EEC 581 Computer Architecture

Lec 5 – Instruction Level Parallelism
(2.3 Branch prediction)

Chansu Yu
Electrical and Computer Engineering
Cleveland State University

Acknowledgement …

• Part of class notes are from
  – David Patterson
  – Electrical Engineering and Computer Sciences
  – University of California, Berkeley

  – http://www.eecs.berkeley.edu/~patterson
  – http://www-inst.eecs.berkeley.edu/~cs252
Outline

- ILP (2.1)
- Compiler techniques to increase ILP (2.1)
- Loop Unrolling (2.2)
- Static Branch Prediction (2.3)
- Dynamic Branch Prediction (2.3)
- Overcoming Data Hazards with Dynamic Scheduling (2.4)
- Tomasulo Algorithm (2.5)
- Conclusion

Overview of Chap. 2 & 3 (again)

- Pipelined architecture allows multiple instructions run in parallel (ILP)
- But, it has data and control hazard problems

- How can we avoid or alleviate the hazard problems in pipelined architecture?
Static Branch Prediction

- To reorder code around branches, need to predict branch statically when compile

- Simplest scheme is to predict a branch as taken
  - Average misprediction = untaken branch frequency = 34% SPEC

- More accurate scheme predicts branches using profile information collected from earlier runs, and modify prediction based on last run:

### Dynamic Branch Prediction

- Why does prediction work?
  - Underlying algorithm has regularities
  - Data that is being operated on has regularities
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems

- Is dynamic branch prediction better than static branch prediction?
  - Seems to be
  - There are a small number of important branches in programs which have dynamic behavior
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
- Problem: in a loop, 1-bit BHT will cause two 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

![Diagram of branch prediction]

There are 10 loops.
First branch: predict ???, Actually Taken
Second: predict Taken, Actually Taken
Third: Predict Taken, Actually Taken
...  
Tenth: Predict Taken, Actually Not-Taken
...
First: Predict Not-Taken, Actually Taken
Prediction accuracy = 80%
Dynamic Branch Prediction

• 1-bit predictor

Lower bits of PC

If actually Not-Taken

If actually Taken

BHT

BHT

BHT

Actually NT

Actually T

Actually T

Actually NT

Predict Taken
if predictor = 10 or 11

Predict Taken
Predict NT

Weakly T

Strongly T

Weakly NT

Strongly NT

(Fig. 2.4 ???)

Dynamic Branch Prediction

• 2-bit predictor

Lower bits of PC

If actually Not-Taken

If actually Taken

BHT

BHT

BHT

Predict Taken
if predictor = 10 or 11

Predict Taken
Predict NT

Weakly T

Strongly T

Weakly NT

Strongly NT

(Fig. 2.4 ???)
Dynamic Branch Prediction

• 2-bit predictor

![Diagram showing the 2-bit predictor and branch history table (BHT).]

- Instruction memory
- Instruction
- Prediction (NT or T) (00/01/11/10)
- IF/ID
- addi $1, $0, 10
- Loop: ...
- ...
- addi $1, $1, -1
- beq $1, $0, Loop

Prediction accuracy = ???

BHT Accuracy

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

• 4096 entry table:
Correlating Branches

Code example showing the potential

If (d==0)
    d=1;
If (d==1)
    ...

Assemble code

BNEZ R1, L1
DADDIU R1, R0, #1
L1: DADDIU R3, R1, #1
BNEZ R3, L2
L2:
...

Observation: if (d==0)-branch is taken, then (d==1)-branch is taken too

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

Correlated Branch Prediction

• Idea: Branches are correlated, sometimes
  – Utilize “m” most recently executed branches to help predictions
  – Maintain $2^m$ history tables and use a proper $n$-bit branch history table based on “m” most recent branches

• $(m,n)$ predictor
  – (1, 2) predictor: 2 history tables, each of which is a 2-bit predictor
  – (2, 2) predictor: 4 history tables, each of which is a 2-bit predictor
  – (12, 2) predictor: 4K history tables, each of which is a 2-bit predictor
Correlating Branches

- (1, 2) predictor

Lower bits of PC

01 10
T NT

If actually Not-Taken

If actually Taken

OR

1-bit Global branch history

Correlating Branches

- (2, 2) predictor

Lower bits of PC

01 10
T NT

If actually Not-Taken

If actually Taken

OR

2-bit Global branch history
Correlating Branches

- (12, 2) predictor ???

How many 2-bit predictors?

How many entries per predictor?

How about (12, 2) predictor with 1 entry per predictor?

12-bit Global branch history

Accuracy of Different Schemes

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

Frequency of Mispredictions

- 4096 entries: 2-bits per entry
- Unlimited entries: 2-bits/entry
- 1024 entries (2,2)
Tournament Predictors

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

Lower bits of PC

“Local predictor”

“Global predictor”

12-bit Global branch history

Selector

It requires two mispredictions before changing the preferred predictor
Tournament Predictors

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

- **Global predictor**
  - 4K entries index 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 (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 prediction

Calculate the target address:
(PC+4) or (PC+4+16-bit offset<<2)

Branch target calculation is costly and
stalls the instruction fetch.
Branch Target Buffers (BTB)

- 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 (BTB)

© 2003 Elsevier Science (USA). All rights reserved.
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 (2.1)
- Compiler techniques to increase ILP (2.1)
- Loop Unrolling (2.2)
- Static Branch Prediction (2.3)
- Dynamic Branch Prediction (2.3)
- Overcoming Data Hazards with Dynamic Scheduling (2.4)
- Tomasulo Algorithm (2.5)
- Conclusion