CHIMAERA: A High-Performance Architecture with a Tightly-Coupled Reconfigurable Functional Unit

Kynan Fraser

Disclaimer

This presentation is based on a paper written by Zhi Alex Yi, Andreas Moshovos, Scott Hauck and Prithviraj Banerjee. The paper is as named in the title.

All proposals, implementation, testing, results and figures and tables have been done by the aforementioned peoples.

These slides however have been produced by me for educational purposes.

Outline

- Background
- Introduction
- Chimaera architecture
- Compiler support
- Related work (not covered)
- Evaluation
  - methodology
  - modelling RFUOP latency
  - RFUOP analysis
  - working set of RFUOP's
  - performance measurements
- Summary

Background

Customized vs Flexibility
Benefits vs Risks
Reconfigurable solution??
Multimedia platforms
Introduction

CHIMAERA
Reconfigurable hardware and compiler

- Coupled RFU (Reconfigurable Functional Unit)
- Implements application specific operations
- 9 inputs to 1 output
- Fairly simple compiler

Introduction – Potential Advantages

- Reduce execution time of dependent instructions
  - tmp=R2–R3; R5=tmp+R1
- Reduce dynamic branch count
  - if (a>88) a=b+3
- Exploit subword parallelism
  - a = a + 3; b = c << 2 (a,b,c halfwords)
- Reduce resource contention

Chimaera Architecture

- Reconfigurable Array
  - programmable logic blocks
- Shadow Register File
- Execution Control Unit
- Configuration Control and Caching Unit

RFUOP unique ID
In-order commits
Single Issue RFUOP scheduler
Worst case 23 transistor levels (for one logic block)
**Compiler Support**

- Automatically maps groups of instructions to RFUOP's
- Analyses DFG's
- Schedules across branches
- Identifies sub-word parallelism (disabled in this case due to endangered correctness)
- Look later at how many can instructions actually map to RFUOP's

---

**Configuration**

<table>
<thead>
<tr>
<th>Component</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar Core</td>
<td></td>
</tr>
<tr>
<td>Branch predictor</td>
<td>6th OSILARE</td>
</tr>
<tr>
<td>Scheduler</td>
<td>Out-of-order issue of up to 4 operations per cycle, 128-entry re-order buffer (ROB), 32-entry load/store queue (LSQ)</td>
</tr>
<tr>
<td>Functional units</td>
<td>4 integer ALUs, 1 integer MULT, 1 FP adder, 1 FP multiplier</td>
</tr>
<tr>
<td>Functional unit latencies</td>
<td>Integer ALU, integer MULT, integer ADD/12, FP adder 12, FP MULT 6, FP DIF 12, load/store 1</td>
</tr>
<tr>
<td>Instruction cache</td>
<td>L1: Direct-Mapped, 32-byte block, 1 cycle hit latency</td>
</tr>
<tr>
<td>Data cache</td>
<td>L2: Direct-Mapped, write-back, write-allocate, non-blocking, 32-byte blocks, 1 cycle hit latency</td>
</tr>
<tr>
<td>L2 cache</td>
<td>Unified 4-way set associative, 128-byte, 12 cycles hit latency</td>
</tr>
<tr>
<td>Main memory</td>
<td>Infinite size, 160 cycles hit latency</td>
</tr>
<tr>
<td>Fetch Mechanism</td>
<td>Up to 4 instructions per cycle</td>
</tr>
</tbody>
</table>

**Reconfigurable Functional Unit**

<table>
<thead>
<tr>
<th>Component</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheduler</td>
<td>6 entries, each entry corresponds to a single RFUOP Single Issue, Single Write-back per cycle. An RFUOP core issue if all its inputs are available and no other instance of the same RFUOP is currently executing.</td>
</tr>
<tr>
<td>Functional Unit / RA</td>
<td>32 rows. Each RFUOP occupies as many rows as instructions of the original program it replaced. (pessimistic) With a single instance of each RFUOP can be active at any given point in time.</td>
</tr>
<tr>
<td>Configuration Loading</td>
<td>1st level configuration cache of 12 configuration rows (32 x 256 bytes). Configuration loading is modeled by reporting access to the rest of the memory hierarchy. Execution stalls for the duration of configuration loading.</td>
</tr>
<tr>
<td>RFUOP Latency</td>
<td>Various model simulated. See Section 3.3.1.</td>
</tr>
</tbody>
</table>

