18-447 Lecture 20: ILP to Multicores

James C. Hoe
Department of ECE
Carnegie Mellon University
Midterm 2 Class Distribution

Minimum: 6.0  Median: 47.5  Maximum: 70.0  Mean: 45.26  Std Dev: 14.41
# Midterm 2 Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>1:TLB</th>
<th>2:except</th>
<th>3:energy</th>
<th>4:poll</th>
<th>5:draw</th>
<th>6:trace</th>
<th>7:table</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>possible</td>
<td>6</td>
<td>8</td>
<td>12</td>
<td>8</td>
<td>19</td>
<td>12</td>
<td>14</td>
<td>70</td>
</tr>
<tr>
<td>average</td>
<td>4.8</td>
<td>3.3</td>
<td>5.8</td>
<td>4.6</td>
<td>8.2</td>
<td>9.3</td>
<td>9.3</td>
<td>45.3</td>
</tr>
<tr>
<td>stdev</td>
<td>1.8</td>
<td>2.8</td>
<td>4.1</td>
<td>3.0</td>
<td>2.1</td>
<td>3.3</td>
<td>5.6</td>
<td>14.4</td>
</tr>
<tr>
<td>max</td>
<td>6.0</td>
<td>8.0</td>
<td>12.0</td>
<td>8.0</td>
<td>10.0</td>
<td>12.0</td>
<td>14.0</td>
<td>70.0</td>
</tr>
<tr>
<td>median</td>
<td>6.0</td>
<td>4.0</td>
<td>7.0</td>
<td>5.5</td>
<td>8.0</td>
<td>11.0</td>
<td>14.0</td>
<td>47.5</td>
</tr>
<tr>
<td>min</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>6.0</td>
</tr>
</tbody>
</table>
Housekeeping

• Your goal today
  – transition from sequential to parallel
  – enjoy (only first part, before OOO, on 447 exam)

• Notices
  – get going on Lab 4, now 3 weeks left
  – Handout #17: HW6, due Monday 5/2 noon

• Readings (advanced optional)
  – MIPS R10K Superscalar Microprocessor, Yeager
  – Superscalar Club!!
Parallelism Defined

• $T_1$ (work measured in time):
  – time to do work with 1 PE

• $T_\infty$ (critical path):
  – time to do work with infinite PEs
  – $T_\infty$ bounded by dataflow dependence

• Average parallelism:

  $$ P_{\text{avg}} = \frac{T_1}{T_\infty} $$

• For a system with $p$ PEs

  $$ T_p \geq \max\{ \frac{T_1}{p}, T_\infty \} $$

• When $P_{\text{avg}} \gg p$

  $$ T_p \approx \frac{T_1}{p}, \text{aka "linear speedup"} $$

$$\begin{align*}
x &= a + b; \\
y &= b \times 2 \\
z &= (x - y) \times (x + y)
\end{align*}$$
**ILP: Instruction-Level Parallelism**

- Average $\text{ILP} = \frac{T_1}{T_\infty} = \frac{\text{no. instruction}}{\text{no. cyc required}}$

  - code1: $\text{ILP} = 1$
    - i.e., must execute serially
  - code2: $\text{ILP} = 3$
    - i.e., can execute at the same time

  code1:  
  - $r1 \leftarrow r2 + 1$
  - $r3 \leftarrow r1 / 17$
  - $r4 \leftarrow r0 - r3$

  code2:  
  - $r1 \leftarrow r2 + 1$
  - $r3 \leftarrow r9 / 17$
  - $r4 \leftarrow r0 - r10$
Superscalar Speculative Out-of-Order Execution
Exploiting **ILP** for Performance

Scalar in-order pipeline with forwarding

- operation latency (**OL**) = 1 base cycle
- peak **IPC** = 1  // no concurrency
- require **ILP** ≥ 1 to avoid stall
Superpipelined Execution

\( \text{OL} = M \text{ minor-cycle}; \) same as 1 base cycle

peak \( \text{IPC} = 1 \) per minor-cycle \hspace{1cm} // has concurrency though

required \( \text{ILP} \geq M \)

Achieving full performance requires always finding \( M \) “independent” instructions in a row
Superscalar (Inorder) Execution

**OL** = 1 base cycle

peak **IPC** = N

required **ILP** ≥ N

Achieving full performance requires finding N “independent” instructions on every cycle.
Lab 4: 2-way, In-order Superscalar

Pipe A
I-cache
Reg File Read
ALU
D-cache
Reg File Write

Pipe B
2 X fetch bandwidth
2 X read ports
2 X Logic
Can’t always double resources
2 X write ports

