ILP and Dynamic Execution #2: Branch Prediction, Multiple Issue

Prof. David A. Patterson

Review Tomasulo

- Reservations stations: *implicit register renaming* to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard
  - Allows loop unrolling in HW
- Not limited to basic blocks
  (integer units gets ahead, beyond branches)
- Today, helps cache misses as well
  - Don’t stall for L1 Data cache miss (insufficient ILP for L2 miss?)
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Pentium III; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264
Tomasulo Algorithm and Branch Prediction

• 360/91 predicted branches, but did not speculate: pipeline stopped until the branch was resolved
  - No speculation; only instructions that can complete
• Speculation with Reorder Buffer allows execution past branch, and then discard if branch fails
  - just need to hold instructions in buffer until branch can commit

Case for Branch Prediction when Issue N instructions per clock cycle

1. Branches will arrive up to $n$ times faster in an $n$-issue processor
2. Amdahl’s Law => relative impact of the control stalls will be larger with the lower potential CPI in an $n$-issue processor
7 Branch Prediction Schemes

1. 1-bit Branch-Prediction Buffer
2. 2-bit Branch-Prediction Buffer
3. Correlating Branch Prediction Buffer
4. Tournament Branch Predictor
5. Branch Target Buffer
6. Integrated Instruction Fetch Units
7. Return Address Predictors

Dynamic Branch Prediction

- Performance = $f(\text{accuracy, cost of misprediction})$
- Branch History Table: Lower bits of PC address index table of 1-bit values
  - Says whether or not branch taken last time
  - No address check (saves HW, but may not be right branch)
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 9 iterations before exit):
  - End of loop case, when it exits instead of looping as before
  - First time through loop on next time through code, when it predicts exit instead of looping
  - Only 80% accuracy even if loop 90% of the time
Dynamic Branch Prediction
(Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction twice: (Figure 3.7, p. 249)

- Red: stop, not taken
- Green: go, taken
- Adds hysteresis to decision making process

Correlating Branches

Idea: taken/not taken of recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
- Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction

- (2,2) predictor: 2-bit global, 2-bit local
**Accuracy of Different Schemes**

(Figure 3.15, p. 257)

4096 Entries 2-bit BHT
Unlimited Entries 2-bit BHT
1024 Entries (2,2) BHT

---

**Re-evaluating Correlation**

- Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:
  
<table>
<thead>
<tr>
<th>Program</th>
<th>Branch %</th>
<th>Static</th>
<th># = 90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>14%</td>
<td>236</td>
<td>13</td>
</tr>
<tr>
<td>eqntott</td>
<td>25%</td>
<td>494</td>
<td>5</td>
</tr>
<tr>
<td>gcc</td>
<td>15%</td>
<td>9531</td>
<td>2020</td>
</tr>
<tr>
<td>mpeg</td>
<td>10%</td>
<td>5598</td>
<td>532</td>
</tr>
<tr>
<td>real gcc</td>
<td>13%</td>
<td>17361</td>
<td>3214</td>
</tr>
</tbody>
</table>

- Real programs + OS more like gcc
- Small benefits beyond benchmarks for correlation? problems with branch aliases?
Predicated Execution

• Avoid branch prediction by turning branches into conditionally executed instructions:
  \[ \text{if (x) then } A = B \text{ op } C \text{ else NOP} \]
  - 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: 64 1-bit condition fields selected so conditional execution of any instruction
  - This transformation is called “if-conversion”

• 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

BHT Accuracy

• Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table

• 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%

• For SPEC92, 4096 about as good as infinite table
Tournament Predictors

• Motivation for correlating branch predictors is 2-bit predictor failed on important branches; by adding global information, performance improved
• Tournament predictors: use 2 predictors, 1 based on global information and 1 based on local information, and combine with a selector
• Hopes to select right predictor for right branch

Tournament Predictor in Alpha 21264

• 4K 2-bit counters to choose from among a global predictor and a local predictor
• Global predictor also has 4K entries and is indexed by the history of the last 12 branches; each entry in the global predictor is a standard 2-bit predictor
  - 12-bit pattern: ith bit 0 => ith prior branch not taken;
    ith bit 1 => ith prior branch taken;
• Local predictor consists of a 2-level predictor:
  - Top level a local history table consisting of 1024 10-bit entries; each 10-bit entry corresponds to the most recent 10 branch outcomes for the entry. 10-bit history allows patterns 10 branches to be discovered and predicted.
  - Next level Selected entry from the local history table is used to index a table of 1K entries consisting a 3-bit saturating counters, which provide the local prediction
• Total size: $4K \times 2 + 4K \times 2 + 1K \times 10 + 1K \times 3 = 29K$ bits!
  (~180,000 transistors)
% of predictions from local predictor in Tournament Prediction Scheme

