Lecture 7 - Branch Prediction

Why predict branches?

- Pipelining
  - w/o prediction branches would cause stalls
- Why not?

What is a prediction?

- Address of next instructions (BTB)
- The prediction (a bit!)
- Confidence

When do we "update" our predictor?

- At the end
- Speculative update
  - Global history register
  - Counters?

What do we predict on?

- type of branch
- history of global branch outcomes
- address of the branch
- direction of the branch

- inputs to the branch
- feedback from previous branch outcomes
  - 2 bit saturating counters
  - history of outcomes for *that* branch
    - TTTNTTNNTNNT

Can we do better?

- Increase branch counter memory
  - Remove interference
- Profile-driven ISA hints
- Speculative threads to compute branch outcomes
- Get rid of the branches entirely, guess the outcome of computation
- Re-do genetic paper w/wider language
- Use program structure
- Not free:
  - More time, more power, more area, more design complexity

Your questions

- How would you combine predictors, identifying branches as one type or other?
- Static ('always take back branches' or something more complicated?) seems to win at least 50% of the time; is this good?
- - Non-choosing branch predictors?
- Did this case study of genetic programming influence later jump target prediction? (there was little previous research)
- Danger of overtraining the predictor to the programs used for fitness function?
- Today do predictor designs in commercial processors use the results of learning algorithms?
- What (approx.) are typical jump target, load/store address prediction, and other accuracies these days?
- Do processors continue to do more speculative execution in last 4 years?
- Do current cache sizes alleviate most of the issues with correlation?
- How does their oracle mechanism for selective history work? Is it possible to simulate such a device in hardware?
- Are modern branch predictors based more on PA or GA prediction?
- How well do the test cases actually reflect generic, common use, programs?
- How well do current genetic algorithms do generating branch predictors?
- Can this notation be modified to handle multiple simultaneous predictions?
- Is this notation or something similar used at present? Does it have significant use outside of providing a basis for creating genetic algorithms?
- How easy are the generated algorithms to implement in hardware?
- The Evers paper talks about predictors which are specialized to certain

Your questions

- Has anybody actually used this approach to build a real predictor?
- Dependence prediction is missing in the list of of the predictors
- De we use way-predictors?
- How do you decide on-the-fly which previous branches are the most important?
- In a hybrid predictor, how do you determine which of the two predictors to trust for a given branch?
- How much are we limited by space in processor to store all of this data during execution?
- Can this method come up with more complex concepts such as eliminating uncorrelated branches from history (similar to the idea from the Evers paper)?
- Does the random starting point affect the outcomes?
- They talk about using previous opcodes as one possible input to predictors, but did they do this in their trials?
- The experiments use a number of different benchmarks. What characteristics differentiate these benchmarks and how do these characteristics affect the type of branch predictor needed?
  - Are there other types of good branch predictors that are not mentioned in this paper?
- Did they (or anyone else) continue the genetic programming experiment with more operations and fundamental units?
  - If so, did they find that any additional types of operations or inputs have good effects? (i.e. Can we use other dynamic inputs other than