An ahead pipelined alloyed perceptron with single cycle access time

David Tarjan
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904
dtarjan@cs.virginia.edu

Kevin Skadron
Dept. of Computer Science
University of Virginia
Charlottesville, VA 22904
skadron@cs.virginia.edu

Mircea Stan
Dept. of Electrical and
Computer Engineering
University of Virginia
Charlottesville, VA 22904
mircea@virginia.edu

ABSTRACT

The increasing pipeline depth, aggressive clock rates and execution width of modern processors require ever more accurate dynamic branch predictors to fully exploit their potential. Recent research on ahead pipelined branch predictors [11, 19] and branch predictors based on perceptrons [10, 11] have offered either increased accuracy or effective single cycle access times, at the cost of large hardware budgets and additional complexity in the branch predictor recovery mechanism. Here we show that a pipelined perceptron predictor can be constructed so that it has an effective latency of one cycle with a minimal loss of accuracy. We then introduce the concept of a precomputed local perceptron, which allows the use of both local and global history in an ahead pipelined perceptron. Both of these two techniques together allow this new perceptron predictor to match or exceed the accuracy of previous designs except at very small hardware budgets, and allow the elimination of most of the complexity in the rest of the pipeline associated with overriding predictors.

1. INTRODUCTION

The trend in recent high-performance commercial microprocessors has been towards ever deeper pipelines to enable ever higher clockspeeds [2, 7], with the width staying about the same from earlier designs. This trend has put increased pressure on the branch predictor from two sides. First, the increasing branch misprediction penalty puts increased emphasis on the accuracy of the branch predictor. Second, the sharply decreasing cycle time makes it difficult to use large tables or complicated logic to perform a branch prediction in one cycle. The consequence has been that recent designs often use a small one cycle predictor backed up by a larger and more accurate multi-cycle predictor. This increases the complexity in the front end of the pipeline, without giving all the benefits of the more accurate predictor.

Recently, it was proposed [8, 11, 19] that a branch predictor could be *ahead pipelined*, using older history or path information to start the branch prediction, with newer information being injected as it became available. While there is a small decrease in accuracy compared to the single cycle version of the same predictor, the fact that a large and accurate predictor can make a prediction with one or two cycles latency more than compensates for this.

Using a different approach to reducing the effective latency of a branch predictor, a pipelined implementation for the perceptron predictor [10] was also proposed. The perceptron predictor is a new predictor which is based not on two bit saturating counters like almost all previous designs, but on a simple neural network.

The main contributions of this paper are:

- We show that a path-based perceptron predictor with one cycle effective access latency can be constructed with neglibible loss in prediction accuracy, obviating the need for any preliminary predictor and all the complexity associated with such a design.
- We show that the prediction of a perceptron using local branch history can be precomputed, removing all of the delay associated with the computation and most of the delay from the table lookup from the critical path. This allows a local perceptron to be used in a one cycle predictor. The effect is to shorten the global pipeline at given accuracy, thus reducing the amount of state that needs to be checkpointed and simplifying the branch recovery mechanism of the global predictor.

This paper is organized as follows: Section 2 gives a short introduction to the perceptron predictor and gives an overview of related work, Section 3 talks about the impact of delay on branch prediction and how it has been dealt with up to now, as well as the complexity involved in such approaches, Section 4 introduces the ahead pipelined path-based perceptron, Section 5 then introduces the concept of a precomputed local perceptron, Section 6 describes the simulation infrastructure used for this paper, Section 7 shows results both for table-based predictors as well as comparing the ahead pipelined alloyed perceptron with prior proposals. Finally, Section 8 concludes the paper and Section 9 gives an outlook on future work.

2. THE PERCEPTRON PREDICTOR AND RELATED WORK

2.1 The Idea of the Perceptron

The perceptron is a very simple neural network. Each perceptron is a set of weights which are trained to recognize patterns or correlations between their inputs and the event to be predicted. A prediction is made by calculating the dot-product of the weights and an input vector. The sign of dot-product is then used as the prediction. In the context of branch prediction, each weight represents the correlation of one bit of history (global, path or local) with the branch to

be predicted. In hardware, each weight is implemented as an n-bit signed integer, where n is typically 8 in the literature, stored in an SRAM-Array. The input vector consists of 1's for taken and -1's for not taken branches. The dot-product can then be calculated as a sum with no multiplication circuits needed.