Accuracy of Branch Prediction

* Profile: branch profile from last execution (static in that in encoded in instruction, but profile)
Accuracy v. Size (SPEC89)

<table>
<thead>
<tr>
<th>Total predictor size (Kbits)</th>
<th>0%</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
<th>5%</th>
<th>6%</th>
<th>7%</th>
<th>8%</th>
<th>9%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correlating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tournament</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need Address at Same Time as Prediction

- Branch Target Buffer (BTB): Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can’t use wrong branch address (Figure 3.19, p. 262)

Branch PC    Predicted PC

No: branch not predicted, proceed normally (Next PC = PC+4)
Yes: instruction is branch and use predicted PC as next PC
Extra prediction state bits
Special Case Return Addresses

• Register Indirect branch hard to predict address
• SPEC89 85% such branches for procedure return
• Since stack discipline for procedures, save return address in small buffer that acts like a stack: 8 to 16 entries has small miss rate

Pitfall: Sometimes bigger and dumber is better

• 21264 uses tournament predictor (29 Kbits)
• Earlier 21164 uses a simple 2-bit predictor with 2K entries (or a total of 4 Kbits)
• SPEC95 benchmarks, 21264 outperforms
  - 21264 avg. 11.5 mispredictions per 1000 instructions
  - 21164 avg. 16.5 mispredictions per 1000 instructions
• Reversed for transaction processing (TP)!
  - 21264 avg. 17 mispredictions per 1000 instructions
  - 21164 avg. 15 mispredictions per 1000 instructions
• TP code much larger & 21164 hold 2X branch predictions based on local behavior (2K vs. 1K local predictor in the 21264)
Dynamic Branch Prediction Summary

• Prediction becoming important part of scalar execution
• Branch History Table: 2 bits for loop accuracy
• Correlation: Recently executed branches correlated with next branch.
  - Either different branches
  - Or different executions of same branches
• Tournament Predictor: more resources to competitive solutions and pick between them
• Branch Target Buffer: include branch address & prediction
• Predicated Execution can reduce number of branches, number of mispredicted branches
• Return address stack for prediction of indirect jump

Getting CPI < 1: Issuing Multiple Instructions/Cycle

• Vector Processing: Explicit coding of independent loops as operations on large vectors of numbers
  - Multimedia instructions being added to many processors
• Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, Pentium III/4
• (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates (TBD)
  - Intel Architecture-64 (IA-64) 64-bit address
    » Renamed: “Explicitly Parallel Instruction Computer (EPIC)”
  - Will discuss in 2 lectures
• Anticipated success of multiple instructions lead to Instructions Per Clock_cycle (IPC) vs. CPI
**Getting CPI < 1: Issuing Multiple Instructions/Cycle**

- **Superscalar MIPS:** 2 instructions, 1 FP & 1 anything
  - Fetch 64-bits/clock cycle; Int on left, FP on right
  - Can only issue 2nd instruction if 1st instruction issues
  - More ports for FP registers to do FP load & FP op in a pair

<table>
<thead>
<tr>
<th>Type</th>
<th>Pipe Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int. instruction</td>
<td>IF ID EX MEM WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF ID EX MEM WB</td>
</tr>
<tr>
<td>Int. instruction</td>
<td>IF ID EX MEM WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF ID EX MEM WB</td>
</tr>
</tbody>
</table>

- 1 cycle load delay expands to 3 instructions in SS
  - instruction in right half can’t use it, nor instructions in next slot

**Multiple Issue Issues**

- **issue packet:** group of instructions from fetch unit that could potentially issue in 1 clock
  - If instruction causes structural hazard or a data hazard either due to earlier instruction in execution or to earlier instruction in issue packet, then instruction does not issue
  - 0 to N instruction issues per clock cycle, for N-issue

- Performing issue checks in 1 cycle could limit clock cycle time: \(O(n^2-n)\) comparisons
  - \(=>\) issue stage usually split and pipelined
  - 1st stage decides how many instructions from within this packet can issue, 2nd stage examines hazards among selected instructions and those already been issued
  - \(=>\) higher branch penalties \(=>\) prediction accuracy important
Multiple Issue Challenges

- While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:
  - Exactly 50% FP operations AND No hazards

