The University of Texas at Austin Department of Computer Sciences Technical Report TR-08-12

Low-Power, High-Performance Analog Neural Branch Prediction

Renée St. Amant   Daniel A. Jiménez*   Doug Burger

The University of Texas at Austin   *Rutgers University
Department of Computer Sciences   Department of Computer Science

Abstract

Shrinking transistor sizes and a trend toward low-power processors have caused increased leakage, high per-device variation and a larger number of hard and soft errors. Maintaining precise digital behavior on these devices grows more expensive with each technology generation. In some cases, replacing digital units with analog equivalents can allow similar computation to be performed at higher speed and lower power. The units that can most easily benefit from this approach are those whose results do not have to be precise, such as various types of predictors. This paper describes an analog implementation of a neural branch predictor, which uses current summation to approximate the expensive dot-product computation required in digital perceptron predictors. The analog neural predictor we simulate is able to produce an accuracy equivalent to a digital neural predictor that requires 128 additions per prediction. The analog version, however, can run in 200 picoseconds, with the analog portion of the prediction computation less than 0.4 milliwatts at a 45 nm technology, which is negligible compared to the power required for the table lookups in this and conventional predictors.

1 Introduction

Branch prediction remains one of the key components of high performance in processors that exploit single-thread performance. Modern branch predictors achieve high accuracies on many codes, but further improvements are needed if processors are to continue improvement of single-threaded performance. Although the industry has recently turned to multi-core designs, some of which do not incorporate branch prediction [1], improving single-core performance will be essential once chips can contain more processors than the average number of available threads.

Neural branch predictors have shown promise in attaining high prediction accuracies, able to accurately predict linearly separable branches. While the recently proposed L-TAGE [19] predictor achieves slightly higher accuracy than the published neural designs, it is unclear which class of predictor will eventually reach the highest accuracies, since both classes show continuing improvements.

The perceptron predictors, while competitive in accuracy, have significantly worse power and energy characteristics than even the complex L-TAGE predictor. The requirement of computing a dot product for every prediction, with potentially tens or even hundreds of elements, has limited perceptron predictors to be an interesting research subject, not suitable for industrial adoption in their current form.

This paper evaluates a neural predictor design that uses analog circuits to implement the power-intensive portion of the prediction loop. Perceptron weights, stored digitally, are converted into analog currents, and are combined with other weights using current summation. The design steers the current representing the magnitude of each weight to one of two wires based on the sign of that weights contribution to the perceptron output. A comparator determines which wire has a higher current, and produces a single-bit prediction, effectively operating as an ADC. By adjusting the widths of the transistors that convert weights to current, each weight can trivially be scaled by a predetermined factor to adjust its importance to finding branch correlation, improving overall prediction accuracy.

We evaluate the design using both Cadence circuit simulation and predictor simulation using standardized branch traces. Using these “analog-assisted digital” techniques, we a show improved prediction accuracy of 5.5% over the best previously proposed neural predictor and comes within 3% of the accuracy of the recently proposed L-TAGE predictor. This mixed-signal neural design is able to issue predictions in 200 picosecond in a 45nm technology, consuming less than one milliwatt of power. Designs such as these are a good match not only for branch prediction, but for any computation that be approximate in nature, in future CMOS regimes where the devices behave much more like analog devices, and where power is the dominant constraint.

2 Related work

This section briefly reviews the concept of neural branch prediction and discusses prior research on analog circuits for neural networks.

2.1 The Perceptron Predictor

Most proposals for neural branch predictors derive from the perceptron branch predictor [6, 7]. In this context, a perceptron is a vector of $h + 1$ small integer weights where h is the *history length* of the predictor. A table of n perceptrons is kept in a fast memory. A branch history shift register of the h most recent branch outcomes (1 for taken, -1 for not taken) is also kept. The table of perceptrons and the shift register are analogous to the table of counters and shift register in traditional two-level predictors [26].

To predict a branch, a perceptron is selected using a hash function of the branch address, e.g. taking the address modulo n. The output of the perceptron is computed as the dot product of the perceptron and the shift register plus an extra *bias weight* in the perceptron. If the output is at least 0, then the branch is predicted taken; otherwise it is predicted not taken. The branch history shift register is speculatively updated and corrected on a misprediction.

When the branch outcome becomes known, the perceptron that provided the prediction is updated. If the prediction is incorrect, or if the magnitude of the output is below a certain threshold, the perceptron is trained. The bias weight is incremented or decremented if the branch is taken or not taken, respectively. For each branch outcome in the history register, the corresponding weight in the perceptron is incremented if the predicted branch has the same outcome, otherwise the weight is decremented. Arithmetic on perceptron weights saturates at the maximum and minimum values for the bit width of the weights. This training algorithm finds correlations between the current branch outcome and the outcomes of branches in the recent history. By strengthening positive correlations and weakening negative correlations, the perceptron learns the behavior of the branch. If there is no correlation between two branches, the corresponding weight will tend to be 0. If there is high positive or negative correlation, the corresponding weight will have a large magnitude. Figure 1 illustrates the concept of a perceptron producing a prediction and being trained. The top row of bipolar values represents the perceptron input units from the branch history, 1 for taken and -1 for not taken. Weights learned by online training label the arcs between the input units and the output unit. The output is the dot product of the history elements and the weights. In this example, the perceptron incorrectly predicts that the branch is taken. The weights are adjusted so that the next time the same inputs are encountered the prediction will be not taken.

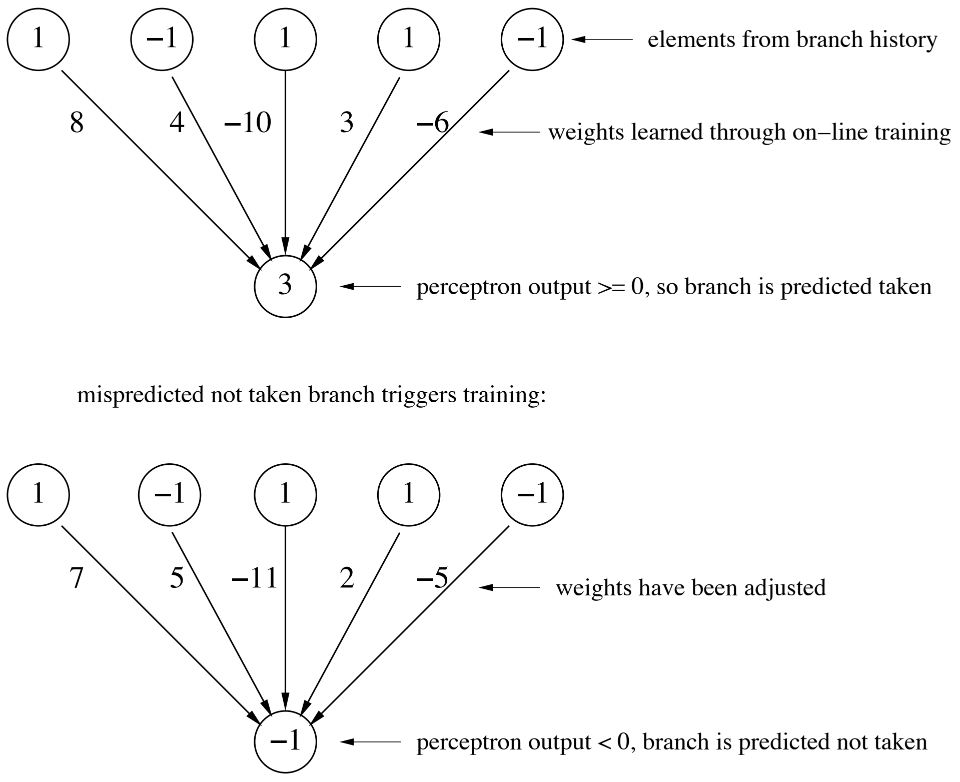
1
−1
1
1
−1
elements from branch history
8
4
−10
3
−6
weights learned through on−line training
3
perceptron output >= 0, so branch is predicted taken
mispredicted not taken branch triggers training:
1
−1
1
1
−1
7
5
−11
2
−5
weights have been adjusted
−1
perceptron output < 0, branch is predicted not taken

