Appears in the Proceedings of the 41^{st} Annual IEEE/ACM International Symposium on Microarchitecture

Low-Power, High-Performance Analog Neural Branch Prediction

Renée St. Amant
Department of Computer Sciences
The University of Texas at Austin
stamant@cs.utexas.edu

Daniel A. Jiménez
Department of Computer Science
The University of Texas at San Antonio
dj@cs.utsa.edu

Doug Burger
Microsoft Research
One Microsoft Way, Redmond, WA 98052
dburger@microsoft.com

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 allows 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. We introduce the Scaled Neural Predictor (SNP)*, a highly accurate prediction algorithm that is infeasible in a purely digital implementation, but can be implemented using analog circuitry. Our analog implementation, the* Scaled Neural Analog Predictor (SNAP)*, uses current summation in place of the expensive digital dot-product computation required in neural predictors. We show that the analog predictor can outperform digital neural predictors because of the reduced cost, in power and latency, of the key computations. The SNAP circuit is able to produce an accuracy nearly equivalent to an infeasible digital neural predictor that requires 128 additions per prediction. The analog version, however, can run at 3GHz, with the analog portion of the prediction computation requiring approximately 7 milliwatts at a 45nm technology, which is small compared to the power required for the table lookups in this and conventional predictors.*

1. Introduction

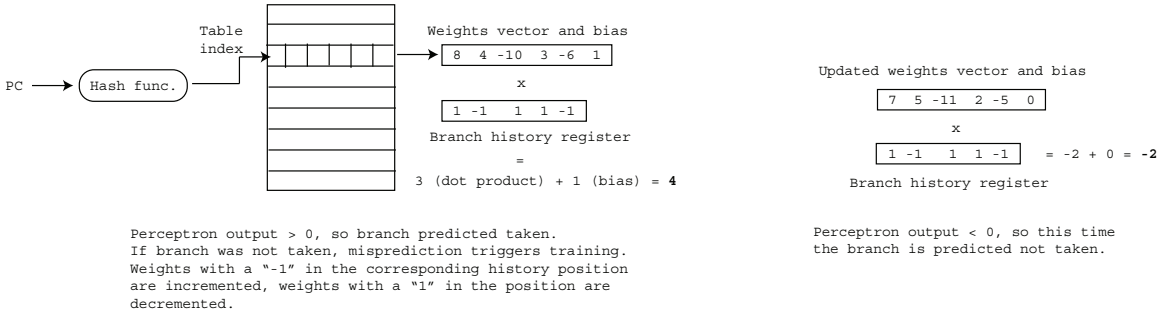
Branch prediction remains one of the key components of high performance in processors that exploit single-threaded performance. Modern branch predictors achieve high accuracies on many codes, but further developments are needed if processors are to continue improving single-threaded performance. Accurate branch prediction will remain important in general-purpose processors, especially as the number of available cores exceeds the number of available threads.

Neural branch predictors have shown promise in attaining high prediction accuracies and until recently achieved the highest accuracies. While the recently proposed L-TAGE [1] 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.

Neural predictors, however, have traditionally shown significantly worse power and energy characteristics than even the complex L-TAGE predictor. The original perceptron predictor achieved high accuracy with untenable power and latency characteristics. Subsequent designs reduced predictor latency at the expense of some accuracy, but still remained uncompetitive from a power perspective. The requirement of computing a dot product for every prediction, with potentially tens or even hundreds of elements, has limited neural predictors to be an interesting research subject, not suitable for industrial adoption in their current form.

This paper evaluates an aggressive neural predictor design that uses analog circuits to implement the power-intensive portion of the prediction loop. The circuit employs lightweight digital-to-analog converters (DACs) to convert digitally stored weights into analog currents, which are then combined using current summation. To compute the perceptron dot product, the circuit steers positive weights to one wire and negative weights to another. A comparator determines which wire has a higher current and produces a single-bit digital prediction, effectively operating as an analog-to-digital converter (ADC).

We introduce the *Scaled Neural Predictor (SNP)*, a prediction algorithm that employs two accuracy-improving features that are infeasible to implement in a strictly digital design. The first is the use of up-to-date information for generating the weights' index. Current neural predictors leverage a technique called *ahead pipelining* [2] to reduce the prediction latency, but which results in the path being hashed with stale PC values, losing potential accuracy. The second feature scales the weights by an inverse linear function, effectively multiplying each weight by a constant. We show that an analog implementation, the *Scaled Neural Analog Predictor (SNAP)*, achieves accuracy close to L-TAGE (5.18 mispredictions per thousand instructions compared to 4.91), differing



(a) The first time the branch is accessed

(b) The next time the branch is accessed

Figure 1. Perceptron prediction and training

by .27 MPKI on average, and an improvement of .22 MPKI over the piecewise linear branch predictor, one of the best previously proposed neural predictor designs. We evaluate SNAP using Cadence circuit simulation and show that this mixed-signal neural design is able to issue predictions at 3GHz in a 45nm technology, consuming less than eight milliwatts of power for the analog computation, and resulting in only a .12 MPKI decrease in accuracy from an infeasible digital SNP implementation.

2. Background on Neural Predictors

Most proposals for neural branch predictors derive from the perceptron branch predictor [3], [4]. 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 global history shift register of the h most recent branch outcomes (1 for taken, 0 not taken) is also kept. The shift register and table of perceptrons are analogous to the shift register and table of counters in traditional global two-level predictors [5], since both the indexed counter and the indexed perceptron are used to compute the prediction.

To predict a branch, a perceptron is selected using a hash function of the branch PC, e.g. taking the PC mod n. The output of the perceptron is computed as the dot product of the perceptron and the history shift register, with the 0 (not-taken) values in the shift registers being interpreted as -1. Added to the dot product is an extra *bias weight* in the perceptron, which takes into account the tendency of a branch to be taken or not taken without regard for its correlation to other branches. If the dot-product result is at least 0, then the branch is predicted taken; otherwise, it is predicted not taken. Negative weight values denote inverse correlations. For example, if a weight with a -10 value is multiplied by -1 in the shift register (i.e. not taken), the value $-1 * -10 = 10$ will be added to the dot-product result, biasing the result toward a taken prediction since the weight indicates a negative correlation with the not-taken branch represented by the history bit. The magnitude of the weight indicates the strength of the positive or negative correlation. As with other predictors, the branch history shift register is speculatively updated and rolled back on a misprediction.

