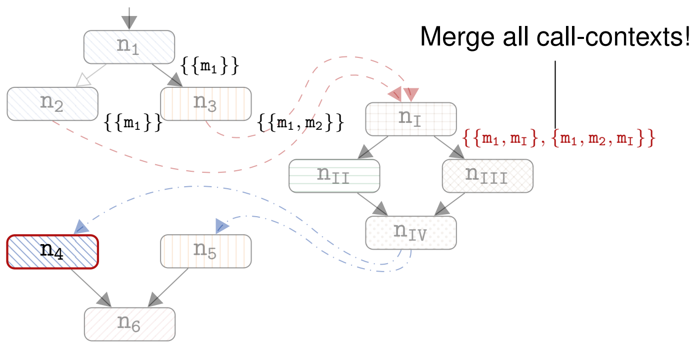
Issues with Call-Context Sensitivity



Issues with Call-Context Sensitivity



Issues with Call-Context Sensitivity



Issues with Call-Context Sensitivity

Merge all call-contexts!

n_1

$\{\{m_1\}\}$

n_2 $\{\{m_1\}\}$ n_3 $\{\{m_1, m_2\}\}$

n_I $\{\{m_1, m_I\}, \{m_1, m_2, m_I\}\}$

n_{II} n_{III}

n_{IV}

$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{III}, m_{IV}\}\}$

n_4 n_5

$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{III}, m_{IV}\}\}$

n_6

Issues with Call-Context Sensitivity

n_1

$\{\{m_1\}\}$

n_2 $\{\{m_1\}\}$

n_3 $\{\{m_1, m_2\}\}$

Merge all call-contexts!

n_I

$\{\{m_1, m_I\}, \{m_1, m_2, m_I\}\}$

n_{II} n_{III}

n_{IV}

$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{III}, m_{IV}\}\}$

n_4 n_5

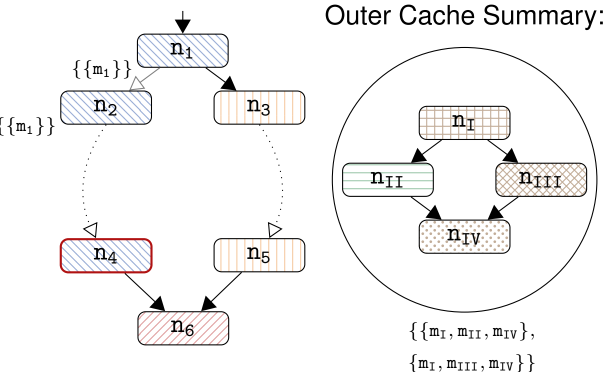
$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_2, m_I, m_{III}, m_{IV}\}\}$

n_6

Bogus cache states are propagated out of functions.

Call-Contexts using Outer Cache Summaries

Example: Outer Cache Summaries



Example: Outer Cache Summaries

n_1

$\{\{m_1\}\}$

n_2

n_3

$\{\{m_1\}\}$

$\{\{m_1\}\} \cdot \{\{m_I, m_{II}, m_{IV}\}, \{m_I, m_{III}, m_{IV}\}\}$

n_4

n_5

n_6

Outer Cache Summary:

n_I

n_{II}

n_{III}

n_{IV}

$\{\{m_I, m_{II}, m_{IV}\},$
$\{m_I, m_{III}, m_{IV}\}\}$

Example: Outer Cache Summaries

n_1

$\{\{m_1\}\}$

n_2 n_3

$\{\{m_1\}\}$

$\{\{m_1\}\} \cdot \{\{m_I, m_{II}, m_{IV}\}, \{m_I, m_{III}, m_{IV}\}\}$

n_4 n_5

$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\}\}$

n_6

Outer Cache Summary:

n_I

n_{II} n_{III}

n_{IV}

$\{\{m_I, m_{II}, m_{IV}\},$
$\{m_I, m_{III}, m_{IV}\}\}$

Outer Cache Summaries

Allow to quickly compute the cache state **after a function call**:[2]

- Capture two facets:
  - Paths that do access the analyzed memory block. ($\mathcal{A}$ Summaries)
  - Paths that do not access the analyzed memory block. ($\mathcal{C}$ Summaries)

[2] Or any other sub-graph of the ICGF, including loops.

Outer Cache Summaries

Allow to quickly compute the cache state **after a function call**:[2]

- Capture two facets:
  - Paths that do access the analyzed memory block. ($\mathcal{A}$ Summaries)
  - Paths that do not access the analyzed memory block. ($\mathcal{C}$ Summaries)
- Can be reused during the analysis of other memory blocks.

