Mixed-Signal Approximate Computation: A Neural Predictor Case Study

As transistors shrink and processors trend toward low power, maintaining precise digital behavior grows more expensive. Replacing digital units with analog equivalents sometimes allows similar computation to be performed at higher speed using less power. As a case study in mixed-signal approximate computation, the authors describe an enhanced neural prediction algorithm and its efficient analog implementation.

Renée St. Amant
University of Texas at Austin

Daniel A. Jiménez
University of Texas at San Antonio

Doug Burger
Microsoft Research

In future processor designs, the chip's power budget will likely limit the amount of logic used to accelerate program performance. One potential path for advancement is to leverage power-efficient analog circuits by composing mixed-signal units that increase program performance while incurring only a small increase in power consumption. Figure 1 shows a model for programmable, general-purpose, mixed-signal approximate computation. The design must minimize the power required to convert parameters from digital to analog and results from analog to digital. Additionally, a low-power inner loop of analog storage and computation should amortize the power consumed by conversions. In this scheme, a string of computations executes with intermediate values placed in analog storage before the circuit converts the final result to a digital value that can be stored in a digital memory construct. Interaction with the digital domain is necessary, however, to constrain noise accrual in the analog domain. Digital feedback periodically adjusts for analog noise and updates analog values to keep computational accuracy at an acceptable level. Analog storage is a necessary component of this model and could consist of capacitors, floating-gate memory cells, or phase-change memory cells.

As a case study in mixed-signal approximate computation, this article evaluates an aggressive neural predictor design that uses analog circuits to implement the power-intensive portion of the prediction loop. The circuit uses lightweight digital-to-analog converters (DACs) to convert digitally stored weights into analog currents, which it then combines 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,

Published by the IEEE Computer Society

effectively operating as an analog-to-digital converter (ADC).

Branch prediction remains a key component for enhancing single-threaded performance in general-purpose processors. Neural branch predictors have shown promise in attaining some of the highest prediction accuracies, but they traditionally have poor power and energy characteristics. The need to compute a dot product for every prediction, with potentially tens or even hundreds of elements, has limited neural predictors' applicability, making them unsuitable for industrial adoption in their current form.

A new prediction algorithm—the *scaled neural predictor* (SNP)—uses an accuracy-improving feature, which is infeasible to implement in a purely digital design, that scales weights using an inverse linear function, effectively multiplying each weight by a constant. The analog implementation—the *scaled neural analog predictor* (SNAP)—achieves accuracy close to L-Tage[1] and an improvement over the piecewise linear branch predictor, one of the best previous neural predictor designs. (See the ''Background on Neural Predictors'' sidebar for related work in this area.)

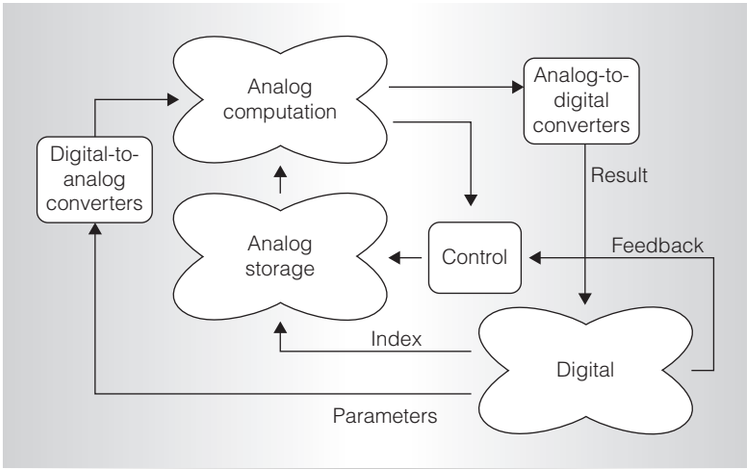

Figure 1. Mixed-signal approximate computation model.

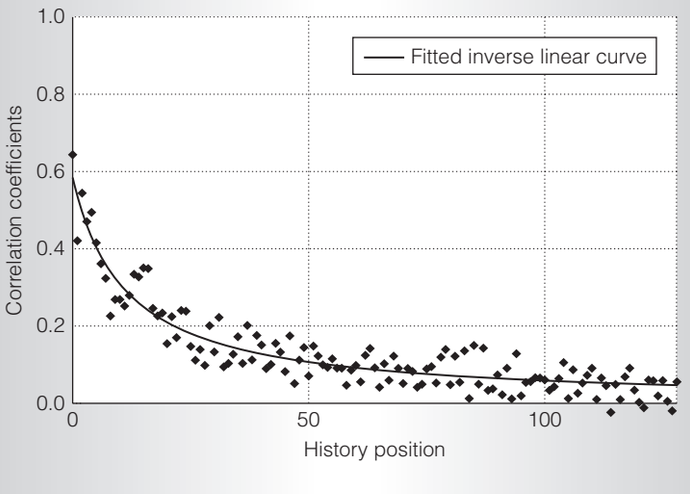

Figure 2. Weight position and branch outcome correlation.

Analog-enabled neural prediction algorithm

The power and latency reductions afforded by analog design allow improvements to neural predictors that are infeasible in a purely digital domain. The primary improvement enabled by the analog design is the ability to scale individual weights using predetermined coefficients based on history position.

