Cache Persistence Analysis for Embedded Real-Time Systems

Christoph Cullmann

Department of Computer Science
Saarland University

AbsInt Angewandte Informatik GmbH

14 February 2013

SAARLAND UNIVERSITY

COMPUTER SCIENCE

Outline

WCET Analysis

- embedded systems are more and more wide-spread
- they are used for safety-critical tasks:
  fly-by-wire or airbag controllers
- functional and timing correctness required!
- scheduling analysis needs upper bounds of task execution time
- WCET analysis can provide such bounds

Cache Analysis

- WCET analysis needs timing model of underlying hardware
- modern embedded systems employ caches
- caches have a major impact on timing behaviour
- precise cache analysis needed to compute useful bounds

Cache Analyses - State of the Art

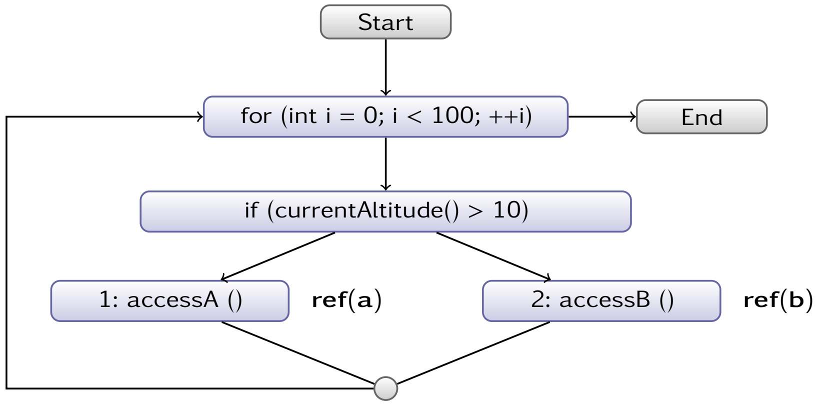
- Must Cache Analysis
  - under-approximation of the cache contents
  - classifies sure-hits
- May Cache Analysis
  - over-approximation of the cache contents
  - classifies sure-misses

Challenge - Analysis Example

- 2-way LRU cache
- accessA() and accessB() reference memory blocks a and b
- a and b map to the same cache set
- currentAltitude() can have arbitrary value per iteration
- code to analyse:

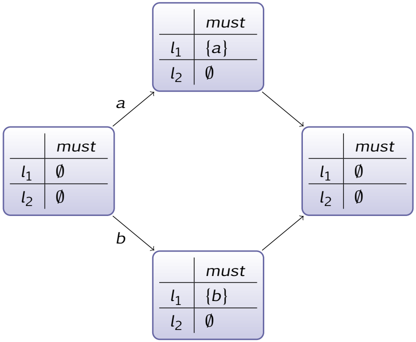
```
void runningExample () {
  for (int i = 0; i < 100; ++i) {
    if (currentAltitude() > 10) accessA ();
    else accessB ();
  }
}
```

Control Flow Graph



Must Cache Analysis

- analysis of loop body
- initial value: empty set



	must
l_1	$\emptyset$
l_2	$\emptyset$

	must
l_1	$\{a\}$
l_2	$\emptyset$

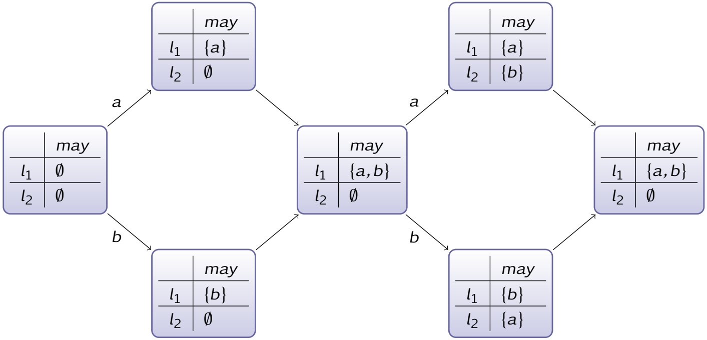
	must
l_1	$\{b\}$
l_2	$\emptyset$

	must
l_1	$\emptyset$
l_2	$\emptyset$

- result: no references classified as sure-hits

May Cache Analysis

- analysis of loop body
- initial value: empty set



may l_1 $\emptyset$ l_2 $\emptyset$
a
may l_1 $\{a\}$ l_2 $\emptyset$
b
may l_1 $\{b\}$ l_2 $\emptyset$
may l_1 $\{a,b\}$ l_2 $\emptyset$
a
may l_1 $\{a\}$ l_2 $\{b\}$
b
may l_1 $\{b\}$ l_2 $\{a\}$
may l_1 $\{a,b\}$ l_2 $\emptyset$

