Lecture 14: Trace Scheduling, Conditional Execution, Speculation, Limits of ILP

Professor Alvin R. Lebeck
Computer Science 220
Fall 1999

Administrative

• HW #2 Due Friday
• Midterm I
  – Chapters 1-4
• Read Complexity Effective Processors
Review: Getting CPI < 1
Multiple Instructions/Cycle

- Two variations:
  - **Superscalar**: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
    - IBM PowerPC, Sun SuperSparc, DEC Alpha, HP 8000
  - **Very Long Instruction Words (VLIW)**: fixed number of instructions (16) scheduled by the compiler
    - Joint HP/Intel agreement in 1998?

---

Loop Unrolling in SuperScalar

<table>
<thead>
<tr>
<th>Integer instruction</th>
<th>FP instruction</th>
<th>Clock cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F0,0(R1)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>LD F6,-8(R1)</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>LD F10,-16(R1)</td>
<td>ADDD F4,F0,F2</td>
<td>3</td>
</tr>
<tr>
<td>LD F14,-24(R1)</td>
<td>ADDD F8,F6,F2</td>
<td>4</td>
</tr>
<tr>
<td>LD F18,-32(R1)</td>
<td>ADDD F12,F10,F2</td>
<td>5</td>
</tr>
<tr>
<td>SD 0(R1),F4</td>
<td>ADDD F16,F14,F2</td>
<td>6</td>
</tr>
<tr>
<td>SD -8(R1),F8</td>
<td>ADDD F20,F18,F2</td>
<td>7</td>
</tr>
<tr>
<td>SD -16(R1),F12</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>SD -24(R1),F16</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>SUBI R1,R1,#40</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>BNEZ R1,LOOP</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>SD -32(R1),F20</td>
<td></td>
<td>12</td>
</tr>
</tbody>
</table>

Unrolled 5 times to avoid delays (+1 due to SS)
12 clocks, or 2.4 clocks per iteration
### Loop Unrolling in VLIW

<table>
<thead>
<tr>
<th>Memory reference 1</th>
<th>Memory reference 2</th>
<th>FP operation 1</th>
<th>FP op. 2</th>
<th>Int. op/ branch</th>
<th>Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F0,0(R1)</td>
<td>LD F6,-8(R1)</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F10,-16(R1)</td>
<td>LD F14,-24(R1)</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F18,-32(R1)</td>
<td>LD F22,-40(R1)</td>
<td>ADDD F4,F0,F2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD F26,-48(R1)</td>
<td>ADDD F12,F10,F2</td>
<td>ADDD F8,F6,F2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD 0(R1),F4</td>
<td>SD -8(R1),F8</td>
<td>ADDD F20,F18,F2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SD -16(R1),F12</td>
<td>SD -24(R1),F16</td>
<td>ADDD F24,F22,F2</td>
<td>6</td>
<td>SUBI R1,R1,#48</td>
<td>8</td>
</tr>
<tr>
<td>SD -32(R1),F20</td>
<td>SD -40(R1),F24</td>
<td>ADDD F28,F26,F2</td>
<td>7</td>
<td>BNEZ R1,LOOP</td>
<td>9</td>
</tr>
</tbody>
</table>

Unrolled 7 times to avoid delays
7 results in 9 clocks, or 1.3 clocks per iteration
Need more registers in VLIW

### Limits to Multi-Issue Machines

- **Inherent limitations of ILP**
  - 1 branch in 5: How to keep a 5-way VLIW busy?
  - Latencies of units: many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent operations to keep machines busy

- **Difficulties in building HW**
  - Duplicate FUs to get parallel execution
  - Increase ports to Register File
    - VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Increase ports to memory
  - Decoding SS and impact on clock rate, pipeline depth
Limits to Multi-Issue Machines

• Limitations specific to either SS or VLIW implementation
  – Decode issue in SS
  – VLIW code size: unroll loops + wasted fields in VLIW
  – VLIW lock step => 1 hazard & all instructions stall
  – VLIW & binary compatibility is practical weakness

Software Pipelining Example

Before: Unrolled 3 times
1  LD  F0,0(R1)
2  ADDD F4,F0,F2
3  SD  0(R1),F4
4  LD  F6,-8(R1)
5  ADDD F8,F6,F2
6  SD  -8(R1),F8
7  LD  F10,-16(R1)
8  ADDD F12,F10,F2
9  SD  -16(R1),F12
10 SUBI R1,R1,#8
11 BNEZ R1,LOOP

