BRANCH PREDICTORS

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- Announcements
  - Homework 3 release: Sept. 25th

- This lecture
  - Dynamic branch prediction
  - Counter based branch predictor
  - Correlating branch predictor
  - Global vs. local branch predictors
Big Picture: Why Branch Prediction?

- **Problem:** performance is mainly limited by the number of instructions fetched per second
- **Solution:** deeper and wider frontend
- **Challenge:** handling branch instructions
Big Picture: How to Predict Branch?

- **Static prediction (based on direction or profile)**
  - Always not-taken
    - Target = next PC
  - Always taken
    - Target = unknown

- **Dynamic prediction**
  - Special hardware using PC
Recall: Dynamic Branch Prediction

- Hardware unit capable of learning at runtime
  1. Prediction logic
     - Direction (taken or not-taken)
     - Target address (where to fetch next)
  2. Outcome validation and training
     - Outcome is computed regardless of prediction
  3. Recovery from misprediction
     - Nullify the effect of instructions on the wrong path
Branch Prediction

- **Goal:** avoiding stall cycles caused by branches
- **Solution:** static or dynamic branch predictor
  - 1. prediction
  - 2. validation and training
  - 3. recovery from misprediction
- **Performance** is influenced by the frequency of branches \((b)\), prediction accuracy \((a)\), and misprediction cost \((c)\)
Goal: avoiding stall cycles caused by branches

Solution: static or dynamic branch predictor

1. prediction
2. validation and training
3. recovery from misprediction

Performance is influenced by the frequency of branches \((b)\), prediction accuracy \((a)\), and misprediction cost \((c)\)

\[
\text{Speedup} = \frac{\text{Old Time}}{\text{New Time}} = \frac{\text{CPI}_{\text{old}}}{\text{CPI}_{\text{new}}} = \frac{1 + bc}{1 + (1 - a)bc}
\]
A pipelined processor requires 3 stall cycles to compute the outcome of every branch before fetching next instruction; due to perfect forwarding/bypassing, no stall cycles are required for data/structural hazards; every 5th instruction is a branch.

Compute speedup gained by a branch predictor with 90% accuracy
Problem

- A pipelined processor requires 3 stall cycles to compute the outcome of every branch before fetching next instruction; due to perfect forwarding/bypassing, no stall cycles are required for data/structural hazards; every 5th instruction is a branch.
- Compute speedup gained by a branch predictor with 90% accuracy

\[
\text{Speedup} = \frac{1 + 0.2 \times 3}{1 + 0.1 \times 0.2 \times 3} = 1.5
\]
Bimodal Branch Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last branch
Bimodal Branch Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last branch

```plaintext
while(1) {
  for(i=0; i<10; i++) {
  }
}
for(j=0; j<20; j++) {
}
```
Bimodal Branch Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last branch

```c
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```
Bimodal Branch Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last branch

- Shared predictor

- Two mispredictions per loop

```c
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```
Bimodal Branch Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last branch

- Shared predictor

- Two mispredictions per loop

Accuracy = 26/30 = 0.86

How to improve?
Bimodal Branch Predictors

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

```c
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```
Bimodal Branch Predictors

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

```plaintext
while(1) {
    for(i=0; i<10; i++) {
    }
    for(j=0; j<20; j++) {
    }
}
```
Bimodal Branch Predictors

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken
- One misprediction on loop exit
- Accuracy = \( \frac{28}{30} = 0.93 \)
Bimodal Branch Predictors

- Two-bit branch predictor
  - Increment if taken
  - Decrement if untaken

- One misprediction on loop exit
- Accuracy = 28/30 = 0.93

- How to improve?
  - 3-bit predictor?

- Problem?
  - A single predictor shared among many branches

```
while(1) {
    for(i=0; i<10; i++) {
    }
}
```

```
for(j=0; j<20; j++) {
}
```
Using Multiple Counters

- How to assign a branch to each counter?

Program code

... branch-1...
... branch-2...
... branch-3...

PC

Counters
Using Multiple Counters

How to assign a branch to each counter?

Program code

...  
branch-1  
...  
branch-2  
...  
branch-3
Using Multiple Counters

- How to assign a branch to each counter?

Program code

... branch-1 ...
... branch-2 ...
... branch-3

1. How many branches are in a program?
2. How many counters are used?
Using Multiple Counters

- How to assign a branch to each counter?

1. How many branches are in a program?
2. How many counters are used?

Cost = n\(2^a\) bits

Program code

... branch-1...
... branch-2...
... branch-3...
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing

Program code

```
... branch-1 ...
... branch-2 ...
... branch-3
```

Least significant bits are used to select a counter
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing

Program code

