Investigating Hardware Micro-Instruction Folding in a Java Embedded Processor

Flavius Gruian[1] Mark Westmijze[2]

[1]Lund University, Sweden
flavius.gruian@cs.lth.se

[2]University of Twente, The Netherlands
m.westmijze@student.utwente.nl

Java Technologies for Real-time and Embedded Systems, 2010

Outline

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 **J**ava **E**mbedded **P**rocessor,
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)

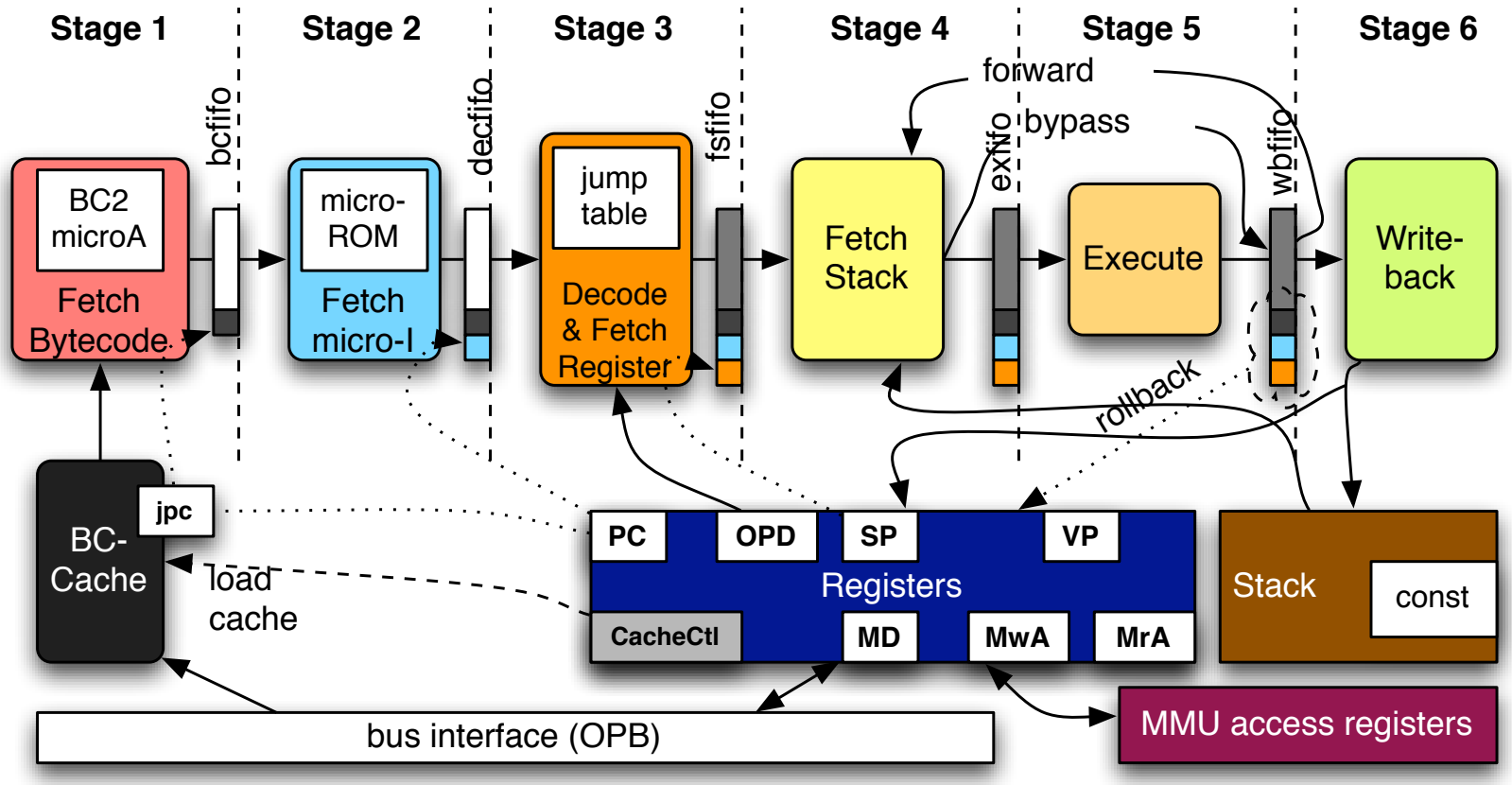
Original Processor Architecture

BlueJEP

BlueSpec System Verilog **J**ava **E**mbedded **P**rocessor,
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]