When the branch outcome becomes known, the perceptron that provided the prediction may be updated. The perceptron is trained on a misprediction or when the magnitude of the perceptron output is below a specified threshold value. Upon training, both the bias weight and the h correlating weights are updated. The bias weight is incremented or decremented if the branch is taken or not taken, respectively. Each correlating weight in the perceptron is incremented if the predicted branch has the same outcome as the corresponding bit in the history register (positive correlation) and decremented otherwise (negative correlation) with saturating arithmetic. If there is no correlation between the predicted branch and a branch in the history register, the latter's corresponding weight will tend toward 0. If there is high positive or negative correlation, the weight will have a large magnitude.

Figure 1 illustrates the concept of a perceptron producing a prediction and being trained. A hash function, based on the PC, accesses the weights table to obtain a perceptron weights vector, which is then multiplied by the branch history, and summed with the bias weight (1 in this case) to form the perceptron output. In this example, the perceptron incorrectly predicts that the branch is taken. The microarchitecture adjusts the weights when it discovers the misprediction. With the adjusted weights, assuming that the history is the same the next time this branch is predicted, the perceptron output is negative, so the branch will be predicted not taken.

The accuracy of the perceptron predictor compared favorably with other branch predictors of the time and a number of optimizations put the perceptron predictor within reach of an efficient digital implementation. For instance, the dot-product computation can be done using an adder tree, since multiplying by a bipolar value (i.e. 1 or -1) is the same as adding or subtracting [4]. The increment/decrement operations in the training algorithm can be quickly performed by $n + 1$ counters in parallel. Later research would significantly improve both latency and accuracy [6], [7], [8].

2.1. Improvements to Neural Branch Predictors

Many improvements have been proposed for the perceptron predictor. Seznec showed that accuracy can be improved by using a redundant representation of the branch history [6]. Jiménez improved both latency and accuracy with the path-based neural predictor [7] using ahead pipelining, which was further refined and generalized as piecewise linear branch prediction [8]. Ahead pipelining gradually computes a prediction by adding weights ahead of time so that only the bias weight must be added to the total for a prediction to be made. While this technique reduces the latency required for predictions to manageable levels, it does not address the power required for the predictor; an equivalently large number of weight additions must be made at each prediction, the difference being that all but one of the additions are for future predictions. In addition, since the future weights are not indexed with the PC of the actual predicting branch, ahead pipelining results in predictor accuracy losses, though the use of path information improves accuracy beyond that of the original perceptron predictor.

Subsequently, Seznec combined the summation and training approach of neural predictors with the idea of hashing into several tables of counters to produce O-GEHL, a neural-inspired predictor with high accuracy [9]. Neural and neural-inspired predictors achieved the highest reported prediction accuracy until recently, when Seznec introduced L-TAGE [1], a predictor based on partial matching that took first prize at the second Championship Branch Prediction competition (CBP-2). L-TAGE is also more feasible to implement from a power perspective than the digital neural and neural-inspired predictors described to date.

2.2. Analog Circuits for Neural Networks

There has been an extensive amount of research on the use of analog circuits to model neural networks [10], [11]. 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 brain and the computing capabilities 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 [12], [13]. In these techniques, weights are represented by currents and summed using Kirchhoff's current law. In [12], Kramer explores charge and conductance summation in addition to current summation and describes an efficient dot-product computation array. Schemmel et al. present a mixed-mode VLSI design that stores 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 and then compared to produce an output. Although the analog predictor design described in this paper uses similar current-steering and comparison techniques, it differs in storage, analog/digital boundaries, and application. Other researchers have concurrently developed a voltage-mode mixed-signal design of the original perceptron predictor[14].

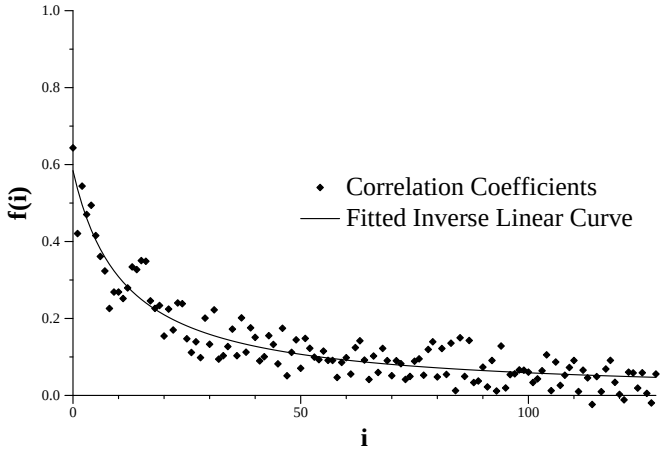


Figure 2. Weight position and branch outcome correlation

3. Analog-Enabled Neural Prediction Algorithm

The analog circuit design described in the next section makes two major improvements to a path-based neural predictor feasible through power and latency reductions. The resulting algorithm, the Scaled Neural Predictor, includes several other improvements that are not restricted to analog implementations. The two primary improvements made possible by the analog design are (1) the elimination of ahead pipelining and (2) the scaling of individual weights by predetermined coefficients, based on their history position. Both of these improvements increase predictor accuracy.

Removal of Ahead Pipelining: 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. That computation is necessarily ahead-pipelined through an array of adders to keep the latency tractably low. By adding the summands for the dot product before the branch to be predicted is fetched, some accuracy is lost because the weights chosen may not be optimal, given that they were not chosen using the PC of the branch to be predicted. Ahead pipelining thus increases destructive aliasing, though the latency benefits were worth the loss in accuracy. Because the analog design can sum all of the weights quickly when the actual branch is being predicted, the latency reduction afforded by ahead pipelining is unnecessary.