- If more instructions issue at same time, greater difficulty of decode and issue:
  - Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue; (N-issue ~O(N^2-N) comparisons)
  - Register file: need 2x reads and 1x writes/cycle
  - Rename logic: must be able to rename same register multiple times in one cycle! For instance, consider 4-way issue:
    ```
    add r1, r2, r3  add p11, p4, p7
    sub r4, r1, r2  sub p22, p11, p4
    lw r1, 4(r4)     lw p23, 4(p22)
    add r5, r1, r2  add p12, p23, p4
    ⇒ sub p22, p11, p4
    lw r1, 4(r4)     lw p23, 4(p22)
    add r5, r1, r2  add p12, p23, p4
    ```
  - Result buses: Need to complete multiple instructions/cycle
    - So, need multiple buses with associated matching logic at every reservation station.
    - Or, need multiple forwarding paths

Dynamic Scheduling in Superscalar
The easy way

- How to issue two instructions and keep in-order instruction issue for Tomasulo?
  - Assume 1 integer + 1 floating point
  - 1 Tomasulo control for integer, 1 for floating point

- Issue 2X Clock Rate, so that issue remains in order

- Only loads/stores might cause dependency between integer and FP issue:
  - Replace load reservation station with a load queue; operands must be read in the order they are fetched
  - Load checks addresses in Store Queue to avoid RAW violation
  - Store checks addresses in Load Queue to avoid WAR,WAW
Register renaming, virtual registers versus Reorder Buffers

• Alternative to Reorder Buffer is a larger virtual set of registers and register renaming

• Virtual registers hold both architecturally visible registers + temporary values
  - replace functions of reorder buffer and reservation station

• Renaming process maps names of architectural registers to registers in virtual register set
  - Changing subset of virtual registers contains architecturally visible registers

• Simplifies instruction commit: mark register as no longer speculative, free register with old value

• Adds 40-80 extra registers: Alpha, Pentium,…
  - Size limits no. instructions in execution (used until commit)

How much to speculate?

• Speculation Pro: uncover events that would otherwise stall the pipeline (cache misses)

• Speculation Con: speculate costly if exceptional event occurs when speculation was incorrect

• Typical solution: speculation allows only low-cost exceptional events (1st-level cache miss)

• When expensive exceptional event occurs, (2nd-level cache miss or TLB miss) processor waits until the instruction causing event is no longer speculative before handling the event

• Assuming single branch per cycle: future may speculate across multiple branches!
Limits to ILP

- **Conflicting studies of amount**
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication

- **How much ILP is available using existing mechanisms with increasing HW budgets?**

- **Do we need to invent new HW/SW mechanisms to keep on processor performance curve?**
  - Intel MMX, SSE (Streaming SIMD Extensions): 64 bit ints
  - Intel SSE2: 128 bit, including 2 64-bit Fl. Pt. per clock
  - Motorola AltaVec: 128 bit ints and FPs
  - Supersparc Multimedia ops, etc.

---

Limits to ILP

Initial HW Model here; MIPS compilers.
Assumptions for ideal/perfect machine to start:

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

Also:
- unlimited number of instructions issued/clock cycle;
- perfect caches;
- 1 cycle latency for all instructions (FP *,/);
Upper Limit to ILP: Ideal Machine
(Figure 3.34, page 294)

Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>Integer: 18 - 60</th>
<th>FP: 75 - 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>54.8</td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>62.6</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>17.9</td>
<td></td>
</tr>
<tr>
<td>fpppp</td>
<td>75.2</td>
<td></td>
</tr>
<tr>
<td>doducd</td>
<td>118.7</td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>150.1</td>
<td></td>
</tr>
</tbody>
</table>

More Realistic HW: Branch Impact
(Figure 3.38, Page 300)

Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

<table>
<thead>
<tr>
<th>Program</th>
<th>Integer: 6 - 12</th>
<th>FP: 15 - 45</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>li</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>fpppp</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>doducd</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>tomcatv</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
More Realistic HW: Renaming Register Impact

Figure 3.41, Page 304

FP: 11 - 45

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

Integer: 5 - 15

Program

Infinite 256 128 64 32 None

More Realistic HW: Memory Address Alias Impact

Figure 3.43, Page 306

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

FP: 4 - 45 (Fortran, no heap)

Integer: 4 - 9

Program

Perfect Global/Stack perf; Inspec. heap conflicts None Assem.
Realistic HW for ’00: Window Impact
(Figure 3.45, Page 309)

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

Integer: 6 - 12
FP: 8 - 45

How to Exceed ILP Limits of this study?

