

# PARALLEL PATH BASED LOCAL PERCEPTRON BRANCH PREDICTOR

Nikolay Laptev

Computer Science Department – University of California, Santa Barbara, CA, U.S.A.



## ABSTRACT

As the number of pipeline stages increases we become hostage to the penalty imposed by misprediction of branches. Previous works have shown that a neural branch predictor remains victorious among its peers by achieving much lower misprediction rates on similar sized hardware budget than traditional approaches. In this paper we implement 3 variations of perceptron based predictor and give each its verdict. We look at the modified base-line perceptron predictor, then at the history and address based path perceptron, and finally we look at a history based path guided perceptron combined with a local predictor. We discover that on average an improvement of 5% in misprediction rates can be seen by moving towards the latter kind of predictor.

## 1 INTRODUCTION

In this day and age speculative learning and prefetching of data grow to be overly important. This modern study stems from machine learning, which we use to predict the future based, most often, on the past. This showed to be great success in improving the performance of computing, however, the greatest bottleneck that seems to hold down the performance is branch misprediction. Longer pipelines require flushing when prediction is false, thus the field of branch prediction is growing in its important and greater accuracy is needed.

There are always trade-offs. As computer scientists we are used to this hackneyed fact. In branch prediction it is important to balance accuracy with performance. Given a certain hardware budget we must build the fastest and most accurate branch predictor possible.

In this paper we will exploit parallelism in branch predictors and look at aliasing. By combining a predictor that can be parallelized, such as a path based perceptron, with a local predictor which solves the problem of aliasing, we can have the best of two worlds, speed and accuracy. However, there is a trade-off as we explain further in the paper.

## 2 BACKGROUND AND RELATED WORK

There are numerous papers on perceptron based predictors but two stand out. In [1] Ipek et al, describes many approaches to cope with problems of aliasing and linearly-inseparable branches (non-linearity). They describe partitioning to reduce linear inseparability (the more partitions the more likely there is a chance of existence of a boolean function dividing the data within each partition). In [2] Monchiero et al, describes the address-based

perceptron and a history based perceptron. The main insight here is to use PC sensitive predictor (address based) with a history sensitive predictor (history based) to achieve lesser aliasing effects.

Both of these papers are very aware of the aliasing problem that branch predictors (including perceptron based) are facing. [1] uses modification approach, where authors simply modify the original perceptron predictor and make it inverted, partitioned or cache based, where as [2] uses a Tournament predictor strategy, where it simply uses two predictors that work best under different circumstance. (i.e. address based perceptron works best where there are the same history bits mapping to the same index and history based works best when address bits of a specific branch map to the same location)

In [1] authors developed an effective strategy, based on their results, of dealing with both aliasing and linear inseparability, however the implementation of some of their techniques will be expensive from both points of view, hardware budgeting and design. Also, the implementation of their design, may be accurate, but will be hard to parallelize, which would negatively impact performance.

[2] also implements an effective strategy for dealing with aliasing issues. It is fast, easy to implement and can be parallelized (because it is a path based perceptron). The hardware budget it requires can be extensive however, (2xNormal perceptron predictor).

Some of the solutions to the shortcomings of the above mentioned papers can be achieved by tailoring predictor to a specific program (weather it is most sensitive to history or PC) and allocating a significant portion of space for that purpose. Another suggestion would be to

use prior combinational techniques (such as a tournament predictor) and implement that strategy with a parallel path based perceptron predictor.

### 3.0 METHODOLOGY

In this paper we have implemented base-line perceptron, path based (address/history) perceptron and finally the combination of parallel (path) based history and local predictor. We have studied several programs (we picked those that are history sensitive and address sensitive) to judge the performance of predictors. The main application is GAIM (a popular instant messenger program). We also ran our predictor on INTEL based tests as well as several interesting Unix commands.

For GAIM, the most amount of mispredicted branches occurs during the start-up of the program (based upon tests). At this time we are unable to provide the code snippet to prove this.

We also ran the standard INTEL based tests and found interesting results (that MEDIA 1 is incredibly history based test where as SERVER 1 is not). This type of information can help us modify our predictor to suit a given program. Furthermore, on the fly branch prediction optimization can be possible given such statistics.

