Overview

September 2004

Advanced Multithreading Architecture with Hardware based Thread Scheduling

Ye Lu
Sakir Sezer
John McCanny

FPL 2010 in Milan
• Introduction to multithreading
• Architecture overview
• Thread scheduler
• Pipeline control
• Synthesis and Simulation results
• Conclusions
Introduction

- Hardware support for effectively execute multiple threads on a single core

- Taxonomy
  - Fine Grained MT (IMT)
  - Coarse Grained MT (BMT)
  - Simultaneous MT

- Goals of the proposed architecture
  - Reduced hardware overhead
  - Low context switching penalty
  - Maintaining single thread performance

Preferred for softcore implementation
Architecture overview

- Advanced thread management
- Zero context switch penalty
- “Temporary execution”
- 5 pipeline stages
- Supporting 4 threads each core
- MIPS ISA for prototyping
Thread scheduler

- Pre-decodes the instruction
- Controls the thread switching
- Maintains the thread state
- Tracks the status of execution and memory access stages
- Determines the next active thread
• Pre-decoding logic detects the instruction
• Finite state machine manages the thread state
• Arbitration logic decides the next active thread
- **ACTIVE**: Thread is executing
- **WAIT**: Thread is waiting for branch decision or memory access
- **READY**: Thread is ready to be executed
- **Temp**: Instruction detected but no alternate thread

**States:**
- TEMP: Instruction detected but no alternate thread
- WAIT: Thread is waiting for branch decision or memory access
- READY: Thread is ready to be executed
- ACTIVE: Thread is executing

**Transitions:**
- TEMP to WAIT: Other thread ready
- TEMP to ACTIVE: Instr detected && (!ready Thread)
- WAIT to READY: Latency complete
- ACTIVE to READY: Latency complete
- ACTIVE to TEMP: Win arbitration
- READY to ACTIVE: Ld, Beq, Bne, etc., && ready thread
Thread Arbitration

- Least recently served scheme
- Matrix arbiter
• Date are launched and latched at different clock edges, thus the scheduler is working at doubled clock rate

• Moving the thread arbitration logic out of the critical path and using one-hot encoding to increase the speed
• Supporting multiple thread to share the pipeline resource

• Resolving the possible pipeline hazard
  ➢ Control hazard
  ➢ Data hazard

• Using thread tags to distinguish instructions from different threads
Pipeline Control

(1) Control hazard
(2) Forwarding
(3) Load delay
(4) Pipeline lock & flush
1. Control hazard
   - Hidden by the thread switching
   - When no alternate thread and mis-prediction happens
     ✓ Exam the ID of instructions
     ✓ Flush the corresponding pipeline registers
     ✓ Change the PC

2. Conventional RAW data hazard
   - No effect to the performance
     ✓ Exam the ID of instructions
     ✓ Resolved by the forwarding logic
3. Load delay (RAW following load)
   - Hidden by the thread switching
   - When no alternate thread and dependence happens
     - Can not be resolved by the forwarding path
     - Stall the PC and IF/ID pipeline register

4. Cache missing without alternate thread
   - Stall the pipeline
   - When any other thread ready, switch to the ready thread, flush the pipeline, recover the context
Thread 0
(1) Add r10, r11, r8
(2) Lw r12, r16, 24
(3) Lw r14, r18, 32
(4) Add r18, r16, r20

Thread 1
(5) Add r16, r15, r10
(6) Sub r10, r11, r8
(7) Beq r12, r13, Label
(8) Add r11, r13, r6
(9) Add r14, r17, r8

Label:
(10) Lw r4, r14, 100
(11) Add r13, r15, r10
(12) Mul r16, r18, r20

Assembly code

Execution

Thread 0

Thread 1

Pipe Lock
Branch

T

Active
Wait
Ready
• Logic overhead

<table>
<thead>
<tr>
<th>Components</th>
<th>Number of ALUTs</th>
<th>Number of Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-decoding logic</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>FSM (total)</td>
<td>30</td>
<td>16</td>
</tr>
<tr>
<td>Arbitration logic</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>Control hazard logic</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Forwarding path</td>
<td>78</td>
<td>0</td>
</tr>
<tr>
<td>Load delay control</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>133</td>
<td>22</td>
</tr>
</tbody>
</table>

• Clock rate

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interconnection Delay</td>
<td>0.708 ns (45% of total)</td>
</tr>
<tr>
<td>Cell Delay</td>
<td>0.774 ns (50% of total)</td>
</tr>
<tr>
<td>uTco</td>
<td>0.066 ns (4% of total)</td>
</tr>
<tr>
<td>Fmax</td>
<td>324Mhz</td>
</tr>
</tbody>
</table>
Compared to a conventional BMT architecture

Reference:
Ye Lu, Sakir Sezer, John McCanny, Design and Analysis of an Advanced Static Blocked Multithreading Architecture, SOCC 2010
Conclusions

• We present a multithreading architecture that manages thread according to the thread state and instruction decoding.

• The architecture can eliminate the context switching penalty while maintaining the single thread performance.

• We demonstrate its performance and feasibility for softcore implementation.
Thanks for your attention

Questions?