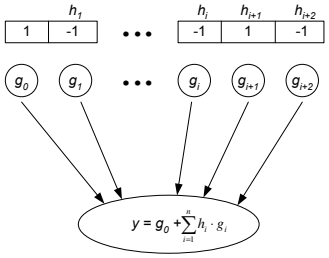

Figure 1: The perceptron assigns weights to each element of the branch history and makes its prediction based on the dot-product of the weights and the branch history plus a bias weight to represent the overall tendency of the branch. Note that the branch history can be global, local or something more complex.

2.2 Related Work

The perceptron predictor was originally introduced in [13] and was shown to be more accurate than any other then known global branch predictor. The original perceptron used a Wallace tree adder to compute the output of the perceptron, but still incurred more than 4 cycles of latency. The recently introduced path-based perceptron [11] hides most of the delay by fetching weights and computing a running sum along the path leading up to each branch. The critical delay of this predictor is thus the sum of the delay of a small SRAM-Array and one small adder. It is estimated that a prediction would be available in the second cycle after the address became available. For simplicity we will call this predictor the *overriding perceptron*, since it can only act as a second level overriding predictor, and not as a standalone predictor.

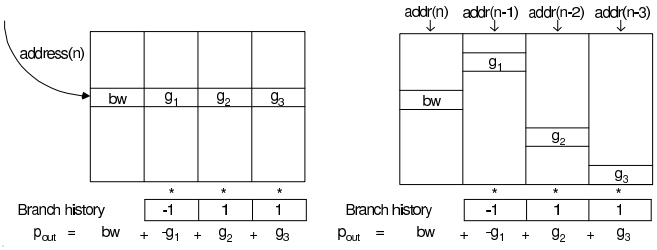

Figure 2: (left) The global perceptron fetches all weights based on the current branch address. (right) The path-based perceptron fetches weights along the path leading up to the branch and computes a running partial sum in the pipeline.

Seznec studied the performance of a theoretical interference-free global perceptron [17]. He then introduced several improvements which could improve the accuracy of a global perceptron at very large hardware budgets, without looking at delay. He noted that the number of additions necessary to compute any perceptron could be cut in half by looking at two adjacent weights in the weight table as the weights of the combination of the two branches. He also introduced the idea of using different hashing functions (similar to skewed caches and branch predictors) to fetch different weights and assigning multiple weights per branch, thereby lessening the impact of aliasing.

Ipek et al.[8] investigated inverting the global perceptron, by using global history to address the weights and using a combination of global history and address bits as inputs for the perceptron. This allowed them to prefetch the weights from the weight tables, eliminating the latency of the SRAM-Arrays, but they still incurred the delay of the Wallace-tree adder. There is no need for this kind of inversion for a pipelined perceptron, since the critical path is already reduced to a small SRAM-array and a single adder. They also looked at incorporating concepts from traditional caches, i.e. two-level caching of the weights, pseudo-tagging the perceptrons and adding associativity to the weight tables.

Most of the improvements proposed by Seznec and Ipek et al. are orthogonal to our work and exploring possible synergies could be the subject of future work.

3. DELAY IN BRANCH PREDICTION

An ideal branch predictor uses all the information which is available at the end of the previous cycle to make a prediction in the current cycle. In a table-based branch predictor this would mean using a certain mix of address, path and history bits to index into a table and retrieve the state of a two-bit saturating counter (a very simple finite state machine), from which the prediction is made.

3.1 Overriding Prediction Schemes

Because of the delay in accessing the SRAM-Arrays and going through whatever logic is necessary, larger predictors oftentimes cannot produce a prediction in the same cycle. This necessitates the use of a small but fast single cycle predictor to make a preliminary prediction, which can be overridden [12] several cycles later by the main predictor. Typically this is either a simple bimodal predictor or, for architectures which do not use a BTB, a next line predictor as is used by the Alpha EV6 and EV7 [4].

This arrangement complicates the design of the front of the pipeline in several ways. Most obviously, it introduces a new kind of branch misprediction and necessitates additional circuitry to signal an overriding prediction to the rest of the pipeline.