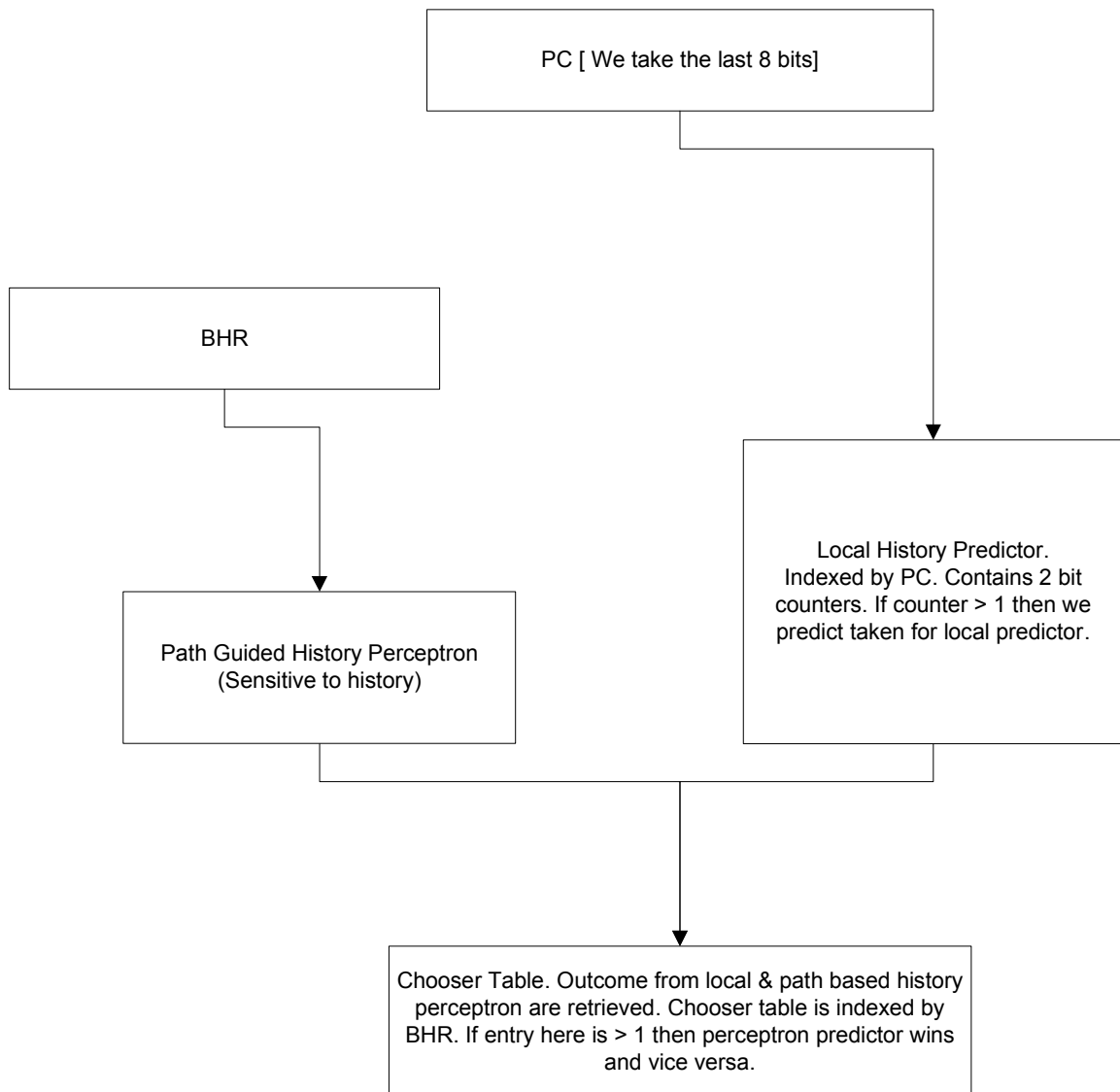
Several Unix commands were run with Valgrind just for entertainment. [du, pstree, ls, diff] We found that du performed the worst where as diff the best.

## 4 BRANCH PREDICTION ARCHITECTURE

Our approach is based upon several of the papers mentioned in the this paper thus far and in the references. We largely give credit to

[1] and [2] for the overall basis. The basic approach involved in implementing a history based path guided perceptron (We also implement history based and address based perceptron as described in [2] for comparison) On top of this we implement a traditional local based predictor (indexed by PC).

When making a prediction, we compute the outcome of our history based path guided perceptron and local predictors. Then, referring to the 'chooser' table which is a 4096 entry table of 2 bit saturated counters (indexed by a 12bit global history) we make the final prediction. If *chooser* entry is greater than 1 perceptron's prediction will be chosen.,



otherwise local predictor wins.