Figure 1: Perceptron prediction and training

The accuracy of the perceptron predictor compared favorably with other branch predictors of the time. Also, a number of optimizations put the perceptron predictor within reach of an efficient digital implementation. For instance, the dot product computation can be accomplished using an adder tree since multiplying by a bipolar quantity (i.e. 1 or -1) is the same as adding or subtracting [7]. The increment/decrement operations in the training algorithm can be quickly performed by $n + 1$ adders in parallel. Later research would significantly improve both latency and accuracy.

2.2 Improved Neural Branch Predictors

There have been many proposed improvements to the perceptron predictor. Seznec showed that accuracy can be significantly improved by using a redundant representation of the branch history [17]. Jiménez improved both latency and accuracy with the path-based neural predictor [2]; this proposal chooses the weights for a perceptron based on the path leading to the branch to be predicted, allowing for an ahead-pipelined implementation. This idea was further refined and generalized as piecewise linear branch prediction [3]. Ahead-pipelining allows a branch predictor to mitigate latency by taking actions related to prediction well before the branch to be predicted has been fetched [20]. Seznec combined the summation and training approach to aggregating predictions from neural predictors with the ideas of hashing into several tables of counters to produce O-GEHL, a neural-inspired predictor with high accuracy [18].

2.3 Branch Predictor State of the Art

Neural predictors have been among the most accurate in the literature, but recently Seznec introduced L-TAGE, a predictor based on prediction by partial matching that is both more accurate and more feasible in terms of implementation than previous neural designs [19]. L-TAGE took first prize at the second Championship Branch Prediction competition (CBP-2).

2.4 Analog Circuits for Neural Networks

There has been an extensive amount of research on the use of analog circuits to model neural networks [11, 15]. With the resurgence of interest in neural networks in the late 1980s and advancements made in computing hardware, parallels were drawn between computation in the human brain and the computing capabilites of machines. Analog circuits were a natural choice for implementation because the brain's signals more closely resemble analog electrical signals than digital ones.

Neural networks are often characterized by a compute-intensive dot-product operation. This operation has been implemented most efficiently with analog current summing techniques [9, 16]. In these techniques, weights are represented by currents and simply summed using Kirchhoff's current law. In [9], Kramer explores charge and conductance summation in addition to current summation and describes an efficient dot-product computation array. The work of Schemmel et al. [16] most closely resembles our own. In this work, the authors present a mixed-mode VLSI design that store weight values in analog current memory cells. Current values are summed on an excitatory or inhibitory input line based on a sign bit for each weight. The total excitatory and inhibitory currents are then compared to produce an output.

3 Effective Path-Based Neural Prediction

The original path-based neural predictor computes the dot product of a vector of weights chosen according to the path leading up to the branch to be predicted. The computation is ahead-pipelined through an array of adders so that the summands for the dot product can be summed before the branch to be predicted is fetched. This scheme was motivated by the need to improve the latency of the perceptron predictor, but it also fortuitously improved accuracy, path information improves the ability of the predictor to find correlations.

We consider an improved path-based neural predictor assuming that the power-intensive constraints on the dot product computation are lifted. We use this improved design later to guide a practical implementation using a combination of analog and digital circuits. However, the analog design makes feasible two modifications to the path-based neural predictor algorithm, which improve its overall accuracy.

- *No ahead-pipelining:* Ahead-pipelining in the original path-based neural branch predictor led to a loss of potential accuracy because, when the weights are being collected along the path to a branch, it is not clear for which branch they will be used. We will not ahead-pipeline the predictor so we can use information about which branch is being predicted to better choose the weights for that branch. This feature is accomplished by incorporating the address of the branch to be predicted in the hash function used to choose weights.
- *Coefficients:* Accuracy can be improved by multiplying each weight in the dot product computation by a coefficient. We have found through experimentation that some weights are more correlated with branch outcome than others. Weights related to more recent history tend to have a higher correlation with branch outcome than other weights. Figure 2 illustrates this concept for a perceptron predictor with a history length of 128. The x-axis represents the position in the history of a weight ($x = 0$ represents the bias weight). The y-axis gives the average correlation coefficient (Pearson's r) between actual branch outcome and the prediction obtained by using only the weight in position x. Clearly, the bias weight has the highest correlation with branch outcome. The first few weights are also highly correlated with branch outcome, while the remaining weights are less correlated.

By multiplying weights with coefficients proportional to their correlation, the predictor's overall accuracy is higher. We compute these coefficients using a function $f(i)$ fitted to the correlation coefficients, also illustrated in Figure 2, which was was generated using the publically distributed traces for the CBP-2 competition.

We also modify the organization of the weights. In the original paper, the number of weights vectors was chosen as an arbitrary integer. We restrict this number to a power of two so that the table of weights can easily be implemented using SRAM memories. We group weights into blocks of 8 columns for more efficient addressing, with a separate memory for the bias weights. Thus, instead of having to keep $h + 1$ separate memories, we keep only $h/8 + 1$. Figure 3 shows a high-level diagram of the data path for the prediction step. This figure illustrates the components of the algorithm for which digital implementation is infeasible. In the next section we will see how these components can be implemented with analog circuits.

