Cache Persistence Analysis for Embedded Real-Time Systems

Christoph Cullmann

Department of Computer Science
Saarland University

AbsInt Angewandte Informatik GmbH

14 February 2013
Outline

1 Introduction and Motivation
   ▶ WCET Analysis
   ▶ Cache Analysis
   ▶ Challenge

2 Cache Persistence
   ▶ Intuition
   ▶ Definition
   ▶ Analyses
   ▶ Implementation

3 Evaluation
   ▶ Synthetic Benchmarks
   ▶ Real-World Examples

4 Conclusion
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:

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

Start

for (int i = 0; i < 100; ++i)

if (currentAltitude() > 10)

1: accessA ()
2: accessB ()
ref(a)   ref(b)

End
Must Cache Analysis

- analysis of loop body
- initial value: empty set

![Diagram showing analysis of loop body with initial value empty set and resulting no references classified as sure-hits]
## May Cache Analysis

- analysis of loop body
- initial value: empty set

<table>
<thead>
<tr>
<th></th>
<th>may</th>
<th></th>
<th>may</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$\emptyset$</td>
<td>$l_1$</td>
<td>${a}$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_2$</td>
<td>$\emptyset$</td>
<td>$l_2$</td>
<td>$\emptyset$</td>
<td>$l_2$</td>
</tr>
</tbody>
</table>

- 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:

```c
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
  ⇒ 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:

```java
for (int i = 0; i < 100; ++i)
if (currentAltitude() > 10)
1: accessA ()
ref(a)
2: accessB ()
ref(b)
```
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:

```
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)
        if (currentTemperature() > 40)
            4: accessA ()     ref(a)
            5: accessB ()     ref(b)
```

Christoph Cullmann
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:

