Value Prediction

Riley Tuttle
ELE548

May 4, 2018

1 Abstract

Branches and data depencies are two of the last places in the processor design space that leave room for improvement. Branching has had solutions proposed and implemented leaving only the data dependency design space to improve on. To that end I have attempted to implement a version of value prediction proposed in Perais and Seznec 2014 [4]. This implemenation was written inside the CVP-1 framework.

While it is still unfinished the implementation can predict values. With SPEC CPU2006 benchmarks provided by the competition, my vtage implementation achieved an average of 31.786% correct predictions. Confidence thresholds were implemented but do not currently work due to problems with the framework. If however they were used with a condfidence max and threshold of 3 my vtage predictor predicted correctly 8.485% of the time and incorrectly .351% of the time. The other 91% of the time it would not predict.

2 Related Works

Value prediction is introduced by Lipasti et al 1996. In 2014 Perais and Seznec propose a method for storing values in tables organized by increasingly longer length branching histories based on Seznec's branching predictors from 2006 and 2011.

3 Paper Outline

The rest of the paper will be outlined as follows. An explanation of the provided framework. A brief explanation of the vtage predictor. Then some implemenation issues will be addressed. The results will be explained. Then the future work. Then the conclusion and references.

4 The Framework

4.1 The CVP-1 simulator

Provided to me was the cvp-1 simulator. It consisted of an abstract class header file that defined three important function signatures that need to be implemented.

Function 1 getPrediction() This function provided the dynamic sequence number, the pc and the piece number (loaded values could be split into up to 3 64 bit pieces). It returns a boolean to decide if the framework should speculate or not (should be based on confidence of the prediction).

Function 2 speculativeUpdate() This function provided the same information as the getPrediction() function along with all the necessary information to construct the current branching status. It does not provide the true value of the instruction.

Function 3 updatePredictor() This function provides a sequence number and the true value for a predictable instruction.

4.2 Traces

The traces are the SPEC CPU2006 traces and were provided by the competition.

4.3 Evaluation Metrics

While a few different metrics are provided by the framework, the most intuitive to use are correct and incorrect predictions. However due to that fact that I could not utilize the confidence thresholds (explained in section 6) I had to add some information to the correct and incorrect predictions count. Specifically I had to record the confidence measure at each correct and incorrect prediction so that I could calculate for myself what the perdiction performance could have been with confidence thresholds in place.

5 Vtage Predictor

The predictor is a modified version of an ITTAGE implementation (Seznec 2011 [3]). A simplified desription of how the predictor works follows (For more detailed description see Seznec et al 2011 [3] and 2014 [4]):

Step 1 getPrediction is called. The pc is hashed with the global branch history to get the tag and index of a potential prediction entry. Tags and indicies of all history lengths are hashed simultaneously.

Step 2 The predictor searches for the longest and second longest matching tags. A decision is made on which to use based on different state variables.

Step 3 speculativeUpdate is called. Use given information to reconstruct branch decisions and update global branch history.

Step 4 updatePredictor is called. Use the true value of the instruction to update predictions. Rehash the tags and indicies based on the prediction's pc and a copy of the history before it was updated. We see if the selected entry matched the true value and update the entry accordingly. If the prediction was correct increment the confidence counter. If it is incorrect create a longer history entry with the new true value.

It is important to note that this is only a brief description of the overall process. There are many complications that come from implementation. For instance multiple pcs can have a prediction before an update is called. The implementation then requires a queue of unupdated predictions.

6 Implementation Issues

After the issues I had just getting the framework to work properly (Boost Libraries not compiling correctly) and some early mistakes (messing up a pointer, etc) I was left with problems that arose from assumptions made by the implementation of ITTAGE that I based my vtage implementation off of. Firstly there was an assumption that the updatePredictor function would always be called after the getPrediction function. However in this framework that was not the case. Because the values could be split into up to three pieces there could be up to three predictions with the same pc before an update function is called. When the update function is finally called we no longer have access to the pc that made the prediction and is now being updated (we only have the dynamic sequence number that made the prediction). This forced me to implement a queue of unupdated predictions indexed by the dynamic sequence number.