No!
Limitations of Inorder Pipeline

- Achieved **IPC** of inorder pipelines degrades rapidly as **NxM** approaches **ILP**
- Despite high concurrency potential, pipeline never full due to frequent dependency stalls!!
Out-of-Order Execution

- **ILP** is scope dependent

\[
\begin{align*}
\text{ILP}=1 & \quad \begin{cases}
  r1 & \leftarrow r2 + 1 \\
r3 & \leftarrow r1 / 17 \\
r4 & \leftarrow r0 - r3 \\
r11 & \leftarrow r12 + 1 \\
r13 & \leftarrow r19 / 17 \\
r14 & \leftarrow r0 - r20 \\
\end{cases} \\
\text{ILP}=2
\end{align*}
\]

Accessing **ILP**=2 requires not only (1) larger scheduling window but also (2) out-of-order execution
Superscalar Speculative Out-of-Order Execution
Instruction Micro-Dataflow

- Maintain a buffer of many pending instructions, a.k.a. reservation stations (RSs)
  - wait for functional unit to be free
  - wait for required input operands to be available
- Decouple execution order from who is first in line (program order)
  - select inst’s in RS whose operands are available
  - give preference to older instructions (heuristical)
- A completing instruction (producer) signals dependent instructions (consumer) of operand availability (producer-push resolution)
Tomasulo’s Algorithm [IBM 360/91, 1967]

• Dispatch an instruction to a **RS** slot after decode
  – decode received from RF either operand value or placeholder **RS-tag**
  – mark RF dest with **RS-tag** of current inst’s **RS** slot
• Inst in **RS** can issue when all operand values ready
• Completing instruction, in addition to updating RF dest, broadcast its **RS-tag** and value to all **RS** slots
• **RS** slot holding matching **RS-tag** placeholder pickup value
WAW and WAR

- No WAW and WAR before because
  - single write stage
  - write stage at the end (later than any read stage)
  - in-order progression in pipeline
Removing False Dependencies

• With out-of-order execution comes WAW and WAR hazards
• Anti and output dependencies are false dependencies on register names rather than data

\[
\begin{align*}
  r_3 & \leftarrow r_1 \text{ op } r_2 \\
  r_5 & \leftarrow r_3 \text{ op } r_4 \\
  r_3 & \leftarrow r_6 \text{ op } r_7
\end{align*}
\]

• With infinite number of registers, anti and output dependencies avoidable by using a new register for each new value
Register Renaming: Example

Original

\[
\begin{align*}
  r1 & \leftarrow r2 / r3 \\
  r4 & \leftarrow r1 * r5 \\
  r1 & \leftarrow r3 + r6 \\
  r3 & \leftarrow r1 - r5
\end{align*}
\]

Renamed

\[
\begin{align*}
  r1 & \leftarrow r2 / r3 \\
  r4 & \leftarrow r1 * r5 \\
  r8 & \leftarrow r3 + r6 \\
  r9 & \leftarrow r8 - r5
\end{align*}
\]
On-the-fly HW Register Renaming

- Maintain mapping from ISA reg. names to physical registers
- When decoding an instruction that updates ‘rₓ’:
  - allocate unused physical register tᵧ to hold inst result
  - set new mapping from ‘rₓ’ to tᵧ
  - younger instructions using ‘rₓ’ as input finds tᵧ
- De-allocate a physical register for reuse when it is never needed again?

r₁ ← r₂ / r₃
r₄ ← r₁ * r₅
r₁ ← r₃ + r₆

^^^^^^when is this exactly?
Superscalar Speculative
Out-of-Order Execution
Control Speculation

• For want of a large window of instructions
  – if 14% of avg. instruction mix is control flow, what is average distance between control flow?
  – instruction fetch must make multiple levels of branch predictions (condition and target) to fetch far ahead of execution and commit

• Modern CPUs can have over 100 instructions in out-of-order execution scope

• Question:
  – how much more ILP is uncovered with look ahead
  – how much useful work is done during look ahead

Ans: not much and not much
Speculative Out-of-order Execution

• A mispredicted branch after resolution must be rewound and restarted ASAP!

