Dynamic branch prediction with perceptrons

https://ieeexplore-ieee-org.aurarialibrary.idm.oclc.org/document/903263

Slide 79:

#1. Thank you Rohith for giving brief explanation about ISL_Tage, Now I will be continuing with Perceptrons.

Slide 80:

#2 Background and related work:

Branch Prediction with Perceptrons

Here I will describe perceptrons, and explain how they can be used in branch prediction, and discuss their strengths and weaknesses. Our method is essentially a two-level predictor, replacing the pattern history table with a table of perceptrons.

The perceptron was introduced in 1962 to study brain function. We consider the simplest of many types of perceptrons, a single-layer perceptron consisting of one artificial neuron connecting several input units by weighted edges to one output unit.

We can use a perceptron to learn correlations between particular branch outcomes in the global history and the behavior of the current branch. These correlations are represented by the weights. The larger the weight, the stronger the correlation, and the more that particular branch in the global history contributes to the prediction of the current branch.

Perceptrons were introduced to the branch prediction arena by Jiménez and Lin [2], where they found that perceptrons are often more effective than gshare, a respected branch predictor in use today. They also produced a hybrid predictor that combined gshare and perceptrons, and often outperformed them both.

Related work: Slide 81

Most two-level predictors cannot consider long history lengths, which becomes a problem when the distance between correlated branches is longer than the length of a global history shift register [7]. Even if a PHT- pattern history table scheme could somehow implement longer history lengths, it would not help because longer history lengths require longer training times for these methods [18].

Variable length path branch prediction [23] is one scheme for considering longer paths. It avoids the PHT capacity problem by computing a hash function of the addresses along the path to the branch. It uses a complex multi-pass profiling and compiler-feedback mechanism that is impractical for a real architecture, but it achieves good performance because of its ability to consider longer histories.

The simplicity of a perceptron is exactly what makes it well suited to the branch prediction problem. Given the short clock cycles that permeate the current state-of-the-art processors, very little computation can be done in the time allotted for prediction. The amount of specialized data that can be accessed in this time is limited, too. Because of this, heavier-weight machine learning techniques such as decision trees, back-propagation neural networks, and nearest neighbor are infeasible. However, perceptrons can be trained with a simple update rule, and can render decisions in roughly the time required to perform a few additions. They are also candidates for pipelining.

Michaud and Seznec produced an internal publication [4] that evaluated the effectiveness of the perceptron branch predictor. They found that including a few bits from the address of a branch as input to the perceptron showed an improvement, largely due to improving linear separability. Their approach was also shown to be more cost-efficient than adding more entries in the table of perceptrons.

Agrawal and Woo investigated perceptrons as part of a class project at CMU, [1]. Their contribution was a simple "patch" that on one benchmark provided a 49% improvement over classical perceptron performance. The patch consisted of tracking the number of branches executed in total, and the number of branches since the last misprediction.

Slide 82: Neural networks

Over the past few years, branch prediction has become an increasingly important feature in the modern microprocessor. This is largely due to the ever-growing emphasis on processor speed: as pipeline lengths increase, more CPU time is wasted on a branch mispredict. There is also the desire to exploit instruction-level parallelism: if a branch predictor can tell the CPU its confidence level for a particular prediction, the CPU can better decide if it should speculatively execute both branch outcomes (low confidence), or simply execute the predicted outcome, freeing functional units for use on other tasks. There are two main types of branch prediction

Neural networks

Artificial neural networks learn to compute a function using example inputs and outputs. Neural networks have been used for a variety of applications, including pattern recognition, classification [8], and image understanding [15], [13].

Slide 83-:

Why perceptrons?

Perceptrons are a natural choice for branch prediction because they can be efficiently implemented in hardware. Other forms of neural networks, such as those trained by back-propagation, and other forms of machine learning, such as decision trees, are less attractive because of excessive implementation costs. For this work, we also considered other simple neural architectures, such as Adaline [25] and Hebb learning [8], but we found that these were less effective than perceptrons. One benefit of perceptrons is that by examining their weights, i.e., the correlations that they learn, it is easy to understand the decisions that they make. By contrast, a criticism of many neural networks is that it is difficult or impossible to determine exactly how the neural network is making its decision. Techniques have been proposed to extract rules from neural networks [21], but these rules are not always accurate. Perceptrons do not suffer from this opaqueness; the perceptron's decision-making process is easy to understand.

