Branch Prediction Through Neural Networks

Minchan Kwon
1012871803, minchan.kwon@mail.utoronto.ca
https://colab.research.google.com/drive/11H6NKhgIopaloeac4qGjvwn-xY32lDh_?usp=sharing

Project Description

The goal of this project is to use a deep learning model to predict branches that occur when a program is running. A branch is an instruction that tells the processor to divert to a completely different part of the program. Branches may happen in an if-then-else statement or in a conditional loop. Such statements in higher level programming languages are translated to branch instructions in assembly and the decision to take the branch is determined based on whether the condition was met or not. Accurate branch prediction can save computation resources and accelerate performance on modern processors that take advantage of deeply pipelined designs. Although modern branch predictors predict most conditional branches with almost perfect accuracy, Chit-Kwan Lin and Stephen J. Tarsa argue that there is room for improvement.[2]

```
1   if(argc==4) //Data-dependent branch
2   {
3     fout = new FSout(argv[3]);
4     fout->printSet(0, NULL, TRANSACTION_NO);
5   }else
6     fout = NULL;
7   if(0 && fptree->Single_path(0))
8   {
9     Fnode* node;
10    i=0;
11    for(node=fptree->Root->leftchild; node!=NULL; node=node->leftchild
      )
12    {
13      list[0]->FS[i++]=node->itemname;
14    }
15    if (fout) //H2P branch
16      fptree->generate_all(fptree->itemno, 0, fout);
```

Listing 1: Example of an H2P Branch

Listing 1 illustrates an example of a Hard-To-Predict(H2P) branch that modern branch predictors often struggle with. The conditional branch at line 15 completely depends on the data-dependent branch at line 1. This project aims to use CNNs and LSTMs to predict such H2P branches, compare the performance of both models and decide on a model that is both capable of predicting H2P branches and requires less hardware resources such as memory and silicon area.

I run the PARSEC benchmark suite [1] and use Intel Pin to record the sequence of the branch instruction address and the branch decisions during execution. For each branch I want to predict, the model input is a fixed-length window of the global branch history (most recent decisions leading up to the branch of interest). The window length is an important hyperparameter because it directly affects inference latency, model size, and memory requirements. The model outputs a binary prediction for the branch: 1 = taken, 0 = not-taken. Because branches are related to previous branch events, deep learning can help predict branches by learning the branch history. A more detailed explanation will be provided in the "Primary Model" section.

Data Processing & Baseline Model

As described above, I ran the PARSEC benchmark suite developed by Princeton University on a Linux system. Intel Pin, a dynamic binary instrumentation tool was used to track the branch behaviors of the benchmark. For collecting new test data, I will run the benchmark on different inputs provided by the benchmark suite. I collected the global branch history of the benchmark program "fluidanimate" provided in the suite. The global history is a csv file consisting of every branch PCs and the corresponding branch direction. Then a 2-bit saturating counter is used to make a prediction for every branch PC in the global history. This is the baseline model for this project which will be explained in a moment. The predictions made by the saturating counter will be used to identify the H2P branches. I have chosen branch PCs with less than 90% prediction accuracy and sufficient samples(in the order of ten thousands) to be our H2P branches on which our models will be trained. The tables below show the statistics for the global branch history and an H2P branch(PC=0x5fc281fa141d) from the global history.

Raw History	
Total Branches	50,000,000
Unique Branch PCs	1755
Total Taken	34,317,550
Total Not-Taken	15,682,450
Taken Rate	68.64%

Baseline Prediction for 0x5fc281fa141d	
Total Occurrences	65,136
Predicted as Taken	48,852
Predicted as Not-Taken	16,284
Prediction Accuracy	75.0%

Training Set for 0x5fc281fa141d	
Training Samples	45,595
One-hot Dimension	2048
Taken Rate	75.11%

Instead of using the raw PC(e.g. 0x5fc281fa141d) for input, I choose only the lower 10 bits of the PC and concatenate with the branch direction. This is hashed to a decimal value using the following hash function: $((PC << 1) + direction)\&(2^{11} - 1)$. This value represents the index of a one-hot vector. The table below shows a snippet of one training sample from the training set for for PC=0x5fc281fa141d. Every decimal number is the index of the one-hot vector. These integer indices are later converted to actual one-hot vectors for training and inference.