• Much trickier than 5-stage pipeline . . .
  – can rewind to an intermediate speculative state
  – a rewound branch could still be speculative and itself be discarded by another rewind!
  – rewind must reestablish both architectural state (register value) and microarchitecture state (e.g., rename table)
  – rewind/restart must be fast (not infrequent)

• Also need to rewind on exceptions . . . but easier
Nested Control Flow Speculation
Mis-speculation Recovery can be Speculative
Instruction Reorder Buffer (ROB)

• Program-order bookkeeping (circular buffer)
  – instructions enter and leave in program order
  – tracks 10s to 100s of in-flight instructions in different stages of execution

• Dynamic juggling of state and dependency
  – oldest finished instruction “commit” architectural state updates on exit
  – all ROB entries considered “speculative” due to potential for exceptions and mispredictions
In-order vs Speculative State

• In-order state:
  – cumulative architectural effects of all instructions committed in-order so far
  – can never be undone!!

• Speculative state, as viewed by a given inst in ROB
  – in-order state + effects of older inst’s in ROB
  – effects of some older inst’s may be pending

• Speculative state effects must be reversible
  – remember both in-order and speculative values for an RF register (may have multiple speculative values)
  – store inst updates memory only at commit time

• Discard younger speculative state to rewind execution to oldest remaining inst in ROB
You have seen this in your pipeline

<table>
<thead>
<tr>
<th></th>
<th>( t_0 )</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
<th>( t_6 )</th>
<th>( t_7 )</th>
<th>( t_8 )</th>
<th>( t_9 )</th>
<th>( t_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></td>
<td>( l_0 )</td>
<td>( l_1 )</td>
<td>( l_2 )</td>
<td>( l_3 )</td>
<td>( l_4 )</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>( l_h )</td>
<td>( l_{h+1} )</td>
<td>( l_{h+2} )</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>( l_0 )</td>
<td>( l_1 )</td>
<td>( l_2 )</td>
<td>( l_3 )</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>( l_h )</td>
<td>( l_{h+1} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td>( l_0 )</td>
<td>( l_1 )</td>
<td>( l_2 )</td>
<td>( l_3 )</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>( l_h )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td>( l_0 )</td>
<td>( l_1 )</td>
<td>( l_2 )</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>( l_0 )</td>
<td>( l_1 )</td>
<td>( l_2 )</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
</tr>
</tbody>
</table>

- Flush the pipeline at anytime to start from after last instruction committed
- Decode in ID needs to consider pending “speculative” state updates in EX/MEM/WB, on top of committed RF states
Superscalar Speculative OOO All Together

Read [Yeager 1996, IEEE Micro] if you are interested
Truth about Superscalar Speculative OOO

• If memory speed kept up with core speed, we would still be building in-order pipelines

• But, by 2005 we were seeing e.g., Intel P4 at 4+GHz

• Speculative OOO has really been about
  – finding independent work to do after cache hit&miss
  – getting to future cache misses as early as possible
  – overlapping multiple cache misses for BW

<table>
<thead>
<tr>
<th>Cache</th>
<th>Size</th>
<th>Access Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 D-cache</td>
<td>16KB</td>
<td>~4 cycles</td>
</tr>
<tr>
<td>L2 D-cache</td>
<td>1024KB</td>
<td>~18 cycles</td>
</tr>
<tr>
<td>Main memory</td>
<td>~50ns</td>
<td>~180 cycles</td>
</tr>
</tbody>
</table>

\[ t_1 = 4 \text{ cycle int (9 cycle fp)} \]
\[ t_2 = 18 \text{ cycle int (18 cycle fp)} \]
\[ t_3 = \sim 50\text{ns or 180 cycle} \]
At the 2005 Peak of Superscalar OOO

<table>
<thead>
<tr>
<th></th>
<th>Alpha 21364</th>
<th>AMD Opteron</th>
<th>Intel Xeon</th>
<th>IBM Power5</th>
<th>MIPS R14000</th>
<th>Intel Itanium2</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock (GHz)</td>
<td>1.30</td>
<td>2.4</td>
<td><strong>3.6</strong></td>
<td>1.9</td>
<td>0.6</td>
<td>1.6</td>
</tr>
<tr>
<td>issue rate</td>
<td>4</td>
<td>3 (x86)</td>
<td>3 (rop)</td>
<td><strong>8</strong></td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>pipeline int/fp</td>
<td>7/9</td>
<td>9/11</td>
<td><strong>22/24</strong></td>
<td>12/17</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>inst in flight</td>
<td>80</td>
<td>72 (rop)</td>
<td>126 rop</td>
<td><strong>200</strong></td>
<td>48</td>
<td>inorder</td>
</tr>
<tr>
<td>rename reg</td>
<td>48+41</td>
<td>36+36</td>
<td>128</td>
<td>48/40</td>
<td>32/32</td>
<td>328</td>
</tr>
<tr>
<td>transistor (10^6)</td>
<td>135</td>
<td>106</td>
<td>125</td>
<td>276</td>
<td>7.2</td>
<td>592</td>
</tr>
<tr>
<td>power (W)</td>
<td><strong>155</strong></td>
<td>86</td>
<td>103</td>
<td>120</td>
<td>16</td>
<td>130</td>
</tr>
<tr>
<td>SPECint 2000</td>
<td>904</td>
<td>1,566</td>
<td>1,521</td>
<td>1,398</td>
<td>483</td>
<td><strong>1,590</strong></td>
</tr>
<tr>
<td>SPECfp 2000</td>
<td>1279</td>
<td>1,591</td>
<td>1,504</td>
<td>2,576</td>
<td>499</td>
<td><strong>2,712</strong></td>
</tr>
</tbody>
</table>
## At peak minus 5 years

