Relational Cache Analysis for Static Timing Analysis

Sebastian Hahn  Daniel Grund

ECRTS 2012
Static Timing Analysis

- influence of the hardware on execution time
  - caches, pipelines, ...
- tight bounds require micro-architectural analysis, e.g. cache analysis
Static Cache Analysis

- Approximate cache content at each program point
- Classify memory references as cache hit or cache miss

<table>
<thead>
<tr>
<th>Must-cache analysis</th>
<th>May-cache analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>➤ under-approximation</td>
<td>➤ over-approximation</td>
</tr>
<tr>
<td>➤ classify hits</td>
<td>➤ classify misses</td>
</tr>
</tbody>
</table>
Static Cache Analysis

Challenges

- Initial cache contents unknown
- Need to combine analysis information
- Need to determine information about $w$, $x$, $y$, $z$

```
\begin{align*}
    w & \in [0x60810, 0x60810] \\
    x & \in [0xc0000, 0xc0008] \\
    y & \in [0x60814, 0x60814] \\
    z & \in [0x00000, 0xfffff]
\end{align*}
```
Static Cache Analysis

Two-step Approach

1. Approximate accessed addresses by Value Analysis

Approximate accessed addresses by Value Analysis

Approximate cached memory blocks by Cache Analysis

Sebastian Hahn and Daniel Grund

Relational Cache Analysis

ECRTS 2012

1. Approximate accessed addresses by Value Analysis

- \( w \in [0x60810, 0x60810] \)
- \( x \in [0xc0000, 0xc0008] \)
- \( y \in [0x60814, 0x60814] \)
- \( z \in [0x00000, 0xfffff] \)
Static Cache Analysis for LRU replacement policy

Two-step Approach

1. Approximate accessed addresses by Value Analysis
2. Approximate cached memory blocks by Cache Analysis
Example

1. Address Information
   \[ a[0], a[1] \in [0x00000, 0xffffff] \]

2. Cache Information
   \{0x60810\}

```c
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```
Example

1. Address Information
   \[a[0], a[1] \in [0x00000, 0xfffff]\]

2. Cache Information

   \{0x60810\}

Intangibility of Memory Blocks
\(\Rightarrow\) not guaranteed to be cached
Example

