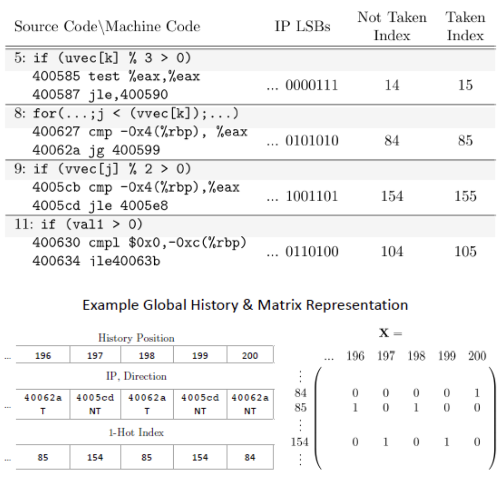
A Proposal on Evaluating a new 3D CNN-based Branch Predictor.

Project Significance and background

As the depth of the out-of-order processor pipeline getting deeper, as well as the introduction of speculative execution, it's becoming more pivotal to have an accurate branch predictor to hide the delay and the mis-prediction penalty from processors. As conventional static or Look-up Table (LUT) based branch predictors are hitting the prediction accuracy bottleneck, some researchers are exploring the field of Machine Learning (ML) to seek an even more powerful solution towards the problem. A branch predictor, in nature, is a binary classifier which ML solutions excel at. In the past 20 years, scientists have proposed numerous of ML-based branch predictors [1-8]. One of the tendencies of the proposed ML models on branch predictors was that the size, dimension, and the number of layers used in the ML models has been increasing; and the accuracy of the models are also increasing over years.

One of the most recent papers [7] proposed to use a one-hot sparse matrix as the input to the CNN model, which encodes the global branch history entry index, take/not take information, as well as PC of the branch instruction into a single sparse matrix (fig 1). This yield encouraging results.



Source Code\Machine Code	IP LSBs	Not Taken Index	Taken Index
`5: if (uvec[k] % 3 > 0)`			

`400585 test %eax,%eax`
`400587 jle,400590`
	... 0000111	14	15

`8: for(...;j < (vvec[k]);...)`			

`400627 cmp -0x4(%rbp), %eax`
`40062a jg 400599`
	... 0101010	84	85

`9: if (vvec[j] % 2 > 0)`			

`4005cb cmp -0x4(%rbp),%eax`
`4005cd jle 4005e8`
	... 1001101	154	155

`11: if (val1 > 0)`			

`400630 cmpl $0x0,-0xc(%rbp)`
`400634 jle40063b`
	... 0110100	104	105

Example Global History & Matrix Representation

History Position

...	196	197	198	199	200

IP, Direction

...	40062a T	4005cd NT	40062a T	4005cd NT	40062a NT

1-Hot Index

...	85	154	85	154	84

$$\mathbf{X} = \begin{array}{c|ccccc} & \dots & 196 & 197 & 198 & 199 & 200 \\ \vdots & & & & & & \\ 84 & & 0 & 0 & 0 & 0 & 1 \\ 85 & & 1 & 0 & 1 & 0 & 0 \\ \vdots & & & & & & \\ 154 & & 0 & 1 & 0 & 1 & 0 \\ \vdots & & & & & & \end{array}$$

Fig 1, the encoding of global branch history matrix. With 1's in (IP << 1 + Direction)

My proposal is to add another set of information into the global branch history matrix, which is *branch offset*. It is rather intuitive that the history of branching was repetitive in time (e.g. T, T , NT, T, T, NT; T for taken and NT for not taken) and in memory space. For example, in a nested loop, whenever the program executes to the end of one iteration of nested loop, the branch instruction either points to the start of the loop or points to outside of nested loop. This process adds to the repetitiveness of branching in space. It would make sense to also include distance of the branch target into account.

To include the branch offset information into the CNN model, we have to perform some changes to the encoding of the global branch history matrix (Fig 2).