For the majority of testing my predictor I hardcoded that it should predict all time regardless of confidence. This was helpful for me to see what was going on at every stage of prediction for multiple predictions. The assumption that I made was that the framework operated the same way regardless of the prediction being made or not. Then once I was content with how my predictor worked I could filter out some incorrect predictions by predicting based on a confidence threshold. What I found out was that the framework only called the updatePredictor function reliably if you predicted at every instruction. The updatePredictor function is the only place where I know what the true value was. Without that I effectively cannot make longer history entries or update confidence counts. With my current implemenation this caused a negative feedback loop where initially there would be no predictions because confidence would be low. However it would never update the potential predictions by incrementing confidence counters so the confidence would never increase enough for the predictor to decide to predict. I was not able to find a solution for this.

7 Results

Since this was the first value prediction competition and I can find no existing implementations I can only compare my implemenation to the base/sample predictor that was provided by the framework. It was a simple 2 level predictor. Because there was a problem with how the framework only called the update function on predictions I had to predict every time the getPrediction function was called even if I knew the prediction had a low confidence. In order to compare to the base predictor accurately I changed it so that it too would predict regardless of confidence. After running the two predictors through all the traces provided by the framework I found that my vtage implemenation correctly predicted on average 31.786% of the time vs the base predictor's 24.637%. So on average 7% better. If we compare my vtage implementation across the different types of traces we can see that it seemed to perform better for the integer computations over the floating point computations. see figures 1, 2, and 3.

8 Future Work

Because there were many issues with the implemenation there is a lot of room for improvements to my predictor. Obviously fixing those issues could increase performance but I think an even larger margin of improvement can be gained by fixing a core problem with my predictor. It is only able to predict values that have happened before. Imagine a simple program that loads 1 value from memory and loops through only ever incrementing the value by 1. Even though this is an easily detectable pattern and calculable result, my predictor would be wrong every single time. However I think this could be remedied by at least 1 extra field in the table entry. In addition to the table entry that contains a tag, value, counter, and usefulness bit described in Seznec 2014 [4] I could add a difference field where the table entry could remember the difference between its stored value and the previous prediction made by that pc and history hash. I say at least 1 entry because I was assuming some linear function. The difference field being the slope equivalent and the base prediction being the offset. If however the pattern was non linear there would need to be more fields in the entry.

In addition I would like to add in the forward probability counters described in Riley and Zilles 2006 [5] and Pareis and Seznec 2014 [4]. These counters are not meant to increase performance but instead to decrease hardware overhead without significantly affecting performance.

9 Conclusion

Even though my vtage implemetation is laden with problems it still predicts values and is right a significant amount of the time. Here I define significant as much better than a random prediction would be. Even an intelligent guess would be to predict zeros all the time and that will give an average of 9.8% correct predictions through all the traces which is about 20% worse than my vtage implementation. I know that on average 30% prediction is terrible when you consider that 70% of the predictions are wrong. However I think it is important to note that with the use of confidence thresholds I would be able to bring down the incorrect predictions while maintaining the correct predictions. In fact when I did this I was able to eliminate almost all incorrect predictions (reduced from 68.214% to .351% incorrect predictions) while maintaining 8.485% of correct predictions.

If I was to modify further so that the value predictor was capable of predicting the difference between predictions along with the base prediction I know I could capture many more correct predictions. Not to mention that without other implemenations it' is hard to judge the performance of my implementation. Although I expect that Seznec's implementation (likely available with the competition results in June) will have a high performance, something in the 90% range of correct predictions.

References

[1] M. Lipasti, C. Wilkerson, and J. Shen. Value locality and load value prediction. *Proc. of ASPLOS*, 1996.

[2] A. Seznec and P. Michaud. A case for (partially) TAgged GEometric history length branch prediction. *JILP*, 8:1-23, 2006.

[3] A. Seznec. A 64-Kbytes ITTAGE indirect branch predictor. *INRIA/IRISA*, 2011.

[4] A. Perais and A. Seznec. Practical Data Value Speculation for Future High-end Processors *IRISA/INRIA*, 2014.

[5] N. Riley and C. B. Zilles. Probabilistic counter updates for predictor hysteresis and stratification. In *Proc. of HPCA*, pages 110-120, 2006.