Perceptrons

A perceptrons is a tool developed in Machine Learning as a simple model of a human neuron. The perceptron receives a set of input signals x, and multiplies each of the signals according to the perceptron’s set of weights w. It then adds the results together, and outputs 1 if the sum is ^{3}0, and –1 otherwise. More formally, given a vector of inputs and a vector of weights, the perceptron’s output is sgn(x·w). For branch prediction, the input to the perceptron is usually the global history of branch outcomes, and semantically, the output is “taken” if the sum is ^{3}0, and “not taken” if the sum is

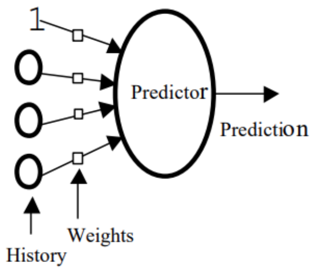


Figure 1: A Perceptron.

Slide 84:

How is perceptron used for Branch prediction?

For branch prediction, the input to the perceptron is usually the global history of branch outcomes, and semantically, the output is “Taken” if the sum is 1 and “not taken” if the sum is -1.

Slide 85: Perceptrons Diagram

Slide 86:

Types of branch predictors that uses neural network

Static branch prediction

Neural networks have been used to perform static branch prediction [3], where the likely direction of a branch is predicted at compile-time by supplying program features, such as control flow and opcode information, as input to a trained neural network. This approach achieves an 80% correct prediction rate, compared to 75% for static heuristics [1], [3]. Static branch prediction performs worse than existing dynamic techniques, but is useful for performing static compiler optimizations.

Dynamic Branch Prediction

Dynamic branch prediction has a rich history in the literature. Recent research focuses on refining the two-level scheme of Yeh and Patt [26]. In this scheme, a pattern history table (PHT) of two-bit saturating counters is indexed by a combination of branch address and global or per-branch history. The high bit of the counter is taken as the prediction. Once the branch outcome is known, the counter is incremented if the branch is taken, and decremented otherwise. An important problem in two-level predictors is aliasing [20], and many of the recently proposed branch predictors seek to reduce the aliasing problem [17], [16], [22], [4] but do not change the basic prediction mechanism. Given a generous hardware budget, many of these two-level schemes perform about the same as one another [4].

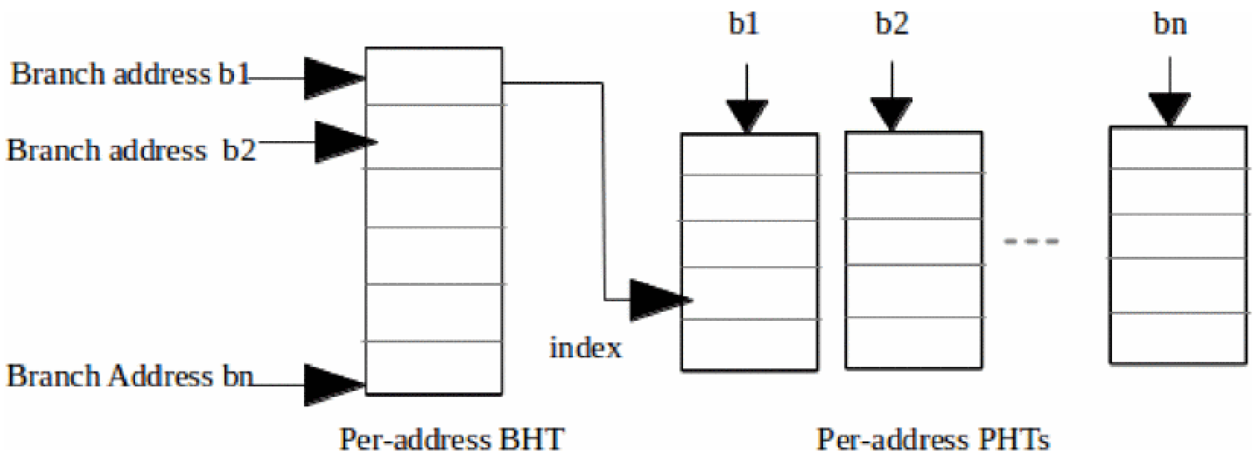
Slide 87-88:

Perceptron Predictor

The perceptron predictor [6] uses a simple neural network, the perceptron instead of the two-bit counters. It learns a target Boolean function f(x1, x2,…,xn) inputs that predicts whether a particular branch will be taken or not. Here x is the bits of a global branch history shift register. X is bipolar, −1 represents the not taken and 1 represents the taken outcome. A perceptron is represented as a vector of weights w0,w1,…,wn. Here, the weights are signed integers. The output is calculated as the dot product of the weight vector w0,w1,…,wn and the input vector x0,x1,…,xn(x0 provides the bias input and is always set to 1). The output y of a perceptron is as below:

$$y=w0+\sum ni=1wi*xi$$

Figure 3 shows a diagram of a perceptron predictor internals.