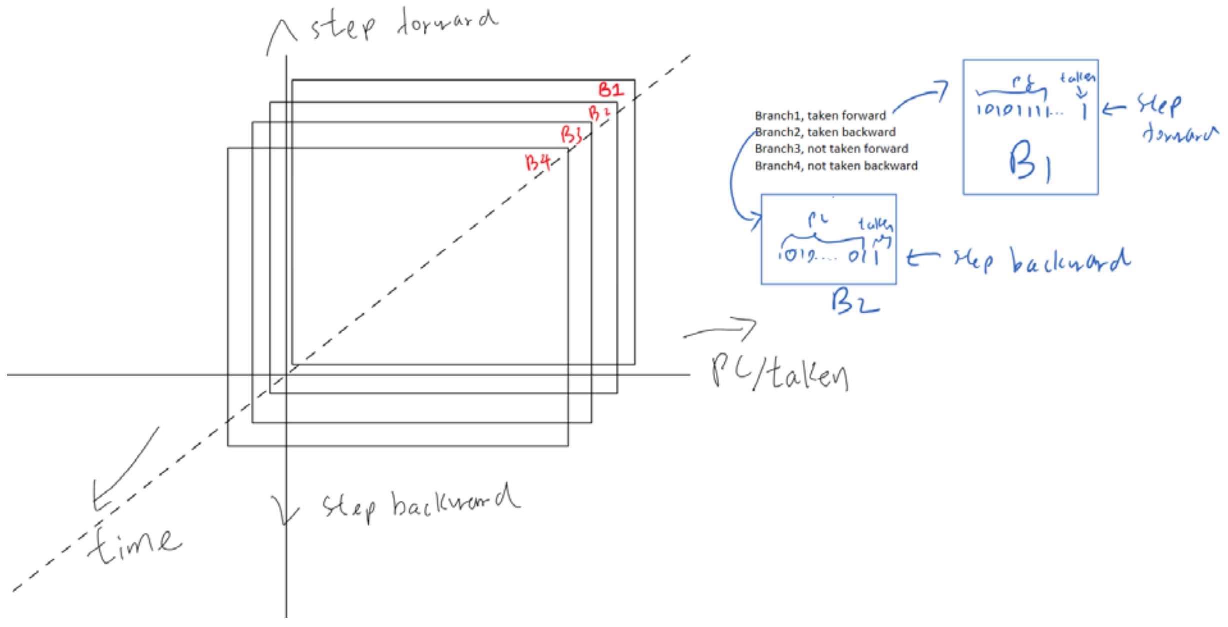


Fig 2 proposed encoding scheme of the global branch history matrix.

There are 3 dimensions within the Global Branch History Matrix (GBHM): PC/taken, Step, and Time.

For example, we have a 16*16*16 GBHM, and if we have a branch instruction at PC = 16'b 0001_1010, and the branch is taken, points forward 2 instructions, then the encoding of this entry of information is: 16'b0011_0101, located in the (16 / 2 + 2) row in the history entry (which is a 16*16 matrix) with all other entries are 0's. for every branch taken / not taken, the GBHM pops the least recent branch history and pushes in the new entry. Each updated GBHM matrix becomes one unit of training data towards the ML model.

Objective

I propose to evaluate whether using this new design of GBHM with branch offset as new added information within ML model would yield decent / better result compared to other ML based predictors. Specifically, I will:

1. Run the Gem5 simulator on certain workload and establish the interface to extract necessary branching information to construct the GBHM data.
2. Encode the GNHM dataset into trainable data format.
3. Design or reference some of the current CNN module design that excel in binary classification.
4. Train the binary classification model with an input as the GBHM and output as the taken / not taken (1/0) for the next branch instruction.
5. Evaluate the accuracy of the trained model by interfering Gem5's branch predictor with the model I have and running multiple benchmarks.
6. Perform a cross-comparison to other pre-existing ML models using the same training data and compare the effeteness of the new branch predictor.

Progress report 1

Following the objectives listed in the proposal, I have finished step 1 and 2; and plan to finish 3 and 4 before the next progress report.

Specifically, I have:

1. Modified the Gem5 base predictor object so that it could log each BTB and update event and prediction event. The prediction event contains information on the taken / not taken information and the training tag; and the BTB contains the branch PC, target, and span parameter required for constructing the training data.
2. Wrote and complied a simple C-based image processing benchmark which computes the histogram of an input image, this benchmark have many nested loops and I find it suitable for collecting branching data for now. This will act as a 'placeholder benchmark' for now to collect data. Future benchmark is expected to contain more variants of workload.
3. Wrote a python script to format the branch history log into NumPy training set. Currently, I observed that the vase majority of the branch jumps take place in a narrow range (100 words at most) so I followed the proposed training data structure and map the "step" info naively as (step = (target - PC) >> 2). This means the, for the 32*32*32 data entry I'm using, every branch history entry will be placed at row# 16 + step. I don't expect this mapping will work fine in the future as there may will be larger branch jumps and this mapping function is expected to be enhanced in the future.

References

[1] (1997) Evidence-Based Static Branch Prediction Using Machine Learning

[2] (2001) Dynamic Branch Prediction with Preceptrons

[3] (2002) Branch Prediction with Neural Networks - Hidden layers and Recurrent Connections

[4] (2004) Branch Prediction with Neural Networks - Hidden layers and Recurrent Connections

[5] (2005) A Study on Deep Belief Net for Branch Prediction

[6] (2017) A Study on Deep Belief Net for Branch Prediction

[7] (2019) Improving Branch Prediction by Modeling Global History with Convolutional Neural Networks

[8] (2020) Dynamic Branch Prediction Using Machine Learning