While traditionally processors checkpointed the state of all critical structures at every branch prediction, this method does not scale for processors with a very large number of instructions in flight. Moshovos proposed the use of selective checkpointing at low confidence branches [16]. Since the number of low confidence branches is much higher for the first level predictor than for the overriding predictor, this negates much of the benefit of selective checkpointing. Other proposals [1, 5] for processors with a very large number of instructions in flight similarly rely on some kind of confidence mechanism to select whether to checkpoint critical structures or not. As mentioned above, the overriding scheme introduces a new kind of branch misprediction. In a normal pipeline even without overriding, all structures which are checkpointed because of branch predictions must

predictor type	Amount of state to be checkpointed in bits
overriding perceptron	$\sum_{i=2}^{x-1} 1 + \lceil \lg(i-1) \rceil$ bits
ahead pipelined perceptron	$(w \cdot x) + \sum_{i=2}^{x-1} 1 + \lceil \lg(i-1) \rceil$ bits
table-based	$2^{x-1} - 1$ bits for most significant bits

Table 1: Amount of state to be checkpointed for each type of predictor. x is the pipeline depth of each predictor and w is the number of bits for each weight in the perceptron predictor.

depth of pipeline	Amount of state to be checkpointed in bits
13	133
18	195
20	221
32	377
34	405
37	447

Table 2: Example of amount of state to be checkpointed for an overriding perceptron with 8-bit weights. We use the pipeline depth determined to be optimal in [11] as examples.

be able to recover from a BTB misprediction, signaled from the front of the pipeline, or a full direction misprediction, which is signaled from the end of the pipeline. The case of the slower predictor overriding the faster one introduces a new possible recovery point, somewhere between the first two.

3.2 Ahead-Pipelined Predictors

A solution to this problem, which was introduced in [11], was to "'ahead pipeline"' a large gshare predictor. The access to the SRAM-Array is begun several cycles before the prediction is needed with the then current history bits. Instead of retrieving one two-bit counter, 2^m two-bit counters are read from the table, where m is the number of cycles it takes to read the SRAM-Array. While the array is being read m new predictions are made. These bits are used to choose the correct counter from the 2^m counters retrieved from the Array.

In an abstract sense, the prediction is begun with incomplete or old information and newer information is injected into the ongoing process. This means that the prediction can stretch over several cycles, with the only negative aspect being that only a very limited amount of new information can be used for the prediction. An ahead pipelined predictor obviates the need for a separate small and fast predictor, yet it introduces other complications. In the case of a branch misprediction, the state of the processor has to be rolled back to a checkpoint. Because traditional predictors only needed one cycle, no information except for the PC (which was stored anyway) and the history register(s) were needed.

3.3 Checkpointing Ahead-Pipelined Predictors

For an ahead pipelined predictor, all the information which is in flight has to be checkpointed or the pipeline would incur several cycles without a branch prediction being made. This would effectively lengthen the pipeline of the processor, increasing the branch misprediction penalty. The reason that an ahead pipelined predictor can be restored from a checkpoint on a branch misprediction and an overriding predictor cannot, is that the ahead pipelined predictor only uses old information to retrieve all the state which is in flight, while the overriding predictor would use new information, which would be invalid in case of a branch misprediction.

This problem was briefly mentioned in [19] in the context of 2BC-gskew predictor and it was noted that the need to recover in one cycle could limit the pipeline length of the predictor. In a simple gshare the amount of state grows exponentially with the depth of the branch predictor pipeline, if all the bits of new history are used. Hashing the bits of new history down in some fashion of course reduces the amount of state in flight.

For an overriding perceptron, all partial sums in flight in the pipeline need to be checkpointed. See Table 1 for the formulas used to determine the amount of state to be checkpointed. Since the partial sums are distributed accross the whole predictor in pipeline latches, the checkpointing tables and associated circuitry must also be distributed. The amount of state that needs to be checkpointed/restored and the pipeline length determine the complexity and delay of the recovery mechanism. Shortening the pipeline and/or reducing the amount of state to be checkpointed per pipeline stage will reduce the complexity of the recovery mechanism.

A final factor to consider is the loss of accuracy in an ahead pipelined predictor with increasing delay. Since this was not explicitly investigated in [9], we investigate the response of some basic predictors and the pipelined perceptron predictor to delay. The first predictor is the gshare predictor, since it serves as the reference predictor in so many academic studies of branch predictors. Unlike the gshare.fast introduced in [9], we vary the delay from zero to four cycles. Our ahead pipelined version of the gshare predictor also differs from the gshare.fast presented in [9], as can be seen in Figure 3.

