Federation: Repurposing Scalar Cores for Out-of-Order Execution

Michael Boyer, David Tarjan, Kevin Skadron
University of Virginia
boyer@cs.virginia.edu
Outline

- Motivation
- Overview of Federation
- Architectural details:
  - Issue queue (IQ)
  - Load-store queue (LSQ)
- Performance, power, and area impact
- Conclusion
Motivation

Best architecture depends on workload:

Lots of parallelism

Limited parallelism

How do we choose at design time without knowing the workload characteristics?
Basic Idea

Allow the architecture to adapt at runtime:

Throughput (Homogeneous)

Mixture (Heterogeneous)

Latency (Homogeneous)
Key Insights

- Large, multi-threaded register files can be repurposed to support out-of-order execution
- If cores are small, single-cycle communication between neighbors is feasible
- Leverage prior work on scaling large out-of-order cores in order to reduce overhead
Adaptive Pipeline (2)

<table>
<thead>
<tr>
<th></th>
<th>In-Order Core 1</th>
<th>In-Order Core 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OO Structures</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>I$</td>
<td>I$</td>
</tr>
<tr>
<td>Fetch</td>
<td>Fetch</td>
<td>Fetch</td>
</tr>
<tr>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
</tr>
<tr>
<td>RF</td>
<td>RF</td>
<td>RF</td>
</tr>
<tr>
<td>EXE</td>
<td>EXE</td>
<td>EXE</td>
</tr>
<tr>
<td>BP</td>
<td>Bus</td>
<td>BP</td>
</tr>
<tr>
<td>Bus</td>
<td>Alloc</td>
<td>Bus</td>
</tr>
<tr>
<td>Alloc</td>
<td>Ren</td>
<td>Alloc</td>
</tr>
<tr>
<td>Ren</td>
<td>IQ</td>
<td>Ren</td>
</tr>
<tr>
<td>IQ</td>
<td>Bus</td>
<td>IQ</td>
</tr>
<tr>
<td>Bus</td>
<td></td>
<td>Bus</td>
</tr>
<tr>
<td></td>
<td>D$</td>
<td>D$</td>
</tr>
</tbody>
</table>
Consumer-Based Issue Queue

- Consumers subscribe to their producers at rename
- Producers set ready bits of their consumers at execute
- Fixed number of subscriber slots can cause stalls

- Select uses static priority encoder based on IQ position rather than oldest-first
Simplified Load-Store Queue

- Memory Alias Table (MAT)
- No store forwarding
- No conservative waiting on stores
- Only detect memory order violations after they have occurred and flush the pipeline when the offending instruction commits
MAT Example

st 0x13, r5
ld r1, 0x13
MAT Example

```
st 0x13, r5
ld r1, 0x13    EXE
```

Id executes and increments counter

<table>
<thead>
<tr>
<th>MAT</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</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>
MAT Example

st 0x13, r5   COM
ld r1, 0x13

st commits and
sets flag
MAT Example

\[ \text{ld r1, 0x13 COM} \]

\[ \text{ld commits, sees flag, and flushes pipeline} \]

MAT Table:

<table>
<thead>
<tr>
<th>MAT</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>!</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Flush**
MAT Example

ld r1, 0x13

MAT is reset and execution resumes
## Area Overhead

<table>
<thead>
<tr>
<th>Core Type</th>
<th>Area (mm²)</th>
<th>Scalar Core Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-way in-order</td>
<td>1.91</td>
<td>-</td>
</tr>
<tr>
<td>Federated out-of-order</td>
<td>3.97</td>
<td>0.07</td>
</tr>
<tr>
<td>2-way out-of-order</td>
<td>5.07</td>
<td>2.65</td>
</tr>
<tr>
<td>4-way out-of-order</td>
<td>11.19</td>
<td>5.85</td>
</tr>
</tbody>
</table>
Simulation Methodology

- **Simulator:**
  - SimpleScalar 3.0
  - Wattch

- **Workloads:**
  - SPEC2000 benchmarks
  - Simpoint: select 100 million instructions
Performance Impact

- Consumer-based issue queue: 0.00%
- Pseudo-random scheduling: 2.67%
- MAT: 1.71%
- Commit-time branch recovery: 5.46%
Performance Results

The image contains a bar chart showing performance results for different configurations. The x-axis represents various configurations: Scalar IO, 2-way IO, Federated OO, 2-way OO, and 4-way OO. The y-axis represents average IPC (Instructions Per Cycle) with values ranging from 0 to 1.4. The chart compares three benchmarks: spec, specint, and specfp, indicated by different colors (spec - dark maroon, specint - blue, specfp - yellow). The chart visually represents the performance comparison across these benchmarks and configurations.
Energy Efficiency Results

Normalized BIPS^3/Watt

Scalar IO  2-way IO  Federated OO  2-way OO  4-way OO

spec  specint  specfp
Energy-Area Efficiency Results

![Normalized BIPS^3/(Watt*mm^2)]

- **Scalar IO**
- **2-way IO**
- **Federated OO**
- **2-way OO**
- **4-way OO**

Legend:
- **spec**
- **specint**
- **specfp**
Conclusions

- Two in-order cores can be Federated \emph{at runtime} to form a two-way out-of-order core with low area overhead
- Federated core approximately doubles single-thread performance of underlying cores
- Best energy-area efficiency of options studied
- Federation allows an architecture to dynamically trade off throughput and single-thread latency
Future Work

Software support

System-level architecture

Architecture of individual cores

Low-level implementation
Technology Transfer

- Liaison interactions:
  - AMD: Srilatha Manne
  - Intel: Perry Wang & Jamison Collins
  - Quarterly meetings

- Publications:
  - DAC 2008
  - Longer tech report published on SRC website
  - Journal version currently in submission

- Presentations:
  - Intel, December 2007 (hosted by Pradeep Dubey)

- Internships:
  - Intel, Summer 2007 (unrelated to this work)
Acknowledgments

This work was supported by:
SRC/AMD Fellowship
SRC grant: 2007-HJ-1607
NSF grants: CCR-0306404
CNS-0509245
IIS-0612049

Thanks to:
Michael Frank
Doug Burger
Mircea Stan
Jeremy Sheaffer