Applications of Machine Learning Techniques to Systems

Chi Ho Yue
Ilya Katsnelson

Outline

- Dynamic Branch Prediction with Perceptrons
  - Why Perceptrons
  - How does Perceptrons work
  - Linearly Separability
  - Performance
- Automated Predictors Synthesis
  - Predictor Notation
  - Use of Genetic Programming
  - Performance
- Questions and Discussion

Introduction

- Modern computers architecture increasingly rely on speculation to boost ILP
- Accurate prediction increases the performance benefit of speculation
- Recent effort to improve branch prediction focus on Aliasing to eliminate destructive interference
- Perceptrons target at improving the accuracy itself

Why Perceptrons

- Traditional dynamic branch predictor with Pattern History Table (PHT) hardware requirement increases exponentially with history length
- Perceptrons can use longer history for higher accuracy since its required hardware resource scale linearly with the history length
- Simple and easy to implement in hardware compared to other forms of neural network

How Perceptrons Work (1)

- A perceptron is assigned to every single static branch
- A perceptron consists of one artificial neuron connecting several input units by weighted edges to one output unit, as a function Y(x0, …,xn) of n inputs
- The xi represent the bits of a global branch history shift register

How Perceptrons Work (2)

- It uses a weight function Y to make branching decision

  Y = w0 + w1*x1 + w2*x2 + … + wn*xn
- The w's are weight in signed integers, assigned to each bits of history
- When an branch is taken, its corresponding xi = 1, when the branch is not taken, xi =- 1
- Branch is taken when Y is positive

Training Perceptrons

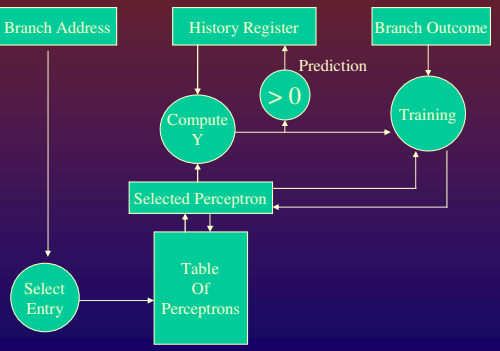
- A Perceptron is update every time a branch is executed

```
if (Real_taken != Y) or (Y < threshold)
    for i := 0 to n do
        wi := wi + (Y * xi)
    end for
end if
```


Training Concept

- When there is mostly agreement on xi (positive correlation), the weight (wi) becomes positively large
- When there is mostly disagreement (negatively correlation), the weight becomes negatively large
- There is a threshold to indicate that a perceptron is trained enough
- Take less training time than gshare, bimode, which are dynamic branch predictors using BHT

Perceptron Predictor Block Diagram



Step of using Perceptrons

1) Branch address is hashed to produce an index into the table of perceptrons
2) The corresponding perceptron is fetch into a register
3) Y is computed using weights and global history
4) Decided prediction using Y
5) Update the perceptron cell when actual output is known

Linear Separability

- Since Y is in integer, there is an problem when
  Y = sum(xi * wi) = 0 // a hyper-plane of xi's
- A function is linearly separable iff there exist values for w0..n such that all of the true instances can be separated from all of the false instances by hyper-plane
- An "XOR branch correlation" can confuse perceptrons because weights are being cancelled
- Hybrid with traditional dynamic branch prediction technique such as gshare to compensate this problem

Example of Linear Separability

Example of linearly in-separable correlation

```
if(cond_A){ ... ... }
if(cond_B){ ... ... }

// XOR cond_A and cond_B
cond_C = cond_A ^ cond_B;
if(cond_C){ ... ... }
```

Example of linearly separable correlation

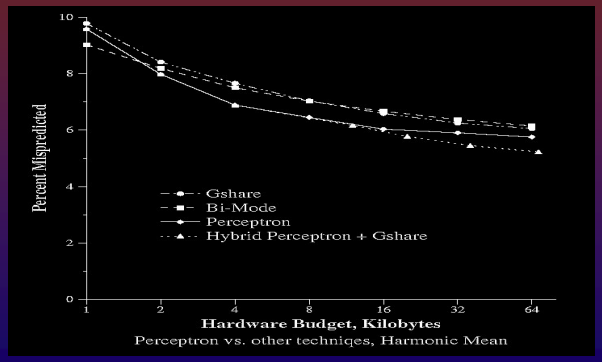
```
if(cond_A){ ... ... }
if(cond_B){ ... ... }

// OR cond_A and cond_B
cond_C = cond_A | cond_B;
if(cond_C){ ... ... }
```

Performance

- In comparison on SPEC 2000, with a 4K byte hardware budget, perception has a mis prediction rate of 6.89%, an improvement of 10.1% over gshare and 2.1% over bimode which use PHT method
- Ability to consider much longer history length than traditional schemes, helps b/c correlated branches can be a large distance apart

Performance Comparison



Critique

- Strength
  - Simple to implement and hardware cost efficient
  - Can make use of long branch history
- Weakness
  - Only accurate when branches are linear separable
  - Concentrate mostly on global history but less on local
  - Not sure if the prediction can be computed in a single cycle when history length becomes longer