Scaling Weights by Coefficients: The vector of weights represents the contribution of each branch in a given history to predictability, but each branch does not contribute equally; unsurprisingly, more recent weights tend to have a stronger correlation with branch outcomes. Figure 2 quantifies this

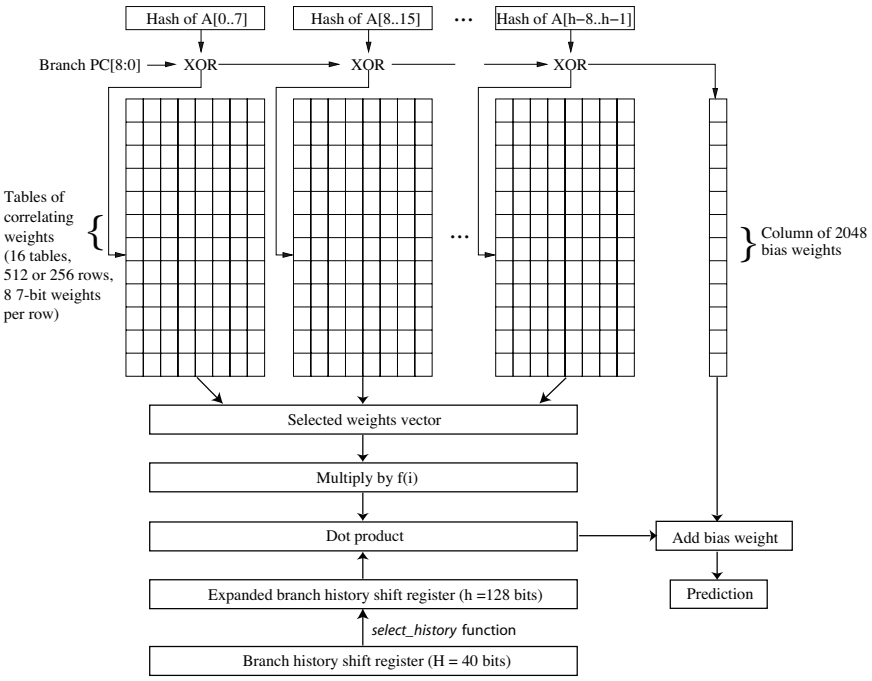


Figure 3. Prediction data path

non-uniform correlation for a neural predictor with a history length of 128. The x-axis represents the position of a weight in the history ($x = 0$ 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. The bias weight has the highest correlation with branch outcome, and the first weights have a much stronger correlation than the later weights. The correlation coefficients were generated using the publicly distributed traces for the CBP-2 competition; for the function $f(i)$, fitted to the correlation coefficients, the best f for a weight in position i is $f(i) = 1/(a + bi)$, where $a = 0.1111$ and $b = 0.037$. By multiplying weights with coefficients proportional to their correlation, the predictor achieves higher accuracy. The analog design described in the next section achieves the weight scaling by varying transistor sizes, whereas a digital implementation would need to perform a number of power-prohibitive multiplications for each prediction.

3.1. Algorithm for the Scaled Neural Predictor

The neural prediction algorithm presented below achieves higher accuracies than previously proposed neural algorithms. The higher accuracies result from (1) accessing the weights using a function of the PC and the path (with no ahead pipelining), (2) breaking the weights into a number of independently accessible tables, (3) scaling the weights by the coefficients as previously described, and (4) taking the dot product of a modified global branch history vector and the scaled weights. Figure 3 shows a high-level diagram of the prediction algorithm and data path. This diagram does not represent the actual circuit implementation, which is described in the next section.

The two key parameters of the predictor are h, the length of the vector with which the dot product is computed, and r, the number of rows in each weights table. In this design, $h = 128$ and $r = 256$, 512, or 2048. Other inputs to the predictor are A, a vector of the low-order bit of each of the past h branch addresses (A is effectively a path vector), and H, the global branch history register. This design uses a history register H of 40 bits.

The two components of the dot-product computation are the history vector and the weights vector. The history vector consists of $h = 128$ bits, which is expanded from the 40 bits of H with a function called *select_history*, shown in Figure 4 for completeness. The use of redundant history can improve prediction accuracy [6], so this predictor replicates the 40 branch history bits to obtain the required 128, which indeed provides higher accuracy than simply using the history bits of the most recent 128 branches.

The second component of the dot-product computation, the weights vector, is obtained by reading eight weights from each of 16 tables, as well as a single weight from a table of bias weights. The first table, containing the weights for the most recent history bits, has 512 entries because the most recent weights are the most important. The bias weights table has 2048 entries. The other tables each have 256 entries, to keep the total predictor state under 32KB. The tables

Table 1. Predictor parameters

Parameter	Value
Number of columns of W	129 (1 bias weight + 128 correlating weights)
Number of rows for W	Bias: 2048, W[0..7]: 512, W[8..127]: 256
Bits per weight	7 for columns 0..56, 6 for columns 57..128

are partitioned into 16, rather than just one large indexed row of 128 weights, because the separation reduces aliasing and achieves higher accuracy with low additional complexity. The path-based neural and piecewise linear branch predictors also separate the weights into tables, but this division was necessary to support ahead-pipelining; accuracy is the sole motivation for separate tables in the SNP design. To index each table, an eight-bit fraction of the A vector is XORed with the low-order eight bits of the branch PC, resulting in an eight-bit index for one of the 256 rows. In the first table with 512 entries, an extra bit of the branch PC is XORed with a second address bit from the most recent branch to produce a nine-bit index. The bias weight table is indexed with 11 lower-order bits from the branch PC. Table 1 shows the parameters for the weight tables, and Figure 5 shows the pseudocode for the prediction algorithm.

3.2. Predictor Updates

Updating the predictor consists of three phases, some of which can occur in parallel.

Updating histories: When the outcome of a branch becomes known, it is shifted into H. The lowest-order bit of the branch's address is shifted into A. A high-accuracy implementation must keep speculative versions of H and A that are restored on a misprediction.

Training the predictor: If the prediction was incorrect, or if the magnitude of the predictor output was under a set threshold, then the predictor invokes its training algorithm. As in previous neural predictors, the weights responsible for the output are incremented if the corresponding history outcome matches the current branch outcome, and decremented otherwise. The weights use saturating arithmetic.