Six Stages Micro-Programmed Pipeline



Bytecode Folding Theory

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

stack code ≈7 bytes	3-address code ≈4 bytes
iload a iload b iadd istore c	add a, b, c

Bytecode Folding Theory

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

stack code ≈7 bytes	3-address code ≈4 bytes
iload a iload b iadd istore c	add a, b, c

- 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

stack code ≈7 bytes	3-address code ≈4 bytes
iload a iload b iadd istore c	add a, b, c

- 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)

Folding Pattern	Length
ppoc	4
poc	3
ppc	3
pc	2
oc	2
po	2

Pre-design Estimates

How much is the number of executed clock cycles reduced?

Pre-design Estimates

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

Architectural Changes

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

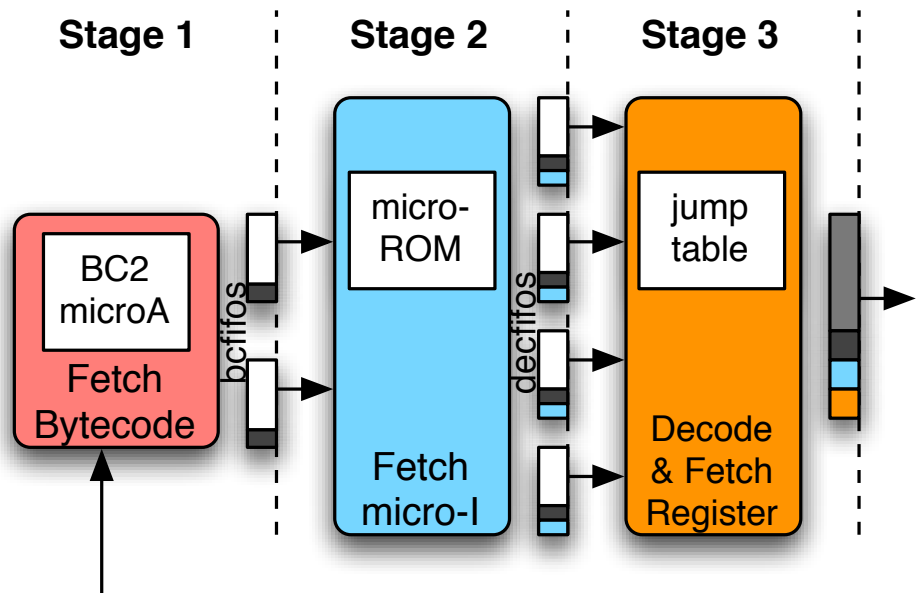


Figure: Handling 2 bytecodes, 4 micro-instructions simultaneously.

Setup and Tools

Synthesis $\longrightarrow$ device area, maximum clock frequency

- FPGA, Xilinx Virtex-5 (XC5VLX30-3)
- BSV compiler 2006.11, *BSV* $\longrightarrow$ *Verilog*
- Xilinx EDK 9.1i, *Verilog* + *IPs* $\longrightarrow$ *System*
- Xilinx ISE 9.1i, *System* $\longrightarrow$ *FPGA*
- Chipscope, to calibrate simulation