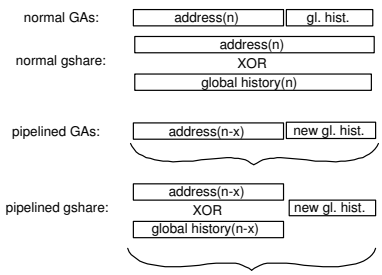

Figure 3: The ahead pipelined versions of each predictor uses the address information from several cycles before the prediction to initiate the fetch of a set of counters. In the case of the gshare predictor these are XOR'ed with the then current global history. The new bits of global history which become available between beginning to access the pipelined table and when the counters become available from the senseamps are used to select one counter from the 2^x retrieved to make the actual prediction.

In general, when we say a predictor has delay x, we mean that only address bits from cycle (n - x), where cycle n is the cycle in which the prediction is needed, are used. In the case of the gshare predictor, we XOR the address(n - x) during cycle (n - x) with the speculative global history shift register and start to fetch a group of 2^x 2-bit counters from the prediction table. We then use the newest x bits of global history, which become available while the table lookup is still in progress, to select one counter from this group.

A bimodal predictor can similarly be pipelined, by using only the address bits from address(n - x) to initiate the table read. The bimodal predictor in this case becomes similar to a GAs [22] (also known as *gselect* [15]), but it uses a past address as opposed to the present address used by a GAs.

4. THE AHEAD PIPELINED PERCEPTRON

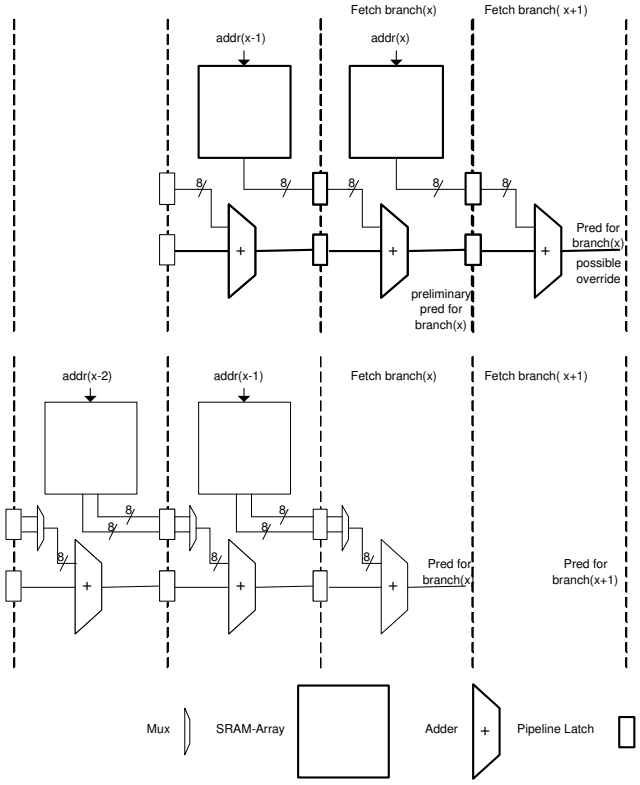

Figure 4: (top)The original proposal for a pipelined perceptron uses the current address in each cycle to retrieve the weights for the perceptron.
(bottom) Our proposed design uses addresses from the previous cycle to retrieve two weights and then chooses between the two at the beginning of the next cycle. Note that the mux could be moved ahead of the pipeline latch if the prediction is available early enough in the cycle.

To bring the latency of the pipelined path-based perceptron down to a single cycle, it is necessary to decouple the table access for reading the weights from the adder. We note that using the address from the cycle $n-1$ to initiate the reading of weights for the branch prediction in cycle n would allow a whole cycle for the table access, leaving the whole cycle when the prediction is needed for the adder logic. We can use the same idea as was used for the ahead pipelined table based predictors to inject one more bit of information (whether the previous branch was predicted taken or not taken) at the beginning of cycle n. We thus read two weights, select one based on the prediction which becomes available at the end of cycle n-1, and use this weight to calculate the result for cycle n. While this means that one less bit of address information is used to retrieve the weights, perceptrons are much less prone to the negative effects of aliasing than table based predictors.

