Branch Prediction Schemes

ECE404: Advance Microprocessor System

Raj Parihar
Motivation

Why do we need (dynamic) branch predictors?

- Branches are very frequent
  - More than 20% INS are Branches
- Pipeline stall (bubble) is required to know the exact outcome / target of branch
  - Longer pipelines increase the latency; even worse!
- Use a naïve predictor (always Taken or Not Taken)
  - Approx 50% of the time correct results
- Branch penalty is too HIGH
  - If miss-predicted, Squash all the INS after branch
- That’s why we need really smart branch predictors
References

- “Alternative Implementations of Two-Level Adaptive Branch Prediction” by Tse-Yu Yeh and Yale N. Patt

- “Combining Branch Predictors” by Scott McFarling


- “Dynamic History-Length Fitting: A Third Level of Adaptivity for Branch Prediction” by Toni Juan et al

- “Neural Methods for Dynamic Branch Predictor” by Daniel A. Jimenez et al.
Outline

- Branch Prediction: Basics
- Various Branch Predictors
- BP in Commercial Design
- Enhancement Techniques
- Schemes to Reduce Interference
- Some Simulations and Results
- Conclusion
Conditional Branches

- High performance systems use multi-level branch predictors
- Two aspects of conditional branch prediction
  - Branch outcome: Taken or Not Taken
  - Branch Address: if Taken then to Where?
- What about unconditional branches?
  - Don’t even bother!
  - Compilers are smart enough to deal with them
- State-of-the-art gives approx ~ 98.8% hit rate
Basic Branch Prediction

- Bimodal Branch Prediction

- Local Branch Prediction Schemes
  - Per branch address

- Global Branch Prediction Schemes
  - Combined all branch addresses
Bi-model Branch Predictors

Pattern History Table

PC
13 bits
0000
8191

Reference: Combining Branch Predictions, Scott McFarling. MRL TN-36
Two-Level Branch Predictor

Global History Register (Table)

Pattern History Table
Two Level Branch Predictor (Cont…)

Global History Register (Table)

Pattern History Table

PC

13 bits

12

0

$2^{13} - 1$

$= 8191$
Combining Branch Predictors

- Meta Predictor → (2-level, Bi-model)

- GHT/GHR

- PHT

- PC

- 13 bits

- \(2^{13} - 1\)

- 12 bits

- 4095
Branch Predictors in Commercial Processors

POWER4, Alpha GS 21264 and Intel
POWER4: Core

- **Branch Prediction Unit**
- Instruction Fetch Unit
- Decode, Crack, Group
- Issue Queues
- LD/ST Queue
- Execution units
  - FP Execution units
  - Fixed Point EX units
  - BR Execution unit
  - CR Execution unit
POWER4: Branch Prediction Unit

- Three set of branch-history tables
  - Local predictor (Traditional BHT)
    - 16K entry, indexed by branch address, 1-bit prediction
  - Global predictor
    - 11-bit global history vector (Similar to GHR)
    - GHR is XORed with Branch address to index the HT
    - 16K entry global history table, 1-bit prediction
  - Selector Table
    - Keeps track of better predictor (global or local)
    - 16K entry global history table, 1-bit prediction
POWER 4 Branch Prediction (Cont…)

- Fetching is re-directed based on prediction
- Eventually branches are executed in BR unit
- Upon execution predictor tables are updated
- Dynamic branch prediction can be overdriven by software, if needed
- Link stack to predict the target of branches
- A target address of branch-to-count is often repetitive
Alpha 21264: BP

- Composed of 3 units
  - Local predictor
  - Global predictor
  - Choice predictors

- Local Predictor
  - 2-level, per-branch HT
  - 1K table entry, 3-bit SC
  - VPC [11:2] of current address
Alpha 21264 (Cont...)

- Global predictor
  - Uses 12-most recent br
  - 4K-entry global HT
  - 2-bit saturating counter

- Choice Predictor
  - Monitors the history of local & global predictors
  - 4K-entry table 2-bit each
  - Chooses the best of Two
Intel Processors

- 386/ 486
  - All branches are statically predicted Not Taken
- Pentium III
  - 2-level, local histories
  - 2-bit saturating counters (Lee-Smith)
- Pentium M
  - Combines 3 predictors
    - Bimodal, Global and Loop predictor
  - Loop predictor analyzes branches to see if they have loop behavior
    - Moving in one direction (taken or NT) a fixed number of times
Branch Prediction: Insights

What makes them *nearly* perfect?
Potential Interferences in Two-Level

- Interferences are caused by multiple branch instructions being mapped to the same table entry.

- Types of interference:
  - Neutral Interference
  - Positive Interference
  - Negative Interference

- **Negative** interference is more dominant than positive interference.

