Generating Fast Sequential Code from Concurrent Programs

Jia Zeng, Cristian Soviani, Stephen A. Edwards

Department of Computer Science, Columbia University

www.cs.columbia.edu/~jia,soviani,sedwards

jia,cristian,sedwards@cs.columbia.edu
Motivation

- Dynamic vs. Static Scheduling

<table>
<thead>
<tr>
<th></th>
<th>Dynamic</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flexibility</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Overhead</td>
<td>run time</td>
<td>compile time</td>
</tr>
<tr>
<td>Behavior</td>
<td>unpredictable</td>
<td>predictable</td>
</tr>
</tbody>
</table>

- Embedded system requirement: low run-time cost, quick response, predictable
  ⇒ Static scheduling
Motivation Cont.

- Hardware Implementation and Simulation

⇒ Simulation is required; Speed is never fast enough
Overview

- Ferrante, Mace & Simons, 1984: Using PDG
- Cytron et al., 1991: Generating PDG
- Simons & Ferrante, 1993: External Edge
- Our approach: Natural Concurrent Programs
if (a == 1)
    b = 0;
    d = 1;
if (b == 0)
    c = 1;

- fork (region)
- predicate
- statement
- control arc
- data arc
(partial order)
From PDG to SCFG: Trivial?

Make it sequential directly
Execute one by one
From PDG to SCFG: Non-trivial

No way to be sequential unless to add guard variable or copy
An Example: Reconstructing PDG 0

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 1

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 2

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 3

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>$A = 1$</th>
<th>Test A</th>
<th>$C = 1$</th>
<th>$C = C + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test B</td>
<td>$A = 1$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 4

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test B</td>
<td>A = 1</td>
<td>Test A</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 5

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test B</td>
<td>A = 1</td>
<td>Test A</td>
<td>C = 1</td>
<td>-</td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 6

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test V</td>
<td>A = 1</td>
<td>Test A</td>
<td>C = 1</td>
<td></td>
</tr>
</tbody>
</table>
An Example: Reconstructing PDG 6

<table>
<thead>
<tr>
<th>orig</th>
<th>Fork</th>
<th>Test B</th>
<th>A = 1</th>
<th>Test A</th>
<th>C = 1</th>
<th>C = C+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy</td>
<td>Fork</td>
<td>Test V</td>
<td>A = 1</td>
<td>Test A</td>
<td>C = 1</td>
<td>C = C+1</td>
</tr>
</tbody>
</table>
An Example: Whole process

if (B){
    V = 1;
    A = 1;
}
else
    V = 0;
if (A)
    C = 1;
if (V)
    {}
else
    C = C + 1;
More complex situations:

converge control flow
More complex situations:

more forks & more data flow
Experimental Results

Generated C code for examples running on 2.5 GHz Pentium 4, Linux
Conclusion

- A technique to generate fast sequential code from PDG for concurrent programs;
- Useful for embedded software and simulation.
- Speed improvement
Acknowledgement

Prof. Edwards (Advisor)
Cristian Soviani (Partner)

Thank you all!
Questions & Answers
Illegal PDG

- Predicate least common ancestor rule
- No post-dominance rule
Non-concise PDG

- Conflict between control/control fbws
- Conflict between control/data fbws
Algorithm Complexity