*Fig. 1: Sketch of basic outline of our predictor*

In updating the chooser table, we increment the counter (at global history index) if local

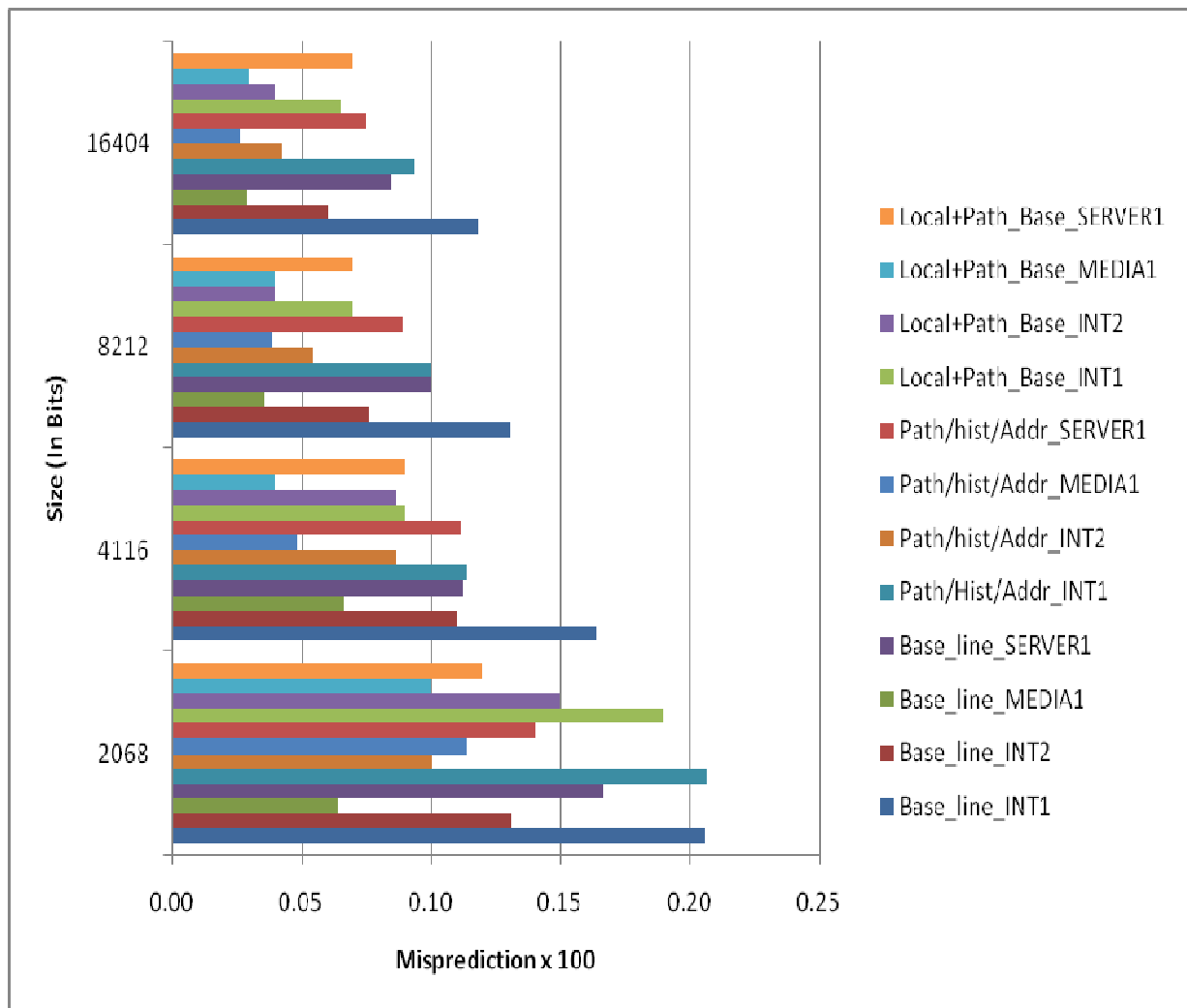
predictor was right and perception was wrong. (and vice versa)

We update the perceptron based predictor if the last prediction does not agree with the outcome and the computed sum is less than Theta (our threshold). When updating perceptron, we increment the weight (i) if global registry bit agrees with the outcome and vice versa. At the end we update the global registers and path table (which is just the table containing PCs of N previous branches). Lastly we update the local predictor in the similar manner described in the literature.