Updating the training threshold: An adaptive threshold training algorithm is used to dynamically adjust the threshold at which training will be invoked for a correct prediction. This algorithm is the same as the one used for O-GEHL [9]: 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 observed that good accuracy is achieved when the training algorithm is invoked equally many times after correct and incorrect predictions [9]; this threshold training strategy strives to achieve that balance.

```
function select_history (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 mod 8                        for odd- and even-numbered tables
  else
    j := i mod 8 + i/4
  endif
  select := H[j]..H[j + 7]
end
```

Figure 4. Select history bits based on weight position

```
function prediction (pc: integer) : { taken , not_taken }
begin
  sum := W[pc mod n, 0]                 Initialize to bias weight
  for i in 1 .. h by 8 in parallel      For all h/8 weight tables
    k := (hash(A[i..i + 7]) xor pc) mod n   Select a row in the table
    for j in 0 .. 7 in parallel         For all weights in the row
      q := select_history(H, i)[j]      Select branch history
      sum := sum + W[k, i + j + 1] × q  Add to dot product
    end for
  end for
  if sum >= 0 then                      Predict based on sum
    prediction := taken
  else
    prediction := not_taken
  endif
end
```

Figure 5. SNP algorithm to predict branch at PC

3.3. Predictor Accuracy Results

The accuracy of the Scaled Neural Predictor was measured using a trace-driven simulator, derived from the CBP-2 contest infrastructure [15], that 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. This predictor design restricts its hardware budget similarly so that its results are comparable to other published results.

As is common practice, the predictor was tuned using a set of training traces, and accuracy experiments were run on a different set of traces. The tuning traces, provided in the CBP-2 infrastructure, were derived from the SPEC CPU2000 and SPECjvm98 benchmark suites. Each trace file represents a single SimPoint [16] of 100 million instructions for the benchmark. A different set of traces, including SPEC CPU2006, were simulated to evaluate predictor accuracy; although some of the same benchmarks appear in the CBP-2 traces, there is no overlap in the traces in terms of portions of program intervals simulated.

We compare the Scaled Neural Predictor against two other predictors from the literature: the piecewise linear branch predictor [8] and L-TAGE [1]. The piecewise linear branch predictor 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. For piecewise linear branch prediction, the search space was exhaustively explored to find the set of parameters that yielded the best accuracy on the CBP-2 traces. L-TAGE includes a 256-entry loop predictor that accurately predicts loops with long trip counts; an identical loop predictor, included in the hardware budget, was added to the Scaled Neural Predictor as well as the piecewise linear branch predictor. The piecewise linear branch predictor was also extended to exercise the adaptive threshold training algorithm used in O-GEHL [9].

Figure 6 compares the accuracies of a range of neural predictors and L-TAGE. The leftmost three bars represent previously published neural predictors (perceptron, path-based, and extended piecewise linear, as described above). The fourth bar represents L-TAGE, showing the best prediction accuracy of all predictors evaluated. The fifth bar shows the accuracy of an idealized path-based predictor without ahead pipelining. Note that with knowledge of the predicting branch PC, path-based branch prediction is equivalent to piecewise linear branch prediction, as the improvement of piecewise linear prediction over path-based prediction is a technique to incorporate more branch PC bits into the final prediction. Thus, the contributions in this paper can be seen as an improvement to both predictors. Comparing the second and fifth bars shows the benefits of removing ahead pipelining and adding the loop predictor and adaptive thresholds. The sixth and final bar shows the Scaled Neural Predictor, which includes the coefficient scaling of weights. The accuracy is close to L-TAGE (5.06 compared to 4.91), differing only by .15 MPKI on average. The Scaled Neural Predictor outperforms L-TAGE on five benchmarks, but on some benchmarks (such as gcc), L-TAGE shows considerably lower misprediction rates. Even with the aggressive tuning/updating of previous neural predictors, the Scaled Neural Predictor shows that it is the most accurate neural predictor measured to date, though it is not competitive from a power perspective.

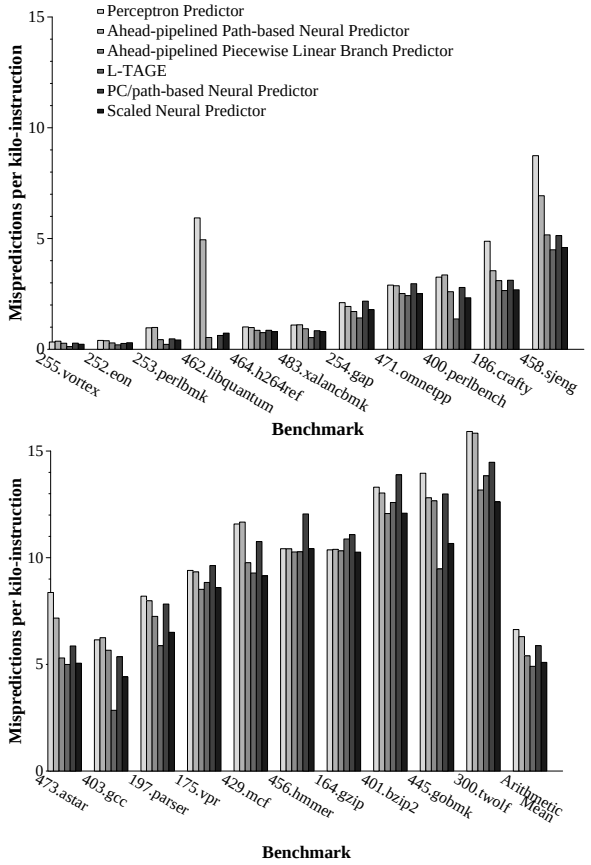


Figure 6. Comparing neural predictors and L-TAGE accuracy

4. Analog Circuit Design

This section describes the implementation of the Scaled Neural Analog Predictor (SNAP). 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, as well as a train / don't train output based on a threshold value. The design utilizes analog current-steering and summation techniques to execute the dot-product operation. The circuit design shown in Figure 7 consists of four major components: current-steering digital-to-analog converters (DACs), current splitters, current to voltage converters, and comparators.

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 were chosen because of their high speed and simplicity. There is one DAC per weight, each consisting of a current source and a bias transistor as well as one transistor corresponding to each bit in the weight.