<table>
<thead>
<tr>
<th></th>
<th>Alpha 21264</th>
<th>AMD Athlon</th>
<th>Intel P4</th>
<th>MIPS R12000</th>
<th>IBM Power3</th>
<th>HP PA8600</th>
<th>SUN Ultra3</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock (MHz)</td>
<td>833</td>
<td>1200</td>
<td><strong>1500</strong></td>
<td>400</td>
<td>450</td>
<td>552</td>
<td>900</td>
</tr>
<tr>
<td>issue rate</td>
<td>4</td>
<td>3 (x86)</td>
<td>3 (rop)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>pipeline int/fp</td>
<td>7/9</td>
<td>9/11</td>
<td><strong>22/24</strong></td>
<td>6</td>
<td>7/8</td>
<td>7/9</td>
<td>14/15</td>
</tr>
<tr>
<td>inst in flight</td>
<td>80</td>
<td>72(rop)</td>
<td><strong>126</strong> rop</td>
<td>48</td>
<td>32</td>
<td>56</td>
<td>inorder</td>
</tr>
<tr>
<td>rename reg</td>
<td>48+41</td>
<td>36+36</td>
<td><strong>128</strong></td>
<td>32+32</td>
<td>16+24</td>
<td>56</td>
<td>inorder</td>
</tr>
<tr>
<td>transistor (10^6)</td>
<td>15.4</td>
<td>37</td>
<td>42</td>
<td>7.2</td>
<td>23</td>
<td>130</td>
<td>29</td>
</tr>
<tr>
<td>power (W)</td>
<td>75</td>
<td><strong>76</strong></td>
<td>55</td>
<td>25</td>
<td>36</td>
<td>60</td>
<td>65</td>
</tr>
<tr>
<td>SPECint 2000</td>
<td>518</td>
<td>524</td>
<td>320</td>
<td>286</td>
<td>417</td>
<td>438</td>
<td></td>
</tr>
<tr>
<td>SPECfp 2000</td>
<td><strong>590</strong></td>
<td>304</td>
<td>549</td>
<td>319</td>
<td>356</td>
<td>400</td>
<td>427</td>
</tr>
</tbody>
</table>
Performance (In)efficiency

• To hit “expected” performance target
  – push frequency harder by deepening pipelines
  – used the 2x transistors to build more complicated microarchitectures so fast/deep pipelines don’t stall (i.e., caches, BP, superscalar, out-of-order)

• The consequence of performance inefficiency is

limit of economical cooling [ITRS]

Recall

[Borkar, IEEE Micro, July 1999]

2005, Intel P4 Tehas 150W
Efficiency of Parallel Processing

Better to replace 1 of this by 2 of these; Or N of these

Power ≈ Perf^{1.75}

[Energy per Instruction Trends in Intel® Microprocessors, Grochowski et al., 2006]
## At peak plus 1 year