```c
int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
```

1. Address Information
   \[ a[0], a[1] \in [0x00000, 0xfffff] \]

2. Cache Information
   \[
   \{0x60810\}
   \]

Intangibility of Memory Blocks
\[\Rightarrow\] not guaranteed to be cached
Example

1. Address Information
   \( a[0], a[1] \in [0x00000, 0xfffff] \)

2. Cache Information

   \[
   \begin{array}{c}
   \text{int } tmp1 = a[0]; \\
   \text{int } tmp2 = a[1]; \\
   a[0] \quad = tmp2; \\
   a[1] \quad = tmp1;
   \end{array}
   \]

   Intangibility of Memory Blocks
   \( \Rightarrow \) not guaranteed to be cached

   Excessive Information Loss
Example

\[
\begin{align*}
\text{int} & \quad \text{tmp1} = a[0]; \\
\text{int} & \quad \text{tmp2} = a[1]; \\
a[0] & \quad = \text{tmp2}; \\
a[1] & \quad = \text{tmp1};
\end{align*}
\]

1. Address Information
\[a[0], a[1] \in [0x00000, 0xfffff]\]

2. Cache Information

Intangibility of Memory Blocks
\[\Rightarrow \text{not guaranteed to be cached}\]

Excessive Information Loss
\[\Rightarrow \text{not guaranteed to be cached anymore}\]
Multiple Cache Sets

Unknown Address

Index

? \rightarrow \{0x60810\} \rightarrow ...

? \rightarrow \{0xc0808\} \rightarrow ...

? \rightarrow \{0x4060c\}
Multiple Cache Sets

Multiple Aging
⇒ any cache set might be affected
Precisely determined addresses are not necessary for precise cache analysis.
Precisely determined addresses are not necessary for precise cache analysis.

But relations between addresses.
Outline

1 Introduction and Problem

2 Relational Cache Analysis
   - Symbolic Names
   - Relational Framework
   - Relations and Congruence Information
   - Cache Analysis

3 Implementation and Evaluation
Symbolic Names

Definition (Symbolic Name)
Unique identifier for an occurrence of an address expression

... int tmp1 = a[0];
int tmp2 = a[1];
a[0] = tmp2;
a[1] = tmp1;
...

... add r5, r1, 0 bind s2
ld r10, [r5] deref. s2
add r6, r1, 4 bind s3
ld r11, [r6] deref. s3
st [r5], r11 deref. s2
st [r6], r10 deref. s3
...

Sebastian Hahn and Daniel Grund
Relational Cache Analysis
ECRTS 2012
Abstract Caches

- symbolic names as abstract cache elements

<table>
<thead>
<tr>
<th>{s_2}</th>
<th>{s_1}</th>
</tr>
</thead>
</table>

during execution, the memory block represented by \( s_1 \) has at most age 1

- symbolic names abstract from concrete addresses
  \[ \rightarrow \] all memory references tangible
Relations between Symbolic Names

... add r6, r1, 4 bind s₃
ld r11, [r6] deref. s₃
...

{s₂}
{s₁}
deref. s₃

Classify reference s₃ as hit?
How are s₁ and s₂ affected?

Use of relational information

- s₃ and s₁ denote the **same memory block** → classify reference as hit
- s₃ and s₂ map to **different cache sets** → s₂ not affected (e.g. no aging)
Overall Framework

Congruence Analysis
- Interval Analysis
- Global Value Numbering
- ...

Relational Cache Analysis
classify \( s_3 \)

\((s_3, s_2)\)?

\(\{s_5, s_4\}\)
\(\{s_2\}\)

(same block)!

Hit
Relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss</td>
<td>same cache set</td>
</tr>
<tr>
<td>ds</td>
<td>different cache set</td>
</tr>
<tr>
<td>sb</td>
<td>same block</td>
</tr>
<tr>
<td>db</td>
<td>different block</td>
</tr>
<tr>
<td>ss db</td>
<td>ss and db</td>
</tr>
<tr>
<td>ss db</td>
<td>ds or sb</td>
</tr>
</tbody>
</table>

sb classify hits

ss db account for cache conflicts

ds exclude possible eviction
Relations

\[ \begin{align*} 
\text{Relation} & \quad \text{Meaning} \\
ss & \quad \text{same cache set} \\
ds & \quad \text{different cache set} \\
sb & \quad \text{same block} \\
db & \quad \text{different block} \\
ss \ db & \quad \text{ss and db} \\
ss \ db & \quad \text{ds or sb} 
\end{align*} \]

Induces partial order \( \sqsubseteq \): 