The source and gate nodes of all of the DAC transistors are tied together so that they share the same gate voltage, V_G, and source voltage, V_S. A current source fixes the current through the bias transistor and because all transistors share the same V_{GS} $(V_G - V_S)$ value, the current through the bias transistor is "mirrored" in the other transistors according to the equation $I_D = \frac{\mu C_{ox}}{2}\frac{W}{L}(V_{GS} - V_t)^2$ [17]. This configuration, known as a current mirror, is a common building block in analog circuit design. Notice that the mirrored current is 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, each transistor draws a current approximately proportional to *W*.

This approach supports near-linear digital-to-analog conversion. For example, for a 4-bit base-2 digital magnitude, the DAC transistor widths would be set to 1, 2, 4, and 8 and draw currents I, 2I, 4I, and 8I, respectively. A switch is used

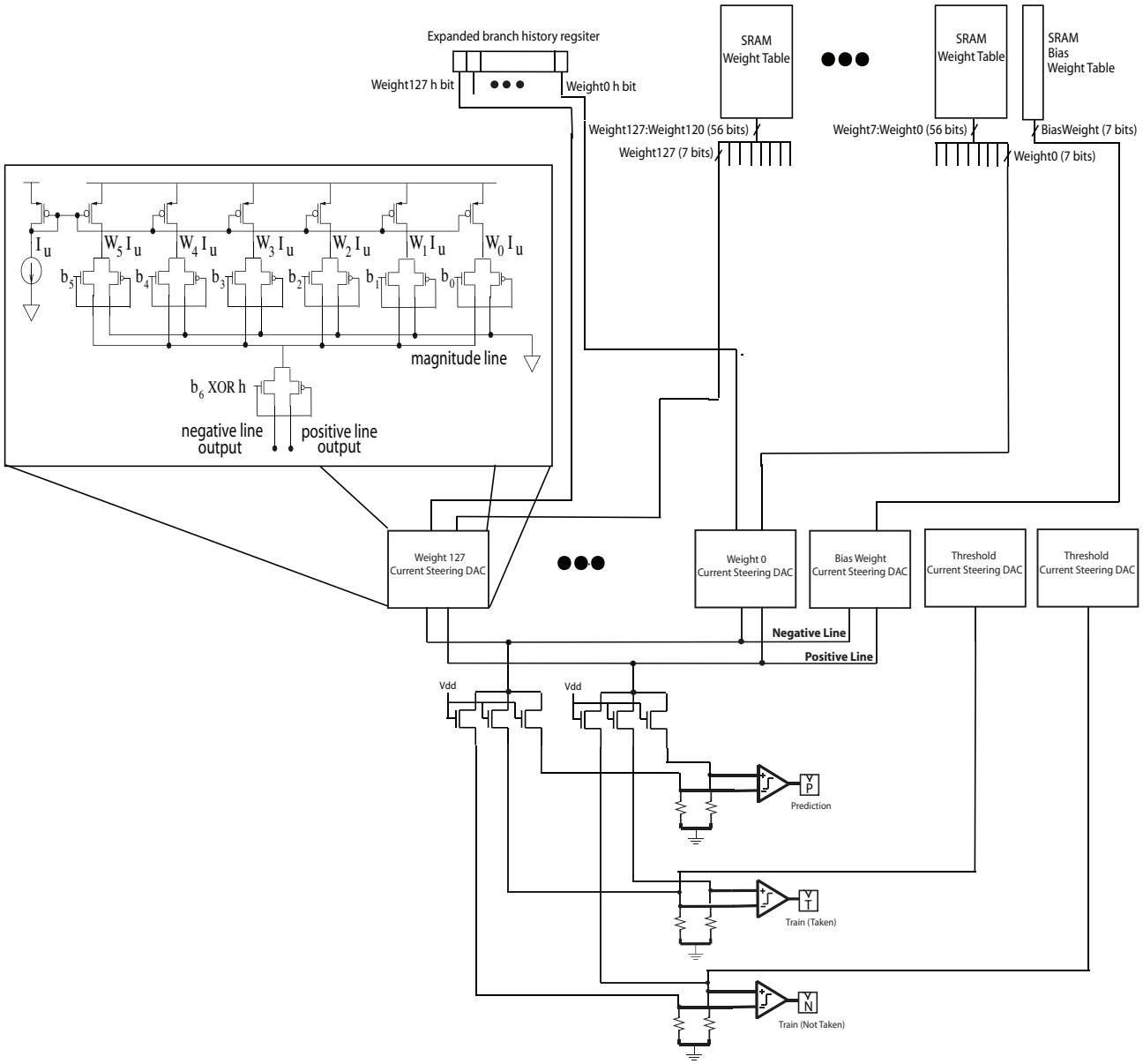


Figure 7. Top-level diagram of a Scaled Neural Analog Predictor (SNAP)

to steer each transistor current according to its corresponding weight bit, where a weight bit of 1 steers the current to the *magnitude line* and a weight bit of 0 steers it to ground. In the example above, if the digital magnitude to be converted is 5, or 0101, currents I and 4I would be steered to the magnitude line, where 2I and 8I would be steered to ground. By the 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 *positive line* or *negative line* based on the XOR of the sign bit for that weight and the appropriate history bit, effectively multiplying the signed weight value by the history bit. The positive 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 Splitter: The currents on the positive line and the negative line are split roughly equally by three transistors to allow for three circuit outputs: a one-bit prediction and two bits that are used to determine whether training should occur. Splitting the current, rather than duplicating it through additional current mirrors, maintains the relative relationship of positive and negative weights without increasing total current draw, thereby avoiding an increase in power consumption.

Current to Voltage Converter: The currents from the splitters pass through resistors, creating voltages that are used as input to the preamplifier and voltage comparators.

Preamplifier and Comparator: Preamplifiers are commonly used in conjunction with comparators in traditional analog circuit design. The preamplifier amplifies the voltage difference between the lines, producing new voltages that will be used as input to the comparator, which improves comparator speed and accuracy; it also acts as a buffer that prevents undesirable kickback effects [17]. The preamplifier requires only a small amount of circuitry, namely a bias current, two transistors, and two resistors. A track-and-latch comparator design [17] 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 one. The comparator outputs a 1 if the voltage corresponding to the positive line outweighs the negative line and a 0 otherwise. For comparator output *P*, a 1 corresponds to a taken prediction and a 0 corresponds to a not-taken prediction.