In the case of a branch misprediction, the pipeline has to be restored the same as an overriding perceptron. However, because the predictor has to work at a one cycle effective latency, additional measures have to be taken. One possibility is to checkpoint not just the partial sums but also one of the two weights coming out of the SRAM-arrays on each prediction. Only the weights which were not selected need be stored, because by definition, when a branch misprediction occurred, the wrong direction was chosen initially. A second possibility is to also calculate the partial sums along the not chosen path. This reduces the amount of state that needs to be checkpointed to only the partial sums, but necessitates additional adders. A third possibility is to only calculate the next prediction, for which no new information is needed, and advance all partial sums by one stage. This would lead to one less weight being added to the partial sums in the pipeline and a small loss in accuracy. The difference between options two and three is fluid and the number of extra adders, extra state to be checkpointed and any loss in accuracy can be weighed on a case by case basis.

For our simulations we assumed the first option, and leave evaluation of the second and third option for future work. The weights are updated based on the committed outcome of the branch and the same training algorithm was used as in [11].

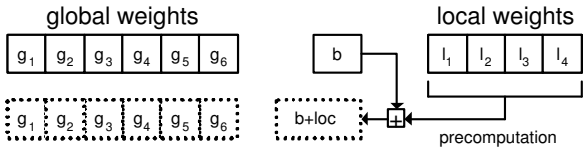

Figure 5: The local part of an alloyed perceptron can be precomputed as soon as the previous prediction for the branch is available. The partial sum consisting of the bias weight b and the sum of the local weights l_i multiplied with their respective speculative history bits can then be stored in place of the bias weight in a normal perceptron.

5. PRECOMPUTING A LOCAL PERCEPTRON

The use of alloyed branch history information [14] in combination with perceptrons was investigated in [11, 13] and shown to be effective. The use of alloyed history was dropped from the overriding perceptron, because the computation of the local part of the alloyed perceptron cannot be pipelined in the same way as the global part.

However it should be noted that when a branch x is predicted for the nth time, almost all the information necessary to calculate the local part of an alloyed perceptron is already

given. The only possible change is that the set of weights can be updated between the nth and (n+1)th prediction or that the outcome of the nth occurrence of branch x is mispredicted. The former is an infrequent event, since perceptrons are quickly trained to the desired outcome and then remain stable for long periods; the later requires a rollback to a non-speculative state.

Note that because of the delay in computing the partial sum, it is possible that the next occurrence of a branch happens before the result can be fully calculated. In effect, the branch overtakes or "loops" the predictor. In this case, the precomputed sum would be stale and most likely wrong. This is most likely to happen for tight inner loops, where the loop branch occurs every cycle or two. In such a case, the global part of an alloyed perceptron would capture the behavior of the branch and would implicitly override the stale local part of the perceptron. The effects of stale precomputed local history on different benchmarks is beyond the scope of this paper and is the target of future work.

Precomputation of a local perceptron is similar to a pipelining of a local history, two-level predictor as proposed in [21]. However, there is no table based predictor which can use so much address, history and path information to make a prediction.

The benefit of integrating a precomputed local perceptron with an ahead pipelined perceptron is two-fold:

First, it has been shown [14] that the use of local and global history will increase the accuracy of a predictor at a given hardware budget.

Second, by shifting weights from the global part of an alloyed perceptron to the local part we reduce the number of pipeline stages in the global part of the predictor. This means fewer partial sums to checkpoint and smaller pipelines to which the checkpoint/recovery information has to be propagated.

Third, several weights can be fetched from the same bank for the local perceptron, saving on decoder area and energy. The use of a Wallace-tree adder for the precomputation of the partial sum instead of n independent adders also saves die area and energy.

6. SIMULATION SETUP

We evaluate the different branch predictors using the 12 SPEC2000 integer benchmarks. All benchmarks were compiled for the Alpha instruction set using the Compaq Alpha compiler with the SPEC *peak* settings and all included libraries. Exploring the design space for new branch predictors exhaustively is impossible in any reasonable timeframe. To shorten the time needed for the design space exploration we used 1 billion instruction traces which best represent the overall behavior of each program. These traces were chosen using data from the SimPoint [20] project. Simulations were conducted using EIO traces for the SimpleScalar simulation infrastructure [3]. We used the sim-bpred simulator from the Simplescalar [3] suite for simulating the accuracy of all branch predictors. In all simulations we ran we assumed that the branch predictor would have to predict only one branch per cycle and that all updates to the weights were immediate. Previous studies have shown [18] that the second assumption can be made with only minimal impact on the accuracy of the results, and the there are known methods how to modify branch predictors to deliver more than one prediction per cycle. We leave the evaluation of the different branch predictors in terms of IPC for future work.

