A Software Pipelining Framework for Simple Processor Cores

• Juergen Ributzka
• David Stephenson
• Timothy Kong
• Dee Lee
• Fred Chow
• Guang R. Gao
Overview

• SiCortex Multiprocessor
• Software Pipelining Framework
• Results
• Future Work
SiCortex Multiprocessor

- RISC
- 6 cores (MIPS 5KF)
- 500 – 700 MHz
- In-Order Execution
- Limited-Dual Issue
- Low Power (12 – 16 Watt)
Software Pipelining Framework

Front-End

FORTRAN → C/C++ → Java → Loop Nest Optimizer

Middle-End

Global Optimizer

Back-End

Code Generator

DDG → MII → MS → MVE → RA → CG
Cyclic Data Dependence Graph
No register anti- and output-dependencies
Only single BBs
Minimum Initiation Interval

• Limited by:
  – Resource Requirements
  – Recurrence Requirements
Modulo Scheduling (MS)

- Huff Modulo Scheduling
  - Lifetime sensitive
  - Uses backtracking
- Hyper Node Reduction Scheduling
  - Lifetime sensitive
  - No backtracking
Modulo Variable Expansion (MVE)

- Separate TN set for every iteration
- # of unrollings depends on longest lifetime

Iteration

C y c l e

TN10 op1 TNXX
... TN11 op1 TNXX
TN20 op2 TN10 TNYY
... ... TN21 op2 TN11 TNYY
TN20 op2 TN10 TNYY
...
Register Allocation (RA)

- Only kernels are fully register allocated
- No regions
  - SWP kernels are not a black box for GRA and LRA anymore
- Currently no register spilling support
• Prologues and epilogues are partially register allocated
• Need several epilogues due to different register sets
Benchmark Results

NAS Parallel Benchmark

<table>
<thead>
<tr>
<th>Speedup</th>
<th>BT</th>
<th>CG</th>
<th>EP</th>
<th>FT</th>
<th>IS</th>
<th>LU</th>
<th>MG</th>
<th>SP</th>
<th>UA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABI n32</td>
<td>1.00</td>
<td>1.15</td>
<td>1.05</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ABI 64</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
<td>0.95</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

- Spilling support needed to enable more loops for SWP
- Screen out loops with small trip count during runtime to reduce SWP overhead
- Hit-under-miss support to overcome current hardware limitations