Investigating Hardware Micro-Instruction Folding in a Java Embedded Processor

Flavius Gruian\textsuperscript{1}  Mark Westmijze\textsuperscript{2}

\textsuperscript{1}Lund University, Sweden
flavius.gruian@cs.lth.se

\textsuperscript{2}University of Twente, The Netherlands
m.westmijze@student.utwente.nl

Java Technologies for Real-time and Embedded Systems, 2010
Outline

1. Introduction
2. Folding BlueJEP
3. Implementation and Experiments
4. Discussion
5. Conclusion
Goal

What are we trying to do?

Implement **bytecode folding** on an **existing Java embedded processor** and **evaluate** the results with respect to:

- theoretical estimates
- absolute speed-up
- performance w.r.t. device area
What are we trying to do?

Implement **bytecode folding** on an **existing Java embedded processor** and **evaluate** the results with respect to:

- theoretical estimates
- absolute speed-up
- performance w.r.t. device area

**Finally...**

**Is it worth it?**
Original Processor Architecture

BlueJEP

BlueSpec System Verilog Java Embedded Processor, a redesign of JOP [M. Schöberl]

- micro-programmed, stack machine core
- predictable rather than high-performance (RT systems)
- JOP micro-instruction set (for ease of programming)
Original Processor Architecture

BlueJEP

**BlueSpec System Verilog Java Embedded Processor**, a redesign of JOP [M. Schöberl]

- micro-programmed, stack machine core
- predictable rather than high-performance (RT systems)
- JOP micro-instruction set (for ease of programming)
- specified in BSV [see JTRES 2007]
BlueJEP Architecture

Six Stages Micro-Programmed Pipeline

Stage 1: BC2 microA
Fetch Bytecode

Stage 2: micro-ROM
Fetch micro-I

Stage 3: jump table
Decode & Fetch Register

Stage 4: Fetch Stack

Stage 5: Execute
forward bypass
rollback

Stage 6: Write-back

PC OPD SP VP
CacheCtl MD MwA MrA

Stack const

bus interface (OPB)

Cache

load cache

BC-Cache

jpc

bcfifo

decfifo

fsfifo

exfifo

wbfifo

OPD

const

CacheCtl

rollback

MMU access registers
Bytecode Folding Theory

Stack machine (JVM) code can be shorter on multi-address machines that emulate them.

<table>
<thead>
<tr>
<th>Stack Code</th>
<th>3-address Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>iload a</td>
<td>add a, b, c</td>
</tr>
<tr>
<td>iload b</td>
<td></td>
</tr>
<tr>
<td>iadd</td>
<td></td>
</tr>
<tr>
<td>istore c</td>
<td></td>
</tr>
</tbody>
</table>

The folding pattern length depends on the available resources (ALUs, memory ports).

Bytecodes are grouped in classes by resource access, e.g.:
- P producer: pushes a value in the stack
- C consumer: pops a value in the stack
- O operation: uses top two and pushes back a result
- S special: not foldable (breaks a pattern)

Approximations:
- Stack code ≈ 7 bytes
- 3-address code ≈ 4 bytes
Bytecode Folding Theory

- stack machine (JVM) code can be shorter on multi-address machines that emulate them

<table>
<thead>
<tr>
<th>stack code</th>
<th>3-address code</th>
</tr>
</thead>
<tbody>
<tr>
<td>≈7 bytes</td>
<td>≈4 bytes</td>
</tr>
</tbody>
</table>

- folding pattern length depends on the available resources (ALUs, memory ports)
Bytecode Folding Theory

- stack machine (JVM) code can be shorter on multi-address machines that emulate them

<table>
<thead>
<tr>
<th>stack code</th>
<th>3-address code</th>
</tr>
</thead>
<tbody>
<tr>
<td>≈7 bytes</td>
<td>≈4 bytes</td>
</tr>
<tr>
<td>iload a</td>
<td>add a, b, c</td>
</tr>
<tr>
<td>iload b</td>
<td></td>
</tr>
<tr>
<td>iadd</td>
<td></td>
</tr>
<tr>
<td>istore c</td>
<td></td>
</tr>
</tbody>
</table>

- folding pattern length depends on the available resources (ALUs, memory ports)
- bytecodes are grouped in classes by resource access, e.g.:
  - P producer: pushes a value in the stack
  - C consumer: pops a value in the stack
  - O operation: uses top two and pushes back a result
  - S special: not foldable (breaks a pattern)
Adopted Folding Scheme