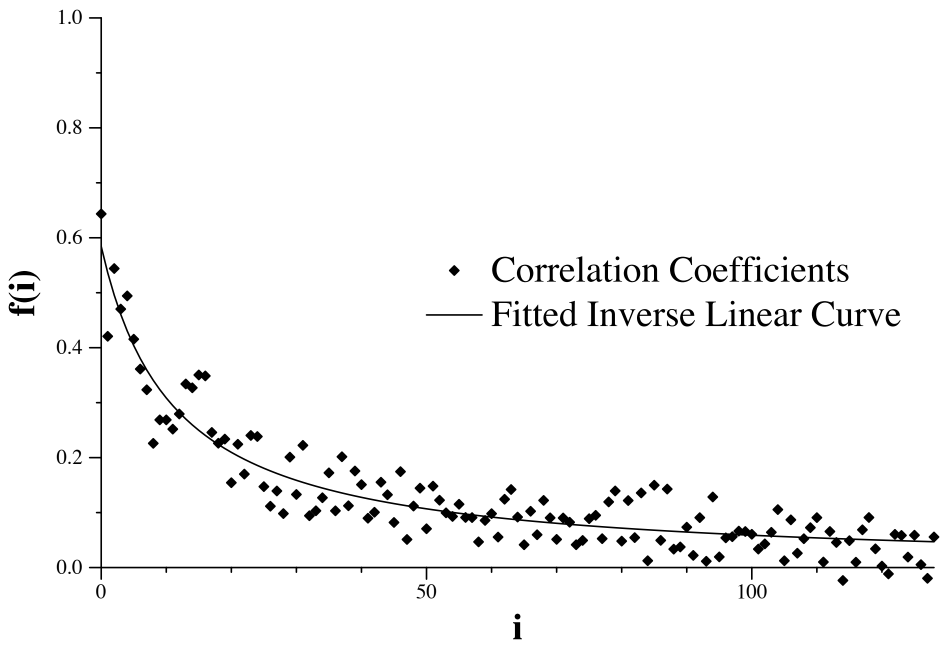

Figure 2: Correlation between weights and branch outcome

3.1 Detailed Neural Algorithm

The neural prediction algorithm presented below differs from previous neural branch predictors. Constraints imposed by the digital implementation of a dot product are removed, and the idea of redundant history is combined with the idea of a path-based predictor. The organization of the weights is also different, requiring fewer tables.

3.2 Prediction

Figure 4 shows our prediction algorithm. There are a number of terms used in the algorithm:

h: The dot product is a sum of h correlating weights plus 1 bias weight where h is fixed for a given design. We require that h be a multiple of 8 to accomodate our practical implementation. h would normally be the history length in other perceptron-based predictors. However, the number of history elements our predictor requires is considerably smaller than h because we redundantly use history elements.

W: This is an $n \times h + 1$ matrix of small integer weights where n is fixed for a given design. Column 0 of the matrix holds the bias weights, i.e., weights that track the tendency of a branch to be taken without respect to other branches. Columns 1 through $h + 1$ hold the correlating weights, i.e., the weights that track the correlation of branch outcome with branches in the history. These latter columns are divided into blocks of 8 weights that are all held in a single table, i.e., columns 1 through $h + 1$ are divided into $h/8$ memories that are independently addressed.

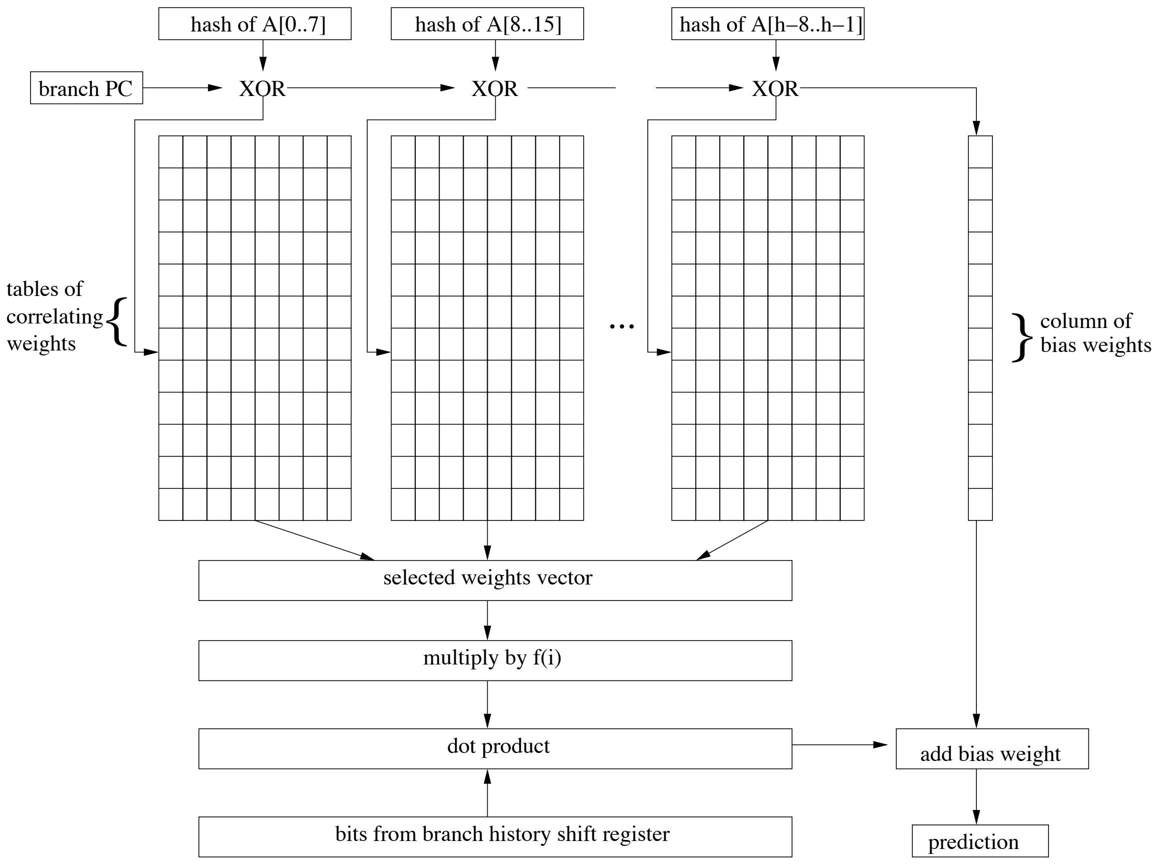

Figure 3: Prediction data path

H: This is the branch history shift register. It is an array of bipolar values with indices 1 through h. $H[i]$ holds the outcome (1 for taken, -1 for not taken) of the i^{th} most recent branch. With the *select_history* function described below, H needs to hold at most $h/4+8$ history elements, e.g. if $h=128$ then H holds 40 bits of history.

A: This array holds the $\log_2 n$ lowest-order bits of recent branch addresses. $A[i]$ holds bits from the address of the branch that produces the outcome recorded in $H[i]$.

