CMSC 411
Computer Systems Architecture
Lecture 9
Instruction Level Parallelism (cont.)

Alan Sussman
als@cs.umd.edu

Outline
• ILP
• Compiler techniques to increase ILP
• Loop Unrolling
• Static Branch Prediction
• Dynamic Branch Prediction
• Overcoming Data Hazards with Dynamic Scheduling
• Tomasulo Algorithm
• Conclusion

Administrivia
• HW #2, on pipelining, due today
• Finish reading Chapter 2 of H&P
• First exam scheduled for next Thursday, March 5
  – on Units 1-3
  – Wanli will be giving it

Correlated Branch Prediction
• Idea: record \( m \) most recently executed branches as taken or not taken, and use that pattern to select the proper \( n \)-bit branch history table

• In general, \((m,n)\) predictor means record last \( m \) branches to select between \( 2^m \) history tables, each with \( n \)-bit counters
  – Thus, old 2-bit BHT is a \((0,2)\) predictor

  – Global Branch History: \( m \)-bit shift register keeping T/NT status of last \( m \) branches.
    – Each entry in table has \( 2^m \) \( n \)-bit predictors.

```c
if (aa == 2) 
  aa = 0;
if (bb == 2) 
  bb = 0;
if (aa != bb) {
```
Correlating Branches

(2,2) predictor w/
- Behavior of recent branches selects between four predictions of next branch, updating just that prediction

Or, 4 addr bits + 2 history bits give us 6-bit index into $2^6 = 64$ predictors, each having two bits: 128 total bits.

Global branch history

Prediction

Calculations

- 4096-entry 2-bit BHT
  - $4k \times 2 = 8k$ bits
  - $4k = 2^{12} \Rightarrow 12$ address bits

- How to use the same # bits w/ a (2,2) predictor?
  - 8k bits w/ 2-bit BHT means 4k BHTs
  - the (2,2) implies an entry has four BHTs
  - $\Rightarrow$ 1k entries, i.e. a (2,2) predictor w/ 1024 entries

Tournament Predictors

• Multilevel branch predictor
• Use $n$-bit saturating counter to choose between predictors
• Usually choice between global and local predictors

Tournament Predictors

Tournament predictor using, say, 4K 2-bit counters indexed by local branch address. Chooses between:

• Global predictor
  - 4K entries indexed by history of last 12 branches ($2^{12} = 4K$)
  - Each entry is a standard 2-bit predictor

• Local predictor
  - Local history table: 1024 10-bit entries recording last 10 branches, index by branch address
  - The pattern of the last 10 occurrences of that particular branch used to index table of 1K entries with 3-bit saturating counters
Comparing Predictors (H&P Fig. 2.8)

- Advantage of tournament predictor is ability to select the right predictor for a particular branch
  - Particularly crucial for integer benchmarks.
  - A typical tournament predictor will select the global predictor almost 40% of the time for the SPEC integer benchmarks and less than 15% of the time for the SPEC FP benchmarks

Pentium 4 Misprediction Rate (per 1000 instructions, not per branch)

- 6% misprediction rate per branch SPECint (19% of INT instructions are branch)
- 2% misprediction rate per branch SPECfp (5% of FP instructions are branch)

Branch Target Buffers (BTB)

- Branch target calculation is costly and stalls the instruction fetch.
- BTB stores PCs the same way as caches
- The PC of a branch is sent to the BTB
- When a match is found the corresponding Predicted PC is returned
- If the branch was predicted taken, instruction fetch continues at the returned predicted PC

Branch Target Buffers

H&P Figure 2.22
Dynamic Branch Prediction Summary

- Prediction becoming important part of execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch
  - Either different branches (GA)
  - Or different executions of same branches (PA)
- Tournament predictors take insight to next level, by using multiple predictors
  - usually one based on global information and one based on local information, and combining them with a selector
  - In 2006, tournament predictors using ≈ 30K bits are in processors like the Power5 and Pentium 4
- Branch Target Buffer: include branch address & prediction

Outline

- ILP
- Compiler techniques to increase ILP
- Loop Unrolling
- Static Branch Prediction
- Dynamic Branch Prediction
- Overcoming Data Hazards with Dynamic Scheduling
- (Start) Tomasulo Algorithm
- Conclusion

Advantages of Dynamic Scheduling

- Dynamic scheduling - hardware rearranges the instruction execution to reduce stalls while maintaining data flow and exception behavior
- Handles cases when dependences unknown at compile time
  - it allows the processor to tolerate unpredictable delays such as cache misses, by executing other code while waiting for the miss to resolve
- Allows code that compiled for one pipeline to run efficiently on a different pipeline
- Simplifies the compiler
- Hardware speculation, a technique with significant performance advantages, builds on dynamic scheduling (later)

HW Schemes: Instruction Parallelism

- Key idea: Allow instructions behind stall to proceed
  - DIVD F0, F2, F4
  - ADDD F10, F0, F8
  - SUBD F12, F8, F14
- Enables out-of-order execution and allows out-of-order completion (e.g., SUBD)
  - In a dynamically scheduled pipeline, all instructions still pass through issue stage in order (in-order issue)
- Will distinguish when an instruction begins execution and when it completes execution; between 2 times, the instruction is in execution
- Note: Dynamic execution creates WAR and WAW hazards and makes exceptions harder
**Dynamic Scheduling Step 1**

- Simple pipeline had 1 stage to check both structural and data hazards: Instruction Decode (ID), also called Instruction Issue
- Split the ID pipe stage of simple 5-stage pipeline into 2 stages:
  - Issue—Decode instructions, check for structural hazards
  - Read operands—Wait until no data hazards, then read operands

**A Dynamic Algorithm: Tomasulo’s**

- For IBM 360/91 (before caches!)
  ⇒ Long memory latency
- Goal: High Performance without special compilers
- Small number of floating point registers (4 in 360) prevented interesting compiler scheduling of operations
  – This led Tomasulo to try to figure out how to get more effective registers — renaming in hardware!

- Why Study 1966 Computer?
  – The descendants of this have flourished!
    » Alpha 21264, Pentium 4, AMD Opteron, Power 5, …

**Tomasulo Algorithm**

- Control & buffers distributed with Function Units (FU)
  – FU buffers called “reservation stations”; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  – Renaming avoids WAR, WAW hazards
  – More reservation stations than registers, so can do optimizations compilers can’t
- Results to FU from RS, not through registers, over Common Data Bus that broadcasts results to all FUs
  – Avoids RAW hazards by executing an instruction only when its operands are available
- Load and Stores treated as FUs with RSs as well
- Integer instructions can go past branches (use branch prediction), also allow FP ops beyond basic block in FP queue

**Tomasulo Organization**

From H&P Figure 2.9

1. From Mem
2. Load Buffers
3. FP Op Queue
4. FP Registers
5. FP registers
6. Mult1
7. Mult2
8. Add1
9. Add2
10. Add3
11. Store Buffers
12. To Mem
13. Common Data Bus (CDB)