- fixed folding pattern approach [picoJava-II]
- micro-instruction level (rather than bytecode level)
- maximum length of four micro-instructions (at most four single instruction bytecodes)

<table>
<thead>
<tr>
<th>Folding Pattern</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>pposc</td>
<td>4</td>
</tr>
<tr>
<td>posc</td>
<td>3</td>
</tr>
<tr>
<td>ppc</td>
<td>3</td>
</tr>
<tr>
<td>pc</td>
<td>2</td>
</tr>
<tr>
<td>oc</td>
<td>2</td>
</tr>
<tr>
<td>po</td>
<td>2</td>
</tr>
</tbody>
</table>
Pre-design Estimates

How much is the number of executed clock cycles reduced?
How much is the number of executed clock cycles reduced?

Processed cycle accurate simulation traces say:

- \( \approx 30\% \) fewer cycles for 0-delay memory
- \( \approx 25\% \) fewer cycles for realistic memory
Increase *fetch* parallelism to allow folding:

- **wider fetch-bytecode stage**: up to four bytecodes must be available simultaneously.
- **multiple bytecode FIFOs**: to feed the next stage with sequences of bytecodes.
- **wider fetch-instruction stage**: up to four different micro-addresses must be read simultaneously.
- **multiple micro-instruction FIFOs**: to provide patterns to the decode stage.
- **folding schemes in the decode stage**: to identify and handle foldable patterns.
Configurability

Highly configurable architecture:
1. bytecode bandwidth (1,2,4)
2. micro-instruction bandwidth (1,2,4)
3. foldable patterns
Configurability

Highly configurable architecture:

1. bytecode bandwidth (1,2,4)
2. micro-instruction bandwidth (1,2,4)
3. foldable patterns

Figure: Handling 2 bytecodes, 4 micro-instructions simultaneously.
Setup and Tools

**Synthesis** → device area, maximum clock frequency

- FPGA, Xilinx Virtex-5 (XC5VLX30-3)
- BSV compiler 2006.11, **BSV** → **Verilog**
- Xilinx EDK 9.1i, **Verilog** + **IPs** → **System**
- Xilinx ISE 9.1i, **System** → **FPGA**
- Chipscope, to calibrate simulation
Setup and Tools

**Synthesis** → device area, maximum clock frequency
- FPGA, Xilinx Virtex-5 (XC5VLX30-3)
- BSV compiler 2006.11, $BSV \rightarrow Verilog$
- Xilinx EDK 9.1i, $Verilog \rightarrow IPs \rightarrow System$
- Xilinx ISE 9.1i, $System \rightarrow FPGA$
- Chipscope, to calibrate simulation

**Simulation** → executed clock cycles
- Desktop, Linux
- BSV compiler 2006.11, $BSV \rightarrow Executable$
- custom tools for parsing the output from instrumented code
Results

Original vs. Folding Configurations (2,2; 2,4)
Original vs. Folding Configurations (4,4)
Introducing folding and more patterns:

+ reduce the executed clock cycles (as in theory), but
- ... greatly reduce the maximum clock frequency
- ... and also greatly increase the required device area
Introducing folding and more patterns:

- reduce the executed clock cycles (as in theory), but
- ... greatly reduce the maximum clock frequency
- ... and also greatly increase the required device area

Performance/area unit gets as low as 1/4 for some designs with maximal folding!

Introducing more simple processors instead of using folding would be more efficient.
Provisions

Reservations:

- using RT-level VHDL instead of BSV may offer better control over the critical path
- introducing more stages may increase clock frequency
- multi-method caches instead of one-method cache would improve overall performance
- other applications than the one we used (GC) could exhibit more folding potential
- more elaborate folding schemes may be more effective
Finally...

**Summary** We evaluated folding schemes for BlueJEP and conclude that the performance greatly decreases although the number of executed cycles is reduced.
Summary  We evaluated folding schemes for BlueJEP and conclude that the performance greatly decreases although the number of executed cycles is reduced.

Observation  Theoretical gains are not enough to show efficiency. **Complete implementations** must be evaluated!
Finally...

**Summary**  We evaluated folding schemes for BlueJEP and conclude that the performance greatly decreases although the number of executed cycles is reduced.

**Observation**  Theoretical gains are not enough to show efficiency. **Complete implementations** must be evaluated!

**Recommendation**  For our case, using several simple processors is potentially more efficient.
Thank you!
Thank you!

Questions?