7. RESULTS

First, we will present results from ahead pipelined versions of basic predictors to show that the impact of delay is universal and has similar but not equal effects on all branch predictors. Prediction accuracy degrades in a non-linear fashion for most predictors with increasing delay, most probably due to increasing aliasing between branches. We will then go on to show the impact of delay on the overriding perceptron predictor, which behaves similarly to table based predictors with respect to increasing delay, despite its very different structure. Finally, we show that the alloyed ahead pipelined perceptron slightly outperforms all previous predictors at larger hardware budgets.

7.1 Table Based Predictors

Figure 6 shows the accuracy of a pipelined GAs predictor. We use only address bits to start the prediction and use the new history bits to select from the fetched predictions. This means that each predictor uses as many bits of global history as its delay in cycles. This of course implies that the predictor with 0 cycles of delay is in fact a bimodal predictor. As can be seen in Figure 6, the addition of global history bits increases the accuracy of such a predictor. For comparison we show non-pipelined GAs predictors with 0 to 4 bits of global history in Figure 7. Such predictors are more accurate than the pipelined GAs predictors with an equivalent number of global history bits. The gshare predictor in

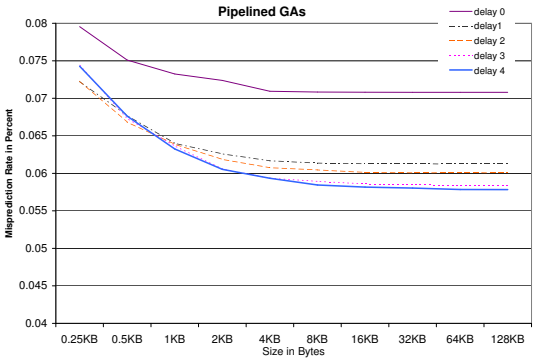

Figure 6: The impact on accuracy of using older addresses on a pipelined GAs predictor. The accuracy of the predictor actually improves with increasing delay, the inclusion of more bits of global history compensating for the effects of increasing delay.

Figure 8 shows a consistent loss of accuracy with increasing delay. However, the increase is not linear, with a predictor with one cycle delay showing only a very mild degradation in comparison to the normal gshare predictor.

7.2 Pipelined Perceptron Predictors

As can be seen in Figure 9, the perceptron predictor behaves similarly to the table based predictors in that the loss of accuracy with increasing delay is not linear. However, it exhibits different behavior from the gshare predictor in that the impact of delay decreases much more quickly with increasing size. We attribute this to the very small number of perceptrons for the smaller hardware budgets, which necessarily means that fewer address bits are used. The loss of

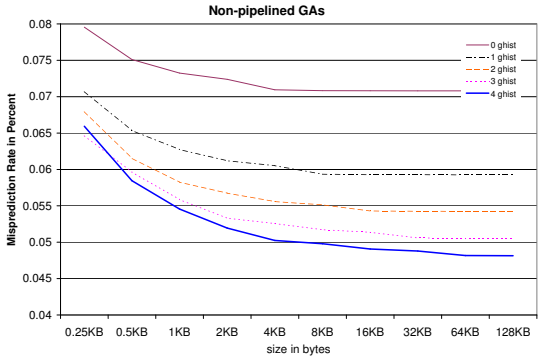

Figure 7: Accuracy of non-pipelined GAs predictors with zero to four bits of global history.

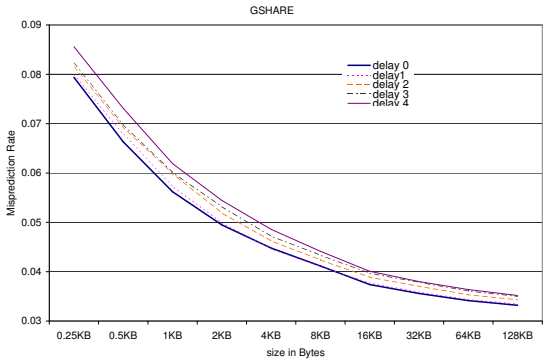

Figure 8: The impact on accuracy of using older addresses on a pipelined gshare predictor. Accuracy decreases with increasing delay. However a one cycle delay leads to only minimal loss of accuracy.

even one bit of address information seems to lead to much increased aliasing. At larger hardware budgets the increasing number of perceptrons and the tendency of the perceptron not to suffer from destructive aliasing to dominate. All the following simulations were done with the ahead pipelined perceptron as described in Section 4, which incurs one cycle of extra latency.

