CSE240A Project: Branch Predictor

Author: Yang, Chengguo PID: A69041855

Overview

This project is based on the 2004 Championship Branch Predictor contest. Each contestant will write a branch predictor that will consume a trace of branches (generated from real execution). In this project, I build 2 predictors based on the skeleton code. The first is a Tournament branch predictor based on the Alpha 21264 processor. The other is a perceptron branch predictor based on Dynamic Branch Prediction with Perceptrons. Hardware budget of both are within 64Kbits + 1024 bits.

Tournament Branch Predictor

Mechanism and Implementation

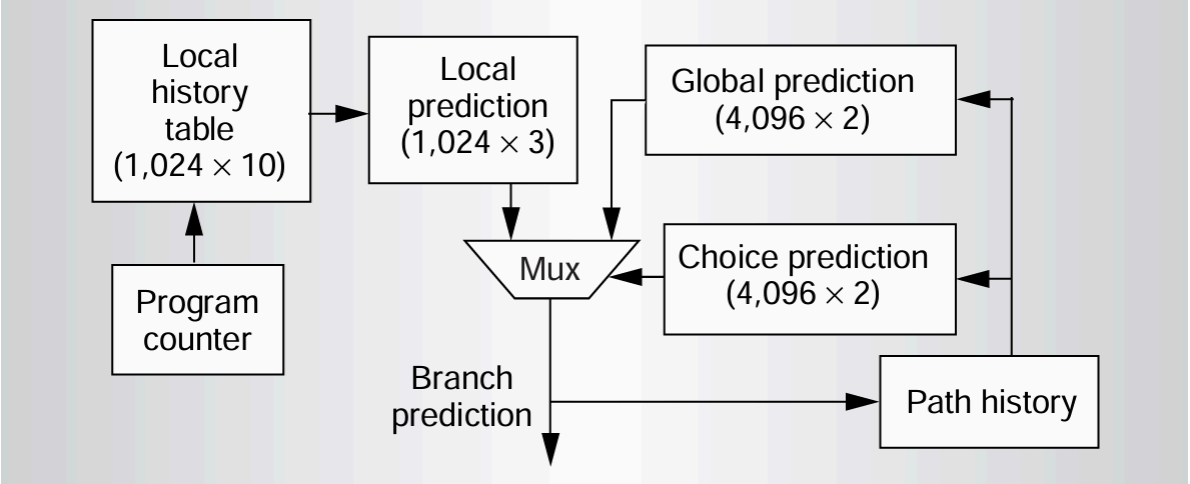


Figure 4. Block diagram of the 21264 tournament branch predictor. The local history prediction path is on the left; the global history prediction path and the chooser (choice prediction) are on the right.

The tournament branch predictor in Alpha 21264 contains 2 predictor module: a global predictor and a local predictor. A chooser will select one of them to predict. According to the paper, the main structure of it is as above.

When a branch occurs, the lower bit of PC is used to index in local history table, whose entries are historical pattern of the branch. The pattern then will be used to index in the local branch history table, whose entries are 3-bit saturating counter. The counter will deliver the local prediction. Meanwile, on the other side, path history which contains history of all branches will be used to index in both global branch history and chooser. Entries of chooser are 2-bit saturating counters, indicating which predictor's result to use.

On train stage, the counters in local and global BHT will increase or decrease by 1 depending on its prediction hit or not. Also, chooser will modify its bias depending on the predictor it chooses hit or not.

About the structure of tournament branch predictor, it mainly contains parts below:

- init_tournament(): Initialization function.
- train_tournament(pc, outcome): train function.
- predict_tournament(pc): do prediction.

Detail can be checked in *predictor.cpp* and *predictor.h*.

Hardware Budget Calc.

