J OP: A Java Optimized Processor for Embedded Real-Time Systems

Martin Schöberl
University of Technology Vienna, Austria
Overview

- Motivation
- Related work
- JOP architecture
- WCET Analysis
- Results
- Conclusions, future work
- Demo
Embedded Systems

- An embedded system is a computer system that is part of a larger system.

Examples

- Washing machine
- Car engine control
- Mobile phone
Real-Time Systems

- A definition by John A. Stankovic:

  In real-time computing the correctness of the system depends not only on the logical result of the computation but also on the time at which the result is produced.
Real-Time Systems

- Imagine a car accident
  - What happens when the airbag is fired too late?
  - Even one ms too late is too late!
- Timing is an important property
- *Conservative* programming styles
RT System Properties

- Often safety critical
- Execution time has to be known
  - Analyzable system
    - Application software
    - Scheduling
    - Hardware properties
  - Worst case execution time (WCET)
Issues with COTS

- COTS are for average case performance
  - *Make the common case fast*
- Very complex to analyze WCET
  - Pipeline
  - Cache
  - Multiple execution units
The Idea

- Build a processor for RT System
  - Optimize for the worst case
- Design philosophy
  - Only WCET analyzable features
    - No unbound pipeline effects
    - New cache structure
  - Shall not be slow
Related Work

- picoJava
  - SUN, never released

- ajile JEMCore
  - Available, RTSJ, two versions

- Komodo
  - Multithreaded Java processor

- FemtoJava
  - Application specific processor
JOP Architecture

- Overview
- Microcode
- Processor pipeline
- An efficient stack machine
- Instruction cache
JOP Block Diagram
JVM Bytecode Issue

- Simple and complex instruction mix
- No bytecodes for *native* functions
- Common solution (e.g. in picoJava):
  - Implement a subset of the bytecodes
  - SW trap on complex instructions
  - Overhead for the trap – 16 to 926 cycles
  - Additional instructions (115!)
JOP Solution

- Translation to microcode in hardware
- Additional pipeline stage
- No overhead for complex bytecodes
  - 1 to 1 mapping results in single cycle execution
  - Microcode sequence for more complex bytecodes
- Bytecodes can be implemented in Java
Microcode

- Stack-oriented
- Compact
- Constant length
- Single cycle
- Low-level HW access

An example

dup: dup nxt // 1 to 1 mapping

// a and b are scratch variables
// for the JVM code.

dup_x1: stm a // save TOS
stm b // and TOS-1
ldm a // duplicate TOS
ldm b // restore TOS-1
ldm a nxt // restore TOS
// and fetch next bytecode
Processor Pipeline

- Bytecode: Fetch, translate and branch
- Microcode: Fetch and branch
- Microcode: Decode
- Microcode: Execute
- Stack: Address generation
- Stack: RAM

Flow:
- Next bytecode
- Bytecode branch condition
- Microcode branch condition
- Branch
- Spill, fill
An Efficient Stack Machine

- JVM stack is a logical stack
  - Frame for return information
  - Local variable area
  - Operand stack
- Argument-passing regulates the layout
- Operand stack and local variables need caching
Stack Access

- Stack operation
  - Read TOS and TOS-1
  - Execute
  - Write back TOS

- Variable load
  - Read from deeper stack location
  - Write into TOS

- Variable store
  - Read TOS
  - Write into deeper stack location
Two-Level Stack Cache

- Dual read only from TOS and TOS-1
- Two register (A/B)
- Dual-port memory
- Simpler Pipeline
- No forwarding logic

- Instruction fetch
- Instruction decode
- Execute, load or store
JVM Properties

- Short methods
- Maximum method size is restricted
- No branches out of or into a method
- Only relative branches
Proposed Cache Solution

- Full method cached
- Cache fill on call and return
  - Cache misses only at these bytecodes
- Relative addressing
  - No address translation necessary
- No fast tag memory
- Simpler WCET analysis
Architecture Summary

- Microcode
- 1+3 stage pipeline
- Two-level stack cache
- Method cache

The JVM is a CISC stack architecture, whereas JOP is a RISC stack architecture.
WCET Analysis

- WCET has to be known
  - Needed for schedulability analysis
  - Measurement usually not possible
    - Would require test of all possible cases

- Static analysis
  - Theory is mature
  - Low-level analysis is the issue
WCET Analysis

- Path analysis
- Low-level analysis (bytecodes)
- Global low-level analysis
- WCET Calculation
WCET Analysis for J OP

- Simple low-level analysis
- Bytecodes are independent
  - No shared state
  - No timing anomalies
- Bytecode timing is known and documented
- Simpler caches
WCET Tool

- Execution time of basic blocks
- Annotated loop bounds
- ILP problem solved
- Simple cache analysis included
  - Only two block cache in loops
  - Will be extended
Results

- Size
  - Compared to soft-core processors

- General performance
  - Application benchmark (KFL & UDP/IP)
  - Various Java systems
## Size of FPGA processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Resources (LC)</th>
<th>Memory (KB)</th>
<th>$f_{\text{max}}$ (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J OP min.</td>
<td>1077</td>
<td>3.25</td>
<td>98</td>
</tr>
<tr>
<td>J OP typ.</td>
<td>1831</td>
<td>3.25</td>
<td>101</td>
</tr>
<tr>
<td>Lightfoot</td>
<td>3400</td>
<td>1</td>
<td>40</td>
</tr>
<tr>
<td>Komodo</td>
<td>2600</td>
<td>?</td>
<td>33/4</td>
</tr>
<tr>
<td>FemtoJ ava</td>
<td>2000</td>
<td>?</td>
<td>4</td>
</tr>
<tr>
<td>NIOS</td>
<td>2923</td>
<td>5.5</td>
<td>119</td>
</tr>
</tbody>
</table>
Application Benchmark

Preformance [iterations]

Embedded Java Systems

Java Optimized Processor
Applications

- Kippfahrleitung
  - Distributed motor control

- ÖBB
  - Vereinfachtes Zugleitsystem
  - GPS, GPRS, supervision

- TeleAlarm
  - Remote tele-control
  - Data logging
  - Automation
JOP in Research

- University of Lund, SE
  - Application specific hardware (Java->VHDL)
  - Hardware garbage collector
- Technical University Graz, AT
  - HW accelerator for encryption
- University of York, GB
  - Javamen – HW for real-time systems
- Institute of Informatics at CBS, DK
  - Real-time GC
  - Embedded RT Machine Learning
JOP for Teaching

- Easy access – open-source
  - Computer architecture
  - Embedded systems

- UT Vienna
  - JVM in hardware course
  - Digital signal processing lab

- CBS
  - Distributed data mining (WS 2005)
  - Very small information systems (SS 2006)

- Wikiiversity
Conclusions

- Real-time Java processor
  - Exactly known execution time of the BCs
  - Time-predictable method cache
  - Simple real-time profile
- Resource-constrained processor
  - RISC stack architecture
  - Efficient stack cache
  - Flexible architecture
Future Work

- Real-time garbage collector
- Instruction cache WC analysis
- Hardware accelerator
- Multiprocessor JVM
- Java computer
More Information

- Two pages short paper
- JOP Thesis and source
  - http://www.jopdesign.com/download.jsp
- Various papers
  - http://www.jopdesign.com/docu.jsp