<table>
<thead>
<tr>
<th></th>
<th>AMD 285</th>
<th>Intel 5160</th>
<th>Intel 965</th>
<th>Intel Itanium2</th>
<th>IBM P5+</th>
<th>MIPS R16000</th>
<th>SUN Ultra4</th>
</tr>
</thead>
<tbody>
<tr>
<td>cores/threads</td>
<td>2x1</td>
<td>2x2</td>
<td>2x2</td>
<td>2x2</td>
<td>2x2</td>
<td>1x1</td>
<td>2x1</td>
</tr>
<tr>
<td>clock (GHz)</td>
<td>2.6</td>
<td>3.03</td>
<td>3.73</td>
<td>1.6</td>
<td>2.3</td>
<td>0.7</td>
<td>1.8</td>
</tr>
<tr>
<td>issue rate</td>
<td>3 (x86)</td>
<td>4 (rop)</td>
<td>3 (rop)</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>pipeline depth</td>
<td>11</td>
<td>14</td>
<td>31</td>
<td>8</td>
<td>17</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>inst in flight</td>
<td>72(rop)</td>
<td>96(rop)</td>
<td>126(rop)</td>
<td>inorder</td>
<td>200</td>
<td>48</td>
<td>inorder</td>
</tr>
<tr>
<td>on-chip$ (MB)</td>
<td>2x1</td>
<td>4</td>
<td>2x2</td>
<td>2x13</td>
<td>1.9</td>
<td>0.064</td>
<td>2</td>
</tr>
<tr>
<td>transistor (10^6)</td>
<td>233</td>
<td>291</td>
<td>376</td>
<td>1700</td>
<td>276</td>
<td>7.2</td>
<td>295</td>
</tr>
<tr>
<td>power (W)</td>
<td>95</td>
<td>80</td>
<td>130</td>
<td>104</td>
<td>100</td>
<td>17</td>
<td>90</td>
</tr>
<tr>
<td>SPECint 2000 per core</td>
<td>1942</td>
<td>(1556*)</td>
<td>1870</td>
<td>1474</td>
<td>1820</td>
<td>560</td>
<td>1300</td>
</tr>
<tr>
<td>SPECfp 2000 per core</td>
<td>2260</td>
<td>(1694+)</td>
<td>2232</td>
<td>3017</td>
<td>3369</td>
<td>580</td>
<td>1800</td>
</tr>
</tbody>
</table>

*3086/+2884 according to www.spec.org

Microprocessor Report, Aug 2006
## At peak plus 3 years

<table>
<thead>
<tr>
<th></th>
<th>AMD Opteron 8360SE</th>
<th>Intel Xeon X7460</th>
<th>Intel Itanium 9050</th>
<th>IBM P5</th>
<th>IBM P6</th>
<th>Fijitsu SPARC 7</th>
<th>SUN T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cores/threads</td>
<td>4x1</td>
<td><strong>6x1</strong></td>
<td>2x2</td>
<td>2x2</td>
<td>2x2</td>
<td>4x2</td>
<td><strong>8x8</strong></td>
</tr>
<tr>
<td>clock (GHz)</td>
<td>2.5</td>
<td>2.67</td>
<td>1.60</td>
<td>2.2</td>
<td>5</td>
<td>2.52</td>
<td>1.8</td>
</tr>
<tr>
<td>issue rate</td>
<td>3 (x86)</td>
<td>4 (rop)</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>pipeline depth</td>
<td>12/17</td>
<td>14</td>
<td>8</td>
<td>15</td>
<td>13</td>
<td>15</td>
<td>8/12</td>
</tr>
<tr>
<td>out-of-order</td>
<td>72(rop)</td>
<td>96(rop)</td>
<td>inorder</td>
<td>200</td>
<td>limited</td>
<td>64</td>
<td>inorder</td>
</tr>
<tr>
<td>on-chip$ (MB)</td>
<td>2+2</td>
<td><strong>9+16</strong></td>
<td>1+12</td>
<td>1.92</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>transistor (10^6)</td>
<td>463</td>
<td><strong>1900</strong></td>
<td>1720</td>
<td>276</td>
<td>790</td>
<td>600</td>
<td>503</td>
</tr>
<tr>
<td>power max(W)</td>
<td>105</td>
<td>130</td>
<td>104</td>
<td>100</td>
<td>&gt;100</td>
<td>135</td>
<td>95</td>
</tr>
<tr>
<td>SPECint 2006</td>
<td>14.4/170</td>
<td><strong>22/274</strong></td>
<td>14.5/1534</td>
<td>10.5/197</td>
<td>15.8/1837</td>
<td>10.5/2088</td>
<td>--/142</td>
</tr>
<tr>
<td>per-core/total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPECfp 2006</td>
<td>18.5/156</td>
<td>22/142</td>
<td>17.3/1671</td>
<td>12.9/229</td>
<td>20.1/1822</td>
<td><strong>25.0/1861</strong></td>
<td>--/111</td>
</tr>
<tr>
<td>per-core/total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Microprocessor Report, Oct 2008
On to Mainstream Parallelism in Multicores and Manycores

Remember, we got here because we need to compute faster while using less energy per operation.