Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture

Anoop Bhagyanath, Tripti Jain and Klaus Schneider

Embedded Systems Group
University of Kaiserslautern

MBMV, 2016
Outline

1 Motivation

2 SCAD

3 Queue-based Code Generation

4 Conclusion
Motivation

Instruction Level Parallelism (ILP)

Superscalar and VLIW Machines

- ILP restricted due to limited number of registers
  - instruction format encoding
  - register file wiring

- Compiler spills variables to main memory

- Number of instructions packed into a VLIW word
Instruction Level Parallelism (ILP)

Expression tree  VLIW code (2 regs)  Superscalar code (2 regs)

\[
\begin{align*}
&\times \\
&/ \\
&- x3 \\
&st \\
x4 \\
&ld / \\
&\times \\
&\end{align*}
\]

3 steps  6 steps  5 steps

Anoop Bhagyanath, Tripti Jain and Klaus Schneider
University of Kaiserslautern
Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture
Exposed Datapath Architectures

- Compiler also controls the data transport
  - bypass registers

- Examples include Raw, TRIPS, Wavescalar, Flexcore, TTAs etc

- Code generators still rely on efficient register mappings
  - register bypassing utilized at later stages
Our Contribution

Synchronous Control Asynchronous Dataflow (SCAD)

- Exposed datapath architecture

Queue-based code generation

- Eliminate register usage improving effective ILP
SCAD Organization

- Grid of processing elements
- FIFO buffers (queues) at inputs and outputs of processing units
- Move instructions \( out \rightarrow in \)
- Move instruction bus (MIB)
- Data transport network (DTN)
- Application-specific
  - Any arbitrary functionality
  - Interconnect choice
SCAD Organization

- each slot in queue \((adr, val)\)
- Move instruction \(out \rightarrow in\)
- Synchronous control via move instruction bus (MIB)
- Processing unit fires if enough data available
- Asynchronous dataflow via Data transport network (DTN)
Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture

SCAD Functionality

Executed x1, x2, x3, x4
SCAD Functionality

Decoded y1

I1

O

I2

0

x4

x3

I2

x2

I1

x1

x1 x2 x3 x4

y1 y2 z1
SCAD Functionality

Decoded y2

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \]
\[ y_1 \quad y_2 \]
\[ z_1 \]

\[ \begin{array}{c}
 0 \\
 0 \\
\end{array} \]

\[ \begin{array}{c}
 0 \\
 0 \\
\end{array} \]

\[ \begin{array}{c}
 l_2 \ x_4 \\
 l_1 \ x_3 \\
 l_2 \ x_2 \\
 l_1 \ x_1 \\
\end{array} \]
Data transported to execute y1

Anoop Bhagyanath, Tripti Jain and Klaus Schneider
University of Kaiserslautern
Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture
Data transported to execute $y_2$
SCAD Functionality

Executed \( y_1, y_2 \)
SCAD Functionality

Anoop Bhagyanath, Tripti Jain and Klaus Schneider
University of Kaiserslautern
Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture
Breadth-First Traversal

Node ordering:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>x2</td>
<td>x3</td>
<td>x4</td>
<td>y1</td>
</tr>
<tr>
<td>y2</td>
<td></td>
<td></td>
<td></td>
<td>y2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>z1</td>
</tr>
</tbody>
</table>

No registers used
Queue depth not limited
Queues scale better
Depth-First Traversal

Current Compilers

- Order nodes of by depth-first traversal
- Minimize register usage
- Optimal code for expression trees
  - Sethi-Ullmann algorithm
  - polynomial time
- Optimal code for directed-acyclic graphs (DAG)
  - proved to be NP-Complete
**Depth-First Traversal in SCAD**

Executed \(x_1, x_2\)

Anoop Bhagyanath, Tripti Jain and Klaus Schneider
University of Kaiserslautern

Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture
Depth-First Traversal in SCAD

Executed y1
Depth-First Traversal in SCAD

Executed x3, x4

Wrong Order!
Queue Machine

- Similar to the more familiar Stack Machine

- One queue to hold
  - operands for execution
  - result of execution

- Turing complete
Anoop Bhagyanath, Tripti Jain and Klaus Schneider

University of Kaiserslautern

Towards Code Generation for the Synchronous Control Asynchronous Dataflow (SCAD) Architecture
Queue Machine

Code Generation for Queue Machine

given DAG | levelized DAG | planar DAG | level-planar DAG | queue program
--|---|---|---|---

```
load x1,1
load x2,2
add 2
dup 2
dup 1
swap
dup 1
mul 1
add 1
store y1
store y2
```

Anoop Bhagyanath, Tripti Jain and Klaus Schneider

University of Kaiserslautern
## Queue code to SCAD code

<table>
<thead>
<tr>
<th>Queue Instruction</th>
<th>Corresponding SCAD Move Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(load x1,1)</td>
<td>[x1-&gt;inp1; load-&gt;opc; 1-&gt;cps]</td>
</tr>
<tr>
<td>(add 1)</td>
<td>[out-&gt;inp1; out-&gt;inp2; add-&gt;opc; 1-&gt;cps]</td>
</tr>
<tr>
<td>(dup 2)</td>
<td>[out-&gt;inp1; dup-&gt;opc; 2-&gt;cps]</td>
</tr>
<tr>
<td>(swap)</td>
<td>[out-&gt;inp1; out-&gt;inp2; swap-&gt;opc]</td>
</tr>
<tr>
<td>(store y1)</td>
<td>[y1-&gt;inp1; out-&gt;inp2; store-&gt;opc]</td>
</tr>
</tbody>
</table>
## Example DAG on Queue machine

<table>
<thead>
<tr>
<th>given DAG</th>
<th>level-planar DAG</th>
<th>queue program</th>
</tr>
</thead>
</table>
| ![Graph](image) | ![Graph](image) | ```
load x1,2
load x2,2
dup 1
swap
dup 1
sub 1
div 1
store y1
store y2
```

2 Dup,1 Swap |
Example DAG on SCAD machine

Executed $x_1, x_2$
Example DAG on SCAD machine

Transported $x_1, x_1$
Example DAG on SCAD machine

Transported $x_2, x_2$
Example DAG on SCAD machine

- Executed: $x_1 - y_1$ and $x_2 / y_2$
- No Swap and Dup used!
- Derived SCAD code is not optimal
Conclusion

Summary

**SCAD architecture**
- exposed datapath

**Queue-based code generation**
- eliminates register usage
- improves effective ILP
- but non-optimal
## Conclusion

### Future Work

- Optimal SCAD code generation for DAGs
- Buffer size analysis
- Performance, cost and timing-predictability comparison
Thank You!

Questions?