- result: no references classified as sure-misses

Summary of Results

- may and must cache analyses fail to provide classifications
- references to *a* and *b* can be hit or miss
- WCET analysis will need to take 100 misses into account

Cache Persistence - Intuition

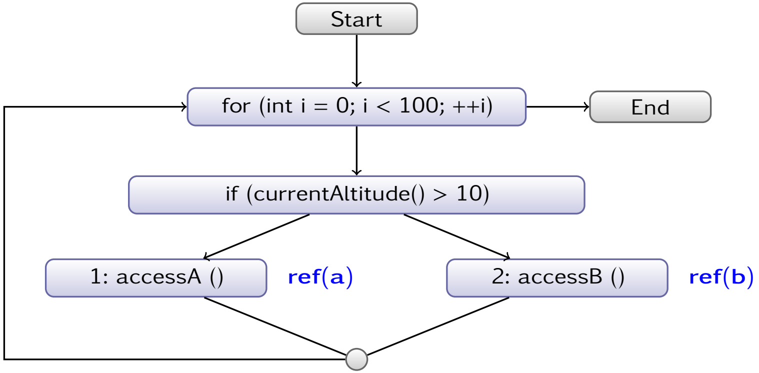
- back to the example:

```
void runningExample () {
  for (int i = 0; i < 100; ++i) {
    if (currentAltitude() > 10) accessA ();
    else accessB ();
  }
}
```

- cache set is large enough to contain both memory blocks
- intuition: only first load of *a* and *b* can miss the cache
  $\Rightarrow$ concept of persistence or first-miss classification

Persistence - Basic Idea

- classify references to memory blocks as persistent
- limit number of misses for all such classified references
- Running example:



Persistence - Definition

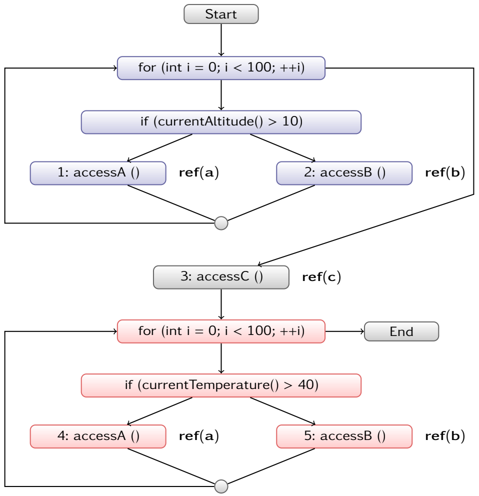
A reference to memory block m in node n is classified as persistent iff for all paths to n one of these conditions holds:

- it is the first reference to m
- the reference will cause a cache hit

$\Rightarrow$ for any path through the graph all persistent references to m can cause at most one cache miss.

Persistence - Extension to Scopes

- extend persistence definition to sub graphs
- Example:



Start
for (int i = 0; i < 100; ++i)
if (currentAltitude() > 10)
1: accessA ()
ref(a)
2: accessB ()
ref(b)
3: accessC ()
ref(c)
for (int i = 0; i < 100; ++i)
End
if (currentTemperature() > 40)
4: accessA ()
ref(a)
5: accessB ()
ref(b)

Persistence - Analyses

- Set-Wise Conflict Counting Persistence Analysis
- Element-Wise Conflict Counting Persistence Analysis
- May Analysis Based Cache Persistence Analysis
- Age-Tracking Conflict Counting Persistence Analysis