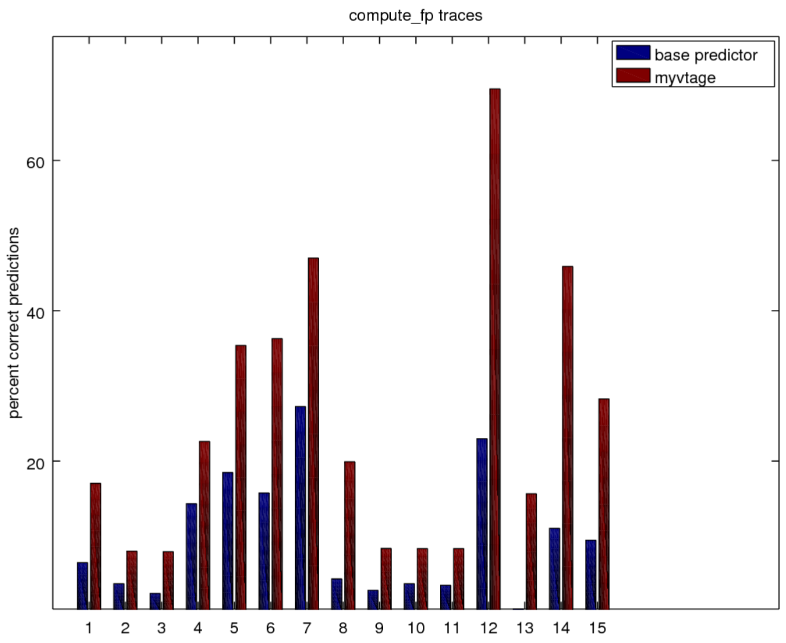

Figure 1: compute_fp traces. base avg = 9.7653, myvtage avg = 25.2433

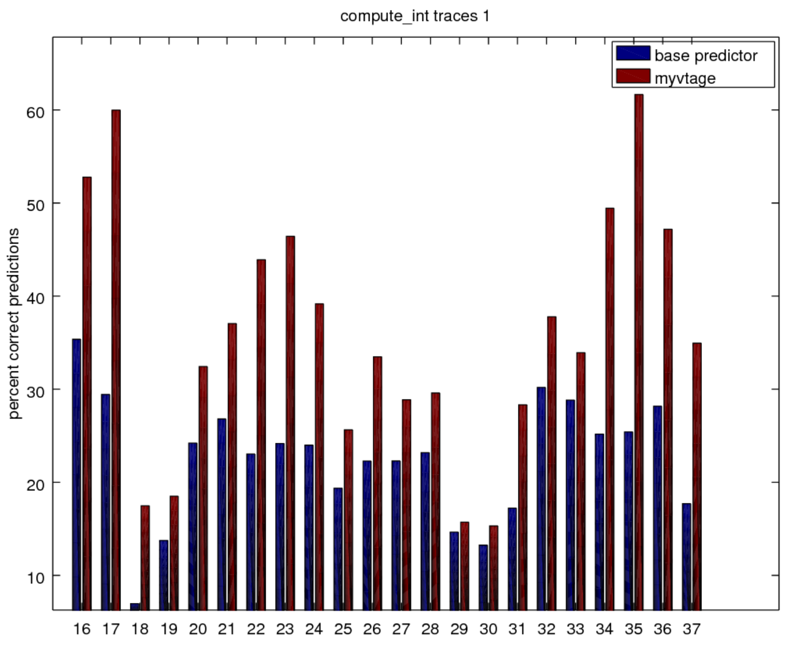

Figure 2: compute_int traces 1 (traces split into 2 groups to more easily view). base avg = 22.588, myvtage avg = 35.414

