Exploring Parallel Out-of-Order Re-execution

David Morano
David Kaeli
Northeastern University
Computer Architecture Research

introduction

• Attempting to take advantage of previous work that has shown some promise in extracting IPC from sequential programs
  – IPC of about 6 for 256 simultaneously re-executing instructions
  – IPC of about 7.5 for 512 simultaneously re-executing instructions
• Applying some of the re-execution ideas to a more conventional superscalar microarchitecture
• Some ideas to be carried forward:
  – retaining binary program compatibility to existing ISAs
  – breaking control and data dependencies
  – time-ordering-tags as the dependency enforcement mechanism
  – designing for re-executions of all instruction types (not just memory loads)
  – dynamic handling of speculative execution and operand management

execution window

Outline

• Introduction
• Execution window
• Basic machine operation
• Issue stations
• Example execution
• Summary
### major components

- **Issue stations**
  - similar to a reservation station
  - holds instruction operation until ready for retirement
  - the instruction operation "issues" from this structure to an available 
    function unit when needed
  - combines some of the functions of an issue window, RUU, and ROB into 
    a common structure
- **Function units**
  - generally the same as existing ones
  - returns result operands to the originating issue station rather than writing 
    results to an RUU or ROB
- **Architected register file**
  - not needed for renaming or speculative results
- **Familiar load-store queue and memory hierarchy**

### basic operation

- Instructions are decoded at fetch time and stored in fetch buffers
- Decoded instructions are dispatched to Issue Stations (IS)
- ISes contend with each other for a FU resource (waiting as needed)
- ISes send an operation along with its input operands to a FU when available
- The IS waits for the FU execution result
- Resulting operand returns to the originating IS
- IS forwards the result operand to other ISes in program-ordered future
- ISes who snarf new (different) input operands proceed to re-execute as 
  needed

### issue station

- Similar to reservation station
- Implements dynamic operand renaming (for registers and memory)

<table>
<thead>
<tr>
<th>instruction address</th>
<th>path</th>
<th>operand bus interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction OP</td>
<td>time-tag</td>
<td>src1</td>
</tr>
<tr>
<td>execution state</td>
<td>predicate</td>
<td>src2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dst1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dst2</td>
</tr>
</tbody>
</table>

- Two input and two output operands are shown (varies w/ ISA)

### operand state block

- Holds all information about one operand
- Includes necessary logic to snoop for updates

<table>
<thead>
<tr>
<th>address</th>
<th>type</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous value</td>
<td>time-tag</td>
<td>path</td>
</tr>
<tr>
<td>next value</td>
<td>sequence</td>
<td></td>
</tr>
</tbody>
</table>

- Operand names take the form -> type : path : time-tag : seq : addr
- Example name for a register -> "register : 1 : 27 : 3 : 6"
- Predicates are operands also but have additional state (not shown)
snoop/snarf operation

result operand forwarding bus

<table>
<thead>
<tr>
<th>tt</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO</td>
<td>LO</td>
</tr>
</tbody>
</table>

loaded by decode or by mem-addr calculation

load at dispatch

<table>
<thead>
<tr>
<th>instr.</th>
<th>time-tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>addr</th>
</tr>
</thead>
</table>

execute or re-execute

repeated for each operand per instruction

snoop/snarf operation

some additional IS state

- Acquiring input operands
- Execution is needed (waiting for FU availability)
- Executing (waiting for FU result to return)
- Executed at least once
- Result operand was requested by another IS
- Result operand needs to be forwarded
- Operand is being forwarded

- Most of these indications also prevent instruction commitment
- Retirement (squash) may still occur under certain conditions

issue station operation

- An instruction gets dispatched to an IS with (choices):
  - initial input operands from:
    - architected register file
    - from a value predictor
  - no initial input operands
- If input operands are available, arbitrate for FU resource, otherwise acquire input operands by requesting them
- After an execution result is available, "forward" the operand
- Continuously snoop for new input operands
- Initiate (arbitrate for FU resource) execution when a changed input operand arrives
- Respond to requests by other ISes for operands
- Track all in-progress conditions for commitment determination

example execution (registers)

code fragment example execution schedule

<table>
<thead>
<tr>
<th>label</th>
<th>TT</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>0</td>
<td>r3 &lt;= 1</td>
</tr>
<tr>
<td>i2</td>
<td>1</td>
<td>r4 &lt;= r3 + 1</td>
</tr>
<tr>
<td>i3</td>
<td>2</td>
<td>r3 &lt;= 2</td>
</tr>
<tr>
<td>i4</td>
<td>3</td>
<td>r5 &lt;= r3 + 2</td>
</tr>
</tbody>
</table>

example execution schedule

<table>
<thead>
<tr>
<th>cycle</th>
<th>execute</th>
<th>forward</th>
<th>snarf</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>i1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>i1[r3=1]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>i2[r3=1], i4[r3=1]</td>
</tr>
<tr>
<td>3</td>
<td>i2, i4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>i3</td>
<td>i2[r4=2], i4[r5=3]</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>i3[r3=2]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>i4[r3=2]</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>i4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>i4[r5=4]</td>
<td></td>
</tr>
</tbody>
</table>

- we want r3 to have value =2 after execution of i3
- we want r5 to have value =4 after execution of i4

- i4 executes in clock 3 after snarfing r3 from i3, resulting in wrong result for r5
- i4 executes again after snarfing r3 from i3, giving correct result for r5
summary

- Proposing a microarchitecture to explore OoO re-execution, but more conventional than our previous designs
  - binary program compatible to existing ISAs
- Will explore:
  - the nature and amount of re-execution that may (or may not) be desirable
  - different types and numbers of resources
  - various interconnection topologies, bus fabrics, and bandwidths
- Microarchitecture can be modified to support a number of hardware mechanisms:
  - various value prediction techniques
  - dynamic execution-time instruction predication
  - dynamically finding and executing control-independent instructions beyond branch joins