In addition, there are three special functions used in the algorithm:

select_history This function selects an 8-entry portion of the H array. Depending on the design of the *select_history* function, redundant history [17] may be used. Figure 5 shows the procedure for selecting 8 bits of history. The procedure alternates between the first 8 bits of history and the 8 bits of history starting at a different position in H for different blocks. Note that many history elements are used redundantly. In our implementation described later, h is 128 but we use only the 40 most recent bits of branch outcome history.

hash The hash function accepts 8 branch addresses and combines them into a single integer to use as an index into one of the $h/8$ tables of correlating weights. The hash function we use is very simple: we choose lower order bits from each address to form a single integer.

f(i) This function accepts an integer i and computes a coefficient to be multiplied by the i^{th} weight. We have found experimentally that the best f is of the form $f(i) = 1/(a + bi)$, where a and b are small real numbers.

```
function prediction (pc: integer) : { taken , not_taken }
begin
    $sum := W[pc \bmod n, 0]$                          Initialize sum to bias weight
    for $i$ in $1$ .. $h$ by $8$ in parallel           For each of $h/8$ tables of weights...
        $k$ := (hash($A[i..i+7]$) xor $pc$) mod $n$    Select a row in this table
        for $j$ in $0$ .. $7$ in parallel              For each weight in this row...
            $q$ := $select\_history(H, i)[j]$          Select an element from the branch outcome history
            $sum := sum + W[k, i+j+1] \times q$        Accumulate into the dot product
        end for
    end for
    if sum $>= 0$ then                                 Compute the prediction from the sum
        prediction := taken
    else
        prediction := not_taken
    endif
end
```

Figure 4: Neural prediction algorithm to predict branch at address pc

```
function select ($H$: array[$1..h$] of bipolar, $i$: integer) :
        array[$0..7$] of bipolar
begin
    if $i/8$ is odd then                               Use a different selection scheme
        $j := i \bmod 8$                               for odd- and even-numbered blocks
    else
        $j := i \bmod 8 + i/4$
    endif
        select := $H[j]..H[j+7]$
end
```

Figure 5: Function for selecting history bits based on weight positions

3.3 Updating the Predictor

Predictor update consists of 3 phases, some of which can occur in parallel:

1. *Update histories.* When the outcome of a branch becomes known, it is shifted into H. The $\log_2 n$ lowest-order bits of address of the branch are shifted into A. A high-accuracy implementation must keep speculative versions of H and A that are restored on a misprediction.

2. *Train.* If the prediction was incorrect, or if the magnitude of the predictor output did not exceed a certain threshold, then the training algorithm is invoked. Just as in previous perceptron predictors, the weights responsible for the output are incremented if the corresponding history outcome matches the current branch outcome, or decremented otherwise. The weights use saturating arithmetic.

3. *Update threshold.* We use adaptive threshold training to dynamically adjust the threshold at which training will be invoked for a correct prediction. The adaptive threshold training algorithm is the same as the one used for O-GEHL [18]: the threshold is increased after a certain number of incorrect predictions, and decreased after a certain number of correct predictions whose outputs were not as large as the current threshold. Seznec observes that good accuracy is achieved when the training algorithm is invoked equally many times after correct and incorrect predictions. This threshold training strategy strives to achieve this balance [18].

3.4 Complexity of digital implementation

Implementing the algorithm as described using digital logic would be prohibitively expensive in terms of time and energy. Since all of the memories are indexed with a value that involves the branch PC, the prediction cannot be ahead-pipelined. Thus, computing the dot product would require that $h + 1$ values be added simultaneously. This operation would require a large number of power-hungry adders as well as a considerable latency [5]. The multiplications of each weight by $f(i)$ is even more problematic, requiring $h+1$ multipliers. Clearly this design is infeasible using only digital logic. However, as we will see in Sections 8 and 5, using a mixture of analog and digital circuitry yields a practical implementation.

4 Implementation of the Improved Predictor

There are several aspects to the computation that must be considered when constructing the mixed-signal design. We describe each of them below.

Retrieving the weights: The weights are grouped into a 1-byte wide array of bias weights and $h/8$ 8-byte wide arrays of history correlating weights. These structures can be efficiently implemented as SRAMs. Each iteration of the two **for** loops in Figure 4 proceed in parallel. The inner **for** loop retrieves one block in one of the memories. The weights tables are chosen to have a power of 2 number of blocks, so computing an index modulo n simply involves choosing lower-order bits of the index.

select_history and hash: Both of these functions simply route bits from one fixed position to another to move elements of the history or to provide an index into the weights tables. They involve no computation.

Multiplication of $W[k, i+j+1] \times q$*:* This multiplication computes a single summand for the dot product. Recall that q is a value from the branch outcome history that can only be 1 or -1. We represent -1 in the history as 0, and we represent the weights as 7-bit signed magnitude values, so this multiplication can be done by passing the magnitude bits and producing a new sign bit as the exclusive-or of the old sign bit and q.

Multiplying weights by coefficients: The function $f(i)$ is used when each DAC is designed. By adjusting transistor widths, the output of the i^{th} DAC will implicitly include a factor of $f(i)$. This multiplication is essentially free; it requires no extra computation. A single exclusive-or delay is incurred when the hash value is combined with the branch PC. This is actually the most computationally intensive component of the prediction step and is no more complex than generating an index into a table for the *gshare* branch predictor [10].

Summation of values: We separate the values to be summed into positive and negative values *sum_pos* and *sum_neg*, adding the positive values to *sum_pos* and adding the magnitudes of the negative values to *sum_neg*. These operations are performed by using simple digital to analog converters (DACs) whose inputs are the magnitudes of the values and whose outputs go to one wire for positive values and another wire for negative values.

Computing the prediction: sum_pos and *sum_neg* are converted from a current to a voltage and given as input to a comparator to determine which is greater. If *sum_pos* is greater, then the circuit predicts that the branch will be taken. If *sum_neg* is greater, then the circuit predicts not taken.

Training: Training is done similarly to the original path-based neural predictor. If the branch is predicted incorrectly or the dot product does not exceed some threshold then the fully digital training algorithm is invoked. Determining whether the dot product exceeds the threshold value in this design requires another comparator to produce a *training signal*.

5 Analog Circuit Design

This section describes an analog implementation of the neural predictor presented above. Its primary function is to efficiently compute the dot-product of 129 signed integers, represented in sign magnitude form, and a binary vector to produce a taken or not taken prediction. In addition, it produces a train / don't train output. It utilizes analog current steering and summation techniques to execute the dot-product operation. The analog circuit design shown in Figure 6 consists of three major components: current-steering digital to analog converters (DACs), a current to voltage converter, and a comparator.

