Slicing Concurrent Programs
Achievements and Open Challenges

Dennis Gifhorn
Programming paradigms group – IPD Snelting
Interference dependence

- Concurrency via threads and shared memory
- Shared-memory communication gives rise to *interference dependence*

**Definition**

Statement $t$ is interference-dependent on statement $s$, if

- $t$ uses a value which is defined by statement $s$, and
- $s$ and $t$ may happen in parallel
Interference dependence

- Concurrency via threads and shared memory
- Shared-memory communication gives rise to *interference dependence*

**Definition**
Statement $t$ is interference-dependent on statement $s$, if
- $t$ uses a value which is defined by statement $s$, and
- $s$ and $t$ may happen in parallel

**Example**

<table>
<thead>
<tr>
<th>thread_1:</th>
<th>thread_2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>$a = \text{input} + c;$</td>
<td>$c = b;$</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$b = \text{input} - 5;$</td>
<td></td>
</tr>
</tbody>
</table>
Interference dependence

- Concurrency via threads and shared memory
- Shared-memory communication gives rise to interference dependence

**Definition**
Statement \( t \) is interference-dependent on statement \( s \), if
- \( t \) uses a value which is defined by statement \( s \), and
- \( s \) and \( t \) may happen in parallel

**Example**

<table>
<thead>
<tr>
<th>thread_1:</th>
<th>thread_2:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \ a = input + c;</td>
<td>4 \ c = b;</td>
</tr>
<tr>
<td>2 \ b = input - 5;</td>
<td>5 \ c = b;</td>
</tr>
</tbody>
</table>
Interference dependence

- Concurrency via threads and shared memory
- Shared-memory communication gives rise to *interference dependence*

**Definition**

Statement $t$ is interference-dependent on statement $s$, if

- $t$ uses a value which is defined by statement $s$, and
- $s$ and $t$ *may happen in parallel*

**Example**

1. thread_1:
   2. $a = \text{input} + c$;
   3. $b = \text{input} - 5$;

4. thread_2:
   5. $c = b$;
cSDG – concurrent SDG

1. thread_1
   a = input + c
   b = input - 5

2. thread_2
   c = b

3. control dep.
4. interference dep.
5. Time-insensitive slices
Two interference dependences may exclude each other
Two interference dependences may exclude each other

⇒ Time-insensitive slices
Time-sensitive Slicing

Time-sensitive path

A context-sensitive path \( \langle n_1, \ldots, n_k \rangle \) in a cSDG is time-sensitive, if \( \forall 1 \leq i < j \leq k \):

- \( n_i \) and \( n_j \) may happen in parallel, or
- \( n_i \) reaches \( n_j \) in the control flow graph
Time-sensitive Slicing

**Time-sensitive path**

A context-sensitive path $\langle n_1, \ldots, n_k \rangle$ in a cSDG is time-sensitive, if $\forall 1 \leq i < j \leq k$:

- $n_i$ and $n_j$ may happen in parallel, or
- $n_i$ reaches $n_j$ in the control flow graph

**Prepending property (Krinke, 2003)**

Let $\pi = \langle n_1, \ldots, n_k \rangle$ be a time-sensitive path in a cSDG $G$. Let $e = n_0 \rightarrow n_1$ be an edge in $G$. Path $\langle n_0, n_1, \ldots, n_k \rangle$ is time-sensitive, if

- $e$ is thread-local, or
- $n_0$ reaches the first element $n$ in $\pi$ that may not happen in parallel to $n_0$
Exploit prepending property:
- Annotate nodes with *state tuples*
  - One entry per thread
  - Contains the first element of that thread in the path taken so far
Time-sensitive Slicing

thread_1

1. $b = \text{input} - 5$
2. $a = \text{input} + c$
3. $c = b$

thread_2

4. control dep.
5. interference dep.

$[2, X]$
Time-sensitive Slicing

```
b = input - 5
a = input + c
c = b
```

**thread_1**

**thread_2**

**control dep.**

**interference dep.**
Time-sensitive Slicing

```
thread_1
b = input - 5
a = input + c
```

```
thread_2
c = b
```

control dep.

interference dep.

```
1
2
3

1
2, X

3

4

4

5

[2, X]

[2, 5]

[1, X]

[1, X]

[2, 5]

b = input - 5

a = input + c

c = b
```
Time-sensitive Slicing

Thread 1:
1. \( b = \text{input} - 5 \)
2. \( a = \text{input} + c \)
3. \( c = b \)

Thread 2:
4. \( c = b \)