A major weakness of perceptrons is their increased computational complexity when compared with two-bit counters. However, it provides better prediction accuracy compared to other popular predictors even at lower resource budget. In a perceptron predictor, the best performance can be achieved by tuning the history length, the number of bits used to represent the weights, and the threshold. A smaller predictor size affects the number of bits of these three parameters, thereby degrading the prediction accuracy in turn.

Slide 89-90:

Training Perceptrons

Once the perceptron output y has been computed, the following algorithm is used to train the perceptron. Let t be -1 if the branch was not taken, or 1 if it was taken, and let θ be the threshold, a parameter to the training algorithm used to decide when enough training has been done.Since t and xi are always either -1 or 1, this algorithm increments the ith weight when the branch outcome agrees with xi, and decrements the weight when it disagrees. Intuitively, when there is mostly agreement, i.e., positive correlation, the weight becomes large. When there is mostly disagreement, i.e., negative correlation, the weight becomes negative with large magnitude. In both cases, the weight has a large influence on the prediction. When there is weak correlation, the weight remains close to 0 and contributes little to the output of the perceptron.

```
if sign($y_{out}$) $\neq t$ or $|y_{out}| \leq \theta$ then
        for $i := 0$ to $n$ do
                $w_i := w_i + tx_i$
        end for
end if
```

Slide 91-92:

Methodology:

We can use a perceptron to learn correlations between particular branch outcomes in the global history and the behavior of the current branch. These correlations are represented by the weights. The larger the weight, the stronger the correlation, and the more that particular branch in the global history contributes to the prediction of the current branch. The input to the bias weight is always 1, so instead of learning a correlation with a previous branch outcome, the bias weight, w0, learns the bias of the branch, independent of the history.

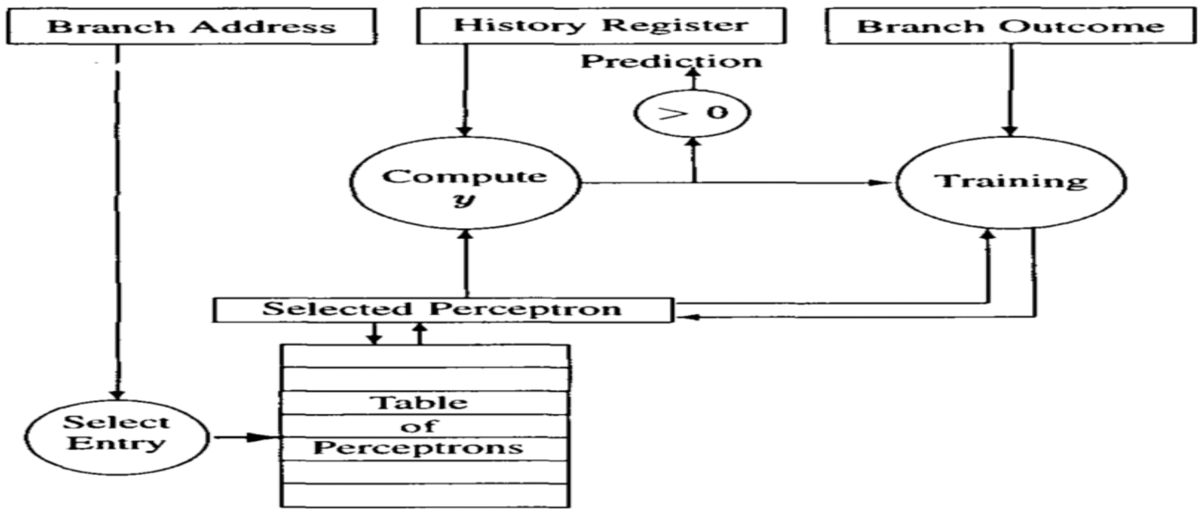


Figure 2 shows a block diagram for the perceptron predictor. The processor keeps a table of N perceptrons in fast SRAM, similar to the table of two-bit counters in other branch prediction schemes. The number of perceptrons, N, is dictated by the hardware budget and number of weights, which itself is determined by the amount of branch history we keep. Special circuitry computes the value of y and performs the training. When the processor encounters a branch in the fetch stage, the following steps are conceptually taken:

1. The branch address is hashed to produce an index $i \in 0..N-1$ into the table of perceptrons.
2. The ith perceptron is fetched from the table into a vector register, P0..n, of weights.
3. The value of y is computed as the dot product of P and the global history register.
4. The branch is predicted not taken when y is negative, or taken otherwise.
5. Once the actual outcome of the branch becomes known, the training algorithm uses this outcome and the value of y to update the weights in P ‘.
6. P is written back to the ith entry in the table.

It may appear that prediction is slow because many computations and SRAM transactions take place in steps 1 through 5. However, Section 6 shows that a number of arithmetic and microarchitectural tricks enable a prediction in a single cycle, even for long history lengths.

Slide 93:

Methodology:

Putting it All Together

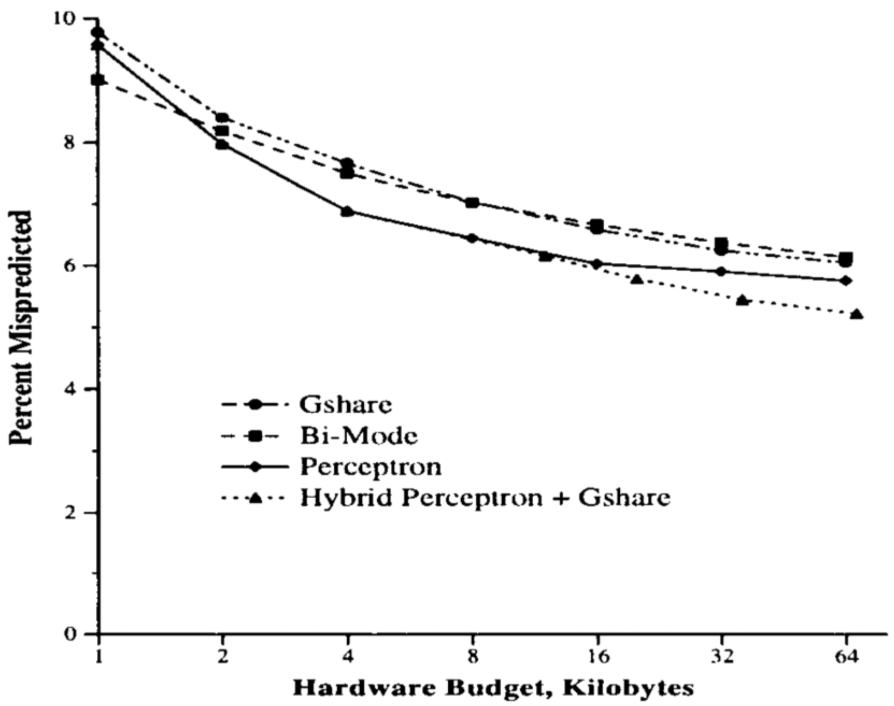
We can use a perceptron to learn correlations between particular branch outcomes in the global history and the behavior of the current branch. These correlations are represented by the weights. The larger the weight, the stronger the correlation, and the more that particular branch in the global history contributes to the prediction of the current branch. The input to the bias weight is always 1, so instead of learning a correlation with a previous branch outcome, the bias weight, w0, learns the bias of the branch, independent of the history.

Tuning the predictors:
We use a composite trace of the first 10 million branches of each SPEC 2000 benchmark to tune the parameters of each predictor for a variety of hardware budgets. For gshare and bi-mode, we tune the history lengths by exhaustively trying every possible history length for each hardware budget, keeping the value that gives the best prediction accuracy. For the perceptron predictor, we find, for each history length, the best value of the threshold by using an intelligent search of the space of values, pruning areas of the space that give poor performance. For each hardware budget, we tune the history length by exhaustive search.