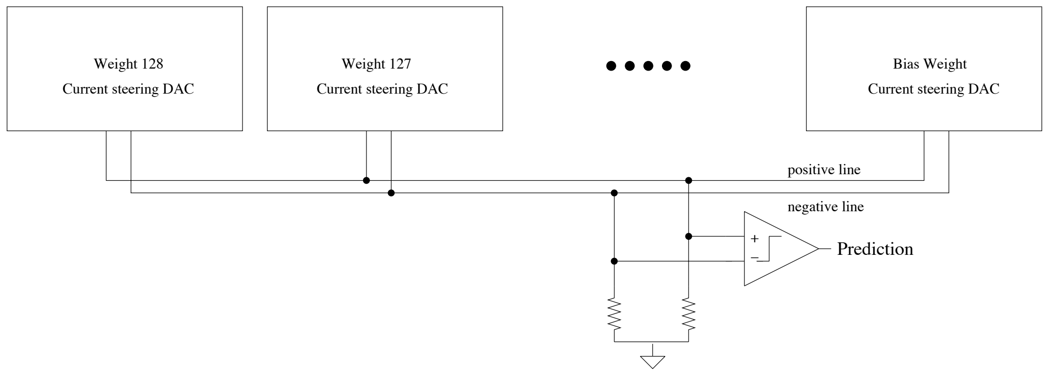

Figure 6: Top-level diagram of an analog neural predictor

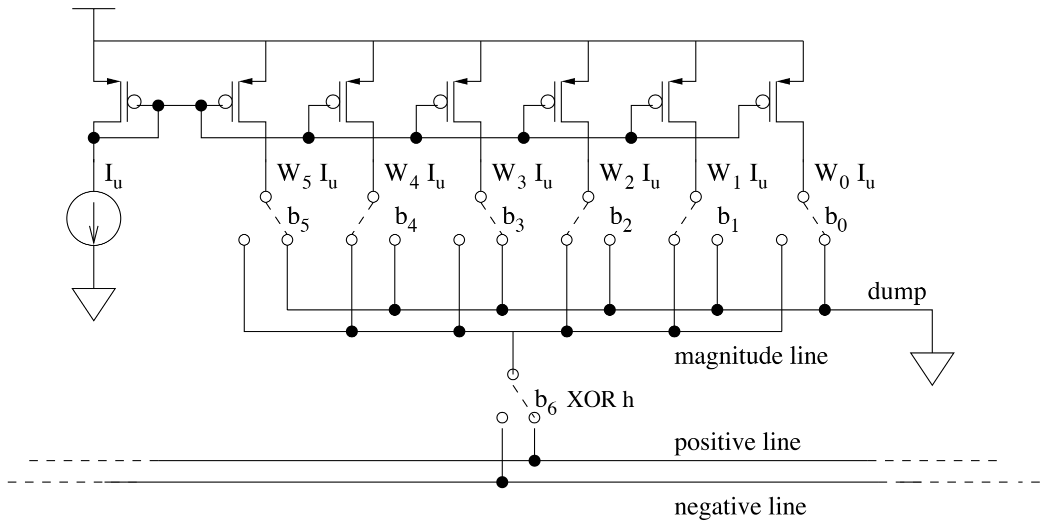

Figure 7: A 6-bit current-steering DAC for one weight. A current proportional to the width of a transistor *W* is steered by the corresponding digital weight bit *b* to the magnitude line if *b* is 1 and to ground otherwise.

Binary Current-Steering DACs With digital weight storage, DACs are required to convert digital weight values to analog values that can be combined efficiently. Although the perceptron weights are 7 bits, 1 bit is used to represent the sign of the weight and only 6-bit DACs are required. Current-steering DACs are able to operate at several GHz and were chosen because of their high speed as well as their simplicity. We use a DAC for each weight, each consisting of a current source and bias transistor as well as transistors corresponding to each bit in the weight. The source and gate nodes of all transistors are tied together so that the gate voltage V_G and source voltage V_S of all transistors are the same. The current source fixes the current through the bias transistor; consequently, because the transistors all share the same $V_{GS}(V_G - V_S)$ value, the current though the bias transistor is "mirrored" in the other transistors according to the current equation $I_D = \frac{\mu C_{ox}}{2} \frac{W}{L}(V_{GS} - V_t)^2$ [8]. This configuration, known as a current mirror, is a common building block in analog circuit design. Notice that the mirrored current is also proportional to *W/L*, where *W* is the width and *L* is the length of the transistor. By holding *L* constant and setting *W* differently for each bit, we draw a current proportional to *W*. This approach supports near-linear digital to analog conversion. For example, for a 4-bit base-2 digital magnitude, we would set the transistor widths to 1, 2, 4, and 8 and draw currents I, 2I, 4I, and 8I, through the respective transistors. The current through each transistor whose corresponding weight bit is 1 is steered to the "magnitude" line. See Figure 7. By the simple properties of Kirchhoff's current law, the magnitude line contains the sum of the currents whose weights bits are 1 and thus approximates the digitally stored weight.

The magnitude value is then steered to a "postive" line or "negative" line based on the xor of the sign bit for that weight and the appropriate history bit, effectively multiplying the signed value by the history bit. The postive and negative lines are shared across all weights and again by Kirchhoff's current law, all positive values are added together and all negative values are added together.

Current to Voltage Converter The current on the positive and negative lines are converted to voltages using two resistors. The current on each line passes through a resistor creating a voltage that is used as input to the preamplifier and comparator.

Preamplifier and Comparator Preamplifiers are commonly used in conjunction with comparators in traditional analog circuit design. The preamplifier can amplify the voltage difference before inputing the new voltages to the comparator, which improves comparator accuracy, and it can act as a buffer that prevents undesirable kickback effects [8]. The preamplifier requires only a small amount of circuitry, namely a bias current, two transistors, and two resistors. A track-and-latch comparator design [8] was chosen based on its high speed capability and simplicity. It compares a voltage associated with the magnitude of the positive weights and one associated with the magnitude of the negative weights. The comparator functions as a one-bit analog to digital converter (ADC) and uses positive feedback to regenerate the analog signal into a digital signal. The comparator outputs a 1, or a taken prediction, if the postive line outweighs the negative line and a 0, or a not taken prediction, otherwise.

Training In addition to a one-bit taken or not taken prediction, the circuit must produce a one-bit training signal to indicate whether or not the weights should be updated. The training bit is set to 1 if the absolute value of the difference between the positive and negative weights is less then the threshold value, and set to 0 otherwise. The training bit is produced in a process analogous to the one described above. Training is not implemented in this study due to time constraints.

Flash storage Rather than using digital adders to increment or decrement digital weight values, weights could be stored in multi-level flash cells [9]. In this case, one cell would be associated with each weight, where the threshold voltage of that cell represents a weight value. When the input voltage at the gate is 0, the channel charge is zero. If the input at the gate is 1, channel charge is proportional to the difference between the gate voltage and the threshold voltage. Threshold voltages are updated by pulsing charge to the floating gate. Flash storage can achieve a compact representation of a large number of weights and reduce power by avoiding table reads and updates. We leave the implementation using flash storage as future work.

6 Methodolgy

This section describes our experimental methodology for evaluating the analog neural predictor.

Parameter	Value
Number of columns of W	129 (1 bias weight + 128 correlating weights)
Number of rows for W	W[0]: 2048, W[1..8]: 512, W[9..128]: 256
Bits per weight	7 for columns 0..56, 6 for columns 57..128

Table 1: Predictor parameters

6.1 Branch predictor simulator

We use a simulator derived from the CBP-2 contest infrastructure [4]. This trace-driven simulator simulates a branch predictor as a C++ class. The CBP-2 contest allowed competing predictors to use approximately 32KB of state to simulate implementable branch predictors. Our simulator also uses this hardware budget so that our results can be compared with previously published results.

The analog neural predictor implementation accepts a number of parameters such as history length, number of weights, transistor widths for each weight, etc. and uses data from the Cadence circuit simulations to compute the prediction and the training signal.

6.2 Tuning the predictor

We searched the design space of 32KB-limited analog neural predictors to choose the set of parameters that produce the best accuracy on a training set of traces. For this tuning, we use the set of traces provided in the CBP-2 infrastructure so that our results will be directly comparable to previously published results. These traces are derived from the SPEC CPU 2000 and SPEC jvm 98 benchmark suites. Each trace file represents a single simpoint [21] of 100 million instructions for the benchmark. (For testing the predictor, we use a different set of traces including SPEC CPU 2006.) To stay within the the 32KB budget, we allow the number of rows in W to vary based on the division of W into 8-weight columns. We also reduce the number of bits per weight in some of the tables. Table 1 shows the results of the tuning for the predictor geometry.