\[ sb \sqsubseteq \overline{ss \ db} \quad \text{and} \quad ds \sqsubseteq \overline{ss \ db} \]

\[ \begin{align*} 
sb & \quad \text{classify hits} \\
ss \ db & \quad \text{account for cache conflicts} \\
ds & \quad \text{exclude possible eviction} 
\end{align*} \]
Congruence Information

Partial Execution Trace $\tau$

\[
\langle s_1 \mapsto 0x60810 \rangle
\circ \langle s_1 \rangle
\circ \langle s_2 \mapsto 0xbfffc0 \rangle
\circ \langle s_2 \rangle
\circ \langle s_3 \mapsto 0xbfffc4 \rangle
\circ \langle s_3 \rangle
\]

\[
rel(\tau, s_1, s_3)
= \hat{rel}(last(\tau, s_1), last(\tau, s_3))
= \hat{rel}(0x60810, 0xbfffc4)
= ds
\]
Congruence Information

Partial Execution Trace $\tau$

\[
\langle s_1 \mapsto 0x60810 \rangle
\circ \langle s_1 \rangle
\circ \langle s_2 \mapsto 0xbffe0 \rangle
\circ \langle s_2 \rangle
\circ \langle s_3 \mapsto 0xbffe4 \rangle
\circ \langle s_3 \rangle
\]

\[
\text{rel}(\tau, s_1, s_3)
= \hat{\text{rel}}(\text{last}(\tau, s_1), \text{last}(\tau, s_3))
= \hat{\text{rel}}(0x60810, 0xbffe4)
= \text{ds}
\]

$\rightarrow$ congruence information has to safely account for both cases

Partial Execution Trace $\tau'$

\[
\langle s_1 \mapsto 0x60810 \rangle
\circ \langle s_1 \rangle
\circ \langle s_2 \mapsto 0xbffe4 \rangle
\circ \langle s_2 \rangle
\circ \langle s_3 \mapsto 0xbffe8 \rangle
\circ \langle s_3 \rangle
\]

\[
\text{rel}(\tau', s_1, s_3)
= \hat{\text{rel}}(\text{last}(\tau', s_1), \text{last}(\tau', s_3))
= \hat{\text{rel}}(0x60810, 0xbffe8)
= \text{ss db}
\]
Congruence Information

Congruence information modelled as one function

\[ cgr_v : \mathcal{N} \times \mathcal{N} \rightarrow \mathcal{R} \]

per program location \( v \).

Definition (Validity of Congruence Information)

Let \( \mathcal{T}_v \) be the set of partial execution traces up to program location \( v \). \( cgr_v \) is called valid if for all \( \tau \in \mathcal{T}_v \) and for all \( s, t \in \mathcal{N} \)

\[ cgr_v(s, t) \sqsubseteq \text{rel}(\tau, s, t). \]
Computing Congruence Information

Global Value Numbering [Rosen, Wegman, and Zadeck, 1988]

\[
\begin{align*}
\text{\textit{vn}} : \text{expressions} & \rightarrow \mathbb{N} \\
\text{\textit{vn}}(e_1) = \text{\textit{vn}}(e_2) & \Rightarrow e_1 \text{ and } e_2 \text{ compute the same value}
\end{align*}
\]

Symbolic names \(s_1\) and \(s_2\) with associated address expressions . . .

- address expressions \(e_1\) and \(e_2\), where \(\text{\textit{vn}}(e_1) = \text{\textit{vn}}(e_2)\)
  \(\Rightarrow\) \(\text{sb}\) relation

- address expressions \(e_1\) and \(e_2 + \text{linesize}\), where \(\text{\textit{vn}}(e_1) = \text{\textit{vn}}(e_2)\)
  \(\Rightarrow\) \(\text{ds}\) relation
Relational Cache Analysis

Similar to Ferdinand’s must cache analysis [Ferdinand, 1997], but

- symbolic names as abstract cache elements instead of memory blocks
  → abstract from concrete addresses

- more general congruence information instead of address information
  → e.g. the address information can be used to compute relations
Outline

1. Introduction and Problem

2. Relational Cache Analysis
   - Symbolic Names
   - Relational Framework
   - Relations and Congruence Information
   - Cache Analysis

3. Implementation and Evaluation
Evaluation Setting

Implementation

- **FIRM** Intermediate representation [Braun, Buchwald, and Zwinkau, 2011]
- x86 assembler graph produced by compilation
- interval analysis and global value numbering as congruence analyses

Three application areas

1. stack-relative memory accesses
2. array accesses within one loop iteration
3. input-dependent memory accesses
Input-dependent Memory Accesses
Taken from Mälardalen benchmarks [Gustafson, Betts, Ermedahl, and Lisper, 2010]