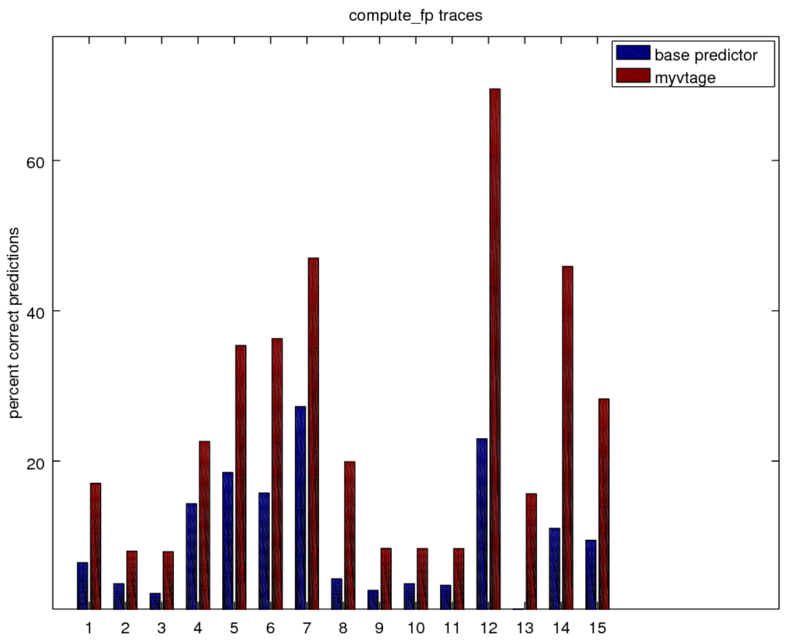

Figure 3: compute_int traces 2

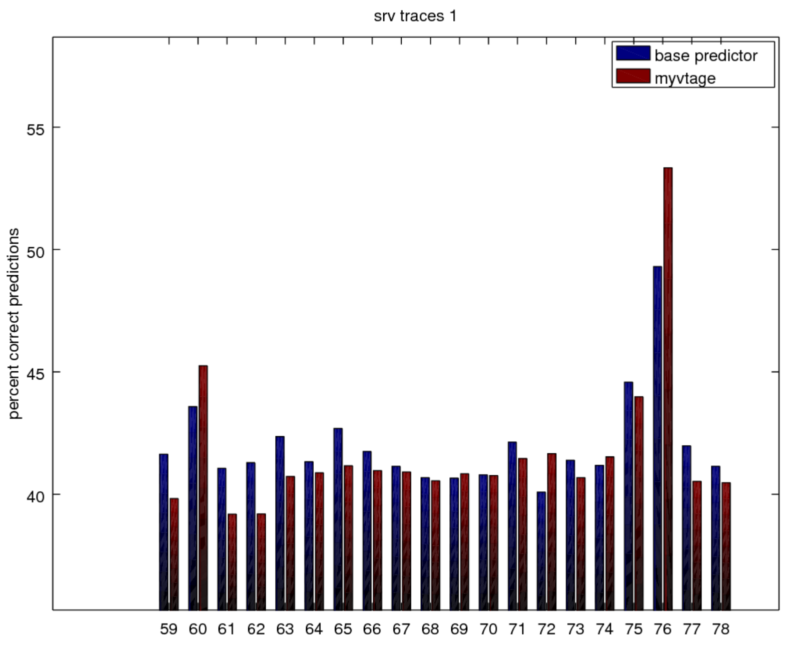

Figure 4: srv traces 1 (traces split into 4 groups to more easily view). base avg = 28.731, myvtage avg = 31.144