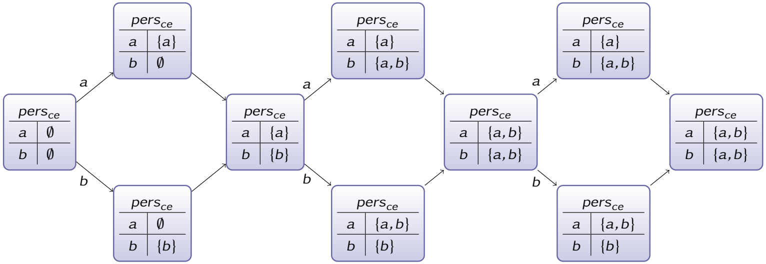
Persistence - Set-Wise Conflict Counting

- collect all memory blocks potentially referenced
- references are persistent, if set doesn't become overfull
- similar to the *first-miss* analysis by Müller
- for our example:

$$\begin{tikzcd}
 & \begin{array}{c} pers_{cs} \\ \hline \{a\} \end{array} \arrow[dr] & & \begin{array}{c} pers_{cs} \\ \hline \{a,b\} \end{array} \arrow[dr] & \\
\begin{array}{c} pers_{cs} \\ \hline \emptyset \end{array} \arrow[ur, "a"] \arrow[dr, "b"'] & & \begin{array}{c} pers_{cs} \\ \hline \{a,b\} \end{array} \arrow[ur, "a"] \arrow[dr, "b"'] & & \begin{array}{c} pers_{cs} \\ \hline \{a,b\} \end{array} \\
 & \begin{array}{c} pers_{cs} \\ \hline \{b\} \end{array} \arrow[ur] & & \begin{array}{c} pers_{cs} \\ \hline \{a,b\} \end{array} \arrow[ur] &
\end{tikzcd}$$

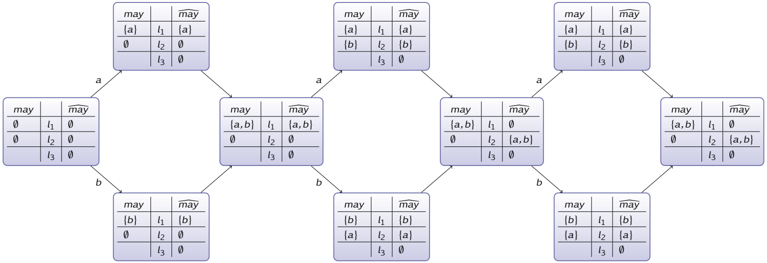
Persistence - Element-Wise Conflict Counting

- for each referenced block:
  collect possible conflicting blocks
- reference is persistent, if the set for the referenced block doesn't become overfull
- similar to the persistence concepts by Huynh et al.
- for our example:


$pers_{ce}$: a ∅, b ∅
a
b
$pers_{ce}$: a {a}, b ∅
$pers_{ce}$: a ∅, b {b}
$pers_{ce}$: a {a}, b {b}
a
b
$pers_{ce}$: a {a}, b {a,b}
$pers_{ce}$: a {a,b}, b {b}
$pers_{ce}$: a {a,b}, b {a,b}
a
b
$pers_{ce}$: a {a}, b {a,b}
$pers_{ce}$: a {a,b}, b {b}
$pers_{ce}$: a {a,b}, b {a,b}

Persistence - May Analysis Based Persistence

- track minimal (*may*) and maximal ($\widehat{may}$) ages of all potentially referenced memory blocks
- reference to a block is persistent, if block is guaranteed to be not evicted
- for our example:



may		$\widehat{may}$
$\emptyset$	l_1	$\emptyset$
$\emptyset$	l_2	$\emptyset$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{a\}$	l_1	$\{a\}$
$\emptyset$	l_2	$\emptyset$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{b\}$	l_1	$\{b\}$
$\emptyset$	l_2	$\emptyset$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{a,b\}$	l_1	$\{a,b\}$
$\emptyset$	l_2	$\emptyset$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{a\}$	l_1	$\{a\}$
$\{b\}$	l_2	$\{b\}$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{b\}$	l_1	$\{b\}$
$\{a\}$	l_2	$\{a\}$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{a,b\}$	l_1	$\emptyset$
$\emptyset$	l_2	$\{a,b\}$
	l_3	$\emptyset$