We use the inverse linear function defined as $f(i)$ in Section 3, scaled appropriately, to determine the widths of the transistors in each DAC. Table 2 shows the results of this sizing for some of the DACs; a complete listing for all 129 DACs is omitted for lack of space.

6.3 Testing the predictor

We produce traces compatible with the CBP-2 framework using representative simpoints from the SPEC CPU 2006 integer benchmarks as well as the SPEC CPU 2000 integer benchmarks not duplicated in SPEC CPU 2006. Although we use some of the same benchmarks as in CBP-2, there is no overlap in the traces in terms of portions of program intervals simulated. Thus, we use different traces for training and testing.

Column Number	**Transistor Widths**					
	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5
0 (bias)	1	2	4	8	16	32
1	1	2	4	8	15	30
2	1	2	3	7	13	26
3	1	1	3	5	11	21
10	-	1	2	3	7	14
20	-	1	1	2	5	9
128	-	1	1	2	4	8

Table 2: Excerpts from the list of transistor widths for the DACs that convert the 6-bit magnitudes of the weights to analog signals

6.4 Comparison to other predictors

We compare our predictor against two other predictors from the literature: piecewise linear branch prediction [3] and L-TAGE [19]. Piecewise linear branch prediction is a neural predictor with high accuracy but high implementation cost. L-TAGE was the winner of the realistic track of the CBP-2 contest and represents the most accurate implementable predictor in the literature. The simulation code for L-TAGE was developed based on the CBP-2 traces just as ours was, so we simply use the source code provided. For piecewise linear branch prediction, we exhaustively explore the search space for the set of parameters that yields the best accuracy on the CBP-2 traces. L-TAGE also includes a 256-entry loop predictor that accurately predicts loops with long trip counts; we include an identical loop predictor, included in the hardware budget, with our predictor as well as with piecewise linear branch prediction. We also modify piecewise linear branch prediction to use the adaptive threshold training from O-GEHL mentioned in Section 3. In addition, we simulated a variety of other predictors from recent literature, including O-GEHL, but we do not include those results because we find that L-TAGE is by far the most accurate of all predictors with constrained hardware budgets in recent literature.

6.5 Circuit simulation

We composed a transistor level implementation of the analog neural predictor in Cadence using Predictive Technology Models (PTMs) at 45nm [14]. These models are the standard for research and circuit design with immature technologies and they take into account numerous non-idealities that become important as transistor sizes shrink. We used Spectre for simulation and assumed a 1V power supply. The maximum delay through the circuit is 200ps and therefore it can safely be clocked at 5GHz.

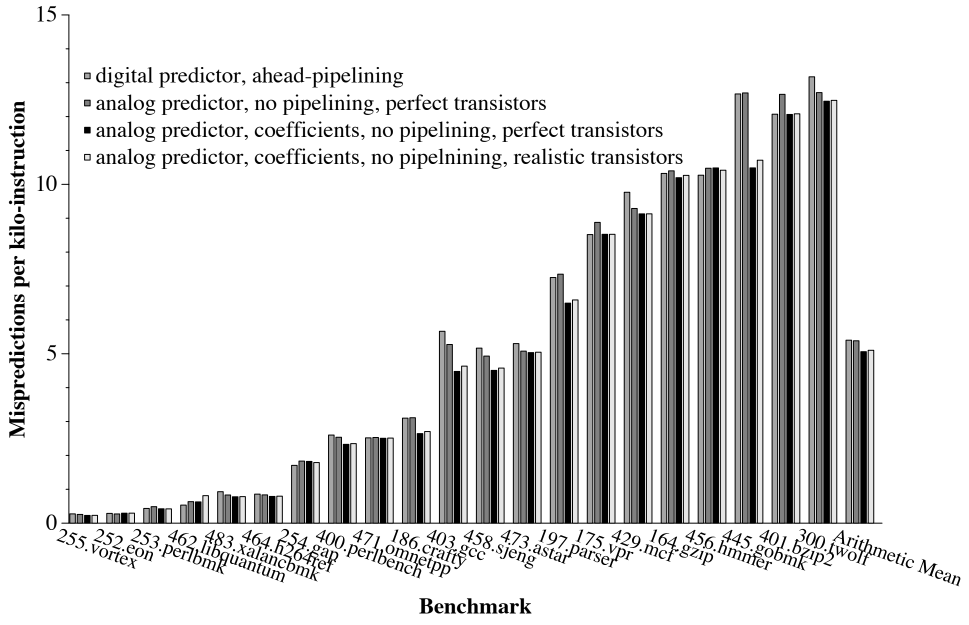

Figure 8: Simulated accuracies of various digital and analog neural predictors

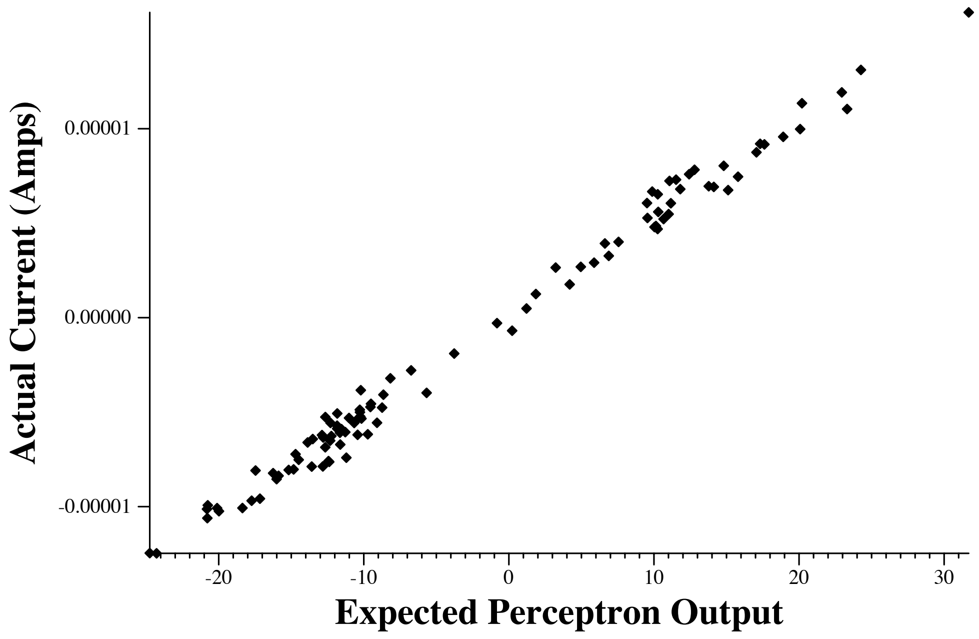

Figure 9: Expected perceptron output versus current difference

7 Results

7.1 Predictor Accuracy

We present results from simulating several versions of the neural predictor. The results show that an implementable analog neural predictor yields accuracy that is superior to previous less feasible digital predictors and