In Figure 10, we compare the overriding perceptron to the ahead pipelined perceptron with and without the use of alloyed history. We use 4 and 8 weights for local history, since those configuration were determined as optimal from our experiments.

We observe that the overriding perceptron predictor performs better at very small hardware budgets than our proposed perceptron. With increasing hardware budgets, the alloyed and ahead pipelined perceptron predictors close the gap and then overtake the pipelined perceptron predictor. We attribute this to the fact that the increase in size makes the increased aliasing in an ahead pipelined perceptron less important.

An interesting result is that the ahead pipelined perceptron outperforms the normal overriding perceptron very slightly at 32 and 64 kilobytes. This could indicate that the use of history information in addressing the weights is beneficial. However, the difference is so small that it could very well just represent noise from our set of benchmarks and traces. Further investigations in this area are necessary to come to

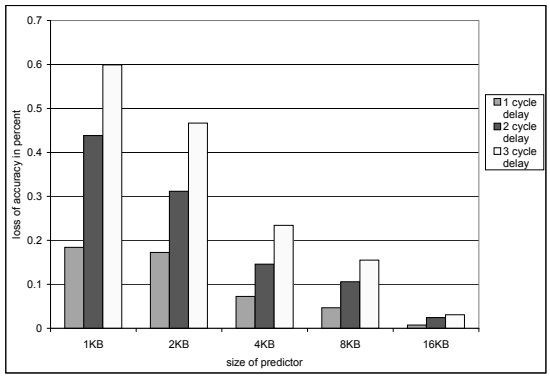

Figure 9: The impact on accuracy of using older addresses to fetch the perceptron weights.

a definitive conclusion.

The benefits of alloyed prediction grow with increasing hardware budgets, with the alloyed perceptron predictors lagging behind at small hardware budgets, but surpassing the purely global predictors at larger budgets.

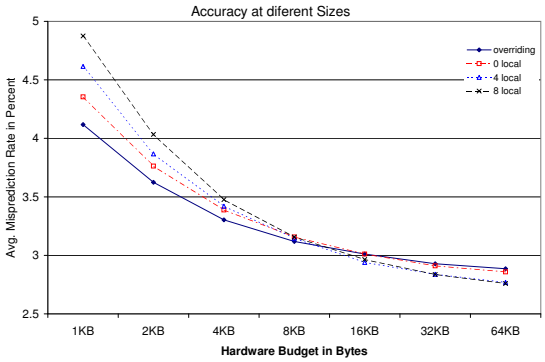

Figure 10: Average misprediction rates for hardware budgets from 1 to 64 kilobytes. All predictors where tuned for optimal accuracy for all sizes.

8. CONCLUSION

We have shown that a perceptron branch predictor can be ahead pipelined to acchieve effective one cycle access latency. This allows the removal of the preliminary predictor necessary for multicycle overriding predictors, considerably reducing the complexity of the branch prediction and fetch engine. It also allows the use of confidence based selective checkpointing mechanisms [1, 5, 16], which rely on a single accurate prediction mechanism. We also show that an alloyed perceptron can be pipelined in the way described above by precomputing the local history part of the perceptron. This further reduces the amount of state that needs to be checkpointed for a given hardware budget, by shortening the global predictor pipeline. When combining these two techniques we can achieve accuracies equal to or better than the best previously published pipelined perceptron predictor [11], while allowing single-cycle prediction latency.

9. FUTURE WORK

It should be noted that precomputation effectively decouples the access latency of the local perceptron from the cycle time of the branch predictor. The latency is thus only bound by the average latency between accesses to the same branch. As mentioned above, it might make sense to use a special loop predictor to filter out loop branches with high loop repetition counts(as has been done in the Pentium M [6]), because loop predictors are much more efficient in terms of hardware for these specific branches. However, some latency can be tolerated and the only critical factor could be area. This can be resolved, perhaps by moving the weight tables for the local perceptron away from the rest of predictor with only the precomputed sums stored locally.

As mentioned above, incorporating some of the improvements proposed by Seznec [17] and Ipek et al. [8] should further improve the accuracy of our predictor, especially at large hardware budgets.