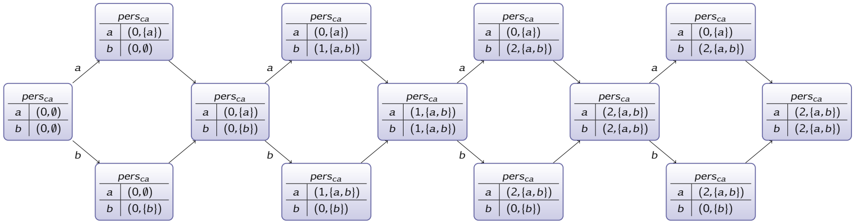
may		$\widehat{may}$
$\{a\}$	l_1	$\{a\}$
$\{b\}$	l_2	$\{b\}$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{b\}$	l_1	$\{b\}$
$\{a\}$	l_2	$\{a\}$
	l_3	$\emptyset$

may		$\widehat{may}$
$\{a,b\}$	l_1	$\emptyset$
$\emptyset$	l_2	$\{a,b\}$
	l_3	$\emptyset$

Persistence - Age-Tracking Conflict Counting

- for each referenced block:
  collect maximal age and possible conflicting blocks
- reference is persistent, if the set for the referenced block doesn't become overfull or age is still small enough
- for our example:



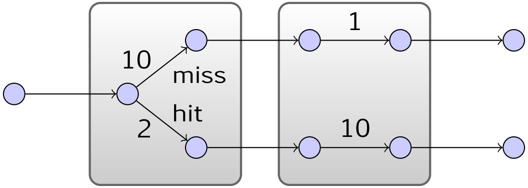
$pers_{ca}$: a (0, ∅), b (0, ∅)
a
$pers_{ca}$: a (0, {a}), b (0, ∅)
b
$pers_{ca}$: a (0, ∅), b (0, {b})
$pers_{ca}$: a (0, {a}), b (0, {b})
a
$pers_{ca}$: a (0, {a}), b (1, {a, b})
b
$pers_{ca}$: a (1, {a, b}), b (0, {b})
$pers_{ca}$: a (1, {a, b}), b (1, {a, b})
a
$pers_{ca}$: a (0, {a}), b (2, {a, b})
b
$pers_{ca}$: a (2, {a, b}), b (0, {b})
$pers_{ca}$: a (2, {a, b}), b (2, {a, b})
a
$pers_{ca}$: a (0, {a}), b (2, {a, b})
b
$pers_{ca}$: a (2, {a, b}), b (0, {b})
$pers_{ca}$: a (2, {a, b}), b (2, {a, b})

Implementation - Challenges

Modern embedded architectures "feature"

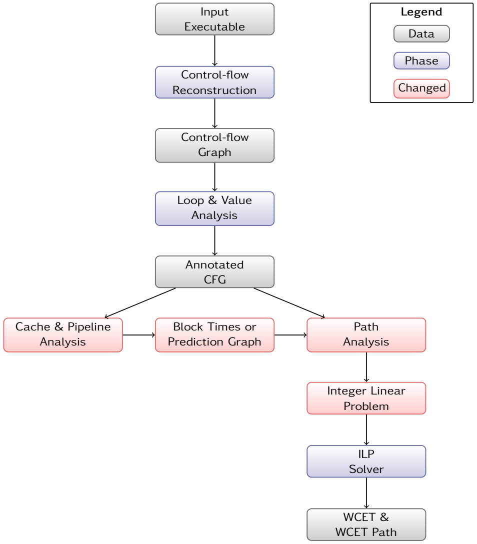
- timing anomalies
- domino effects

⇒ WCET analysis must follow all possible decision paths!



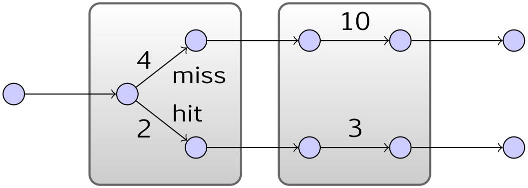
⇒ no cumulative post-pass analysis possible!
⇒ tight integration into analysis framework needed!