- **The Agree Predictor: A Mechanism for Reducing Negative Branch History Interference**, by Sprangle, Chappell, Alsup, and Patt, ISCA 97
Destructive Interference (Aliasing)

- Unrelated branches might accidentally use the same counter
  - If two branches behave differently, the predictor can’t learn the behavior
  - Leads to decreased accuracy

- Almost all known techniques change the microarchitecture
  - Techniques shown to work well in simulation
  - But microprocessor manufacturers still use relatively simple predictors

- Can we reduce destructive interference without changing the processor?
Ways to Reduce Interference

- Larger prediction table
  - Map conflicting branches to different table entries

- Efficient Hash Function
  - Use different mapping schemes to better distribute branches among different entries

- Profiling of branches
  - Separating different classes of branches to use different prediction tables

- Avoid negative interference
  - By converting negative interferences into neutral or positive interference
A bias bit is assigned to each branch in BTB.

PHT gives the info as "agree" or "disagree"
Pattern History Table: Utilization

- We need a very efficient HASH function

- We assume the utilization of PHT entry is uniform
  - Not true though, depends upon the efficiency of HASH function
  - XOR is the simplest HASH function
  - And here are some surprising results 😊
Simplescalar Implementation (Baseline)

L1 Table (GHR)

L2 Table (PHT)

L1 size - 1
AND
L1 index
Long
XOR
Memory Address
>> 2
BR Address

Baseline
Tweaking

Int

0000

8191
Simplescalar Tweak

- Change the `history_reg_size` in .cfg file

- Branch Lookup
  - bpred.c: `bpred_dir_lookup` function
  - XOR upper half of l2index with lower half before indexing L2 table which is kind of PHT

- Branch Update
  - Because L2index is a pointer it is automatically updates the correct entry once `bpred_update` function is called
PHT Utilization

Baseline  GHR = 13bit  
PHT = 8192 Entry  
GHR20  GHR = 20bit  
Simulation: 10 million INS  
BR+GHR  baddr is XORed  
Fast forward: 50 million INS

Number of Branches

PHT Entry [n]

GHR20  Baseline  BR+GHR
PHT Utilization: Art Benchmark

![Graph showing PHT Utilization](image)

- **BR_Baseline**
- **Miss_Baseline**
Impact of GHR Length

- In general, the length of GHR can impact the overall prediction accuracy.
- Longer GHR will affect more entries in PHT, but may reduce or enhance interference.
- Determining the appropriate GHR length is non trivial.
- To allow GHR to be changed dynamically is a possible way to improve performance.
- How to determine the best GHR length is an open research issue.
Variable GHR Length: Simulation

- GHR length VS # of Mispredictions

![Graph showing GHR length vs # of Mispredictions for various applications](image)
Variable GHR Length: Simulation

- GHR length VS % Improvement
Dynamic History Length

- Optimal Branch history length
  - Some prefer short history (less training time)
  - Some require longer history (complex behavior)

- Vary history length
  - Choose through profile/compile-time hints
  - Or learn dynamically

- References
Variable GHR and PHT Utilization

![Graph showing variable GHR and PHT utilization](image-url)
Changing the Branch Predictor

Before 2001, most work refined two-level adaptive branch prediction [Yeh & Patt 92]

- A 1st-level table records recent global or per-branch pattern histories
- A 2nd-level table learns correlations between histories and outcomes
- Refinements focus on reducing destructive interference

Some of the better refinements

- gshare [McFarling `93], agree [Sprangle et al. `97], hybrid predictors [Evers et al. `96], skewed predictors [Michaud et al. `93]
A Machine Learning Approach

- Conditional Branch Prediction is a Machine Learning Problem
  - The machine learns to predict conditional branches

- So why not apply a machine learning algorithm?

- Artificial neural networks
  - Simple model of neural networks in brain cells
  - Learn to recognize and classify patterns
Neuron Based Prediction

- Some of the well known techniques
  - Perceptron based predictors
  - Back propagation
  - Radial basis network
  - Elman network
  - Linear Vector Quantization (LVQ) network

- All are well known complex neural network based approaches

- Lot of computation and implementation overhead
  - Idea is to implement basic/lightweight footprint of above
Basic of Neuron Based Predictor

- The inputs to a neuron are branch outcome histories
  - The last $n$ branch outcomes
  - Can be global or local (per-branch) or both (alloyed)
  - Conceptually, branch outcomes are represented as
    - +1, for taken
    - -1, for not taken

- The output of the neuron is
  - Non-negative, if the branch is predicted taken
  - Negative, if the branch is predicted not taken

- Ideally, each static branch is allocated its own neuron
Perceptron Based Predictors

- Inputs \((x’s)\) are from branch history
- \(n + 1\) small integer weights \((w’s)\) learned by on-line training
- Output \((y)\) is dot product of \(x’s\) and \(w’s\); predict taken if \(y \geq 0\)
- Training finds correlations between history and outcome

\[
y = w_0 + \sum_{i=1}^{n} x_i w_i
\]
Conclusion

- Branch predictors make use of correlation between history of a branch, correlation with other branches and its outcome

- Neural prediction could be incorporated into future CPUs
  - Accuracy is very good; complexity is still a bottleneck
  - Power and energy need to be amortized

- Predictor accuracy is more important for deeper pipelines because the penalty increases with the depth of pipeline
Question!