Training: In addition to a one-bit taken or not-taken prediction, the circuit latches two signals that will be used when the branch is resolved to indicate whether the weights must be updated. Training occurs if the prediction was incorrect or if the absolute value of the difference between the positive and negative weights is less than the threshold value. Rather than actually computing the difference between the positive and negative lines, which would require more complex circuitry, the absolute value comparison is split into two separate cases: one case for the positive weights being larger than the negative weights and the other for the negative weights being larger than the positive ones. Instead of waiting for the prediction output *P* to be produced, which would increase the total circuit delay, all three comparisons are done in parallel. *T* is the relevant training bit if the prediction is taken and *N* is the relevant training bit if the prediction is not taken. To produce *T*, the threshold value is added to the current on the negative line. If the prediction *P* is 1 (taken) and the *T* output is 0, which means the negative line with the threshold value added is larger than the positive line, then the difference between the positive and negative weights is less than the threshold value and the predictor should train. Similarly, to produce *N*, the threshold value is added to the current on the positive line. If the prediction *P* is 0 (not taken) and the *N* output is 1, which means the positive line with the threshold value added is larger than the negative line, then the difference between the negative and positive weights is less than the threshold value. If *C* is the direction of the branch on commit, the following logic formula indicates training: $(C \oplus P) + P\overline{T} + \overline{P}N$.

Table 2. Excerpts from the list of DAC transistor widths

Col. #	**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

4.1. Circuit Evaluation Methodology

We composed a transistor-level implementation of SNAP in the Cadence Analog Design Environment using Predictive Technology Models (PTMs) at 45nm [18]. 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. They include basic effects such as drain-induced barrier lowering, non-uniform doping, and short and narrow channel length effects on threshold voltage, as well as various leakage currents including subthreshold leakage and high-speed models consisting of a substrate resistance network, an intrinsic input resistance model, and a transient non-quasi-static model. All transistors in the design utilize 45nm bulk CMOS model cards that can be found on the PTM website [18]; a description of the BSIM4 model parameters can be found in the user's guide [19].

Spectre transient analyses were used for all circuit simulations. A 1V power supply and a 10% rise/fall time were assumed for each clock speed. Analog power is measured by multiplying the supply voltage by the average current drawn from the power supply. Analog accuracy numbers were generated by characterizing the analog circuit behavior as a statistical error model and mapping it back to the CBP-2 simulation infrastructure.

4.2. Tuning the Predictor Circuit

The SNAP implementation accepts a number of parameters including history length, number of bits per weight, and transistor widths for each weight. 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 is omitted for lack of space.

4.3. Analog Power, Speed, and Accuracy

The SNAP design presents a trade-off between power, speed, and accuracy. The principle factor determining circuit delay is the size of the currents produced in the DACs. Larger currents drive outputs to stabilize sooner, thereby decreasing delay

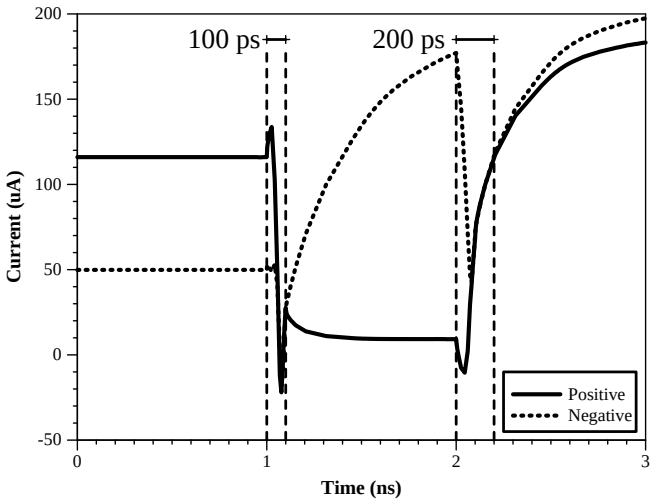


Figure 8. Time required for current differentiation

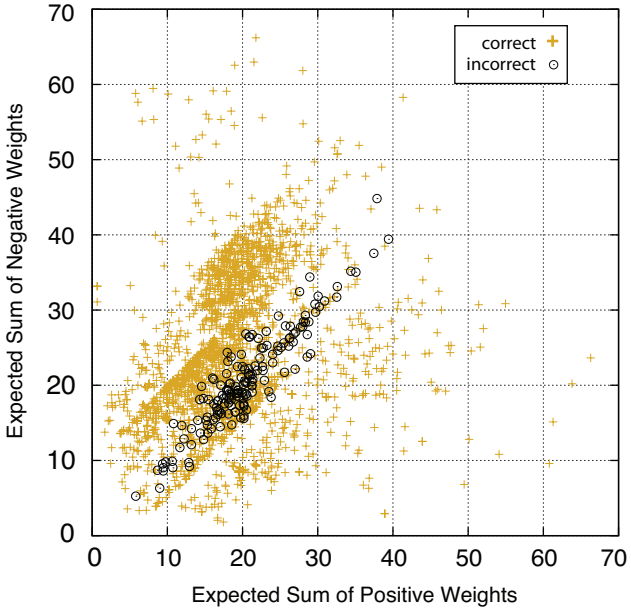


Figure 9. Prediction errors for sum combinations

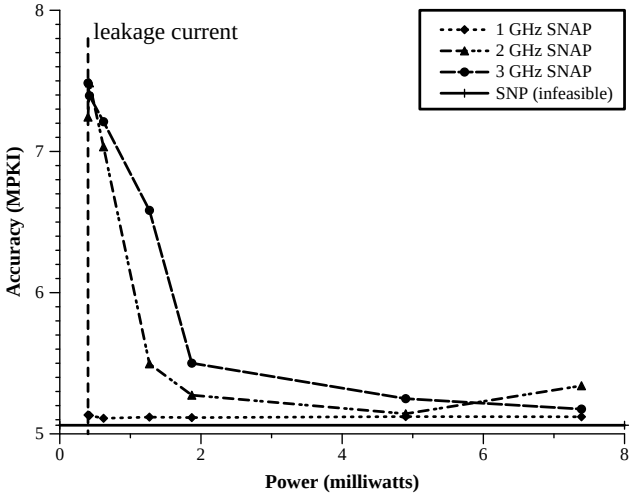


Figure 10. Tradeoff between power and accuracy

through the circuit; however, large currents increase power consumption by increasing the total current draw.