To get best performance within the budget, after tuning, the parameters are set as below.

```
int pcIndexBits = 11;  // Number of bits used for PC index
int lhistoryBits = 12; // Number of bits used for Local History
int pathHistoryBits = 13; // Number of bits used for Path History
int chooserBits = 12;  // Number of bits used for chooser table
```

With parameters above, the hardware budget is as below.

$$2^{11} \cdot 12 + 2^{12} \cdot 3 + 2^{13} \cdot 2 + 2^{12} \cdot 2 = 61440bit$$

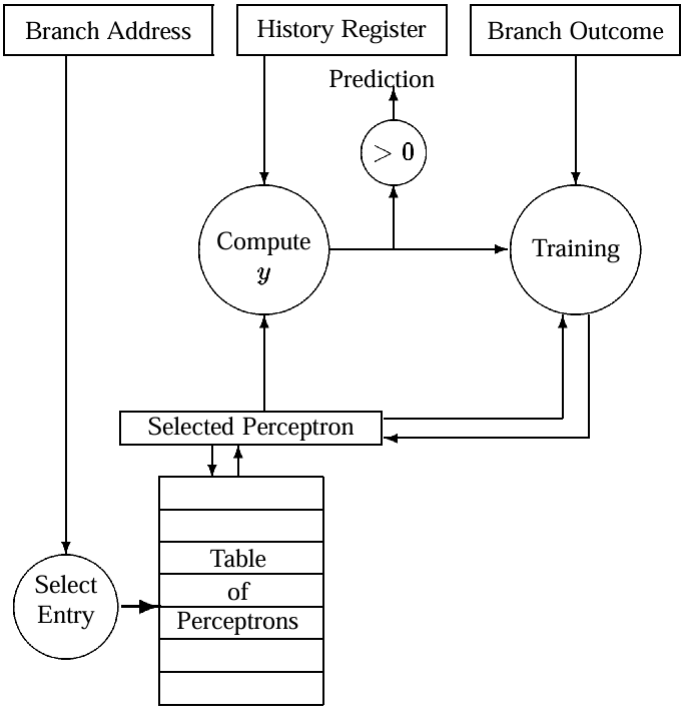
Performance

	Total	Mispredict
U1_Blender.bz2	10000000	292078
U2_Leela.bz2	10000000	957174
U3_GCC.bz2	10000000	159717
U4_Cam4.bz2	10000000	64155

So mispredict rate is 3.68281%

Perceptron Branch Predictor (Custom)

Mechanism and Implementation



For perceptron predictor, it uses the simplest idea of neuron network. One layer of perceptrons are set. Each contains a weight (in my code sometimes I call it bias as well, indicating the bias of perceptron). Additionally a constant bias term is set as well. Each time, the past global history (in my code I call it path history for consistency) is used as index of perceptron table, meaning that each pattern owns a corresponding perceptron. For each "history" in global history, 0 means Nottaken and results in a decrease of sum by weight, 1 means Taken and results in a increase of sum by weight. So weight can be regarded as an impact coefficient of a certain branch's history has on the branch to be predicted (positive weight means positive correlation). After adding the bias term as well, the final sum will be either positive or negative, indicating Taken or Nottaken.

On train stage, additionally, a THETA term is set, saying that if final sum's absolute value is under THETA which indicates an inconfident prediction, this prediction is considered as misprediction as well. Therefore, if the prediction made is wrong or inconfident, weight will be updated (for simplicity, learning rate is set as 1) within a certain range. Finally, global history will be updated as well.

The structure of tournament branch predictor mainly contains initialization, training and predictor part as well. Detail can be checked in *predictor.cpp* and *predictor.h*.

Hardware Budget Calc.

The set of parameters referring to the paper is as belows:

```
// perceptron parameters
#define PERCEPTRON_HISTORY 35    // history length
#define NUM_PERCEPTRONS 256      // number of perceptrons
#define WEIGHT_BITS 7            // weight bit width
#define THETA 80                 // training threshold
```

Consider the constant bias term, the hardware budget is:

$$(35+1)\cdot 256\cdot 7 = 64512bit$$

Performance

	Total	Mispredict
U1_Blender.bz2	10000000	292240
U2_Leela.bz2	10000000	891521
U3_GCC.bz2	10000000	122222
U4_Cam4.bz2	10000000	69113

So mispredict rate is 3.43774%

Conclusion

The performance of these 2 predictors shows that perceptron indeed can beat basic tournament predictor cleanly. But according to the paper, perceptron behaves badly when facing some situations such as context switching, therefore, if I later combine perceptron with gshare and a chooser just like what mentioned in the paper, the performance will be slightly better.