Reducing Control Overhead in Dataflow Architectures

Andrew Petersen
Martha Mercaldi, Steven Swanson, Andrew Putnam, Andrew Schwerin
Mark Oskin and Susan Eggers

University of Washington
Talk Outline

WaveScalar Background
Control Overhead in Dataflow Code
Case Study: Removing Control from a Loop
Overall Results
WaveScalar Processor

Scalability
- Short wires

Complexity
- Simple, replicated unit

Power
- Turn off unneeded tiles
Execution in WaveScalar

\[
\begin{align*}
  a & = e1; \\
  b & = e2; \\
  \text{while (p1) } & \{ \\
    a & = a + b; \\
  \} \\
  c & = a * b;
\end{align*}
\]
Execution in WaveScalar
Execution in WaveScalar
Transforming C to Dataflow

C Code

```c
a = e1;
b = e2;
while (p1) {
    a = a + b;
}
c = a * b;
```

Control Flow Graph

```
a = e1
b = e2
```
```
while (p1)
a = a + b
```
```
c = a * b
```
Transforming C to Dataflow

Control Flow Graph

```
while (p1)
  a = a + b
```

c = a * b

Dataflow Graph

```
a = e1
b = e2
```

c = a * b
Transforming C to Dataflow

Control Flow Graph

Dataflow Graph
Transforming C to Dataflow

Control Flow Graph

Dataflow Graph
Talk Outline

WaveScalar Background

**Control Overhead in Dataflow Code**

Case Study: Removing Control from a Loop

Overall Results
Characterizing the Overhead

- Computation: 27%
- Other Overhead: 4%
- Selects: 35%

Wave-Advances: 34%

Flowchart:
- \(a = e_1\)
- \(b = e_2\)
- \(p_1 \rightarrow \text{select} \quad T \quad F\)
- \(\text{wave advance}\)
- \(a = a + b\)
- \(p_1 \rightarrow \text{select} \quad T \quad F\)
- \(\text{wave advance}\)
- \(c = a \times b\)
Solution: Parallel Instructions

Overhead = 66%

Overhead = 20%

11/2/06  PACT 2006
Issues with Combined Instructions

- In what situations can we combine instructions?
- Can the hardware perform the combined operations?
Talk Outline

WaveScalar Background
Control Overhead in Dataflow Code
**Case Study: Removing Control from a Loop**
Folding Selects and Wave-Advances
Hoisting Unused Values
Overall Results
Folding Selects
Folding Selects
Folding Selects

- Selects fold “up”
Folding Selects

- Selects fold “up”
  1. The Select may only have a single producer instruction.
  2. The producer should not be another Select.
Folding Selects

• Selects fold “up”

1. The Select may only have a single producer instruction.

2. The producer should not be another Select.
Folding Selects

- Selects fold “up”
  1. The Select may only have a single producer instruction.
  2. The producer should not be another Select.
Folding Selects

- Selects fold "up"
  1. The Select may only have a single producer instruction.
  2. The producer should not be another Select.

This Select may not be folded.
Changing the PE

- Can the hardware perform the folded operation?
- Changes are minor: An additional adder and table entry are required.
- Die area increases 2%  
- Cycle time does not increase
Hoisting Unused Values
Hoisting Unused Values

\[
\begin{align*}
&\text{a = e1} \\
&\text{b = e2} \\
&\text{x = e3}
\end{align*}
\]

\[
\begin{align*}
&\text{a = a + b} \\
&\text{c = a * b} \\
&\text{y = a * x}
\end{align*}
\]
Hoisting Unused Values

- The trick is tagging matching
- The values ‘a’ and ‘x’ must match to compute ‘y’
Solution: Patching the Tag

- Create a value that contains the tag needed
- Patch values sent around the loop with this rendezvous value
Hoisting Unused Values

\[
x = e3 \\
T \quad F
\]

\[
a = a + b \\
wave advance \\
T \quad F
\]

\[
y = a \times x \\
wave advance
\]
Hoisting Unused Values
Hoisting Unused Values

\[ x = e3 \]

\[ a = a + b \]

\[ y = a \times x \]
Is Hoisting Profitable?

• Not always
  – And not in the example provided
• Creating a rendezvous value requires that it be sent through the loop
• Patching values requires an additional 3 operations

• Hoisting is only profitable when more than one value is to be hoisted from the loop
  – The rendezvous value code can be shared
Talk Outline

WaveScalar Background
Control Overhead in Dataflow Code
Case Study: Removing Control from a Loop
Overall Results
Impact on Control Overhead

![Bar chart showing overhead for different benchmarks.](chart)

- ammp: WaveAdvance Folding: 40%, Select Folding: 30%, Farhoisting: 10%, Remaining Overhead: 20%, Untargeted Overhead: 5%
- art: WaveAdvance Folding: 35%, Select Folding: 25%, Farhoisting: 15%, Remaining Overhead: 15%, Untargeted Overhead: 8%
- equake: WaveAdvance Folding: 45%, Select Folding: 20%, Farhoisting: 10%, Remaining Overhead: 20%, Untargeted Overhead: 5%
- gzip: WaveAdvance Folding: 30%, Select Folding: 30%, Farhoisting: 10%, Remaining Overhead: 20%, Untargeted Overhead: 5%
- twolf: WaveAdvance Folding: 40%, Select Folding: 25%, Farhoisting: 10%, Remaining Overhead: 20%, Untargeted Overhead: 5%
- average: WaveAdvance Folding: 35%, Select Folding: 25%, Farhoisting: 10%, Remaining Overhead: 20%, Untargeted Overhead: 5%
Impact on Performance

Normalized Performance

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>ammp</th>
<th>art</th>
<th>equake</th>
<th>gzip</th>
<th>twolf</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Far Hoisting</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select Folding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WaveAdvance Folding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td>1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.3</td>
<td>1.3</td>
<td>1.3</td>
</tr>
</tbody>
</table>

1/2/06 PACT 2006 25
Conclusion

- Historically, dataflow architectures have had a high ratio of overhead instructions to useful work.

- Providing combined computation-and-control instructions can decrease control overhead by 80% and increase performance by 25%.

- The combined instructions require minimal additional hardware (2% additional die area).
For more information:

http://wavescalar.cs.washington.edu