[2] Or any other sub-graph of the ICGF, including loops.

Outer Cache Summaries

Allow to quickly compute the cache state **after a function call**:[2]

- Capture two facets:
  - Paths that do access the analyzed memory block. ($\mathcal{A}$ Summaries)
  - Paths that do not access the analyzed memory block. ($\mathcal{C}$ Summaries)
- Can be reused during the analysis of other memory blocks.
- Can be computed independent from each other in parallel.

[2] Or any other sub-graph of the ICGF, including loops.

Persistence / First Miss Classification

Inner Cache Summaries

Allow to quickly compute the cache state **within a callee**:

- Capture two facets:
  - Paths covering the first access to the analyzed memory block. ($\mathcal{B}^{\mathcal{A}}$ Summaries)
  - Paths with recurrent accesses to the analyzed memory block. ($\mathcal{B}^{\mathcal{C}}$ Summaries)

Inner Cache Summaries

Allow to quickly compute the cache state **within a callee**:

- Capture two facets:
  - Paths covering the first access to the analyzed memory block. ($\mathcal{B}^{\mathcal{A}}$ Summaries)
  - Paths with recurrent accesses to the analyzed memory block. ($\mathcal{B}^{\mathcal{C}}$ Summaries)
- Recurrent accesses ($\mathcal{B}^{\mathcal{C}}$ Summaries) provide Persistence classification.

Inner Cache Summaries

Allow to quickly compute the cache state **within a callee**:

- Capture two facets:
  - Paths covering the first access to the analyzed memory block. ($\mathcal{B}^{\mathcal{A}}$ Summaries)
  - Paths with recurrent accesses to the analyzed memory block. ($\mathcal{B}^{\mathcal{C}}$ Summaries)
- Recurrent accesses ($\mathcal{B}^{\mathcal{C}}$ Summaries) provide Persistence classification.
- Virtually for free: Inner Summaries are a byproduct of Outer Summaries.

Analysis Complexity

Experimental Setup

Substantial runtime measurements:

- Intel Core2 Duo (3.16 GHz with 4 GB memory, using a single core).

Experimental Setup

Substantial runtime measurements:

- Intel Core2 Duo (3.16 GHz with 4 GB memory, using a single core).
- All non-recursive benchmarks of the TACLe suite.

Experimental Setup

Substantial runtime measurements:

- Intel Core2 Duo (3.16 GHz with 4 GB memory, using a single core).
- All non-recursive benchmarks of the TACLe suite.
- Three analysis variants:
  - *Baseline*: Precise analysis without summaries.
  - *Outer*: Precise analysis using outer summaries only.
  - *Full*: Precise analysis with outer and inner summaries.

  - 90 minute timeout.

Experimental Setup

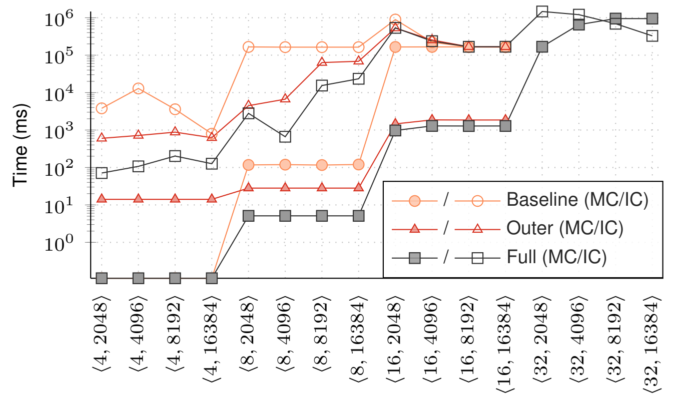
Substantial runtime measurements:

- Intel Core2 Duo (3.16 GHz with 4 GB memory, using a single core).
- All non-recursive benchmarks of the TACLe suite.
- Three analysis variants:
  - *Baseline*: Precise analysis without summaries.
  - *Outer*: Precise analysis using outer summaries only.
  - *Full*: Precise analysis with outer and inner summaries.

  - 90 minute timeout.
- Various standard-instruction-cache and method-cache configurations.