```plaintext
... branch-1...
... branch-2...
... branch-3
```

Least significant bits are used to select a counter

(+): Reduced hardware
(-): Branch aliasing
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing

Cost = $n2^b$ bits

Program code

... branch-1 ...
... branch-2 ...
... branch-3

Least significant bits are used to select a counter

(+) Reduced hardware

(−) Branch aliasing
Using Multiple Counters

How to assign a branch to each counter?

- **Decode History Table (DHT)**
  - Reduced HW with aliasing

- **Branch History Table (BHT)**
  - Precisely tracking branches

Most significant bits are used as tags
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing
  - Branch History Table (BHT)
    - Precisely tracking branches

Most significant bits are used as tags
(+): No aliasing
(−): Missing entries
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing
  - Branch History Table (BHT)
    - Precisely tracking branches

Cost = \((a-b+n)2^b\) bits

Most significant bits are used as tags
(+) No aliasing
(−) Missing entries

hit/miss*
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing
  - Branch History Table (BHT)
    - Precisely tracking branches
  - Combined BHT and DHT
    - BHT is used on a hit
    - DHT is used/updated on a miss
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing
  - Branch History Table (BHT)
    - Precisely tracking branches
  - Combined BHT and DHT
    - BHT is used on a hit
    - DHT is used/updated on a miss

Cost = \((a-b+2n)2^b\) bits
Using Multiple Counters

- How to assign a branch to each counter?
  - Decode History Table (DHT)
    - Reduced HW with aliasing
  - Branch History Table (BHT)
    - Precisely tracking branches
  - Combined BHT and DHT
    - BHT is used on a hit
    - DHT is used/updated on a miss

Cost = \((a-b+2n)2^b\) bits

DHT typically has more entries than BHT
Correlating Branch Predictor

- Executed branches of a program stream may be correlated

```c
while (1) {
    if(x == 0)
        y = 0;
    ...
    if(y == 0)
        x = 1;
}
```
Correlating Branch Predictor

- Executed branches of a program stream may be correlated

```c
while (1) {
    if(x == 0) {
        y = 0;
        ...
    }
    if(y == 0) {
        x = 1;
    }
}
```

branch-1

branch-2
Correlating Branch Predictor

- Executed branches of a program stream may be correlated

```c
while (1) {
    if(x == 0)
        y = 0;
    ...
    if(y == 0)
        x = 1;
}
```

```
while:
    BNEQ R1, R0, skp1
    ADDI  R2, R0, #0
skp1: ...
    BNEQ R2, R0, skp2
    ADDI  R1, R0, #1
skp2:  J while
```
Executed branches of a program stream may be correlated

```c
while (1) {
    if(x == 0)
        y = 0;
    ...
    if(y == 0)
        x = 1;
}
```

Global History Register: an r-bit shift register that maintains outcome history

branch-1

branch-2

\( \text{taken?} \)
Correlating Branch Predictor

- Executed branches of a program stream may be correlated

```java
while (1) {
    if(x == 0)
        y = 0;
    ... 
    if(y == 0)
        x = 1;
}
```

Global History Register: an r-bit shift register that maintains outcome history.
Correlating Branch Predictor

- Executed branches of a program stream may be correlated

```c
while (1) {
    if(x == 0)
        y = 0;
    ...
    if(y == 0)
        x = 1;
}
```

Global History Register: an r-bit shift register that maintains outcome history

Cost = $r + n2^{b+r}$ bits
Global Branch Predictor

- GHR is merged with PC bits to choose a counter
Global Branch Predictor

- GHR is merged with PC bits to choose a counter

Diagram:
- PC
- XOR
- GHR
- Shared Counters
Global Branch Predictor

- GHR is merged with PC bits to choose a counter

\[ \text{Cost} = r + n2^{\text{MAX}\{b, r\}} \text{ bits} \]
Local Branch Predictor

- One GHR per branch
Local Branch Predictor

- One GHR per branch

\[ \text{Cost} = r2^b + n2^r \text{ bits} \]
Local Branch Predictor

- One GHR per branch
Local Branch Predictor

- One GHR per branch

Cost = \( r2^b + n2^{\text{MAX}\{r, b\}} \) bits
Local Branch Predictor

- One GHR per branch

Cost = \( r2^b + n2^{\text{MAX}\{r, b\}} \) bits
Local predictor may work well for some applications, while global predictor works well for some other programs. Include both and identify/use the best one for each branch.
Branch Prediction Summary

- Dedicated predictor per branch
  - Program counter is used for assigning predictors to branches

- Capturing correlation among branches
  - Shift register is used to track history

- Predicting branch direction is not enough
  - Which instruction to be fetched if taken?

- Storing the target instruction can eliminate fetching
  - Extra hardware is required
Branch Target Buffer

- Store tags and target addresses for each branch.
Branch Target Buffer

- Store tags and target addresses for each branch

V Tag Target

PC

Target address

Hit/miss*