## 5 RESULTS

Below is the graph of all branch predictors we have implemented (including the one closely studied in this paper)

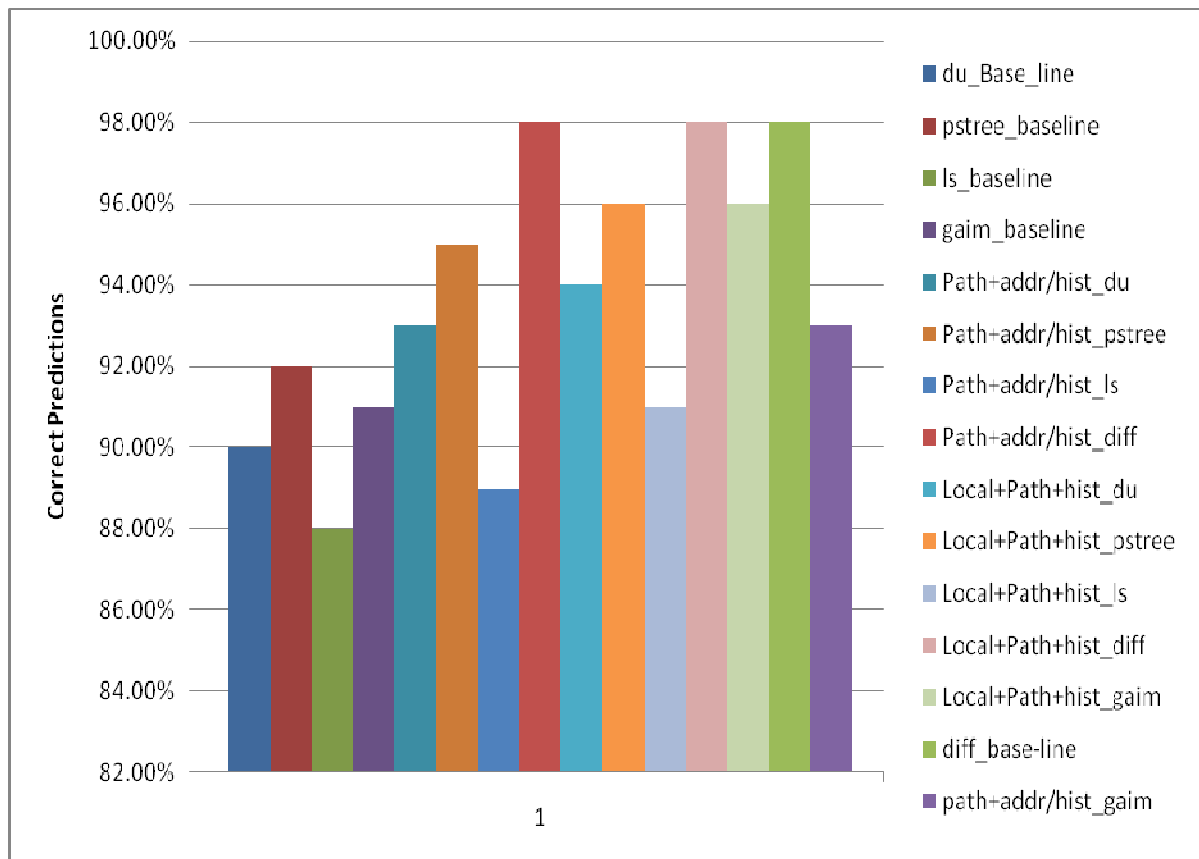
On average our design is about 2% faster than path guided history/address based perceptron and about 7-10% faster than the original perceptron predictor. This is due to the fact that our combined predictor does not suffer from aliasing problem (it is path guided and



implements a local predictor)

This graph illustrates results for GAIM and some linux commands. All predictors were run with 16404 Bits.

There is still room for tuning our predictor, to find the best combination of local + perceptron predictor sizes. This work is still in progress and will be updates shortly.



## REFERENCES

1. Engin Ipek, Sally A. McKee, Martin Schulz, and Shai Ben David, On Accurate and Efficient Perceptron-Based Branch Prediction
2. The combined Perceptron Branch Predictor, Matteo Monchiero, Gianluca Palermo

3. Fast Path-Based Neural Branch Prediction, Daniel A. Jimenez
4. David Tarjan and Kevin Skadron, Merging Path and Gshare Indexing in Perceptron Prediction