Implementation - WCET Analysis Framework

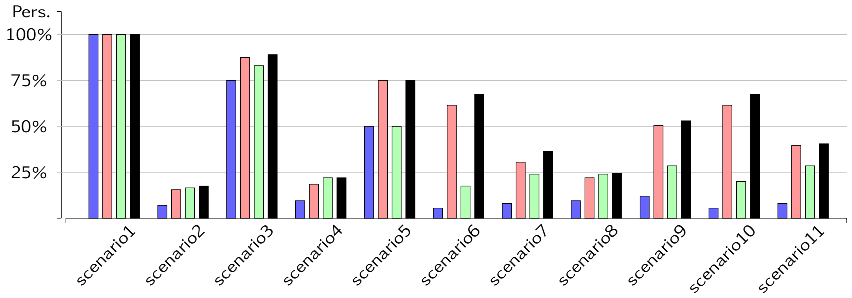


Implementation - Framework Modifications

- integration of persistence analysis into cache & pipeline analysis
- path analysis not on block but on prediction graph level
  $\Rightarrow$ path analysis will use graph created by abstract state evolution of the pipeline
- propagation of persistence classification into prediction graph
  $\Rightarrow$ persistence constraints per edge after hit/miss split in pipeline state graph



Evaluation - Synthetic Benchmarks



🟦 set-wise conflict counting

🟥 element-wise conflict counting

🟩 may-based

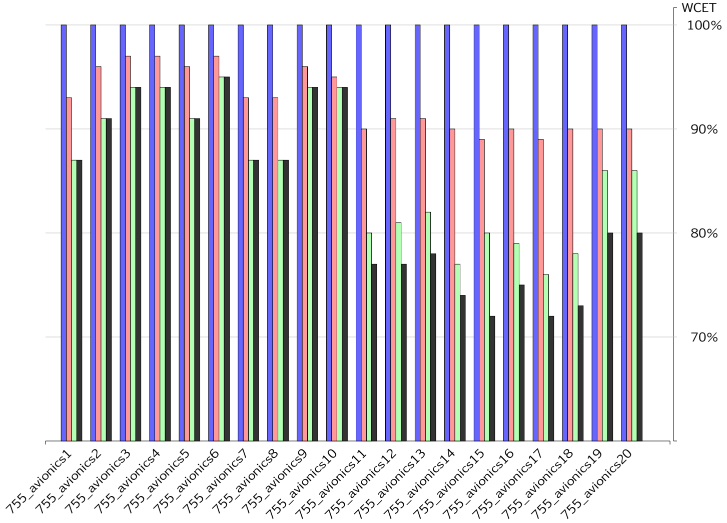
⬛ age-tracking conflict counting

Evaluation - aiT WCET Analyzer

Chosen Architectures & Tests

- ARM7 TDMI
  - a fully timing compositional architecture
  - Tests: synthetic examples from thesis and *WCET benchmarks*
- Freescale MPC5554
  - a compositional architecture with constant-bounded effects
  - Tests: avionics tasks
- Freescale MPC755
  - a non-compositional architecture
  - Tests: avionics tasks

Evaluation - MPC755 Avionics



🟦 S1: ILP Based, Reference = 100%

🟥 S2: Prediction Graph

🟩 S3: Set-Wise Conflict Counting Persistence

⬛ S3: Age-Tracking Conflict Counting Persistence

Evaluation - Summary

- Synthetic Benchmarks
  - best precision: age-tracking conflict counting
- aiT Integration
  - persistence analysis applicable to real-world applications
  - persistence analysis can handle all three architectural classes
  - precision improvement between 7% and 15% on average

Conclusion

Main Contributions:

- Concise formalization of cache persistence
- Presentation of four analyses, including two novel ones:
  - May Analysis Based Cache Persistence
  - Age-Tracking Conflict Counting Cache Persistence
- Soundness proofs for all analyses
- Integration in state-of-the-art WCET analysis framework
  $\Rightarrow$ Analysis of complex architectures possible
- Exhaustive evaluation

Persistence by Ferdinand - Bug

```
void switchLoop () {
  for (int i = 0; i < 100; ++i) {
    switch (somethingUnknown()) {
      case 0: accessA (); break;
      case 1: accessB (); break;
      default: accessC (); break;
    }
  }
}
```

Figure: Loop with switch construct accessing memory blocks a, b or c depending on a condition not known statically.