Dependency arrows:
- Control dependency: \([1,X]\) to \([2,4]\)
- Interference dependency: \([2,X]\) to \([2,5]\)

19. Januar 2011  Dennis Gifhorn – Slicing Concurrent Programs
Time-sensitive Slicing

Runtime costs

- A node can be visited repeatedly with different state tuples
- State tuple entry is node + calling context

\[ O(N^p t), \]
\[ N = \text{number of nodes}, \]
\[ p = \text{max. call depth}, \]
\[ N^p = \text{upper bound for nodes + calling contexts}, \]
\[ t = \text{number of threads} \]
Time-sensitive Slicing

- Practical for programs with approx. 10 kLoc
- Usable for mature languages
  - Interprocedural programs, including recursion
  - Dynamic thread creation inside loops or recursion
- JOANA-Project for full Java bytecode
  (Giffhorn and Hammer, *Precise Analysis of Java Programs using JOANA (Tool Demonstration)*, in *8th IEEE SCAM*, 2008.)
## Empirical Results

<table>
<thead>
<tr>
<th>name</th>
<th>nodes</th>
<th>edges</th>
<th>methods</th>
<th>threads</th>
<th>interf. dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiningPhils</td>
<td>5143</td>
<td>125470</td>
<td>116</td>
<td>2</td>
<td>471</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>6218</td>
<td>51035</td>
<td>161</td>
<td>2</td>
<td>948</td>
</tr>
<tr>
<td>Barcode</td>
<td>12393</td>
<td>64820</td>
<td>271</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>HyperM</td>
<td>17835</td>
<td>97827</td>
<td>277</td>
<td>6</td>
<td>8139</td>
</tr>
<tr>
<td>Podcast</td>
<td>23676</td>
<td>159478</td>
<td>407</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

- 1,000 slices per program:

<table>
<thead>
<tr>
<th>name</th>
<th>size per slice (nodes)</th>
<th>runtime per slice (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens.</td>
</tr>
<tr>
<td>DiningPhils</td>
<td>2,867</td>
<td>2,499 (87%)</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>3,409</td>
<td>3,328 (98%)</td>
</tr>
<tr>
<td>Barcode</td>
<td>3,410</td>
<td>2,974 (87%)</td>
</tr>
<tr>
<td>HyperM</td>
<td>9,222</td>
<td>7,441 (81%)</td>
</tr>
<tr>
<td>Podcast</td>
<td>12,335</td>
<td>8,730 (71%)</td>
</tr>
</tbody>
</table>
Important Topics for Future Research

- Runtime costs
- Prepending property and MHP information
- Interference dependence and reaching definitions
## Runtime Costs

What property of a cSDG has the major influence on the runtime costs?

<table>
<thead>
<tr>
<th>name</th>
<th>nodes</th>
<th>edges</th>
<th>methods</th>
<th>threads</th>
<th>interf. dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiningPhils</td>
<td>5143</td>
<td>125470</td>
<td>116</td>
<td>2</td>
<td>471</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>6218</td>
<td>51035</td>
<td>161</td>
<td>2</td>
<td>948</td>
</tr>
<tr>
<td>Barcode</td>
<td>12393</td>
<td>64820</td>
<td>271</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>HyperM</td>
<td>17835</td>
<td>97827</td>
<td>277</td>
<td>6</td>
<td>8139</td>
</tr>
<tr>
<td>Podcast</td>
<td>23676</td>
<td>159478</td>
<td>407</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>size per slice (nodes)</th>
<th>runtime per slice (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens.</td>
</tr>
<tr>
<td>DiningPhils</td>
<td>2,867</td>
<td>2,499 (87%)</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>3,409</td>
<td>3,328 (98%)</td>
</tr>
<tr>
<td>Barcode</td>
<td>3,410</td>
<td>2,974 (87%)</td>
</tr>
<tr>
<td>HyperM</td>
<td>9,222</td>
<td>7,441 (81%)</td>
</tr>
<tr>
<td>Podcast</td>
<td>12,335</td>
<td>8,730 (71%)</td>
</tr>
</tbody>
</table>
What property of a cSDG has the major influence on the runtime costs?

