Michgan Tech

Branch Prediction Simulation

Project I

Writen by: liang YAN

Computer Science

Last modified: November 2013

Project Requirement

1. Develop a base class which implements a 2-bit saturating counter predictor. The class should accept the predictor size in bytes as a parameter.
2. Use the trace file posted on Canvas to test the predictor.
3. Create a derived class that implements the Gshare predictor. It should use the same number of global history bits as the index used to access the table. Use the trace file to test the predictor.
4. Create a global predictor which uses only the global history and combine it with the simple predictor using a selector predictor, creating a tournament predictor.
5. Assuming you have 2, 4, and 8 kilobytes of space for your predictors, compare the performance of the tournament predictor with that of gshare and simple single predictor.
6. Plot the prediction accuracy of simple 2 bit predictor which only uses PC, the gshare predictor with global bits equal to table index, global only predictor and the tournament predictor.
7. Write a brief report about how to use your predictors. In your report, it will be helpful to cover: the implementation process, the difficulties you have encountered, your fully com- mented C++ program and class descriptions, branch prediction measurements itemized above and a discussion of the results you have obtained.

0.1 Introduction

According to the requirement, we have to design four different predictors, and these predictors have different spaces(2Kbytes,4Kbytes,8Kbytes), meanwhile all these predictors use a 2 bit counter. At last, We need to measure these different predictor by going through a trace file which is composed by instructions.

0.2 Implementation

I choose C++ as the develop language, and using a basic factory pattern for design, put the basic predictor function in base class. The Implementation looks like below:

0.2.1 execute main function

Reading from the trace file, get the exact value by line. The instruction address, Introduction type, the taken or not taken state, and the target address. First I verify the value of instruction type, if it is 0, continue to process the next line, else get in the predictor.

0.2.2 Saturationg counter prediction

I design the saturating counter predictor as the base class.

0.2.2.1 class description

It has the member values like below:

protected:

int bitWidth; // the bit width, here is 2 bit.

int byteSize; // the size of Precditor, here is 2K 4K and 8K.

long long int iTotalCount; // the total number of branch instructers

long long int predictFailCount; // the total number of wrong prediction of branch

float predictionHitRate; // the right prediction rate of this predictor

public:

int entryNum;

int *predictionTable;

It has member functions like below

protected:

long long int ctoll(char s[]); // type change

public:

int GetSize();

void SetSize(int size);

int GetBitWidth();

void SetBitWidth(int width);

long long int GetiTotalCount();

long long int GetpredictFailCount();

float GetpredictionHitRate();

bool predictionProcess(int,int);// the prediction train function.

virtual bool create();// to create the predictor talbe and initilize it.

virtual bool stepSimulation(char *, int);// the main simulation function.

saturaPredictor();

virtual saturaPredictor();

0.2.2.2 Logic flow

When class is constructed, it set defaults values for some member variables, also it is supported to get or set some special value from member function, like the area size and bit width, but considering the prediction table is based on 2bit counter, so it could only be 2 for bitWidth so far. Then the class need to create the prediction table by call create() function. after that the class is just waiting for the main function to call it and input the instruction address and its branch state. In the stepSimulation function, we need to find the prediction table index by instruction address, also remember the total number of branch instructions and the Wrong number which is from the train function predictionProcess().

0.2.3 Global predictor

This class is deveried from base class saturaPredictor, we only add the member variable globalHTindex and redefine the function stepSimulation, the others keep the same.It is because it does not use instruction address to be the prediction table index, it uses a Global history table to keep the history branch state, and try to use the Global history pattern to achive the prediction talbe.

0.2.4 Gshare predictor

This class is deveried from base class saturaPredictor, we only add the member variable globalHTindex and redefine the function stepSimulation, the others keep the same. The main idea of this prediction is to use global and local advantage to create the prediction index.

0.2.5 Tournament predictor

This class is deveried from base class saturaPredictor, we only add the member variable globalHTindex and redefine the function stepSimulation, the others keep the same. The predictor is composed by there different part, two seperate predictors and one slector, they share all the area size. Every time, both two predictors give two prediction values, and the value is choosed by this slector. So the predictors would work by their own ways, we only use the choosen result to verify the Rightness of this predictiors.

0.2.6 Combine predictor

This predictors is not necessary, I design this new one to see what would happen if the Tournament predictor have different member predictors. I add a typeMode variable, we can choose different combination through the value. The reason I design this is because I found the tournaPredictor is not as good as gsharePredictor, it is suppposed to be the best predictor, so I tried different combination, however it is still not better than gsharePredictor.

0.2.7 Measurement

I count all the branch instructions once they go into the stepSimulation of predictors, I also count the wrong prediction numbers(to save execute time). then get a right prediction rate through the class member function. At last put all key data in a result.txt file.