Persistence by Ferdinand - Bug

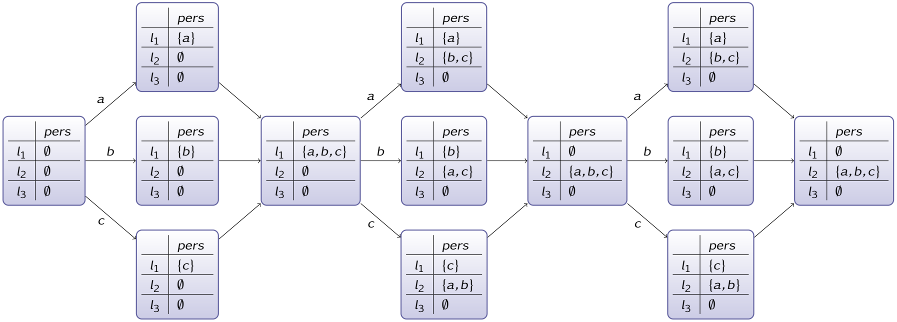

pers
l_1 $\emptyset$
l_2 $\emptyset$
l_3 $\emptyset$
a
b
c
l_1 $\{a\}$ l_2 $\emptyset$ l_3 $\emptyset$
l_1 $\{b\}$ l_2 $\emptyset$ l_3 $\emptyset$
l_1 $\{c\}$ l_2 $\emptyset$ l_3 $\emptyset$
l_1 $\{a,b,c\}$ l_2 $\emptyset$ l_3 $\emptyset$
l_1 $\{a\}$ l_2 $\{b,c\}$ l_3 $\emptyset$
l_1 $\{b\}$ l_2 $\{a,c\}$ l_3 $\emptyset$
l_1 $\{c\}$ l_2 $\{a,b\}$ l_3 $\emptyset$
l_1 $\emptyset$ l_2 $\{a,b,c\}$ l_3 $\emptyset$
l_1 $\{a\}$ l_2 $\{b,c\}$ l_3 $\emptyset$
l_1 $\{b\}$ l_2 $\{a,c\}$ l_3 $\emptyset$
l_1 $\{c\}$ l_2 $\{a,b\}$ l_3 $\emptyset$
l_1 $\emptyset$ l_2 $\{a,b,c\}$ l_3 $\emptyset$

Figure: Fixed point iteration for the persistence analysis by Ferdinand for the switch loop example: After three rounds, the fixed point is reached. The memory references to the three memory blocks *a*, *b* and *c* are classified as *persistent*.

Imprecision - Prefix in Loop

```
void prefixLoop () {
  for (int i = 0; i < NUMBER_OF_EVENTS; ++i) {
    if (i == 0) // only done in first round
      accessA ();
    if (somethingUnknown()) accessB ();
    else accessC ();
  }
}
```

Figure: Loop which accesses three memory blocks *a*, *b* and *c* mapping to the same cache set. *a* is only accessed once in the first iteration.

Imprecision - Switch in Loop

```
void switchLoop () {
  for (int i = 0; i < 100; ++i) {
    switch (somethingUnknown()) {
      case  0: accessA (); break;
      case  1: accessB (); break;
      default: accessC (); break;
    }
  }
}
```

Figure: Loop with switch construct accessing memory blocks a, b or c depending on a condition not known statically.

Imprecision - Persistence in Loop-Body

```
void innerPersistenceLoop () {
  for (int i = 0; i < NUMBER_OF_EVENTS; ++i) {
    if (somethingUnknown1()) accessA ();
    else accessB ();
    accessC ();
    if (somethingUnknown2()) accessA ();
    else accessB ();
  }
}
```

Figure: Loop which accesses three memory blocks *a*, *b* and *c* mapping to the same cache set. *a* and *b* are possibly accessed twice.