Setup and Tools

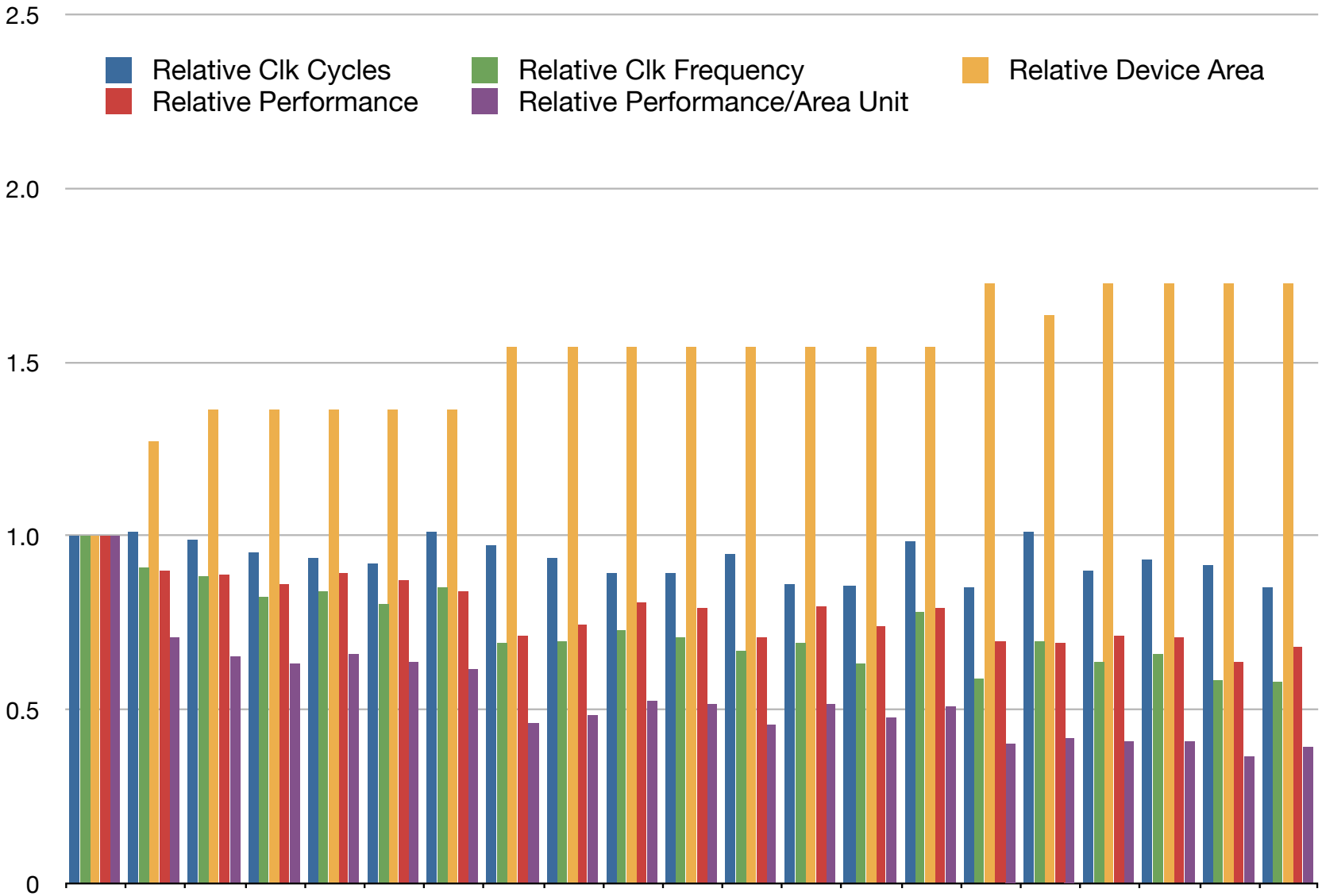
Synthesis → device area, maximum clock frequency