Slide 94-95:

Performance:

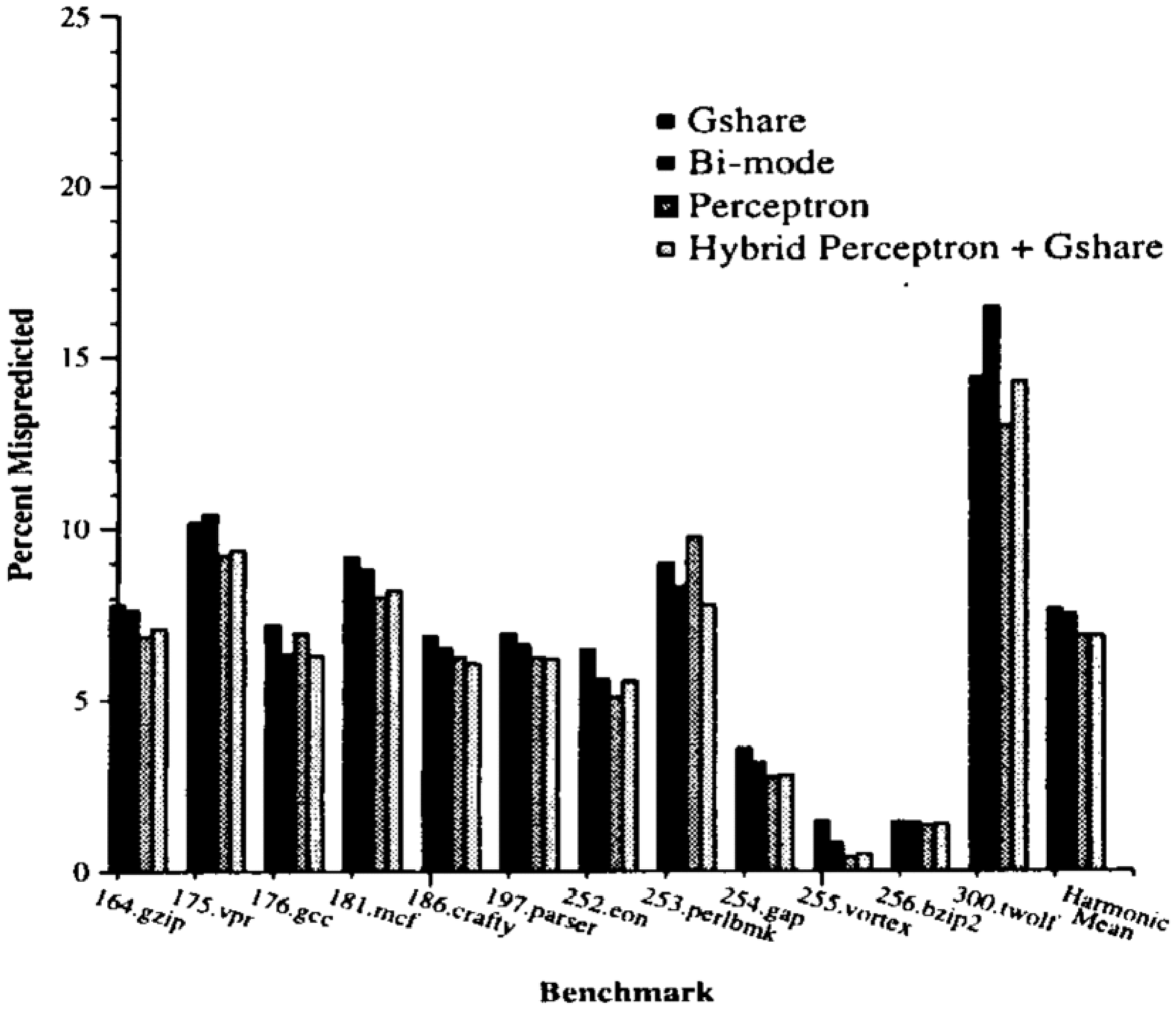
the harmonic means of prediction rates achieved with increasing hardware budgets on the SPEC 2000 benchmarks. The perceptron predictor's advantage over the PHT methods is largest at a 4K byte hardware budget, where the perceptron predictor has a misprediction rate of 6.89%, an improvement of 10.1% over gshare and 8.2% over bimode. For comparison, the bi-mode predictor improves only 2.1% over gshare at the 4K budget. Interestingly, the SPEC 2000 integer benchmarks are, as a whole, easier for branch predictors than the SPEC95 benchmarks, explaining the smaller separation between gshare and bimode than observed previously



Perceptron vs. other techniqes, Harmonic Mean

Slide 96:

To compare the training speeds of the three methods, we examine the first 40 times each branch in the 176. gcc benchmark is executed (for those branches executing at least 40 times). Figure 6 shows the average accuracy of each of the 40 predictions for each of the static branches. The average is weighted by the relative frequencies of each branch. We choose 176. gcc because it has the most static branches of all the SPEC benchmarks.