Evaluation - Results MPC755 Part 1

	S1: ILP Based		S2: Prediction Graph		
	WCET	Time	WCET	Time	Impr. (S1)
755_avionics1	30184	0:21	28212	0:25	6.53%
755_avionics2	82139	0:29	78754	0:36	4.12%
755_avionics3	74724	0:31	72202	0:35	3.38%
755_avionics4	69129	0:24	66788	0:27	3.39%
755_avionics5	80940	0:26	77784	0:31	3.90%
755_avionics6	103588	0:22	100604	0:24	2.88%
755_avionics7	30386	0:21	28346	0:25	6.71%
755_avionics8	31117	0:23	29006	0:27	6.78%
755_avionics9	69078	0:25	66651	0:27	3.51%
755_avionics10	84967	0:31	81894	0:36	3.62%
755_avionics11	895184	1:21	808165	1:32	9.72%
755_avionics12	1042489	1:30	946987	1:50	9.16%
755_avionics13	1188760	1:45	1079622	2:02	9.18%
755_avionics14	595568	1:03	538369	1:05	9.60%
755_avionics15	1074711	1:25	954226	1:34	11.21%
755_avionics16	923855	1:46	830002	1:55	10.16%
755_avionics17	842246	1:46	751052	1:52	10.83%
755_avionics18	991719	1:57	893113	2:01	9.94%
755_avionics19	1093188	1:26	979889	1:41	10.36%
755_avionics20	1108734	1:27	994473	1:42	10.31%

Table: WCET in cycles and analysis runtime in minutes for the *MPC755* with settings S1 and S2.

Evaluation - Results MPC755 Part 2

	S2: Conflicts			S3: Conflicts & Aging		
	WCET	Time	Impr. (S1)	WCET	Time	Impr. (S1)
755_avionics1	26216	0:28	13.15%	26216	0:26	13.15%
755_avionics2	74758	0:37	8.99%	74758	0:37	8.99%
755_avionics3	70206	0:37	6.05%	70206	0:37	6.05%
755_avionics4	64792	0:29	6.27%	64792	0:29	6.27%
755_avionics5	73788	0:40	8.84%	73788	0:40	8.84%
755_avionics6	98608	0:26	4.81%	98608	0:26	4.81%
755_avionics7	26350	0:28	13.28%	26350	0:26	13.28%
755_avionics8	27010	0:29	13.20%	27010	0:29	13.20%
755_avionics9	64655	0:29	6.40%	64655	0:28	6.40%
755_avionics10	79898	0:36	5.97%	79898	0:37	5.97%
755_avionics11	716200	2:30	19.99%	690200	3:41	22.90%
755_avionics12	845059	3:01	18.94%	803926	5:00	22.88%
755_avionics13	976392	3:42	17.86%	931647	6:21	21.63%
755_avionics14	456859	1:48	23.29%	443370	2:37	25.56%
755_avionics15	855662	3:30	20.38%	771011	7:00	28.26%
755_avionics16	725449	3:13	21.48%	692313	5:52	25.06%
755_avionics17	636029	3:13	24.48%	608555	4:37	27.75%
755_avionics18	778054	3:47	21.54%	724181	4:54	26.98%
755_avionics19	940102	3:40	14.00%	874127	7:45	20.04%
755_avionics20	954686	3:26	13.89%	887758	7:53	19.93%

Table: WCET in cycles and analysis runtime in minutes for the *MPC755* with setting S3 and the set-wise or age-tracking conflict counting persistence analysis.

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Extension - Global Must Cache Information

```
void reusedMemoryBlocks () {
  for (int i = 0; i < EVENTS; ++i) {
    accessA ();
    if (somethingUnknownForEachCall()) {
        accessB ();
        accessB ();
    } else {
        accessC ();
        accessC ();
    }
  }
}
```

Figure: Loop which accesses three memory blocks *a*, *b* and *c* mapping to the same cache set. *b* and *c* are always accessed twice inside their if-then-else branch. (Nagar)

Extension - Global May Cache Information

```
void writeToCache () {
  for (int i = 0; i < EVENTS; ++i) {
    if (somethingUnknownForEachCall())
        accessA ();
    writeOnlyB ();
  }
}
```

Figure: Loop which accesses two memory blocks *a* and *b* mapping to the same cache set. *b* is only written, never read.

Extension - Non-LRU Replacements

- k-way PLRU (Reineke)
  $\Rightarrow$ use analysis for $(log_2(k) + 1)$-way LRU
- k-way FIFO (Grund)
  at most k memory blocks referenced $\Rightarrow$ at most one miss per memory block
- k-way MRU (Guan et al.)
  at most k memory blocks referenced $\Rightarrow$ at most k misses per memory block