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

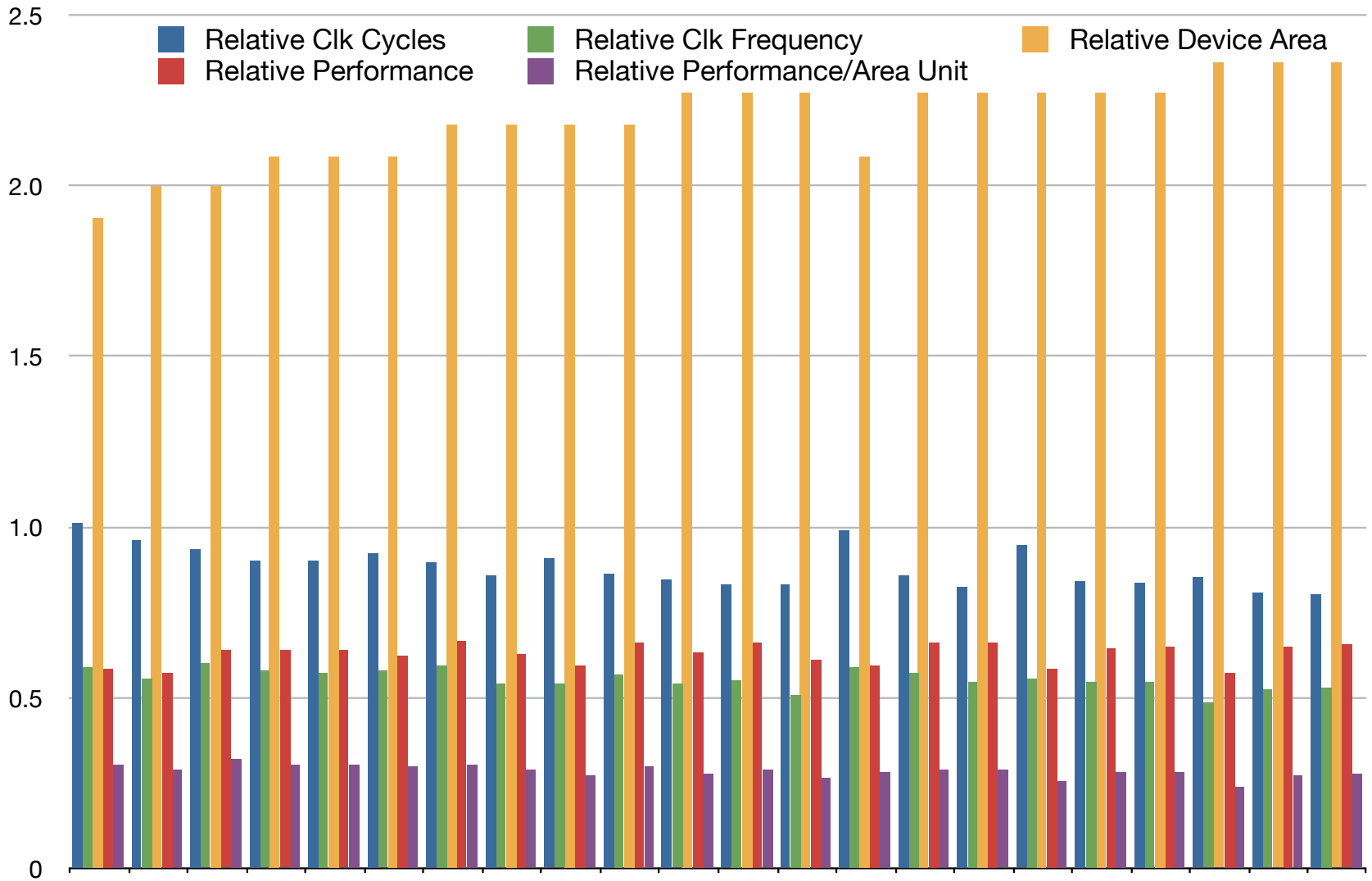
Simulation → executed clock cycles

- Desktop, Linux
- BSV compiler 2006.11, *BSV* → *Executable*
- custom tools for parsing the output from instrumented code

Original vs. Folding Configurations (2,2; 2,4)



Original vs. Folding Configurations (4,4)



Discussion

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

Discussion

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.

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!

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?