After: Software Pipelined
1  SD  0(R1),F4 ; Stores M[i]
2  ADDD F4,F0,F2 ; Adds to M[i-1]
3  LD  F0,-16(R1); Loads M[i-2]
4  SUBI R1,R1,#8
5  BNEZ R1,LOOP

• Symbolic Loop Unrolling
  – Less code space
  – Fill & drain pipe only once
  vs. each iteration in loop unrolling
Review: Summary

• Branch Prediction
  – Branch History Table: 2 bits for loop accuracy
  – Correlation: Recently executed branches correlated with next branch
  – Branch Target Buffer: include branch address & prediction

• SuperScalar and VLIW
  – CPI < 1
  – Dynamic issue vs. Static issue
  – More instructions issue at same time, larger the penalty of hazards

• SW Pipelining
  – Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead

Trace Scheduling

• Parallelism across IF branches vs. LOOP branches

• Two steps:
  – Trace Selection
    » Find likely sequence of basic blocks (trace) of (statically predicted) long sequence of straight-line code
  – Trace Compaction
    » Squeeze trace into few VLIW instructions
    » Need bookkeeping code in case prediction is wrong
Trace Scheduling

Reorder these instructions to improve ILP

Fix-up instructions
In case we were wrong

HW support for More ILP

- **Avoid branch prediction by turning branches into conditionally executed instructions:**
  
  \[
  \text{if } (x) \text{ then } A = B \text{ op } C \text{ else NOP}
  \]
  
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr, IA-64 predicated execution.

- **Drawbacks to conditional instructions**
  
  - Still takes a clock even if “annulled”
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline
HW support for More ILP

- **Speculation**: allow an instruction to issue that is dependent on branch predicted to be taken without any consequences (including exceptions) if branch is not actually taken ("HW undo" squash)
- Often try to combine with dynamic scheduling
- Tomasulo: separate speculative bypassing of results from real bypassing of results
  - When instruction no longer speculative, write results (instruction commit)
  - execute out-of-order but commit in order

Speculation

- 4-way issue
- All of B2 issued speculatively
- Must be squashed
- Could execute B1 in speculative mode
HW support for More ILP

- Need HW buffer for results of uncommitted instructions: reorder buffer
  - Reorder buffer can be operand source
  - Once operand commits, result is found in register
  - 3 fields: instr. type, destination, value
  - Use reorder buffer number instead of reservation station
  - Instructions commit in order
  - As a result, it's easy to undo speculated instructions on mispredicted branches or on exceptions

Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination.

2. Execution—operate on operands (EX)
   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute

3. Write result—finish execution (WB)
   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. Commit—update register with reorder result
   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.
Limits to ILP

- Conflicting studies of amount of parallelism available in late 1980s and early 1990s. Different assumptions about:
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication

Initial HW Model here; MIPS compilers

1. Register renaming—infinitive virtual registers and all WAW & WAR hazards are avoided
2. Branch prediction—perfect; no mispredictions
3. Jump prediction—all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. Memory-address disambiguation—addresses are known & a store can be moved before a load provided addresses not equal

1 cycle latency for all instructions
Upper Limit to ILP
(Figure 4.38, page 319)

More Realistic HW: Branch Impact
(Figure 4.40, Page 323)
Selective Branch Predictor

- 8096 x 2 bits
- Taken/Not Taken
- 2048 x 4 x 2 bits
- Branch Addr
- Global History
- 2
- 00
- 01
- 10
- 11
- 8K x 2 bit Selector
- 11 Taken
- 10
- 01 Not Taken
- 00
- Choose Non-correlator
- Choose Correlator
- 11
- 10
- 01
- 00

Program

- gcc
epresso
- lx
- fppp
- doduc
- tomcatv
- 11
- 15
- 12
- 29
- 54
- 10
- 15
- 12
- 49
- 16
- 10
- 13 12
- 35
- 44
- 19 10 11
- 28
- Infinite 256 128 64 32 None

More Realistic HW: Register Impact

Figure 4.44, Page 328
More Realistic HW: Alias Impact

Figure 4.46, Page 330

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction, 256 renaming registers

Perfect Global/Stack perf; Inspect. None
heap conflicts Assem.

Realistic HW for '9X: Window Impact

(Figure 4.48, Page 332)

Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window

Infinite 256 128 64 32 16 8 4
Braniac vs. Speed Demon (Spec Ratio)

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe)
- vs. 2-scalar Alpha 21064 @ 200 MHz (7 stages)

Next Time

- Complexity Effective Processors
- HW #2 Due