Instruction-Level Parallelism and Its Exploitation (Part III)

ECE 154B

Dmitri Strukov
Dealing With Control Hazards

• Simplest solution to stall pipeline until branch is resolved and target address is calculated
  – would severely limit dynamical scheduling
  – Loop unrolling might help somewhat
Solution 2: Delayed Branches

• If the branch hardware has been moved to the ID stage, then we can eliminate all branch stalls with delayed branches which are defined as always executing the next sequential instruction after the branch instruction – the branch takes effect after that next instruction
  – MIPS compiler moves an instruction to immediately after the branch that is not affected by the branch (a safe instruction) thereby hiding the branch delay

- With deeper pipelines, the branch delay grows requiring more than one delay slot
  - Delayed branches have lost popularity compared to more expensive but more flexible (dynamic) hardware branch prediction
  - Growth in available transistors has made hardware branch prediction relatively cheaper
A. From before branch

\[ \text{add } \$1, \$2, \$3 \]
\[ \text{if } \$2 = 0 \text{ then} \]
\[ \text{delay slot} \]

becomes

\[ \text{if } \$2 = 0 \text{ then} \]
\[ \text{add } \$1, \$2, \$3 \]

B. From branch target

\[ \text{sub } \$4, \$5, \$6 \]
\[ \text{add } \$1, \$2, \$3 \]
\[ \text{if } \$1 = 0 \text{ then} \]
\[ \text{delay slot} \]

becomes

\[ \text{add } \$1, \$2, \$3 \]
\[ \text{if } \$1 = 0 \text{ then} \]
\[ \text{sub } \$4, \$5, \$6 \]

C. From fall through

\[ \text{add } \$1, \$2, \$3 \]
\[ \text{if } \$1 = 0 \text{ then} \]
\[ \text{delay slot} \]

becomes

\[ \text{add } \$1, \$2, \$3 \]
\[ \text{if } \$1 = 0 \text{ then} \]
\[ \text{sub } \$4, \$5, \$6 \]

\[ \text{sub } \$4, \$5, \$6 \]

- A is the best choice, fills delay slot and reduces IC
- In B and C, the sub instruction may need to be copied, increasing IC
- In B and C, must be okay to execute sub when branch fails
Solution 3: Static Prediction

• Branch condition prediction only makes sense if delay for calculation condition is longer than that of target address
• Predict branch is always taken or always not taken
  – Compiler can help by favoring each case
  – Need mechanism to back off (flush pipeline when prediction is wrong)
• Even better solution is via profiling
  – Run program to get branch statistics
  – Explicitly specify direction in instruction
  – Branches are typically bimodal
Dynamic Branch Prediction

• Change the outcome of prediction in the runtime

• Basic 1 and 2-bit predictors:
  – For each branch:
    • Predict taken or not taken
    • If the prediction is wrong two consecutive times, change prediction

• Correlating or two-level predictor:
  – Multiple 2-bit predictors for each branch
  – One for each possible combination of outcomes of preceding $n$ branches

• Local predictor:
  – Multiple 2-bit predictors for each branch
  – One for each possible combination of outcomes for the last $n$ occurrences of this branch

• Tournament predictor:
  – Combine correlating predictor with local predictor
Basic Dynamic Branch Prediction

• Branch History Table (BHT): Lower bits of PC address index table of 1-bit values
  – Says whether or not branch taken last time ( T-Taken, N )
  – No full address check (saves HW, but may be wrong)

• Problem: in a loop, 1-bit BHT will cause 2 mispredictions (avg is 10 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 if 10 iterations per loop on average
2-bit Branch Prediction - Scheme 1

• Better Solution: 2-bit scheme:

- **Red**: stop, not taken
- **Green**: go, taken

(Jim Smith, 1981)
Branch History Table (BHT)

• BHT is a table of “Predictors”
  – 2-bit, saturating counters indexed by PC address of Branch
• In Fetch phase of branch:
  – Predictor from BHT used to make prediction
• When branch completes:
  – Update corresponding Predictor
2-bit Branch Prediction - Scheme 2

Another Solution: 2-bit scheme where change prediction (in either direction) only if get misprediction *twice* :

![Diagram of 2-bit Branch Prediction Scheme 2]

- **Red**: stop, not taken
- **Green**: go, taken

Lee & A. Smith, IEEE Computer, Jan 1984

Both schemes achieve 80-95% accuracy with only a small difference in behavior
Branch Corelations

B1: if (x)
    ...

B2: if (y)
    ...
    z=x&&y

B3: if (z)
    ...