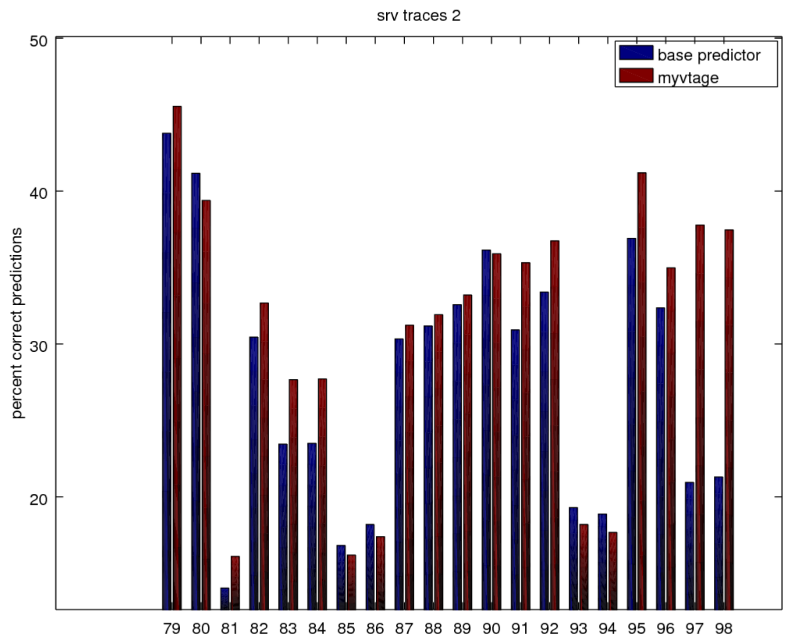

Figure 5: srv traces 2

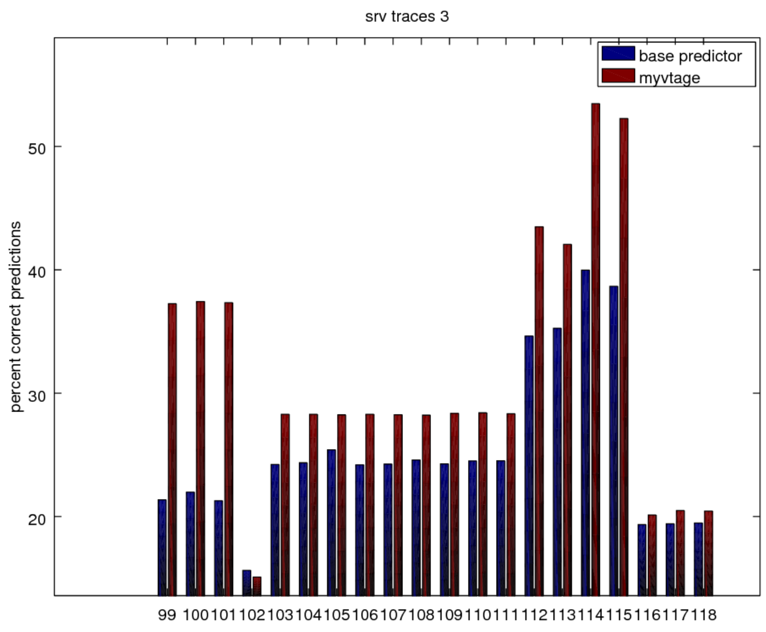

Figure 6: srv traces 3.

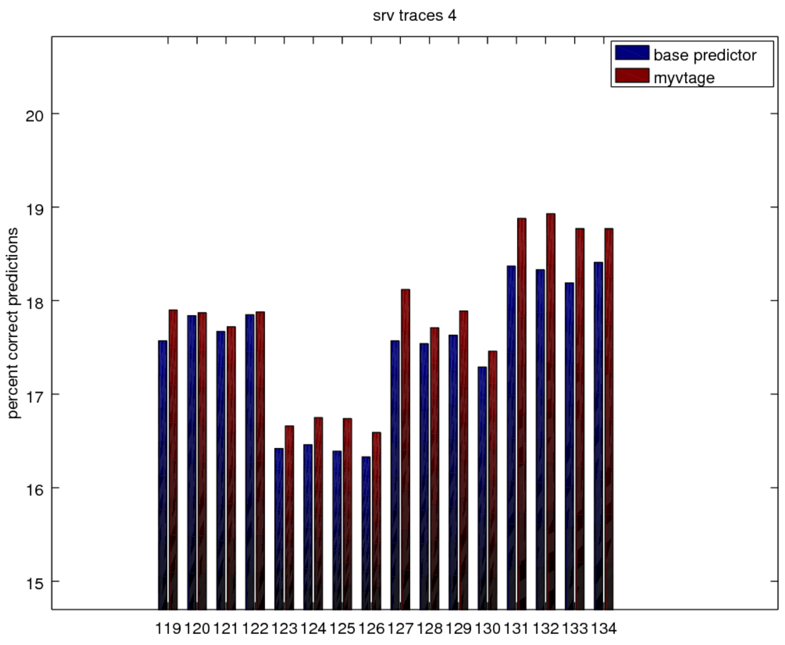

Figure 7: srv traces 4.