Grade Distribution: Midterm 1

Number of Students
Branch Prediction
Key Points: Branch Prediction

• Control hazards occur when we don’t know what the next instruction is
• Mostly caused by branches
Normal operation

Cycles
There are two important parameters of the pipeline that determine the impact of branches on performance:

- Branch decode time -- how many cycles does it take to identify a branch
- Branch resolution time -- cycles until the real branch outcome is known
1. Branches take 19 cycles to resolve
2. Identifying a branch takes 4 cycles.
   1. The P4 fetches 3 instructions per cycle
   2. 126 instructions in flight at once
3. The CPU must keep fetching!!!

- Pentium 4 pipelines peaked at 31 stage!!!
- Current cpus have about 12-14 stages.
Branch Prediction

- Guess which way the branch will go.
- Keep fetching down that path
  - If you are right, great! The pipeline flows smoothly
  - If you are wrong, kill the wrong-path instructions and start fetching at the correct PC
- Turning off BP will reduce performance by at least 90%.
  - Branches are frequent, so stalling would be frequent.
Why are branches predictable?

• Loops
• Error checks
• Programmers write predictable code, because it’s less buggy
• User level predictability
• different branches based on same data
• Spin loops.
• When won’t they be?
  • OS

• Switch statement? (for large fanout/jump tables)
• Data-dependent branches
• if(rand()%2) {}

• What does “predictable” mean?
• predictable in a small amount of state
• Fast!!! << 10 cycles
• How confident are you?
Simple “static” Prediction

- “static” means before run time
- Many prediction schemes are possible
- Predict taken
  - Pros?
- Predict not-taken
  - Pros?
Simple “static” Prediction

• “static” means before run time
• Many prediction schemes are possible
• Predict taken
  • Pros?
• Predict not-taken
  • Pros?

Loops are commons
Simple “static” Prediction

• “static” means before run time
• Many prediction schemes are possible
• Predict taken
  • Pros?  
  Loops are commons
• Predict not-taken
  • Pros?  
  Not all branches are for loops.
Simple “static” Prediction

• “static” means before run time
• Many prediction schemes are possible
• Predict taken
  • Pros? Loops are commons
• Predict not-taken
  • Pros? Not all branches are for loops.

Backward Taken/Forward not taken
Best of both worlds.
A Study of Branch Prediction Strategies

James Smith
Context

- Jim was working on the Cyber 180 series at Control Data Corporation
- Pipelines were getting long in supercomputers
  - One instance (among very many) of SC problems showing up decades before “desktop” problems.
- They had simple, static prediction, but it was not enough
Predict that Branches Will Be taken

- Implementation?
  - Very simple
- Useful when?
  - Loops.
  - Hand-coded
- Variations
  - Not taken
  - Backward-taken, forward not-taken
  - Per-opcode direction.
  - Could add an extra bit in the ISA for taken/not-taken
Predict a branch will do what it did last time

- Implementation?
  - Space constrained. -- hash (cache but no tags)
  - In the icache? Why (not)?
    - # instructions in the icache limits the number of branches I can store predictions for
    - Decoupling BP from I$ lets me store more predictions without increasing icache size.
- Associative table of taken.
- Variations
  - four states strong/weak - taken/not-taken
- Useful when?
- “...about as good as can reasonably be expected”
  - What about “anomalous predictions”?
Backward Taken, Forward not taken

- Relationship to per-op-code static
Impact

• This paper is essentially the undergrad lecture I give about BP
• He’s inventing the wheel.
• Things don’t really get going until the 90s, when BP shows up in “desktop” machines.
• The “confidence” bit at the end shows up 15 years later (and this thursday)
An Analysis of Correlation and Predictability: What Makes Two Level Branch Predictors Work

Marius Evers, Sanjay Patel, Robert S. Chappel, Yale N. Patt
Context

- Branch prediction is clearly important
- A bunch of predictors have been designed
- They are going back to do “basic science” about correlation and predictability
- Should this have been done earlier? Could it have?
Predictability

• Direction correlation -- this branch goes the same direction as a previous branch (or it’s direction is a function of previous directions)
• Path correlation -- That we got here via a certain path predicts the outcome.
Identifying instances

- Execution instance
- Backward branch count
Limit Studies

• Do the impossible, to see how good the possible could be
Measuring Correlation

- Choose the “best” three branches
- Record the taken/not-taken/skipped history for those branches
  - Use that history to index into a table of counters.
- Compare to
  - Gshare -- xor PC and global history to index into a table of two-bit saturating counters.
Figure 4. Selective history vs. gshare and interference-free gshare
The Importance of History Length

- How long do you need to remember?
- “data” state vs “control” state

Figure 5. Accuracy as a function of history length using a 3-branch selective history