Outline

- Dynamic Branch Prediction with Perceptrons
  - Why Perceptrons
  - How does Perceptrons work
  - Linearly Separability
  - Performance
- Automatic predictors synthesis
  - Predictor Notation
  - Use of Genetic Programming
  - Performance
- Questions and Discussion

Automated Prediction Synthesis

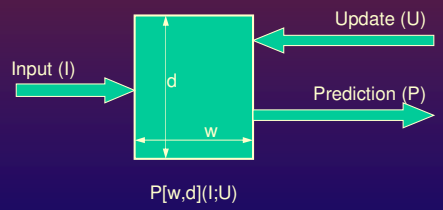
- The quest for more performance lead to increasing use of **speculative execution**
- Speculation requires some guessing or formally – prediction
  - Different architectural or implementation values (see previous presentations and EE282)

Current Predictor Design

- Build based on high-level constructs from earlier research
- Pros:
  - + Well understood working methods
  - + High speed with good hardware optimization
- Cons:
  - – Everything is the same – no new components
  - – Same designs might not apply for all predictors
    - i.e. Branch and data value predictors use different methods
- In order to find more efficient predictor constructs is it better to start from simple primitive predictor

Predictor Notation

- Define a primitive dynamic predictor construct:



BP Language

- Provides formal methods for describing predictors
  - Primitive predictor:
    P[w, d](I,U)
  - Array of n-bits saturating counters with up or down counting:
    Counter[n,d](i;T)=p[n,d](i;If T then P+1 else P-1)
- Specifications can be parsed to create simulations of predictors

Genetic Programming (1)

- Genetic algorithms can be applied to automatically synthesize new predictors:
  1. Create initial population of individuals
  2. Rank fitness of individuals in the population by simulation using BP parser
  3. Apply genetic operations to create new generation
  4. Repeat steps 2 and 3

Genetic Programming (2)

- Each individual is represented by a tree structure = BP expression of the predictor
- The nodes in the tree represent different logical parts of the predictor
  - Primitive predictors;
  - Functions;
  - Terminals

Genetic Programming (3)

- Genetic operations are used to create new generation:
  - Replication – survive the privileged
  - Crossover – randomly exchange the nodes
  - Mutation – randomly modify the nodes or subtrees
  - Encapsulation – randomly seal sets of components from crossover or mutation
- For each operation the input individuals are chosen based on their fitness value

Limit the Growth

- Starting point
- Constraints
  - No more then 512K of memory
  - New expression has to be legal BP expression
- Fitness
  - BP parser used to create simulators
  - Traces from real benchmarks used for the evaluation
  - Evaluation does not have to be accurate – only relative value of solutions is needed

Is This Method of Any Use?

- Many Jump and Branch predictors were generated
- Some amount of fine tuning was performed to get better improvements

Predictor	Mispredict Rate (SPEC)	Predictor	Mispredict Rate (SPEC)
TwoBit[256K]	13.1	GP1	9.5
Shared	6.7	GP2	9.7
Global Hist	7.9	GP3	7.2
Local Hist	7.9	GP4	7.0

Critique

- Too complex – "… even though in size these predictors are comparable to the human-designed predictors they are logically much more complex, and probably not directly implementable"
- Some useful ideas can be taken, but the implementation still requires manual tuning
- No guarantee of completion in fixed time
- The predictor has to be designed off-line before it can be used in hardware

Questions & Discussion (1)

- Perceptrons
  - Why is x0 is always set to 1?
  - What is the importance of the threshold in the training process?
  - Why does Perceptrons prediction require less training time than gshare and bimode which use BHT method?
  - How do Perceptrons take care of aliasing?

Questions & Discussion (2)

- Can we ever develop automated branch predictor generator?
- When is it a better choice to implement a genetic algorithm to generate branch predictors rather then to build a Perceptrons-based predictors or other BHT based predictors?
- What methods do ML mechanisms use to learn?

Questions & Discussion (3)

- Why does it make sense to use ML techniques in systems design?
- What types of problems are 'good' for ML methods?
- Any examples of ML in current computer architecture? Do they give us what we want?
- Which problem does genetic programming solve?
- … and how this all can be applied to CMP?

  Problems that could be solved, methods, trade-offs?
- Is there any alternatives to ML? Can they be effectively used in CMP architecture?

Reference

1) J. Emer, N. Gloy. A Language for Describing Predictors and its Applications to Automatic Synthesis. Proceedings of the 24th International Symposium on Computer Architecture, Denver, CO, June 1997.
2) D. Jimenez, C. Lin. Dynamic Branch Prediction with Perceptrons. Proceedings of the 7th International Symposium on High Performance Computer Architecture, Monterrey, Mexico, January 2001.
3) M. Stephenson, S. Amarasinghe, M. Martin, U. O'Reilly. Meta Optimization: Improving Compiler Heuristics with Machines Learning. Proceedings of the Conference on Programming Languages Design and Implementation, San Diego, CA, June 2003.
4) M. Sakr, D. Chiarulli, B. Horne, C. Giles. Predicting Multiprocessor Memory Access Patterns with Learning Models. Proceedings of the 4th International Conference on Machine Learning, Nashville, TN, July 1997.
5) S. McFarling, "Combining Branch Predictors", WRL Technical Note TN-36, Digital Equipment Corporation, June 1993.