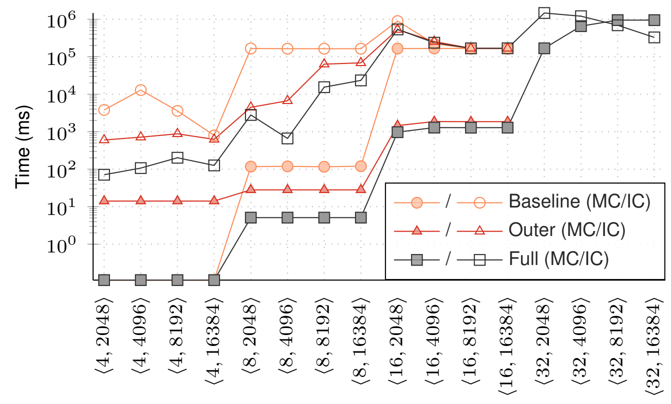
  cache sizes: 2, 4, 8, 16 KB   associativity: 4, 8, 16, 32

Analysis Complexity



Analysis time over all benchmarks for all considered cache configurations (log-scale, lower is better).

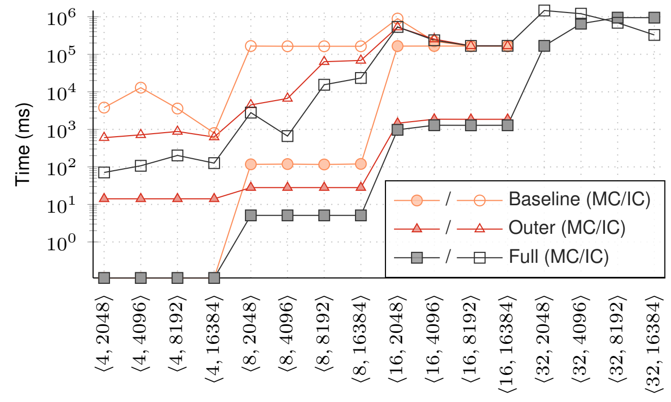
Analysis Complexity



Analysis time over all benchmarks for all considered cache configurations (log-scale, lower is better).

- *Baseline* is 20× - 100× faster than [36].

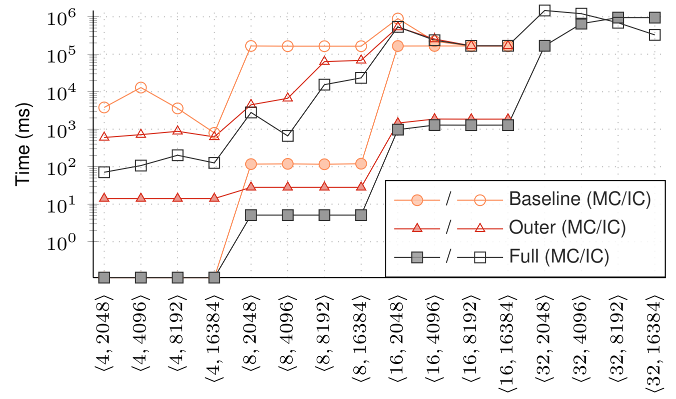
Analysis Complexity



Analysis time over all benchmarks for all considered cache configurations (log-scale, lower is better).

- *Baseline* is 20× - 100× faster than [36].
- *Full* provides speed-ups of up to 200×!

Analysis Complexity



Analysis time over all benchmarks for all considered cache configurations (log-scale, lower is better).

- *Baseline* is 20× - 100× faster than [36].
- *Full* provides speed-ups of up to 200×!
- *Full* reduces memory footprint by up to 42×.

In the Paper

- Full specification in the form of data-flow analyses.

In the Paper

- Full specification in the form of data-flow analyses.
- Examples and illustrations of analysis steps.

In the Paper

- Full specification in the form of data-flow analyses.
- Examples and illustrations of analysis steps.
- Full evaluation of analysis complexity.

In the Paper

- Full specification in the form of data-flow analyses.
- Examples and illustrations of analysis steps.
- Full evaluation of analysis complexity.
- Comparison between standard caches and the method cache.

Outer Cache Summaries

Outer Cache Summary:

n_1

$\{\{m_1\}\}$

n_2 n_3

$\{\{m_1\}\}$

$\{\{m_1\}\} \cdot \{\{m_I, m_{II}, m_{IV}\}, \{m_I, m_{III}, m_{IV}\}\}$

n_4 n_5

$\{\{m_1, m_I, m_{II}, m_{IV}\},$
$\{m_1, m_I, m_{III}, m_{IV}\}\}$

n_6

n_I

n_{II} n_{III}

n_{IV}

$\{\{m_I, m_{II}, m_{IV}\},$
$\{m_I, m_{III}, m_{IV}\}\}$

Questions?