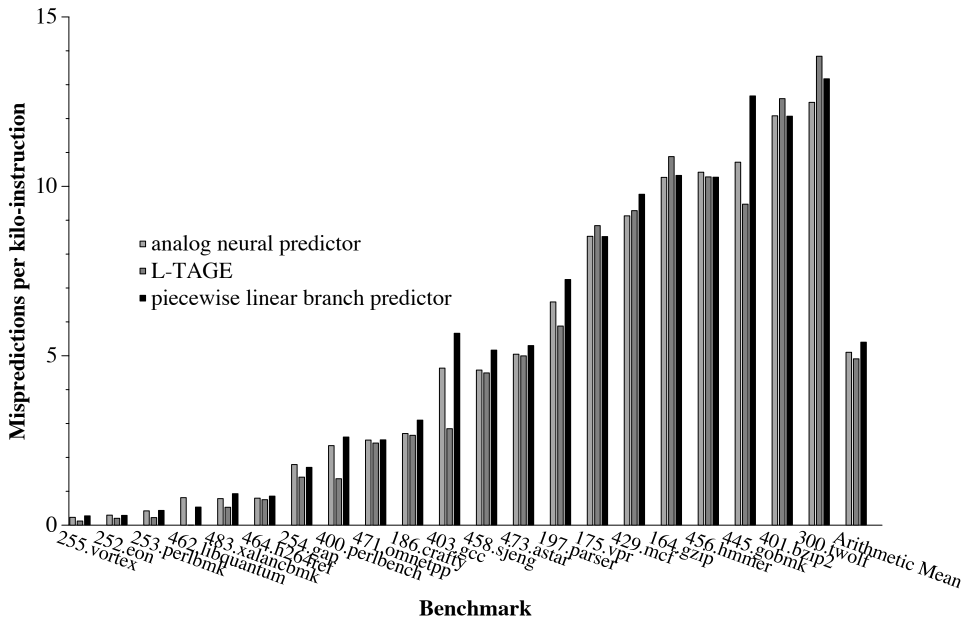

Figure 10: Accuracy of analog neural versus other predictors

competitive with the best branch predictor from the literature.

Figure 8 shows the benefit of moving from a digital to an analog predictor, as well as the impact of simulating real transistors. As a baseline neural predictor, we first simulate a fully digital version of the predictor equivalent to a piecewise linear branch prediction [3] augmented with a loop predictor and dynamically adjusted training threshold. This predictor achieves an average 5.40 mispredictions per kilo-instruction (MPKI). We then simulate an analog version of the predictor with no ahead pipelining, assuming that transistors behave perfectly linearly, which reduces the average MPKI to 5.38. We also simulate the analog predictor with no ahead pipelining and adjust transistor widths to take into account the coefficients from the $f(i)$ function. This predictor achieves an average 5.06 MPKI. Figure 9 shows that the actual current difference produced by the transistors and comparators is somewhat noisy compared with the expected perceptron output. We simulate the same predictor taking into account the non-linearity of the transistors; the average MPKI rises slightly to 5.10. This practical predictor reduces MPKI by 5.5% over the digital implementation. Figure 10 shows the accuracy of our predictor compared to L-TAGE [19], the most accurate implementable predictor to date.

7.2 Frequency of Training

The predictor update requires the use of an array of narrow up/down counters, potentially requiring significant dynamic power. Figure 11 shows how often the weights need to be adjusted for the various benchmarks. On average, the weights need to be adjusted 10% of the time; the other 90% of the time the adders are idle. This observation enables us to explore the potential of multiplexing fewer up/down counters over time.

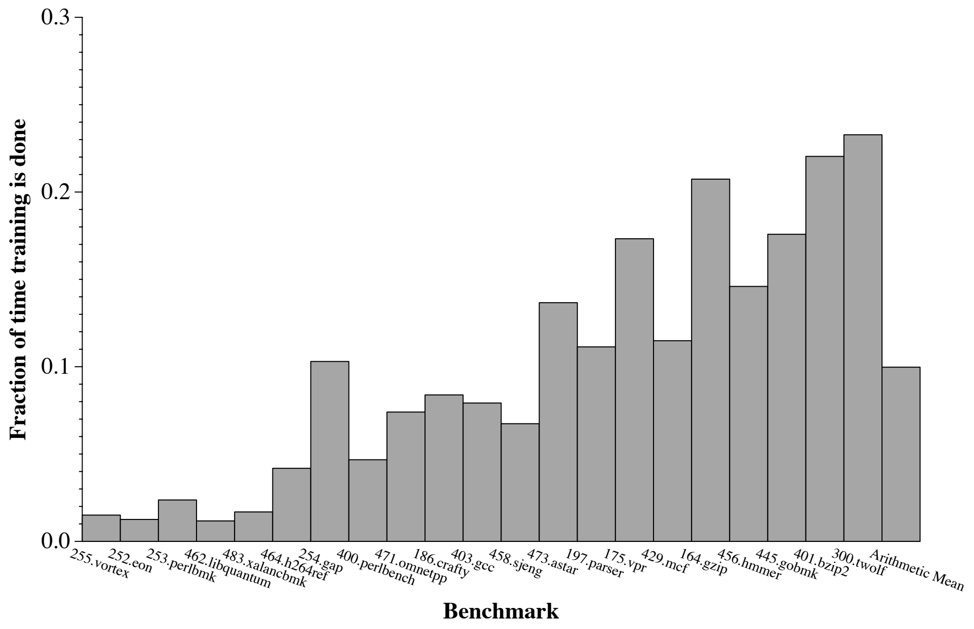

Figure 11: Fraction of dynamic branches that require training

7.3 Power consumption

Total power consumed by the the prediction step of the analog neural predictor includes table look-ups and analog computation. Power is measured by multiplying the supply voltage by the average current drawn from the power supply. With a 1V supply voltage, average power consumption is 0.38mW. Note that the same current is drawn from the power supply regardless of the weight inputs. Also, unlike digital implementations, where power consumption increases with frequency, the power consumption of our analog design remains constant as frequency increases because the average current drawn is constant.

We use CACTI 4.2 [24] with 45nm technology files to measure the dynamic power of the memories of the predictor. Modelling each of the 17 tables in the analog neural predictor as tagless memories with 8-byte lines we estimate that the total dynamic read power at a maximum frequency of 5.4GHz is 0.117 W. For comparison, we estimate the power of the various memory structures of L-TAGE at 0.112 W. Thus, the memory components of the two predictors have comparable dynamic power. Clearly, the analog computation of the dot product is a very small fraction of the total power of the predictor.

8 Implementation Issues

This section addresses issues related to mixed-signal design, in particular, environmental noise and testing.

8.1 Tolerance to noise

On-chip analog circuits are susceptible to subtrate noise caused by digital switching. When a large number of digital circuits are switching, they discharge currents into the substrate that cause the substrate voltage to bounce. Traditional techniques, such as guard rings, can be used to mitigate the affects of substrate noise [23]. A guard ring separates analog and digital circuits and creates a large capacitance that provides noise currents a low impedance path to ground.