**Table 3: Benchmark characteristics**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
<th>Inst. Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaBench Benchmarks:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEPGEN</td>
<td>MPEG encoder</td>
<td>1139 M</td>
</tr>
<tr>
<td>G722enc</td>
<td>CCITT G.722 voice encoder</td>
<td>309 M</td>
</tr>
<tr>
<td>G729dec</td>
<td>CCITT G.721 voice decoder</td>
<td>294 M</td>
</tr>
<tr>
<td>ADPCM enc</td>
<td>Speech compression</td>
<td>6.6 M</td>
</tr>
<tr>
<td>ADPCM dec</td>
<td>Speech decompression</td>
<td>5.6 M</td>
</tr>
<tr>
<td>JPEG</td>
<td>JPEG file compression</td>
<td>12.3 M</td>
</tr>
<tr>
<td>PGP</td>
<td>PGP symmetric key generation. PGP is a public key encryption and authentication application.</td>
<td>3.8 M</td>
</tr>
<tr>
<td>PGP decryption</td>
<td>PGP private decryption</td>
<td>12.5 M</td>
</tr>
<tr>
<td>Honeywell Benchmarks:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comp</td>
<td>Image compression</td>
<td>34.1 M</td>
</tr>
<tr>
<td>Decomp</td>
<td>Image decompression</td>
<td>32.7 M</td>
</tr>
</tbody>
</table>

**Evaluation - Methodology**

- Execution driven timing
- Built over simplescalar
- ISA extension of MIPS
- RFUOP's appear as NOOP's under MIPS ISA
- Previous slide configuration used

**Related Work**

- We are looking at it
- Read section 4 for more information
**Evaluation – Modelling RFUOP Latency**

- First row modelled on original instruction sequence critical path
- Second row modelled on transistor levels and delays

---

**Evaluation – RFUOP Analysis**

- Total number of RFUOP's per benchmark
- Frequency of instruction types mapped to RFUOP's

---

**Evaluation – RFUOP Analysis**

- Look at how many instructions replaced by RFUOP
  - 
    - dest = src1 op src2 op src3
    - 3/4 input/output most
- Look at critical path of instructions replaced

---

**Evaluation – Working set of RFUOP’s**

- Larger working set = more stalls to configure
- Maintaining 4 MRU almost no misses
- 16 rows sufficient

---

**Table 6: RFUOP latency (ns)**

<table>
<thead>
<tr>
<th>Model</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>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>750</td>
<td>900</td>
<td>1050</td>
<td>1200</td>
<td>1350</td>
<td>1500</td>
<td>1650</td>
<td>1800</td>
</tr>
<tr>
<td>Transistor</td>
<td>1000</td>
<td>1100</td>
<td>1200</td>
<td>1300</td>
<td>1400</td>
<td>1500</td>
<td>1600</td>
<td>1700</td>
</tr>
</tbody>
</table>

**Table 7: RFUOP distribution in terms of critical instructions (ns) (range shown is 1 to 17 instructions, iclusions omitted have 0% in all ranges).**

<table>
<thead>
<tr>
<th>Instruction</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>
</tr>
</thead>
<tbody>
<tr>
<td>dest = src1</td>
<td>50%</td>
<td>40%</td>
<td>30%</td>
<td>20%</td>
<td>10%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>op src2</td>
<td>40%</td>
<td>30%</td>
<td>20%</td>
<td>10%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>op src3</td>
<td>10%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
</tbody>
</table>

**Figure 6: RFUOP working set. Cache size is the number of elements that can exist in the RFU.**

**Figure 7: RFUOP configuration size working set. Cache size is the number of rows in the RFU.**


**Evaluation – Performance Measurements**

- Performance under original instruction timing latencies (4 issue)
- Latency of 2C or better still give speed of 11% or greater, 3C not worthwhile

- 3C not worthwhile (only speedup under one benchmark)
- N model improves performance overall
  - due to decreased branches and reduced resource contention

---

**Evaluation – Performance Measurements**

- Performance under transistor timing (4 issue)
- Improvements of 21% even under most conservative transistor timing
- Performance in optimistic models close to 1-cycle model (upper bound)
**Evaluation – Performance Measurements**

- Performance with 8 issue
- Only improvements with C, 1, 2 and N timing
- Relative improvements (to 4 issue) small
- Reason: Because limited to one RFUOP issue per cycle

---

**Evaluation – Performance Measurements**

- Strong relationship between performance improvement and branches replaced by RFUOP's
- Benchmarks with lowest branch reduction have lowest speedup
- Even under pessimistic assumptions Chimaera still provides improvements

---

**Summary**

- Seen the CHIMAERA architecture
- The C compiler that generates RFUOP's
- Maps sequences of instructions into single instruction (9input/1output)
- Can eliminate flow control instructions
- Exploits sub-word parallelism (not here)
- 22% on average of all instructions to RFUOP's
- Variety of computations mapped
- Studied under variety of configurations and timing models
Summary

- 4 way: average 21% speedup for transistor timing (pessimistic)
- 8 way: average 11% speedup for transistor timing (pessimistic)
- 4 way: average 28% speedup for transistor timing (reasonable)
- 8 way: average 25% speedup for transistor timing (reasonable)