Load Latency Tolerance In Dynamically Scheduled Processors

Srikanth T. Srinivasan and Alvin R. Lebeck
Department of Computer Science
Duke University

www.cs.duke.edu/ari/ice
Instruction Level Parallelism

• **Mechanisms**
  – Dynamic Scheduling
  – Branch Prediction
  – Speculative Execution

• **Mechanisms help tolerate Latency**

• **Limitations**
  – Data Dependencies
  – Finite Resources
  – Branch Mispredictions

• **Long Latency Operations exacerbate Limitations**
Motivation

- Criticality of Loads vary
- Ideal: all Loads take 1 cycle
- How close to Ideal can we get?
- Do current Memory systems match variation?
Measuring Load Latency Tolerance

- Increase Latency
  - Uniform [Lizy Kurian, ISCA92]
    » All or Nothing approach
  - Miss Penalties
    » Cache Organization dependent

Our Approach
- How long can each Load wait to complete?
- Remove Memory hierarchy
- Performance level controls Load completion
- Achieve IPC close to Ideal memory system
- Simulate processor with constrained resources

No insight on individual Load Latency Tolerance
Simulation Methodology

• Delay Load completion
• Every cycle, our simulator asks
  – Should Loads complete?
  – Which Loads should complete?
  – When should Loads complete?
  – How many Loads should complete?

• Latency Tolerance = Completion time - Issue time

• Goal: Maximize IPC and Load Latency Tolerance
Should Loads Complete?

Branch-based
- Minimize speculative execution
- Mispredicted Branches alone?

Performance-based
- Free dependent instructions
- Release buffer space
- Metrics
  - Instruction Issue Rate
  - Functional Unit Utilization

Limit-based
- 32 cycle limit
Branch-based Load Completion

- **Which?** All Loads on which Branch is dependent
- **When?** Avoid need for Branch prediction
  - Need for Rollback

Load completion time = $t - 8$

Branch execution time = $t$
Performance-based Load Completion

• **Which ?**
  - Program order
  - Dependence Graph Depth based

• **When ?**
  - Allow pipeline to fill-up with ready instructions

![Dependence Graph Diagram]

**Completion** • **Pipeline fill-up time** • **Detection**

Time (cycles)  t-2 → t
Outline

• Motivation
• Measuring Load Latency Tolerance
  • Experimental Setup
• Results
• Future Work
• Conclusion
Experimental Setup

- Simplescalar
- Checkpointing, Rollback and Sampling
- Benchmarks - Spec95
  - compress, gcc, li, vortex, hydro2d, swim, tomcatv, wave
- Reference data set
  - up to 10 billion instructions with 1% sampling
  - IPC values within 5% of complete simulations
- Baseline processor
  - 8 issue, out-of-order processor
  - 256 RUU entries, 128 LSQ entries
  - 2-level branch predictor with a total of 8192 entries
Goals Revisited

- **Maximize IPC and Latency Tolerance**
  - Evaluate Load Completion Parameters
- **Variation in Load Latency Tolerance**
- **Measured Tolerance vs. Actual Latency**
Branch-based Load Completion

- Performance decreases if Branches not considered
- IPC : mispred == all
- Latency Tolerance : mispred >> all
- Latency Tolerance increases with prediction accuracy
• Less than 2% of Loads completed due to Branches
• No effect on IPC or Latency Tolerance
Performance-based Load Completion

- IPC decreases with threshold
- Latency Tolerance increases as threshold decreases
- IPC : issue4 == fu4
- Latency Tolerance : issue4 >> fu4
Performance-based Load Completion

- IPC : issue4 == fu4
- Latency Tolerance : issue4 >> fu4
Load Completion Parameters - Summary

- **Should Loads complete?**
  - Loads with dependent mispredicted Branches
  - Issue Rate threshold of 4

- **Which Loads should complete?**
  - Dependence Graph based
  - Satisfying Loads in program order not always best

- **When should Loads complete?**
  - Pipeline fill-up time of 2 cycles

- **How long can Loads wait?**
  - 13% to 62% must complete in 1 cycle
  - 2% to 42% can wait for 8 cycles or more
  - IPC within 8% of ideal memory system
Processor Architecture and Latency Tolerance

- IPC within 11% of ideal
- Decreasing issue-width or increasing buffer-space increases Load Latency Tolerance
- 4/256/128 swim: 84% Loads can complete in 32 cycles
Traditional Memory Hierarchies and Latency Tolerance

<table>
<thead>
<tr>
<th>Measured Tolerance</th>
<th>Where Satisfied?</th>
<th>vortex</th>
<th>Measured Tolerance</th>
<th>Where Satisfied?</th>
<th>swim</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L1 Cache</td>
<td></td>
<td></td>
<td>L1 Cache</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>53%</td>
<td>3%</td>
<td>Low</td>
<td>45%</td>
<td>16%</td>
</tr>
<tr>
<td>High</td>
<td>21%</td>
<td>1%</td>
<td>High</td>
<td>23%</td>
<td>7%</td>
</tr>
</tbody>
</table>

- Measured Tolerance does not match actual Latencies
- Caches must differentiate between critical and non-critical Loads
Future Work

- Use Latency Tolerance information in Cache replacement decisions
- Give priority to critical Loads in Memory system Queues
Conclusion

- Measured individual Load Latency Tolerances
- Variation in Latency Tolerance among Loads exists
  - 1% to 62% must complete in 1 cycle
  - Up to 84% can wait for 32 cycles
- Traditional memory systems do not capture variation
Dynamically Scheduled Processors and Latency Tolerance

- Dynamically Scheduled Processors: Limitations
  - Branch Mispredictions
  - Data Dependencies
  - Finite Resources

Characteristic of Program

Machine Dependent

swim
Loads completed and Latency Tolerance

vortex

swim

IPC

ideal
nl4
nl2
nl1
traditional
fixed (8 cycles)