8.2 Testing

Manufacturing testing contributes significantly to production cost and time-to-market. In the last few decades, design-for-test research has focused mainly on digital ICs; consequently, digital techniques used to reduce reliance on external testing equipment, such as automatic test pattern generation (ATPG), scan chains, and built-in self-tests (BISTs), have become sophisticated and cost effective [12]. Our design can utilize existing digital testing techniques because it includes a digital to analog converter and a one-bit analog to digital converter with only a small amount of analog circuitry in between. Scan chains are used to gain access to internal circuit components; testing data is scanned into a chain of registers from chip input pins, the clock is pulsed, and the results are captured in registers and scanned to chip output pins. In our design, various input weights can be scanned in and a one-bit prediction can be scanned out. The circuit component is determined to be "good" if its predictions are consistent with the expected prediction an acceptable percentage of the time. Weight inputs and expected outputs could also be stored in memory and a pass/fail signal could we routed off chip.

For finer grain failure detection, internal analog nodes can be accessed using analog scan chains. In [25] and [22], currents are shifted through the scan chains using current mirrors and switches. In [13], a BIST circuit is proposed to detect unexpected changes in the power supply current. This technique could be used to detect unexpected currents through the transistors in our design.

9 Conclusion

Branch predictors continue to evolve and improve, and it is still an open question whether conventional, multi-table based predictors (such as L-TAGE) or neural predictors will eventually prove the most accurate. This paper described a mixed-signal neural-based design that uses analog techniques to compute the computationally expensive part of the prediction step, using current summation and an analog comparator to perform the dot-product computation.

This design is the first neural predictor whose prediction step is competitive from a power perspective, requiring only 0.38 mW for the computation, in addition to the 117 mW for the digital table lookups, running at 5 GHz at a 45nm technology. By comparison, the table lookups for a comparably-sized L-TAGE predictor consume

112 mW, not counting the L-TAGE predictor's more complex hashing computations. Thus, the computation part of the neural predictor is negligible compared to the other components of these predictors. The dot-product function requires only 200 ps to compute its results and generate a prediction.

This design is the second most accurate predictor design published to date (using the union of the SPECINT 2000 and 2006 benchmarks); the L-TAGE predictor first published in December, 2006 is the most accurate, showing 3.7% fewer mispredictions per 1K instructions (4.91 as opposed to 5.1 for the analog perceptron). The best previous neural predictor, the Piecewise Linear Predictor, achieved an MPKI of 5.4 on the same benchmarks.

While this study showed how to build an accurate neural predictor with a low-power prediction computation, both the table lookups and the training steps (which occur after 10% of the predictions, on average) still require conventional amounts of power. Compared to conventional predictors such as L-TAGE, this analog neural predictor will require comparable table lookup power, and its update power will be higher. Future implementations may reduce the lookup and update power by pushing these functions into the analog portion of the design as well, using multi-level flash cells to adjust the weighted current directly, rather than performing a lightweight DAC.

By making more aggressive computation functions feasible in the prediction loop, analog techniques may open the door to more aggressive neural prediction algorithms, for instance the use of back propagation to compute non-linear functions of correlation. It remains to be seen whether this capability will appreciably reduce mispredictions in a power-efficient manner, but it is a promising direction.

In future process technologies, it is likely that CMOS devices will behave much more like analog devices than digital ones, with more leakage, noise, variation, and unreliability. We expect to see more applications of analog techniques assisting the digital designs to reduce computation power, particularly for computations that can be approximate, such as predictions or data that do not affect the control path of a program. In the long term, maintaining a firm digital abstraction over much of the computation may prove too expensive, leaving only the ability to accelerate workloads that can exploit low-power analog computation, while preventing the accretion of noise in the data.

References

[1] G. Grohoski. Niagara-2: A highly threaded server-on-a-chip. In *Hot Chips 18*, 2006.

[2] D. A. Jiménez. Fast path-based neural branch prediction. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*, pages 243–252. IEEE Computer Society, December 2003.

[3] D. A. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-32)*, June 2005.

[4] D. A. Jiménez. Guest editor's introduction. *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, 9, May 2007.

[5] 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 International Symposium on Microarchitecture (MICRO-33)*, pages 67–76, 2000.

[6] D. A. Jiménez and C. Lin. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High Performance Computer Architecture (HPCA-7)*, pages 197–206, January 2001.

[7] D. A. Jiménez and C. Lin. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems*, 20(4):369–397, November 2002.

[8] D. A. Johns and K. Martin. *Analog Integrated Circuit Design*. John Wiley and Sons, Inc., 1997.

[9] A. H. Kramer. Array-based analog computation. *IEEE Micro*, 16(5):20–29, October 1996.

[10] S. McFarling. Combining branch predictors. Technical Report TN-36m, Digital Western Research Laboratory, 1993.

[11] C. Mead. *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[12] L. S. Milor. A tutorial introduction on research on analog and mixed-signal circuit testing. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 45(10):1389–1407, October 1998.

[13] Y. Miura. Real-time current testing for a/d converters. *IEEE Design Test*, 13(2):34–41, 1996.

[14] Nanoscale Integration and Modeling (NIMO) Group at ASU. *Predictive Technology Models (PTMs)*. http://www.eas.asu.edu/ ptm/.

[15] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf. A reconfigurable vlsi neural network. *IEEE Journal of Solid-State Circuits*, 27(1):67–81, January 1992.

[16] J. Schemmel, S. Hohmann, K. Meier, and F. Schürmann. A mixed-mode analog neural network using current-steering synapses. *Analog Integrated Circuits and Signal Processing*, 38(2-3):233–244, February-March 2004.

[17] A. Seznec. Redundant history skewed perceptron predictors: Pushing limits on global history branch predictors. Technical Report 1554, IRISA, September 2003.

[18] A. Seznec. Analysis of the o-geometric history length branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, pages 394–405, June 2005.

[19] A. Seznec. A 256 kbits l-tage branch predictor. *Journal of Instruction-Level Parallelism (JILP) Special Issue: The Second Championship Branch Prediction Competition (CBP-2)*, 9, May 2007.

[20] A. Seznec and A. Fraboulet. Effective ahead pipelining of instruction block address generation. In *Proceedings of the 30th International Symposium on Computer Architecture*, San Diego, California, June 2003.

[21] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[22] M. Soma. Structure and concepts for for current-based analog scan. *Proceedings of the IEEE Costum Integrated Circuits Conference*, pages 517–520, 1995.

[23] D. K. Su, M. J. Loinaz, S. Masui, and B. A. Wooley. Experimental results and modeling techniques for substrate noise in mixed-signal integrated circuits. *IEEE Journal of Solid-State Circuits*, 28(4):420–430, April 1993.

[24] D. Tarjan, S. Thoziyoor, and N. P. Jouppi. Cacti 4.0. Technical Report HPL-2006-86, HP Laboratories Palo-Alto, June 2006.

[25] S. Wey, C.-L.; Krishman. Built-in self-test (bist) structures for analog circuit fault diagnosis with current test data. *IEEE Transactions on Instrumentation and Measurement*, 41(4):535–539, August 1992.

[26] T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th ACM/IEEE International Symposium on Microarchitecture*, pages 51–61, Nov. 1991.