```
pers_{cs} \emptyset

pers_{cs} \{a\}

pers_{cs} \{b\}

pers_{cs} \{a\}

pers_{cs} \{a, b\}

a

pers_{cs} \{a, b\}

a

pers_{cs} \{a\}

b

pers_{cs} \{a, b\}

b

pers_{cs} \{b\}
```
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:
track minimal (may) and maximal (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:
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:
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

1. Input Executable
2. Control-flow Reconstruction
3. Control-flow Graph
4. Loop & Value Analysis
5. Annotated CFG
6. Cache & Pipeline Analysis
7. Block Times or Prediction Graph
8. Path Analysis
9. Integer Linear Problem
10. ILP Solver
11. WCET & WCET Path

Legend:
- Data
- Phase
- Changed

Christoph Cullmann
Cache Persistence Analysis
14 February 2013
- integration of persistence analysis into cache & pipeline analysis
- path analysis not on block but on prediction graph level
  ⇒ path analysis will use graph created by abstract state evolution of the pipeline
- propagation of persistence classification into prediction graph
  ⇒ persistence constraints per edge after hit/miss split in pipeline state graph
Evaluation - Synthetic Benchmarks

Pers.

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
  ⇒ Analysis of complex architectures possible
- Exhaustive evaluation
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.
**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.
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.
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.
void innerPersistenceLoop() {
    for (int i = 0; i < NUMBER_OF_EVENTS; ++i) {
        if (somethingUnknown1()) accessA();
        else accessB();
        accessC();
        if (somethingUnknown2()) accessA();
        else accessB();
    }
}
### Evaluation - Results MPC755 Part 1

<table>
<thead>
<tr>
<th></th>
<th>S1: ILP Based</th>
<th></th>
<th></th>
<th>S2: Prediction Graph</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCET</td>
<td>Time</td>
<td>WCET</td>
<td>Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics1</td>
<td>30184</td>
<td>0:21</td>
<td>28212</td>
<td>0:25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics2</td>
<td>82139</td>
<td>0:29</td>
<td>78754</td>
<td>0:36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics3</td>
<td>74724</td>
<td>0:31</td>
<td>72202</td>
<td>0:35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics4</td>
<td>69129</td>
<td>0:24</td>
<td>66788</td>
<td>0:27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics5</td>
<td>80940</td>
<td>0:26</td>
<td>77784</td>
<td>0:31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics6</td>
<td>103588</td>
<td>0:22</td>
<td>100604</td>
<td>0:24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics7</td>
<td>30386</td>
<td>0:21</td>
<td>28346</td>
<td>0:25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics8</td>
<td>31117</td>
<td>0:23</td>
<td>29006</td>
<td>0:27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics9</td>
<td>69078</td>
<td>0:25</td>
<td>66651</td>
<td>0:27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics10</td>
<td>84967</td>
<td>0:31</td>
<td>81894</td>
<td>0:36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics11</td>
<td>895184</td>
<td>1:21</td>
<td>808165</td>
<td>1:32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics12</td>
<td>1042489</td>
<td>1:30</td>
<td>946987</td>
<td>1:50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics13</td>
<td>1188760</td>
<td>1:45</td>
<td>1079622</td>
<td>2:02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics14</td>
<td>595568</td>
<td>1:03</td>
<td>538369</td>
<td>1:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics15</td>
<td>1074711</td>
<td>1:25</td>
<td>954226</td>
<td>1:34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics16</td>
<td>923855</td>
<td>1:46</td>
<td>830002</td>
<td>1:55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics17</td>
<td>842246</td>
<td>1:46</td>
<td>751052</td>
<td>1:52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics18</td>
<td>991719</td>
<td>1:57</td>
<td>893113</td>
<td>2:01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics19</td>
<td>1093188</td>
<td>1:26</td>
<td>979889</td>
<td>1:41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>755_avionics20</td>
<td>1108734</td>
<td>1:27</td>
<td>994473</td>
<td>1:42</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table:** WCET in cycles and analysis runtime in minutes for the MPC755 with settings S1 and S2.
### Evaluation - Results MPC755 Part 2

<table>
<thead>
<tr>
<th></th>
<th>S2: Conflicts</th>
<th>S3: Conflicts &amp; Aging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WCET</td>
<td>Time</td>
</tr>
<tr>
<td>755_avionics1</td>
<td>26216</td>
<td>0:28</td>
</tr>
<tr>
<td>755_avionics2</td>
<td>74758</td>
<td>0:37</td>
</tr>
<tr>
<td>755_avionics3</td>
<td>70206</td>
<td>0:37</td>
</tr>
<tr>
<td>755_avionics4</td>
<td>64792</td>
<td>0:29</td>
</tr>
<tr>
<td>755_avionics5</td>
<td>73788</td>
<td>0:40</td>
</tr>
<tr>
<td>755_avionics6</td>
<td>98608</td>
<td>0:26</td>
</tr>
<tr>
<td>755_avionics7</td>
<td>26350</td>
<td>0:28</td>
</tr>
<tr>
<td>755_avionics8</td>
<td>27010</td>
<td>0:29</td>
</tr>
<tr>
<td>755_avionics9</td>
<td>64655</td>
<td>0:29</td>
</tr>
<tr>
<td>755_avionics10</td>
<td>79898</td>
<td>0:36</td>
</tr>
<tr>
<td>755_avionics11</td>
<td>716200</td>
<td>2:30</td>
</tr>
<tr>
<td>755_avionics12</td>
<td>845059</td>
<td>3:01</td>
</tr>
<tr>
<td>755_avionics13</td>
<td>976392</td>
<td>3:42</td>
</tr>
<tr>
<td>755_avionics14</td>
<td>456859</td>
<td>1:48</td>
</tr>
<tr>
<td>755_avionics15</td>
<td>855662</td>
<td>3:30</td>
</tr>
<tr>
<td>755_avionics16</td>
<td>725449</td>
<td>3:13</td>
</tr>
<tr>
<td>755_avionics17</td>
<td>636029</td>
<td>3:13</td>
</tr>
<tr>
<td>755_avionics18</td>
<td>778054</td>
<td>3:47</td>
</tr>
<tr>
<td>755_avionics19</td>
<td>940102</td>
<td>3:40</td>
</tr>
<tr>
<td>755_avionics20</td>
<td>954686</td>
<td>3:26</td>
</tr>
</tbody>
</table>

**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.
Extension - Global Must Cache Information

```c
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)
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)
  ⇒ use analysis for \((\log_2(k) + 1)\)-way LRU

- k-way FIFO (Grund)
  at most \(k\) memory blocks referenced ⇒ at most one miss per memory block

- k-way MRU (Guan et al.)
  at most \(k\) memory blocks referenced ⇒ at most \(k\) misses per memory block