Instruction Level Parallelism

Software View of Computer Architecture COMP9244
Godfrey van der Linden
2006-04-06

Introduction

• Definition of Instruction Level Parallelism (ILP)
• Pipelining
  – Hazards & Solutions
• Dynamic Scheduling
  – Hazards & Solutions
  – Limitations of ILP
  – Power Consumption
• History
• Conclusion
Definition of ILP

• Early processors would use more than one cycle to execute an instruction. Instruction per cycle (IPC) < 1
• Pipelines overlap instruction execution to achieve IPC = 1
• ILP can be defined as - IPC > 1

Background - Pipelining

• RISC style instruction set architectures allowed hardware designers to vastly simplify the implementation of CPUs
• For example the MIPS instruction set architectures uses a maximum 5 cycle to implement any instruction
• RISC was designed for ease of pipelining
Pipelining (cont)

• A pipeline overlaps instructions to complete one instruction per cycle

<table>
<thead>
<tr>
<th>i</th>
<th>IF</th>
<th>ID</th>
<th>EXE</th>
<th>MEM</th>
<th>RWB</th>
</tr>
</thead>
<tbody>
<tr>
<td>i+1</td>
<td>IF</td>
<td>ID</td>
<td>EXE</td>
<td>MEM</td>
<td>RWB</td>
</tr>
<tr>
<td>i+2</td>
<td>IF</td>
<td>ID</td>
<td>EXE</td>
<td>MEM</td>
<td>RWB</td>
</tr>
<tr>
<td>i+3</td>
<td>IF</td>
<td>ID</td>
<td>EXE</td>
<td>MEM</td>
<td>RWB</td>
</tr>
<tr>
<td>i+4</td>
<td>IF</td>
<td>ID</td>
<td>EXE</td>
<td>MEM</td>
<td>RWB</td>
</tr>
</tbody>
</table>

Pipelining - Hazards

• Pipelines are an easy extension to multi-cycle processor design, use the cross cycle latches to cross instruction phase
• There are some problems to be solved, aka pipeline hazards
  – Structural Hazard
  – Read after write(RAW) Hazard
  – Branch Hazard
Pipelining - Hazard Solutions

- Stall pipeline until hazard clears
- RAW Hazard - forwarding from one output phase direct to input phase
- Structural Hazard - duplicate units, eg. Split data & instruction caches
- Branch Hazard - predict a branch not taken and no-op & fetch if wrong

Instruction Level Parallelism (ILP)

- Higher performance means more instructions per second
- Implies higher clock rate or more instructions per clock cycle
- More instructions per cycle after a pipeline is IPC > 1, hence instructions need to complete in parallel
ILP (cont)

- Higher clock rates requires more, simpler & shallower pipeline stages
- However the simple pipeline hazard solutions are less efficacious
  - Branch mis-predicts require more stalls
  - RAW hazards need to be forwarded backwards in time
  - Structural units need to be split and reproduced for shallower phases

ILP Dependencies & Hazards

- True Data Dependencies
  - If inst j needs result of inst i, then j depends on i
- Name Data Dependencies
  - If j writes to reg a, and an earlier i reads a then order must be maintained (WAR)
  - If j writes to reg a, and an earlier i writes a then order must be maintained (WAW)
- Control Dependencies
  - Instructions streams are dependant on branch results
Dynamic Scheduling

- Technique to execute instructions as soon as dependencies are satisfied
- Scoreboarding tracks instructions on a functional unit scheduling execution when data dependencies are satisfied
- Tomasulo extends this with dynamic renaming of registers to clear name dependencies

Tomasulo Algorithm

- 3 Phase instruction execution
  - Issue - ID, if reservation station(RS) available schedule inst with current operands and dependencies
  - Execute - Operands available schedule instruction on functional unit
  - Write Results - On completion write results to register file and any RS that depends on it
Superscalar Processors

• Given a Tomasulo architecture can now add functional units to achieve even more parallelism
• Power4 has 2 Load/Store, 2 Fixed Point, 2 Floating Point, Branch and a CR unit

Branch Prediction

• 1 in 3 - 7 inst’s is a branch. Control dependency stalls destroy IPC
• Modern unit - tournament between local and global branch predictors
• Stack of targets and returned addresses
• Branch prediction, today’s typical unit achieves 95% accuracy
• OO virtual member functions present real problems to branch predictors without value prediction
Speculation

• Extend Tomasulo approach by adding a Reorder buffer (ROB)
  – Changes issue phase as ROB must be allocated
  – Adds commit phase, current head of the ROB FIFO is ‘committed’
  – On a committing failed branch, flush ROB
• Solves precise exception problem too!

Ideal CPU ILP Limits

• What is maximum ILP achievable on an ideal machine

<table>
<thead>
<tr>
<th>SPEC Benchmark</th>
<th>Average Inst. Issued</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>55</td>
</tr>
<tr>
<td>espresso</td>
<td>63</td>
</tr>
<tr>
<td>li</td>
<td>18</td>
</tr>
<tr>
<td>fpppp</td>
<td>75</td>
</tr>
<tr>
<td>doduc</td>
<td>119</td>
</tr>
<tr>
<td>tomcatv</td>
<td>150</td>
</tr>
</tbody>
</table>
Realisable CPU ILP Limits

<table>
<thead>
<tr>
<th>SPEC/</th>
<th>Inf.</th>
<th>256</th>
<th>128</th>
<th>64</th>
<th>32</th>
<th>16</th>
<th>8</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>espresso</td>
<td>15</td>
<td>15</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>li</td>
<td>12</td>
<td>12</td>
<td>11</td>
<td>11</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>fppp</td>
<td>52</td>
<td>47</td>
<td>35</td>
<td>22</td>
<td>14</td>
<td>8</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>dduc</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>tomcatv</td>
<td>56</td>
<td>45</td>
<td>14</td>
<td>22</td>
<td>14</td>
<td>9</td>
<td>6</td>
<td>3</td>
</tr>
</tbody>
</table>

Power Consumption

- [Li03] found IPC/Power were correlated for OS routines on a superscalar
- Using a cycle accurate power simulator of MIPS R10000, 50% of power used by data-path & pipeline structure
- The cite 3 other papers that collaborating their findings
History of ILP

- ’59 IBM 7030 “stretch” - Pipelining
- ’64 CDC 6600 dynamic scheduling using scoreboard
- ’67 IBM 360/91 for dynamic scheduling Tomasulo
- ’94/’95 1st gen superscalars: Pentium, AMD K6, MIPS 12000, PowerPC620
- End ‘90s 2nd gen: PIII, Athlon, Power4, Alpha 21264

Conclusions

- Modern superscalar speculative processors are extremely capable and complex
- Penalties for missed branches are large, OO language techniques are a problem, until IF phase can read registers
- Other penalties are also significant, static scheduling for a processor architecture of instruction stream is recommended
Conclusions (cont)

- Statically scheduling code for processors is very difficult, use compiler technology from chip manufacturer
- Software engineers in high level languages can not influence code scheduling, except with branch hints
- With profiling help an engineer can achieve some 5x performance improvement over generated code in statically scheduled assembler [Grey05]