Slide 97:

Limitations:

Perceptron networks have several limitations. First, the output values of a perceptron can take on only one of two values (0 or 1) because of the hard-limit transfer function. Second, perceptrons can only classify linearly separable sets of vectors. If a straight line or a plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly. However, it has been proven that if the vectors are linearly separable, perceptrons trained adaptively will always find a solution in finite time. You might want to try Linearly Non-separable Vectors. It shows the difficulty of trying to classify input vectors that are not linearly separable.

https://www.mathworks.com/help/deeplearning/ug/perceptron-neural-networks.html

Advatage:

A key advantage of our approach is its ability to utilize long branch history lengths. In our predictor, each static branch is ideally allocated its own perceptron to predict the branch outcome, and the space required by our scheme scales linearly with the history length. In contrast, traditional two-level adaptive schemes use a pattern history table (PHT) of two-bit saturating counters, indexed by a history shift register that stores the outcomes of previous branches. This PHT structure limits the length of the history register to the logarithm of the number of counters. Thus, for the same hardware budget, our predictor can consider much longer histories than PHT-based schemes, which leads to high accuracy

https://people.engr.tamu.edu/djimenez/pdfs/old-perceptron.pdf

Slide 99: Comparison of all five branch predictors

Slide 100: Experimental Setup:

In this work, we use the five different predictor implementations, namely, GShare, a local history based Two-Level predictor, ISL-TAGE, LTAGE and Perceptron from the Championship Branch Prediction-2 (CBP-2) [2] benchmarks. In CBP-2, all predictors are designed for a fixed storage budget. For our experiments, we modified the predictor codes to work with 6 different storage budgets, namely, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB and 64 KB. The storage value mentioned here is the total storage allocated to a predictor for managing all data structures that it needs internally to store and manipulate history information. We performed our experiments on the CBP-2 traces and recorded the Mis-prediction Per Kilo Instructions (MPKI) and latency

- Six different storage budgets are involved in CBP-2 benchmarks, likely 2 KB,4 KB,8 KB,16 KB,32 KB and 64 KB.
- Five different predictors namely GShare, a local history based Two-Level predictor, ISL-TAGE, LTAGE and Perceptron are implemented for all these storage budgets.

Slide 101: Metrics for Comparison

There are two metrics of comparison for all the five predictors over different CPB-2 benchmarks and different storage budgets for each benchmark which are-

1. Mis-prediction Per Kilo Instructions (MPKI)
2. latency

Slide: 102-103:

Results based on MPKI Metrics

Here on X-Axis we have CBP2 Benchmark programs and on Y-Axis we have MPK1.

We have outcome of 6 different sizes 2KB, 4KB, 8KB, 16KB, 32KB, 64KB.

LTAGE – Gshare – ISL-TAGE – Two-level – Perceptron

We have 6 different graphs because we have 6 different storage budgets

Slide 104-105:

Results based on Latency Metrics

Here on X-Axis we have CBP2 Benchmark programs and on Y-Axis we have Time to get latency metrics.

We have outcome of 6 different sizes 2KB, 4KB, 8KB, 16KB, 32KB, 64KB.

LTAGE – Gshare – ISL-TAGE – Two-level – Perceptron

We have 6 different graphs because we have 6 different storage budgets

Slides: 106-107:

Conclusion and Observations:

From Results,

- The predictor with the lowest MPKI at a particular storage point can have the highest MPKI for some other storage point.
- LTAGE has the highest MPKI for all the CBP-2 traces with 2 KB storage size whereas, it does not have the same for all the programs for other storage sizes.
- For the storage size greater than 16KB, LTAGE and ISL-TAGE performed well.
- The performance of a two-level predictor and perceptron are better for low storage budget compared to that of the LTAGE and ISL-TAGE.
- The latency doesn’t largely vary across the points.
- The two-level predictor has the highest latency compared to that of the other predictors for all the storage sizes.
- The perceptron predictor has the lowest MPKI as well as its latency is also comparable with the other predictors.
- Predictors included in modern processors may not always be the best choice for the embedded processors with less storage.

Slides 108-109 :

References

Slide 110:

Thankyou