• B3 can be predicted with 100% accuracy based on the outcomes of B1 and B2

• Basic scheme cannot use this information
Correlating or Two-Level Branches

Idea: taken/not taken of recently executed branches is related to behavior of present 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

Branch address (4 bits)

2-bits per branch
local predictors

Prediction

2-bit recent global branch history
(01 = not taken then taken)
Accuracy of Different Schemes

4096 Entries 2-bit BHT
Unlimited Entries 2-bit BHT
1024 Entries (2,2) BHT
Global vs Local History Predictors

Global:
BHR: A shift register for global history
- Shift in latest result in each cycle
- Provides global context

Local:
BHT: Keeps track of local history
- Select entry based on PC bits & shift in latest result in each cycle
Tournament Predictors

• Motivation for correlating branch predictors: 2-bit local predictor failed on important branches; by adding global information, performance improved

• Tournament predictors: use two 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 (or right context of 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 is 0 => ith prior branch not taken;
    ith bit is 1 => ith prior branch taken;

![Diagram](image)
Tournament Predictor in Alpha 21264

- **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! (~180K transistors)
% of predictions from local predictor in Tournament Prediction Scheme

- **nasa7**: 98%
- **matrix300**: 100%
- **tomcatv**: 94%
- **doduc**: 90%
- **spice**: 55%
- **fpppp**: 76%
- **gcc**: 72%
- **espresso**: 63%
- **eqntott**: 37%
- **li**: 69%
Accuracy of Branch Prediction

- **Profile**: branch profile from last execution (static in that is encoded in instruction, but profile)

![Graph showing accuracy of branch prediction for various programs](chart)

- **tomcatv**: 100% (Profile-based), 99% (2-bit counter), 99% (Tournament)
- **doduc**: 84% (Profile-based), 82% (2-bit counter), 97% (Tournament)
- **fpppp**: 86% (Profile-based), 88% (2-bit counter), 98% (Tournament)
- **li**: 77% (Profile-based), 88% (2-bit counter), 98% (Tournament)
- **espresso**: 82% (Profile-based), 86% (2-bit counter), 96% (Tournament)
- **gcc**: 70% (Profile-based), 88% (2-bit counter), 94% (Tournament)

**Fig 3.40**
Accuracy v. Size (SPEC89)

Local - 2 bit counters

Correlating - (2,2) scheme

Tournament
Lab 3

Table 2: Branch predictor parameters

<table>
<thead>
<tr>
<th>predictor</th>
<th>l1_size</th>
<th>hist_size</th>
<th>l2_size</th>
<th>xor</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAg</td>
<td>1</td>
<td>W</td>
<td>$2^w$</td>
<td>0</td>
</tr>
<tr>
<td>GAP</td>
<td>1</td>
<td>W</td>
<td>$&gt;2^w$</td>
<td>0</td>
</tr>
<tr>
<td>PAg</td>
<td>N</td>
<td>W</td>
<td>$2^w$</td>
<td>0</td>
</tr>
<tr>
<td>PAp</td>
<td>N</td>
<td>W</td>
<td>$2^{N+w}$</td>
<td>0</td>
</tr>
<tr>
<td>gshare</td>
<td>1</td>
<td>W</td>
<td>$2^w$</td>
<td>1</td>
</tr>
</tbody>
</table>

- BHR: (G,P): {Global history, Per-address history}
- PHT: (g,p,s): {Global, Per-address, Set}
  - g: use the BHR output as the address into the PHT
  - p: combine the BHR output with some bits from the PC
  - s: use an arbitrary hashing function for PHT addressing
- 9 combinations: GAg, GAP, GAs, PAg, PAp, PAs, SAg, SAP and SAs
Branch Prediction for Function Returns

- **Problem**: Hard to predict target for returns (15% of all branches) because of indirection (60% accuracy)
- **In most languages, function calls are fully nested**
  - If you call A() → B() → C() → D()
  - Your return targets are PCc → PCb → PCa → PCmain
- **Solution**: Return Address Stack (RAS)
  - A FILO structure for capturing function return addresses
  - Operation: push on function call, pop on function return
  - 16-entry RAS is enough for the typical depth call trees
  - RAS mispredictions:
    - Overflows
    - Long jumps
Return Address Stack

16-entry → few percent of mispredictions
Need Address at the Same Time as Prediction

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

No: branch not predicted; proceed normally (PC+4)

Yes: instruction is branch; use predicted PC as next PC (if predict Taken)

Prediction state bits
Branch Target "Cache"

- Branch Target cache - Only predicted taken branches
- "Cache" - Content Addressable Memory (CAM) or Associative Memory (see figure)
- Use a big Branch History Table & a small Branch Target Cache
Steps with Branch target Buffer

for the 5-stage MIPS

- **Send PC to memory and branch-target buffer**
  - **Entry found in branch-target buffer?**
    - **Yes**
      - **Send out predicted PC**
    - **No**
      - **Is instruction a taken branch?**
        - **Yes**
          - **Taken Branch?**
            - **Yes**
              - Branch correctly predicted; continue execution with no stalls
            - **No**
              - Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
        - **No**
          - Normal instruction execution

- **Enter branch instruction address and next PC into branch-target buffer**
Branch Folding

Supply actual instruction instead of instruction address

• More time for fetch $\rightarrow$ bigger BTB
• or 0 delay branch (or branch folding)
Multiple Issue and Static Scheduling

• To achieve CPI < 1, need to complete multiple instructions per clock

• Solutions:
  – Statically scheduled superscalar processors
  – VLIW (very long instruction word) processors
  – dynamically scheduled superscalar processors
Acknowledgements

Some of the slides contain material developed and copyrighted by Csaba Moritz (U Amherst) and D. Culler (UCB) and instructor material for the textbook