- WAR and WAW hazards through memory: eliminated WAW and WAR hazards through register renaming, but not in memory usage
- Unnecessary dependences (compiler not unrolling loops so iteration variable dependence)
- Overcoming the data flow limit: value prediction, predicting values and speculating on prediction
  - Address value prediction and speculation predicts addresses and speculates by reordering loads and stores; could provide better aliasing analysis, only need predict if addresses =
Workstation Microprocessors 3/2001

| Processor | AMD Athlon | HP PA-8600 | IBM Power-II | Intel Pentium III | Intel Pentium 4 | MIPS R12000 | Sun Ultra-2 | ULTRA
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Rate (MHz)</td>
<td>750/1.2GHz</td>
<td>1.1GHz</td>
<td>900MHz</td>
<td>1.1GHz</td>
<td>1.4GHz</td>
<td>1.2GHz</td>
<td>900MHz</td>
<td>900MHz</td>
</tr>
<tr>
<td>Cache (I/C/L2)</td>
<td>52KB/52/256KB</td>
<td>128KB/256KB</td>
<td>1MB</td>
<td>256KB</td>
<td>512KB</td>
<td>64/32KB</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Issue Rate</td>
<td>4 issue</td>
<td>3 x 86 insr</td>
<td>4 issue</td>
<td>3 x 86 insr</td>
<td>3 x 80 insr</td>
<td>4 issue</td>
<td>3 x 80 insr</td>
<td>3 x 80 insr</td>
</tr>
<tr>
<td>Pipeline Stages</td>
<td>7/9 stages</td>
<td>7/13 stages</td>
<td>9/7 stages</td>
<td>7/12 stages</td>
<td>12/14 stages</td>
<td>22/24 stages</td>
<td>6 stages</td>
<td>6/9 stages</td>
</tr>
<tr>
<td>Out of Order Entries</td>
<td>80 instr</td>
<td>72/83K</td>
<td>56 instr</td>
<td>32 instr</td>
<td>40 ROPOs</td>
<td>128 ROPOs</td>
<td>48 instr</td>
<td>None</td>
</tr>
<tr>
<td>Rename Registers</td>
<td>48/48</td>
<td>36/36</td>
<td>56 total</td>
<td>61/24 fp</td>
<td>40 total</td>
<td>128 total</td>
<td>32/32</td>
<td>None</td>
</tr>
<tr>
<td>L1B Entries</td>
<td>48/32/24</td>
<td>32/24</td>
<td>25 1/0</td>
<td>32/12</td>
<td>48/32</td>
<td>512/32/16</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>Memory I/O</td>
<td>2.066GB/s</td>
<td>2.1GB/s</td>
<td>1.5GB/s</td>
<td>1.6GB/s</td>
<td>1.6GB/s</td>
<td>3GB/s</td>
<td>1.5GB/s</td>
<td>1.5GB/s</td>
</tr>
<tr>
<td>Package</td>
<td>21x21mm</td>
<td>27x27mm</td>
<td>29x29mm</td>
<td>29x29mm</td>
<td>29x29mm</td>
<td>29x29mm</td>
<td>29x29mm</td>
<td>29x29mm</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>300MHz</td>
<td>350MHz</td>
<td>350MHz</td>
<td>350MHz</td>
<td>350MHz</td>
<td>350MHz</td>
<td>350MHz</td>
<td>350MHz</td>
</tr>
<tr>
<td>Max Issue</td>
<td>4 instructions (many CPUs)</td>
<td>Max rename registers: 128 (Pentium 4)</td>
<td>Max BHT: 4K x 9 (Alpha 21264B), 16Kx2 (Ultra III)</td>
<td>Max Window Size (OOO): 126 instructions (Pentium 4)</td>
<td>Max Pipeline: 22/24 stages (Pentium 4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPEC 2000 Performance</td>
<td>Alpha 21264B</td>
<td>3.8X</td>
<td>1.2X</td>
<td>1.3X</td>
<td>1.2X</td>
<td>1.7X</td>
<td>1.5X</td>
<td>1.3X</td>
</tr>
</tbody>
</table>
| Source: Microprocessor Report, www.MPRonline.com

3/23/01

Source: Microprocessor Report, www.MPRonline.com
If time permits: "A Language for Describing Predictors and its Application to Automatic Synthesis", by Emer and Gloy

• What was dynamic branch mechanisms they looked at?
• How did they explore space?
• Did they improve upon current practice?
• How was did they choose between options?

Conclusion

• 1985-2000: 1000X performance
  - Moore’s Law transistors/chip => Moore’s Law for Performance/MPU
• Hennessy: industry been following a roadmap of ideas known in 1985 to exploit Instruction Level Parallelism and (real) Moore’s Law to get 1.55X/year
  - Caches, Pipelining, Superscalar, Branch Prediction, Out-of-order execution, …
• ILP limits: To make performance progress in future need to have explicit parallelism from programmer vs. implicit parallelism of ILP exploited by compiler, HW?
  - Otherwise drop to old rate of 1.3X per year?
  - Less than 1.3X because of processor-memory performance gap?
• Impact on you: if you care about performance, better think about explicitly parallel algorithms vs. rely on ILP?