Evaluating the impact of this new, very accurate single cycle predictor on the performance of a real processor is of great interest to us, and we are working on incorporating the ahead pipelined perceptron with a detailed model of a current high-performance processor.

10. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under grant nos. CCR-0105626, EIA-0224434, and a grant from Intel MRL. We would also like to thank Karthik Sankaranarayanan and Yingmin Li for valuable feedback on early versions of this paper.

11. REFERENCES

[1] H. Akkary, R. Rajwar, and S. T. Srinivasan. Checkpoint Processing and Recovery: Towards Scalable Large Instruction Window Processors. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 423. IEEE Computer Society, 2003.

[2] D. Boggs, A. Baktha, J. M. Hawkins, D. T. Marr, J. A. Miller, P. Roussel, R. Singhal, B. Toll, and K. S. Venkatraman. The Microarchitecture of the Intel Pentium 4 Processor on 90nm Technology. *Intel Technology Journal*, 8(1), February 2004.

[3] D. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors: The Simplescalar Tool Set. Technical Report CS-TR-1996-1308, University of Wisconsin-Madison, 1996.

[4] B. Calder and D. Grunwald. Next Cache Line and Set Prediction. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 287–296. ACM Press, 1995.

[5] A. Cristal, D. Ortega, J. Llosa, and M. Valero. Out-of-Order Commit Processors. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, page 48. IEEE Computer Society, 2004.

[6] S. Gochman, R. Ronen, I. Anati, A. Berkovits, T. Kurts, A. Naveh, A. Saeed, Z. Sperber, and R. C. Valentine. The Intel Pentium M Processor: Microarchitecture and Performance. *Intel Technology Journal*, 7(2):21–59, May 2003.

[7] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal*, 5(1):13, February 2001.

[8] E. Ipek, S. A. McKee, M. Schulz, and S. Ben-David. On Accurate and Efficient Perceptron-Based Branch Prediction. Unpublished Work.

[9] D. Jiménez. Reconsidering Complex Branch Predictors. In *The Ninth International Symposium on High-Performance Computer Architecture, 2003. HPCA-9 2003. Proceedings.*, pages 43–52, 2003.

[10] D. Jiménez and C. Lin. Dynamic Branch Prediction with Perceptrons. In *Proceedings of The Seventh International Symposium on High-Performance Computer Architecture*, pages 197–206, 2001.

[11] D. A. Jiménez. Fast Path-Based Neural Branch Prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 243. IEEE Computer Society, 2003.

[12] D. A. Jiménez, S. W. Keckler, and C. Lin. The Impact of Delay on the Design of Branch Predictors. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, pages 67–76. ACM Press, 2000.

[13] D. A. Jiménez and C. Lin. Neural Methods for Dynamic Branch Prediction. *ACM Trans. Comput. Syst.*, 20(4):369–397, 2002.

[14] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Alloyed Branch History: Combining Global and Local Branch History for Robust Performance. *International Journal of Parallel Programming*, 31:137–177, 2003/4.

[15] S. McFarling. Combining Branch Predictors. Technical Report TN-36, June 1993.

[16] A. Moshovos. Checkpointing Alternatives for High Performance, Power-Aware Processors. In *Proceedings of the 2003 International Symposium on Low Power Wlectronics and Design*, pages 318–321. ACM Press, 2003.

[17] A. Seznec. Redundant History Skewed Perceptron Predictors: pushing limits on global history branch predictors. Technical Report 1554, IRISA, September 2003.

[18] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides. Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 295–306. IEEE Computer Society, 2002.

[19] A. Seznec and A. Fraboulet. Effective Ahead Pipelining of Instruction Block Address Generation. In *Proceedings of the 30th Annual International Symposium on Computer architecture*, pages 241–252. ACM Press, 2003.

[20] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior, 2002. Tenth International Conference on Architectural Support for Programming Languages and Operating Systems, October 2002. http://www.cs.ucsd.edu/users/calder/simpoint/.

[21] T.-Y. Yeh and Y. N. Patt. Two-Level Adaptive Training Branch Prediction. In *Proceedings of the 24th Annual International Symposium on Microarchitecture*, pages 51–61. ACM Press, 1991.

[22] T.-Y. Yeh and Y. N. Patt. A Comparison of Dynamic

Branch Predictors that use Two Levels of Branch History. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 257–266. ACM Press, 1993.