The relative difference between the positive and negative weights also determines when the correct output can be latched. Figure 8 demonstrates the current behavior for two different sets of input weights: one where the positive and negative sums vary greatly in magnitude and one where the sums are similar. In this example, the weights change at 1ns such that the negative sum greatly outweighs the positive, and the two currents quickly diverge. At 2ns the weights change such that the negative sum is only slightly larger than the positive; in this case, the currents require more time stabilize before a winner can be determined.

Figure 9 shows prediction errors for various positive/negative sum combinations; unsurprisingly, errors arise when the two sums are closest in magnitude. These errors occur because the currents were not given sufficient time to stabilize before the output was latched or because the currents produced by the DACs resulted in the wrong line having the larger value, since similar sums allow less room for errors in current values. Incorrect currents can result from non-linearity in the DACs as well as process variation and noise.

Fortunately, similar sums correspond to low prediction confidence, since the dot-product result is close to zero and does not signify a strongly taken or not-taken prediction; errors on low-confidence predictions mitigate the impact of errors on overall prediction accuracy. In addition, this case occurs on a small percentage of the total number of predictions. The simulations run to generate Figure 9 focused on this small space, even though these points are less common, to clearly illustrate the diagonal error band.

The width of the error band shown in Figure 9 increases as clock speed increases or power decreases, causing a decrease in predictor accuracy. Figure 10 shows the relationship between power, speed, and accuracy. The DAC bias currents were adjusted to generate the multiple power levels, and the lowest power shown (0.4mW) corresponds to the DACs running strictly on leakage currents. At 1GHz, high accuracy is maintained over the range of power levels simulated, where leakage current alone achieves an MPKI of 5.13. The drop in accuracy from 4.9mW to 7.4mW at 2GHz is a function of inaccuracy in the DAC currents coupled with the particular behavior of the traces simulated. At 3GHz, increasing power from 1.9mW to 7.4mW shows an improvement of .32 MPKI.

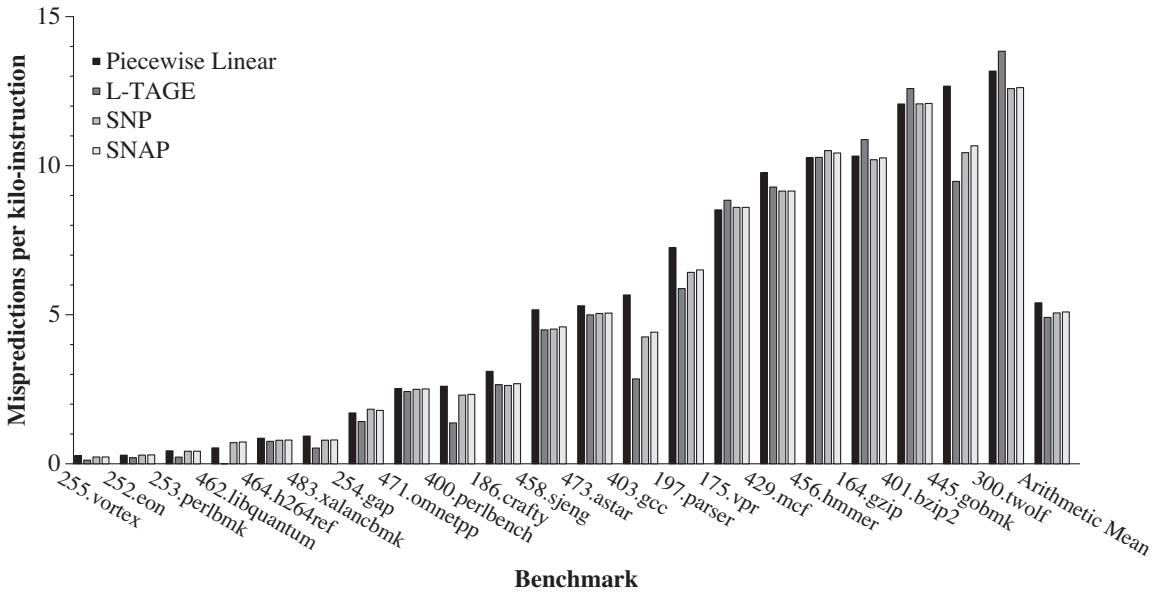


Figure 11. Accuracy of digital vs. analog versions of the Scaled Neural Predictor

4.4. Comparison to Digital Accuracy

To illustrate the loss in potential prediction accuracy due to the use of non-ideal analog circuits, Figure 11 shows the accuracy of the Scaled Neural Analog Predictor compared to an infeasible digital Scaled Neural Predictor. The figure also shows L-TAGE and piecewise linear branch prediction for comparison. The analog version achieves an average accuracy of 5.18 MPKI compared to 5.06 for the digital version; thus, the imprecision of the analog circuit results in only a .12 MPKI decrease in accuracy. Piecewise linear branch prediction, which is less feasible than SNAP but still arguably implementable, results in 5.4 MPKI.

4.5. Power Comparison

Total power consumed by the prediction step of the SNAP design includes table lookups in addition to the analog computation. Running at 3GHz with a 1V supply voltage, the average analog power required to obtain high prediction accuracy is 7.4mW. At slower clock speeds, however, high accuracy can be achieved at lower power levels; for example, at 1GHz, the 0.4mW power configuration achieves an accuracy of 5.13 MPKI. Note that the same current is drawn from the power supply regardless of the weight inputs. Also, for a given bias current configuration, average power consumption remains constant over a range of frequencies because the average current draw remains constant.

