APPLICATION REQUIREMENTS AND EFFICIENCY OF EMBEDDED JAVA BYTECODE MULTI-CORES

JTRES 2010

Martin Zabel, Rainer G. Spallek

Prague, 19.08.2010
Itinerary

- Motivation
- Application Requirements
- SHAP Multi-Core Design
- Performance Evaluation
- Summary
Motivation

- Complexity of applications increases:
  - Raise computational throughput.
  - Decrease latency.

- Previous Approaches: smarter Java bytecode single-cores.
  - Just-In-Time compilation.
  - Instruction-level parallelism: bytecode folding, VLIW packets.
  - Bit-level parallelism.

Application Requirements

Address Spaces

- Code Area:
  - Access frequently.
  - Duplication $\Rightarrow$ Chip-space intensive.
  - Sharing $\Rightarrow$ Efficient method caching.

- Shared Java Heap:
  - UMA/NUMA.
  - Fast atomic operations for monitor lock/unlock.
  - Independent locks, otherwise performance is degraded.
  - Memory bus utilization.

- Shared Peripherals.
Application Requirements

Memory Bandwidth

- SHAP single-core already includes multi-port memory manager:
  - DMA and cache-line filling.
  - Pipelined transactions using outstanding reads.
  - Maximum bandwidth with pipelined memory (ZBT SRAM).

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Data</td>
<td>Cache</td>
<td>Cache</td>
<td>DMA</td>
<td>DMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reply</td>
<td>Data</td>
<td>Cache</td>
<td>Cache</td>
<td>DMA</td>
<td>DMA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Application Requirements

Memory Bandwidth Utilization

<table>
<thead>
<tr>
<th>Application</th>
<th>ML505</th>
<th>DE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queens</td>
<td>3.09%</td>
<td>6.35%</td>
</tr>
<tr>
<td>Lift</td>
<td>10.18%</td>
<td>22.12%</td>
</tr>
<tr>
<td>FScriptME</td>
<td>8.46%</td>
<td>17.60%</td>
</tr>
<tr>
<td>SMMI</td>
<td>13.26%</td>
<td>30.43%</td>
</tr>
<tr>
<td>El-Kharashi</td>
<td>≈ 10%</td>
<td>≈ 21%</td>
</tr>
</tbody>
</table>

Utilization:
\[
u = \frac{m_{\text{Total}}}{e_{\text{Total}}} = \frac{\sum b f_b \cdot m_b}{\sum b f_b \cdot e_b}
\]

Clock Cycle

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ML505</strong></td>
<td></td>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C0</td>
<td></td>
</tr>
<tr>
<td><strong>ML505</strong></td>
<td>C0</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C0</td>
<td></td>
</tr>
<tr>
<td><strong>DE2</strong></td>
<td>C0</td>
<td>W</td>
<td>C1</td>
<td>W</td>
<td>C2</td>
<td>W</td>
<td>C3</td>
<td>W</td>
<td></td>
<td>C0</td>
<td></td>
</tr>
<tr>
<td><strong>DE2</strong></td>
<td>C0</td>
<td></td>
<td>C1</td>
<td></td>
<td>C2</td>
<td></td>
<td>C3</td>
<td></td>
<td></td>
<td>C0</td>
<td></td>
</tr>
</tbody>
</table>
Application Requirements

Conclusion

⇒ UMA setup is suitable if whole memory subsystem can be operated in a pipelined fashion.

⇒ Bandwidth sufficient for up to 10 cores on ML505.
SHAP Multi-Core Microarchitecture

- MMU w. variable port count.
- Full-duplex memory bus with pipelined transactions.
- Multi-threaded, real-time capable cores with local on-chip stack and method cache.
- Exact and fully concurrent non-blocking garbage collector.
- Native execution of Java bytecode.
- Fast atomic operations for independent locks.
- Synthesizable for a variable number of cores.
SHAP Implementation on ML505

- Main evaluation platform: Xilinx ML505 Development Board
  - Virtex-5 XC5VLX50T
  - 1 MB external ZBT SRAM with 32-bit data bus.
  - 80 MHz clock frequency for up to 9 cores.
- Setup: 8 KB stack and 4 KB method cache per core. Minimum I/O.
- Chip-space scales linear:

  \[
  \text{LUTs}(n) \approx 2794 + 2831 \cdot n \\
  \text{FFs}(n) \approx 1933 + 1447 \cdot n \\
  \text{18 kbit BRAMs}(n) = 1 + 2 \cdot n \\
  \text{36 kbit BRAMs}(n) = 1 + 3 \cdot n \\
  \text{Multiplier}(n) = 2 + 3 \cdot n
  \]

- Other platforms available: Xilinx Spartan-3/3E Starter Kit, Altera DE2 Development Board.
Relative Speed-Up

- Measured on: Xilinx ML505 Virtex-5 Development Board.
- Pipelined ZBT SRAM with 32-bit data bus.
Area Efficiency

- Absolute/relative area of LUTs, FFs, BRAMs and multipliers is unknown.
- Speed-Up in relation to count of BRAMs on ML505.
Comparison Against Related Projects

- Other Java bytecode multi-cores: JopCMP, jamuth IP multi-core, REALJava.
- Comparison of absolute performance, etc.
- Using same platform.

⇒ JopCMP on Altera DE2 Development Board.
Comparison Against JopCMP

Absolute Performance on DE2 Board

- SHAP @ 60 MHz, JopCMP @ 90 MHz
- Asynchronous SRAM with only 16-bit data bus.

![Graph showing performance comparison between SHAP and JopCMP](image-url)
Comparison Against JopCMP

**Chip-Space on DE2 Board**

- SHAP implements GC in hardware.
- SHAP-core requires about 23% more LEs than a JopCMP core.
Synchronization Performance

- Sync. limits speed-up ⇒ Keep as short/rare as possible.
- Highly application/API dependent.
- Typical: *synchronized* blocks.
  - Long blocking periods for field update(s).
  - Only small amount for atomic bus access.
    Example: `java.util.concurrent.LinkedBlockingQueue.put()`

- Alternative: compare and swap.
  - Short blocking period: only atomic bus access.
  - Complex code.
Summary

- Multi-core Java bytecode processor w. shared heap.
- Multi-port MMU w. pipelined transactions.
- Chip-space scales linear.
- Application mix:
  - Typical: 10% memory bandwidth utilization.
  - Almost linear speed-up for up to 9 cores.
  - Area efficiency of $\approx 120\% (> 1 \text{ core})$.
- Better absolute performance than related project JopCMP.

http://shap.inf.tu-dresden.de