<table>
<thead>
<tr>
<th>name</th>
<th>nodes</th>
<th>edges</th>
<th>methods</th>
<th>threads</th>
<th>interf. dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiningPhils</td>
<td>5143</td>
<td>125470</td>
<td>116</td>
<td>2</td>
<td>471</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>6218</td>
<td>51035</td>
<td>161</td>
<td>2</td>
<td>948</td>
</tr>
<tr>
<td>Barcode</td>
<td>12393</td>
<td>64820</td>
<td>271</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>HyperM</td>
<td>17835</td>
<td>97827</td>
<td>277</td>
<td>6</td>
<td>8139</td>
</tr>
<tr>
<td>Podcast</td>
<td>23676</td>
<td>159478</td>
<td>407</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>size per slice (nodes)</th>
<th>runtime per slice (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens. (cont.-sens.)</td>
</tr>
<tr>
<td>DiningPhils</td>
<td>2,867</td>
<td>2,499 (87%)</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>3,409</td>
<td>3,328 (98%)</td>
</tr>
<tr>
<td>Barcode</td>
<td>3,410</td>
<td>2,974 (87%)</td>
</tr>
<tr>
<td>HyperM</td>
<td>9,222</td>
<td>7,441 (81%)</td>
</tr>
<tr>
<td>Podcast</td>
<td>12,335</td>
<td>8,730 (71%)</td>
</tr>
</tbody>
</table>
## Runtime Costs

What property of a cSDG has the major influence on the runtime costs?

<table>
<thead>
<tr>
<th>name</th>
<th>nodes</th>
<th>edges</th>
<th>methods</th>
<th>threads</th>
<th>interf. dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiningPhils</td>
<td>5143</td>
<td>125470</td>
<td>116</td>
<td>2</td>
<td>471</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>6218</td>
<td>51035</td>
<td>161</td>
<td>2</td>
<td>948</td>
</tr>
<tr>
<td>Barcode</td>
<td>12393</td>
<td>64820</td>
<td>271</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>HyperM</td>
<td>17835</td>
<td>97827</td>
<td>277</td>
<td>6</td>
<td>8139</td>
</tr>
<tr>
<td>Podcast</td>
<td>23676</td>
<td>159478</td>
<td>407</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>size per slice (nodes)</th>
<th>runtime per slice (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens.</td>
</tr>
<tr>
<td>DiningPhils</td>
<td>2,867</td>
<td>2,499 (87%)</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>3,409</td>
<td>3,328 (98%)</td>
</tr>
<tr>
<td>Barcode</td>
<td>3,410</td>
<td>2,974 (87%)</td>
</tr>
<tr>
<td>HyperM</td>
<td>9,222</td>
<td>7,441 (81%)</td>
</tr>
<tr>
<td>Podcast</td>
<td>12,335</td>
<td>8,730 (71%)</td>
</tr>
</tbody>
</table>
What property of a cSDG has the major influence on the runtime costs?

<table>
<thead>
<tr>
<th>name</th>
<th>nodes</th>
<th>edges</th>
<th>methods</th>
<th>threads</th>
<th>interf. dep.</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiningPhils</td>
<td>5143</td>
<td>125470</td>
<td>116</td>
<td>2</td>
<td>471</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>6218</td>
<td>51035</td>
<td>161</td>
<td>2</td>
<td>948</td>
</tr>
<tr>
<td>Barcode</td>
<td>12393</td>
<td>64820</td>
<td>271</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>HyperM</td>
<td>17835</td>
<td>97827</td>
<td>277</td>
<td>6</td>
<td>8139</td>
</tr>
<tr>
<td>Podcast</td>
<td>23676</td>
<td>159478</td>
<td>407</td>
<td>3</td>
<td>128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>size per slice (nodes)</th>
<th>runtime per slice (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens.</td>
</tr>
<tr>
<td></td>
<td>cont.-sens.</td>
<td>time-sens.</td>
</tr>
<tr>
<td>DiningPhils</td>
<td>2,867</td>
<td>2,499 (87%)</td>
</tr>
<tr>
<td>LaplaceGrid</td>
<td>3,409</td>
<td>3,328 (98%)</td>
</tr>
<tr>
<td>Barcode</td>
<td>3,410</td>
<td>2,974 (87%)</td>
</tr>
<tr>
<td>HyperM</td>
<td>9,222</td>
<td>7,441 (81%)</td>
</tr>
<tr>
<td>Podcast</td>
<td>12,335</td>
<td>8,730 (71%)</td>
</tr>
</tbody>
</table>
Prepending Property and MHP Information

- May-happen-in-parallel analysis:
  Which parts of the threads may happen in parallel?
- Recall the definition of time-sensitive paths:

Time-sensitive path

A context-sensitive path \( \langle n_1, \ldots, n_k \rangle \) in a cSDG is time-sensitive, if \( \forall 1 \leq i < j \leq k : \)