We use CACTI 4.2 [20] with 45nm technology files to measure the dynamic power of the digital portion of the design. The 16 weight tables were modeled as tagless memories with 8-byte lines; the total dynamic read power at maximum frequency is estimated to be 117mW. For comparison, the power consumed by the various memory structures of L-TAGE is estimated at 112mW. Thus, the memory components of the two predictors have comparable dynamic power, and the analog computation of the dot product is a small fraction of the total power consumed by the predictor.

4.6. Frequency of Training

The predictor update requires the use of an array of narrow up/down counters, potentially requiring significant dynamic power. On the various benchmarks simulated, including SPEC CPU2006, the weights need to be adjusted 10% of the time on average; the other 90% of the time the adders are idle. This observation supports the possibility of multiplexing fewer up/down counters over time.

5. Implementation Issues

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

5.1. Process Variation

As technology scales, process variation becomes an increasing concern. Process variation appears in several forms including transistor length, transistor width, and threshold voltage variation. SRAM cells that hold predictor state in traditional table-based predictors and weights vectors in neural predictors are subject to increased access delay due to process variations. In a traditional predictor design, increased access delay can result in incorrect table reads, thereby producing incorrect branch predictions.

A neural predictor with a feedback-training algorithm, however, is well suited to tolerate inaccuracies caused by process variation. In an analog neural predictor, unexpected currents may be produced as a result of an incorrect table read or due to variations in the DAC transistors. The training algorithm described in Section 3 will correct for unexpected currents by adjusting the weights vector appropriately. For example, if a weight produces a current value that is smaller than the expected value, causing the prediction to be wrong or the sum to be less than the threshold value, the weight will be increased; additionally, the non-faulting weights will be adjusted, thereby quickly correcting for the inaccuracy of the faulting weight. Analog neural predictors that adjust a weights vector based on a current summation are therefore more robust against the negative effects of process variation than table-based designs that predict based on the sign bit of a digital counter value.

5.2. Noise Tolerance

On-chip analog circuits are susceptible to substrate 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 [21]. A guard ring separates analog and digital circuits and creates a large capacitance that provides noise currents a low impedance path to ground.

5.3. 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 scan chains, automatic test pattern generation (ATPG), and built-in self-tests (BISTs), have become sophisticated and cost effective [22]. The SNAP 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. For SNAP, 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 be routed off chip.

For finer grain failure detection, internal analog nodes can be accessed using analog scan chains. In [23] and [24], currents are shifted through scan chains using current mirrors and switches. In [25], 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 the SNAP design.

6. 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 highly accurate Scaled Neural Predictor algorithm, made feasible by a mixed-signal design that uses analog current summation and comparison techniques to compute the computationally expensive part of the prediction step.

The 45nm analog design can operate over a range of power and clock-speed configurations, and at 3GHz and 7.4mW, shows an increase of only .12 MPKI over an ideal digital implementation. This design is the first neural predictor whose prediction step is competitive from a power perspective, requiring only a small amount of power for the dot-product computation compared to the 117mW required for the digital table lookups in this design. The table lookups for a comparably-sized L-TAGE predictor consume 112mW, not counting L-TAGE's more complex hashing computations.

The SNAP design is the second most accurate predictor design published to date (using the union of the SPEC CPU2000 and 2006 benchmarks); the L-TAGE predictor first published in December, 2006 is the most accurate, showing 4.91 MPKI as opposed to 5.18 for the the Scaled Neural Analog Predictor. Piecewise linear branch prediction achieves 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) require conventional amounts of power. Compared to conventional predictors such as L-TAGE, the SNAP design 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 memory cells to adjust the weighted currents directly, rather than performing a lightweight D-to-A conversion. In this case, one cell would be associated with each weight, allowing for a compact representation of a large number of weights and reduced power by avoiding table reads and updates. Flash and phase-change memory cells are candidates for storing the analog values.

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. This capability will only be possible if the computations are insensitive to accreted noise and errors (such as convergent numerical algorithms, human/computer interfaces, and media), or if the noise can be periodically reduced through feedback and correction, such as with the digital adjustment of neural predictor weights upon branch mispredictions.

Acknowledgments

Renée St. Amant is supported by an NSF GRF and Daniel A. Jiménez by NSF grants 0751138 and 0545898.

References

[1] 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)*, vol. 9, May 2007.

[2] 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.

[3] 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)*, January 2001, pp. 197–206.

[4] ——, "Neural methods for dynamic branch prediction," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 369–397, November 2002.

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

[6] A. Seznec, "Redundant history skewed perceptron predictors: Pushing limits on global history branch predictors," IRISA, Tech. Rep. 1554, September 2003.

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

[8] ——, "Piecewise linear branch prediction," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA-32)*, June 2005.

[9] A. Seznec, "Analysis of the o-geometric history length branch predictor," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA'05)*, June 2005.

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

[11] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, "A reconfigurable vlsi neural network," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 67–81, January 1992.

[12] A. H. Kramer, "Array-based analog computation," *IEEE Micro*, vol. 16, no. 5, pp. 20–29, October 1996.

[13] 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*, vol. 38, no. 2-3, pp. 233–244, February-March 2004.

[14] O. Kirby, S. Mirabbasi, and T. M. Aamodt, "Mixed-signal neural network branch prediction," unpublished manuscript.

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

[16] 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*, October 2002.

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

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

[19] Bsim Research Group at UC Berkeley, *BSIM4.6.1 mosfet manual user's guide*, 2007, http://www-device.eecs.berkeley.edu/~bsim3/BSIM4/BSIM461/doc/.

[20] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "Cacti 4.0," HP Laboratories Palo-Alto, Tech. Rep. HPL-2006-86, June 2006.

[21] 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*, vol. 28, no. 4, pp. 420–430, April 1993.

[22] 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*, vol. 45, no. 10, pp. 1389–1407, October 1998.

[23] 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*, vol. 41, no. 4, pp. 535–539, August 1992.

[24] M. Soma, "Structure and concepts for current-based analog scan," *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 517–520, 1995.

[25] Y. Miura, "Real-time current testing for a/d converters," *IEEE Design Test*, vol. 13, no. 2, pp. 34–41, 1996.