```c
void fdct(int *block, int lx) {
    ...
    /* Pass 1: process rows. */
    ...
    /* Pass 2: process columns. */
    for (i = 0; i<8; i++) {
        tmp0 = block[0] + block[7*lx];
        tmp7 = block[0] - block[7*lx];
        tmp1 = block[lx] + block[6*lx];
        tmp6 = block[lx] - block[6*lx];
        ...
        block[0] = ...
        block[6*lx] = ...
        block[7*lx] = ...
        block[lx] = ...
        ...
        /* advance to next column */
        block++;
    }
}
```

<table>
<thead>
<tr>
<th>Configuration</th>
<th>references classified as always hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>line size</td>
<td>assoc.</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>
Evaluation

Qualitative Result

Relational Cache Analysis is at least as precise as Ferdinand's Cache Analysis
Summary and Conclusion

Absolute address information not needed for precise cache analysis

- Symbolic names abstract from concrete addresses
- Congruence analysis module provides relations between symbolic names

Congruence Analysis
- Interval Analysis
- Global Value Numbering
- ...

Relational Cache Analysis
- \( \{s_5, s_4\} \)
- \( \{s_2\} \)

\((s_3, s_2)\)?

(classify \( s_3 \))

(same block)!

Hit
Future Work

Congruence Analysis
- New congruence analyses e.g.,
  - the Value-Set Analysis by Balakrishnan and Reps
  - the Congruence Analysis by Wegener at WCET 2012
- Effects of different congruence analyses on the analysis precision

Cache Analysis
- Improve abstract domain
- May analysis

Applications
- Analysing accesses to dynamically allocated data structures
Stack-relative Memory Accesses