0.3 Result anlysis

The result look like below:

Predictor	Table size	total branch instructions	Wrong Precdition	Right prediction rate
saturaPredictor	2048	46680967	8823859	0.810975
globalPredictor	2048	46680967	9153545	0.803913
gsharePredictor	2048	46680967	6165095	0.867931
tournaPredictor	2048	46680967	6778934	0.854782
saturaPredictor	4096	46680967	8820904	0.811039
globalPredictor	4096	46680967	8491487	0.818095
gsharePredictor	4096	46680967	5385872	0.884624
tournaPredictor	4096	46680967	6445010	0.861935
saturaPredictor	8192	46680967	8819563	0.811067
globalPredictor	8192	46680967	7817878	0.832525
gsharePredictor	8192	46680967	4857211	0.895949
tournaPredictor	8192	46680967	6172937	0.867763

From the table, we could find that gsharePredictor is best, then is tournaPredictor , globalPredictor, last is saturaPredictor. From the picture, we could find that almost predictors is improved when increasing the area size except saturaPredictor,probably it get its threshold which tells us that we could not increase the predictor size limitless.

I did not post out the data from combinationPrediction, because when switch the prefer predictor, the result did not change much, when change the combination predictor, it was still not better than gsharePredictior

This is one place I am very curious, all the test results are not over 0.9, we ran through nearly fifty million branch instructions, it should be enough to get a better result from some peoples, I hope I did not do anything wrong.

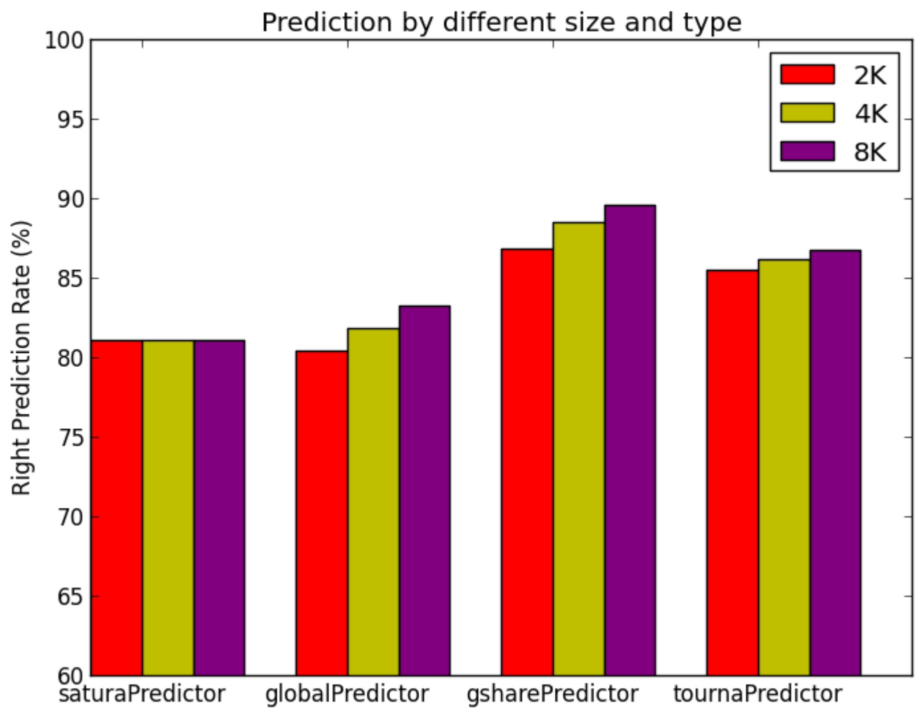

FIGURE 1

0.4 The difficulties

1. To understand how these predictiors work, this is also the purpose of this project. The TA gave me a lot of help for this problem, also I reviewed the class record, finally I figured it out, it felt good.

2. Convert the instruction address to an integal, I did not know why I could not the library function, considering the pattern of the value, I wrote a convert function by myself, and it worked well.

3. About the Tournament, first time, I got a very low rightPredictionRate which made it look very wierd. I checked a very long time and found that I used the rightness of the child prediction as the prediction result, also the prediction result is from 0 1 2 3 not just 0 1.

4. segfault error. I met many segfault errors during programming. This is probably it has many bit operations and some very big data, we this error happeded, I mostly checked the places about prediction table, it helped.

0.5 Conclution

I made a good recongnization for different kinds of predictors from this project. The branch state could be predicted very well through a good predictor. The prediction could be better when increasing the size of precditor, and they also have a bottleneck and could not increase it for ever. Future, I could achieve the predictionProcess function in different ways to make it simulate different bit width counter. we can also add more functions based on the saturaPredictor.