- \( n_i \) and \( n_j \) may happen in parallel, or
- \( n_i \) reaches \( n_j \) in the control flow graph

- Time-sensitivity depends on available MHP information
- In our evaluation it increased precision by 10%
Prepending Property and MHP Information

- May-happen-in-parallel analysis:
  Which parts of the threads may happen in parallel?
- Recall the definition of time-sensitive paths:

**Time-sensitive path**

A context-sensitive path \( \langle n_1, \ldots, n_k \rangle \) in a cSDG is time-sensitive, if \( \forall 1 \leq i < j \leq k: \)

- \( n_i \) and \( n_j \) may happen in parallel, or
- \( n_i \) reaches \( n_j \) in the control flow graph

- Time-sensitivity depends on available MHP information
- In our evaluation it increased precision by 10%
- Cannot be completely exploited by the time-sensitive slicer
Prepending property (Krinke, 2003)

Let $\pi = \langle n_1, \ldots, n_k \rangle$ be a time-sensitive path in a cSDG $G$. Let $e = n_0 \rightarrow n_1$ be an edge in $G$. Path $\langle n_0, n_1, \ldots, n_k \rangle$ is time-sensitive, if

- $e$ is thread-local, or
- $n_0$ reaches the first element $n$ in $\pi$ that may not happen in parallel to $n_0$

Does only hold in case all threads may happen in parallel

- $n_0$ and $n$ may not happen in parallel to the same elements $M$ in $\pi$
- $n$ reaches every $m \in M$
- ‘Reaches’ is transitive, thus it suffices that $n_0$ reaches $n$
Prepending Property and MHP Information

Example

1 thread_1:
2   if (...)
3     x = 1;
4   else
5     fork thread_2;
6   y = x;

7 thread_2:
8   print y;
Prepending Property and MHP Information

Thread 1

1. `thread_1`
2. `if (...)`
3. `x = 1`
4. `fork thread_2`
5. `y = x`

Thread 2

7. `thread_2`
8. `print y`

Control dependency
Data dependency
Interference dependency
Prepending Property and MHP Information

1. thread_1
2. if (...)
3. x = 1
4. fork thread_2
5. y = x
6. print y

control dep.  
data dep.  
interference dep.
Prepending Property and MHP Information

Develop a slicer that is time-sensitive wrt. general MHP information

- Adjust the prepending property?
  ⇒ Which information has to be stored in the state tuples?
- Find a completely different approach?
  - Post-process time-insensitive slices
  - Deactivate invalid parts of the cSDG (use Petri nets?)
  - …
Interference Dependence and Reaching Definitions

**Definition**

Statement $t$ is interference-dependent on statement $s$, if

- $t$ uses a variable $v$ which is defined by statement $s$, and
- $s$ and $t$ may happen in parallel

- Data dependence requires reaching definitions
- Interference dep. ignores reaching definitions
- Undecidable for concurrent interprocedural programs
- Decidable in special cases
Threaded Interaction Reachability Graph

Idea of Qi et al.
(Slicing Concurrent Programs Based on Program Reachability Graphs, in IEEE 10th ICQS, 2010.)

- Unroll the possible interleavings in a
  Threaded Interaction Reachability Graph (TIRG)
- Each node corresponds to a possible interleaving situation
Threaded Interaction Reachability Graph

Thread 1:
1. \text{thread}_1
2. \text{a} = \text{input} + c
3. \text{b} = \text{input} - 5

Thread 2:
4. \text{c} = \text{b}
5. \[1,4\] [2,4] [2,5] [3,5] [3,4] [1,5]
Threaded Interaction Reachability Graph

- Create a SDG from the TIRG
  - Data flow analysis for sequential programs
- Free of interference dependence
  - No time-insensitivity
  - Finds situations, where a def in one thread does not reach a use in another thread
Threaded Interaction Reachability Graph

- Create a SDG from the TIRG
  ⇒ Data flow analysis for sequential programs
- Free of interference dependence
  - No time-insensitivity
  - Finds situations, where a def in one thread does not reach a use in another thread
- Very high precision
- No empirical data ⇒ assumption: TIRGs are huge (need to inline procedures)
- No recursion
- No thread creation inside loops
  ⇒ Extend its power
  ⇒ Develop compression techniques
Literature


JIT Compiler

- Just-in-time (JIT) compiler may reorganize code during program execution.
- Control flow during execution may not correspond to the control flow during the slice.
- Time-insensitive paths may turn time-sensitive.
Java’s JIT compiler may switch statements 2 and 3
Slice is time-sensitive!
Identification of which execution orders are guaranteed