Snippet of One Training Sample							
124	399	483	1983	19	124	399	483
1983	19	124	399	482	653	1315	1983

Figure 1 shows the diagram of a 2-bit saturating counter. A saturating counter is a simple hardware that predicts branch direction based on its current state(i.e, a taken state will always predict 1 and a not-taken state will always predict 0). It is still used in modern computer chips. Transitions between states are based on the actual branch direction of the previous instruction and it saturates at both ends. The saturating counter was able to predict branches with an overall accuracy of 94.71% and a best accuracy of 100% on the fluidanimate benchmark. However, it failed to give good prediction for some branches such as the instruction at PC=0x5fc281fa141d. Figure 2 shows the H2P branches based on the saturating counter's prediction accuracy.

Primary Model

A CNN and an LSTM have been trained on the training set described above. The CNN has two 1-dimensional convolution layers and ReLU activations after each layer. The first layer has 32 filters of size 1, the second layer has 64 filters of size 3. There is a pooling layer with kernel size 4 after the

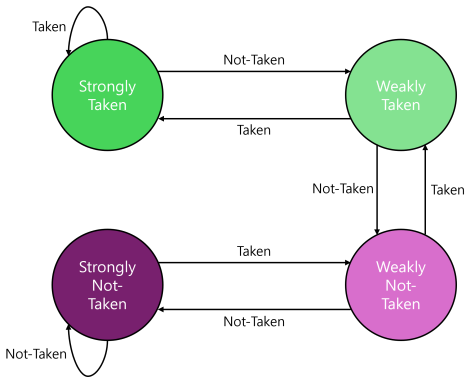

Figure 1: Finite State Diagram of a 2-bit Saturating Counter

H2P Branches Identified By The Baseline Model					
PC	Occurrence	Accuracy	PC	Occurrence	Accuracy
0xf6e5	73,800	50%	0xfce7	36,504	85.1%
0x141d	65,136	75%	0x0f6b	36,504	86.6%
0xf4ee	147,600	75%	0x1c6c	41,236	88.5%
0x27ec	58,867	89.4%			

Figure 2: H2P Branches From Fluidanimate (Only lower 16 bits shown)

final convolutional layer. The classifier has two FC layers. The first FC layer has 128 neurons and the second has 1 neuron connected to a sigmoid activation to output a binary decision. The CNN has a total of 0.48M trainable parameters. The LSTM has 2 layers with 128 hidden units per layer. The final hidden state of the sequence (representing the most recent context) is passed through a single neuron, FC layer and a sigmoid activation to output the probability of the next branch being taken. It has a total of 1.25M trainable parameters.

A CNN is capable of capturing spatial or temporal correlations between branches that may be located far away within the global history. LSTMs are explicitly built to capture long-range dependencies within sequential data like the branch history. These properties provide strong motivation for applying deep learning algorithms to the branch prediction problem. Figure 3 backs this claim as you can see that both models perform much better on certain branches than the saturating counter. You can observe that the branch at PC=0xf6e5 improved from 50% to 100%. Branches 0x141d and 0x0f6b were also predicted with 100% accuracy by both models. However, some branches show very little improvement. This could be due to the history length being too short to capture relevant events, or the branch PC being relatively independent to other branches. To improve test results, I can 1) try varying the history length to see which length works best, 2) adopt different ways to represent branch history (e.g. embeddings instead of one-hot vectors). For hardware optimization, I can try 1) quantizing the weights and 2) reduce the model complexity so far as it doesn't sacrifice the model's accuracy.

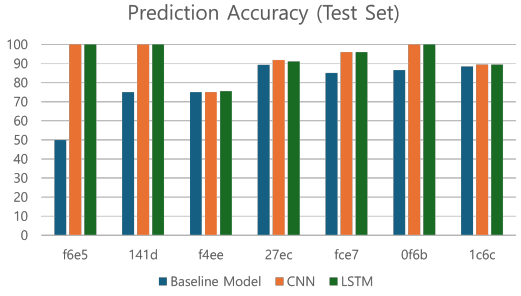

Figure 3: Prediction Accuracy on Test Set

References

[1] cirosantilli. parsec-benchmark. `https://github.com/cirosantilli/parsec-benchmark`, 2023.

[2] Chit-Kwan Lin and Stephen J. Tarsa. Branch prediction is not a solved problem: Measurements, opportunities, and future directions. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, page 228–238. IEEE, November 2019.