```c
int comp(int a1, int a2, int a3,
         int b1, int b2, int b3,
         int c1, int c2, int c3) {
    int p1 = a2 * b3 + a3 * b2;
    int p2 = a3 * b1 + a1 * b3;
    int p3 = a1 * b2 + a2 * b1;

    int p4 = a2 * c3 + a3 * c2;
    int p5 = a3 * c1 + a1 * c3;
    int p6 = a1 * c2 + a2 * c1;

    int p7 = b2 * c3 + b3 * c2;
    int p8 = b3 * c1 + b1 * c3;
    int p9 = b1 * c2 + b2 * c1;

    return p1 * c1 + p2 * c2 + p3 * c3 +
           p4 * b1 + p5 * b2 + p6 * b3 +
           p7 * a1 + p8 * a2 + p9 * a3;
}
```

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Precise SP 0xc000</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>k</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Imprecise SP 0xc000 - 0xc008</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>k</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>
int a[50][50], b[50];

int main(void) {
    int i, j, n = 50, w;
    for (i = 0; i <= n; i++) {
        w = 0;
        for (j = 0; j <= n; j++) {
            a[i][j] = (i + 1) + (j + 1);
            if (i == j)
                a[i][j] *= 10;
            else
                a[i][j] *= 2;
            w += a[i][j];
        }
        b[i] = w;
    }
    return 0;
}

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Number of references classified as always hit</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls</td>
<td>k</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>
Congruence Information

Partial Execution Trace $\tau$

\[
\langle s_1 \mapsto 0x60810 \rangle \\
\circ \langle s_1 \rangle \\
\circ \langle s_2 \mapsto 0xbffc0 \rangle \\
\circ \langle s_2 \rangle \\
\circ \langle s_3 \mapsto 0xbffc4 \rangle \\
\circ \langle s_3 \rangle 
\]

\[
cgr(s_1, s_3) = \hat{rel}(\text{last}(\tau, s_1), \text{last}(\tau, s_3)) \\
= \hat{rel}(0x60810, \bot) \\
= \bot
\]
Congruence Information

Partial Execution Trace $\tau'$

\[
\langle s_1 \mapsto 0x60810 \rangle \\
\circ \langle s_1 \rangle \\
\circ \langle s_2 \mapsto 0xbfffc0 \rangle \\
\circ \langle s_2 \rangle \\
\circ \langle s_3 \mapsto 0xbfffc4 \rangle \\
\circ \langle s_3 \rangle
\]

$cgr(s_1, s_3) = \widehat{rel}(last(\tau, s_1), last(\tau, s_3))$

$= \widehat{rel}(0x60810, \bot)$

$= \bot$

$cgr(s_1, s_3) = \widehat{rel}(last(\tau', s_1), last(\tau', s_3))$

$= \widehat{rel}(0x60810, 0xbfffc4)$

$= ds$

$\rightarrow$ congruence information depends on program location
Abstract Cache Domain

\[
\begin{array}{|c|}
\hline
\{s_1\} \\
\hline
\{s_2\} \\
\hline
\end{array}
\]

\[\mathcal{N} \rightarrow \{0, \ldots, k - 1, \infty\}\]
Abstract Cache Domain

\[
\{ s_1 \} + cgr_v(s_1, s_2) = sb
\]

\[
(N \rightarrow AB^{\leq}) \times (N \times N \rightarrow R)
\]
Abstract Cache Domain

Effective Age Bound — Aliasing Problem

\[
\begin{align*}
\{s_1\} & + cgr_v(s_1, s_2) = sb \\
\{s_2\} & \\
\end{align*}
\]

\[
eab^\leq : (N \to AB^\leq) \times (N \times N \to R) \to (N \to AB^\leq)
\]

\[
eab^\leq(ab, cgr_v) = \lambda s \in N. \min\{ab(t) \mid t \in N \land cgr_v(s, t) = sb\}
\]
Abstract Cache Domain

Effective Age Bound — Normalisation

\[ \{s_1\} + cgr_v(s_1, s_2) = sb \]

\[ \downarrow \]

\[ eab^\leq : (\mathcal{N} \to AB^\leq) \times (\mathcal{N} \times \mathcal{N} \to \mathcal{R}) \to (\mathcal{N} \to AB^\leq) \]

\[ eab^\leq(ab, cgr_v) = \lambda s \in \mathcal{N}. \min\{ab(t) \mid t \in \mathcal{N} \land cgr_v(s, t) = sb\} \]

\[ \downarrow \]

\[ \{s_1, s_2\} + cgr_v(s_1, s_2) = sb \]
Join

\[ \{s_1\} \subseteq \{s_2\} \subseteq \{s_3, s_4\} \cup_{rel} \subseteq \{s_2\} \subseteq \{s_5\} \subseteq \{s_1\} \]

\[ \lambda s \in \mathcal{N}. \max(ab(s), ab'(s)) \]
Classification

classify $s_1$ → [Classification]

\[ Class_{rel}^{\leq}(ab, s) := \begin{cases} H & : ab(s) < \infty \\ T & : \text{otherwise} \end{cases} \]
Classification

classify $s_3$

\[ \text{Class}_{rel}^{<}(ab, s) := \begin{cases} H : \text{ab}(s) < \infty \\ T : \text{otherwise} \end{cases} \]
Update

$c_{gr}(s_1, s_1) = sb$

$U_{rel}^{≤}(ab, s) := \lambda t. \begin{cases} 
0 & : srel = sb \\
ab(t) & : srel \in \{ds, ss db\} \\
ab(t) & : srel \supseteq ss db \land ab(s) \leq ab(t) \\
ab(t) + 1 & : srel \supseteq ss db \land ab(s) > ab(t) \land ab(t) < k - 1 \\
\infty & : srel \supseteq ss db \land ab(s) > ab(t) \land ab(t) \geq k - 1 
\end{cases}$

where $srel = c_{gr}(s, t)$. 
Update

\[ cgr_v(s_1, s_2) = sb \]

\[ U_{rel}^{\leq}(ab, s) := \lambda t. \begin{cases} 
  0 & : srel = sb \\
  ab(t) & : srel \in \{ds, ss db\} \\
  ab(t) & : srel \sqsubseteq_R ss db \land ab(s) \leq ab(t) \\
  ab(t) + 1 & : srel \sqsubseteq_R ss db \land ab(s) > ab(t) \land ab(t) < k - 1 \\
  \infty & : srel \sqsubseteq_R ss db \land ab(s) > ab(t) \land ab(t) \geq k - 1 
\end{cases} \]

where \( srel = cgr_v(s, t) \).
Update

\[
cgr_v(s_1, s_3) = ds
\]

\[
U_{rel}^\leq(ab, s) := \lambda t. \begin{cases} 
0 & : srel = sb \\
ab(t) & : srel \in \{ds, ss db\} \\
ab(t) + 1 & : srel \sqsubseteq_R ss db \land ab(s) \leq ab(t) \\
\infty & : srel \sqsubseteq_R ss db \land ab(s) > ab(t) \land ab(t) < k - 1 \\
\end{cases}
\]

where \( srel = cgr_v(s, t) \).
\begin{equation*}
cgr_v(s_1, s_4) = ss \, db
\end{equation*}

\[ U_{rel}^{\leq}(ab, s) := \lambda t. \begin{cases}
0 & : srel = sb \\
ab(t) & : srel \in \{ds, ss \, db\} \\
ab(t) & : srel \sqsubseteq_R ss \, db \land \ab(s) \leq \ab(t) \\
\ab(t) + 1 & : srel \sqsupseteq_R ss \, db \land \ab(s) > \ab(t) \land \ab(t) < k - 1 \\
\infty & : srel \sqsupseteq_R ss \, db \land \ab(s) > \ab(t) \land \ab(t) \geq k - 1
\end{cases} \]

where \( srel = cgr_v(s, t) \).
Update

\[
cgr_v(s_1, s_5) = \text{ss}
\]

\[
U_{rel}^{\leq}(ab, s) := \lambda t. \begin{cases} 
0 &: srel = \text{sb} \\
ab(t) &: srel \in \{\text{ds, ss db}\} \\
ab(t) + 1 &: srel \supseteq_R \text{ss db} \land ab(s) \leq ab(t) \\
\infty &: srel \supseteq_R \text{ss db} \land ab(s) > ab(t) \land ab(t) < k - 1 \\
\end{cases}
\]

where \(srel = cgr_v(s, t)\).