The vector of weights represents the contribution of each branch in a given history to predictability, but each branch doesn't contribute equally. Unsurprisingly, more recent weights tend to have a stronger correlation with branch outcomes. Figure 2 quantifies this nonuniform correlation for a neural predictor with a history length of 128. The x-axis represents a weight's position 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 using only the weight in position x. The bias weight has the highest correlation with branch outcome, and the weights corresponding to recent branches have a much stronger correlation than those corresponding to older branches.

By multiplying weights with coefficients proportional to their correlation, the predictor achieves higher accuracy. The weights alone partially capture this correlation, but older branches sometimes introduce noise into the prediction result. Scaling weights based on their history position helps eliminate this noise. The analog

Background on Neural Predictors

Most proposals for neural branch predictors derive from the perceptron branch predictor (see Figure A).[1,2] In this context, a *perceptron* is a vector of $h + 1$ small integer weights, where h is the predictor's *history length*. The predictor keeps a table of n perceptrons in fast memory. It also keeps a global history shift register of the h most recent branch outcomes (1 for taken, −1 for not taken).

To predict a branch, the predictor selects a perceptron using a hash function of the program counter (PC)—for example, taking the PC mod n. It computes the perceptron's output as the dot product of the weights vector and the history shift register. An extra bias weight in the perceptron is added to the dot product, adjusting for heavily taken or not-taken branches. A negative weight value denotes an inverse correlation between the direction of the branch in the corresponding history position and the current branch. The magnitude of the weight indicates the strength of the positive or negative correlation. If the dot-product result is at least 0, the branch is predicted taken; otherwise, it's predicted not taken.

When the branch outcome becomes known, the predictor updates the perceptron that provided the prediction. The perceptron is trained on a misprediction or when the magnitude of the perceptron output is below a specified threshold value. 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.

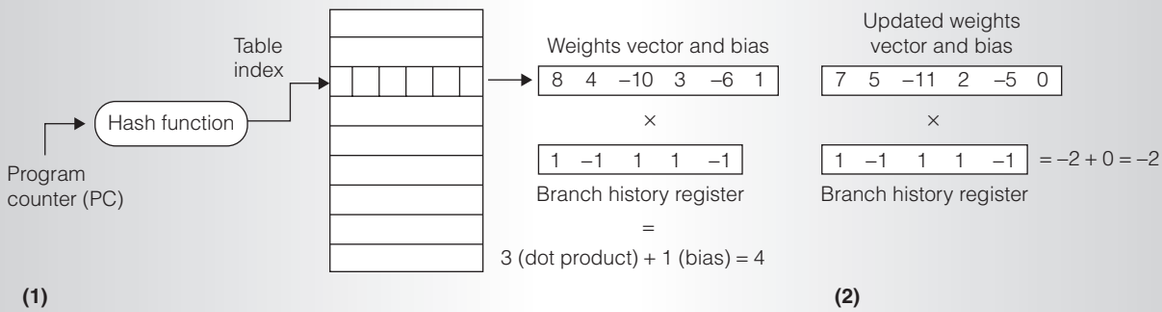

Figure A. Perceptron branch predictor prediction and training. The first time the branch is accessed, the perceptron output is greater than 0, so the branch is predicted taken (1). If the branch was not taken, misprediction triggers training. Weights with a −1 in the corresponding history position are incremented, whereas weights with a 1 in the position are decremented. The next time the branch is accessed, the perceptron output is less than 0 and the branch is predicted not taken (2).

implementation 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.

The SNP algorithm achieves higher accuracies than previously proposed neural algorithms. The higher accuracies result from

- accessing the weights using a function of the program counter (PC) and the path (with no ahead pipelining),
- breaking the weights into several independently accessible tables,
- scaling the weights by the coefficients as previously described, and
- 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 doesn't represent the actual circuit implementation, which we describe in the next section.

The predictor's two key parameters are h, the length of the vector consumed in the dot product, and r, the number of rows in each weights table. In this design, $h = 128$ and $r = 256$, 512, or 2,048. 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

The bias weight is incremented or decremented if the branch is taken or not taken, respectively. Figure A illustrates a perceptron producing a prediction and being trained.

Later research addressed neural predictor power and area[3,4] and significantly improved latency and accuracy.[5] Neural predictors achieved the highest reported prediction accuracy until recently, when Seznec introduced L-Tage,[6] a predictor based on partial matching.L-Tage is also more feasible to implement from a power perspective than the digital neural and neural-inspired predictors described to date.

Neural predictor designs, however, can be modified to use approximate computation, significantly reducing power consumption. In the predictor design shown in Figure A, the weights table index must be precise to ensure that the appropriate weights vector is selected, but the remainder of the prediction step can be imprecise for various reasons: the dot-product result is simply compared to zero, so variations in the result value don't often change the prediction. Similarly, because many weights contribute to this dot-product computation, variations in weight values are also acceptable. Computational imprecision can be traded for decreased power consumption to the extent that prediction accuracy remains acceptably high.

Neural networks are often characterized by a compute-intensive dot-product operation. Researchers have implemented this operation most efficiently with analog current-summing techniques.[7,8] In these techniques, weights are represented by currents and summed using Kirchhoff's current law. Kramer explored charge and conductance summation in addition to current summation and described an efficient dot-product computation array.[7] Schemmel et al. presented a mixed-mode VLSI design that stores weight values in analog current memory cells.[8] The circuit summed currents on an excitatory or inhibitory input line based on a sign bit for each weight and compared the two current values to produce an output. Although the analog predictor design we describe here uses similar current-steering and comparison techniques, it differs in storage, analog/digital boundaries, and application.

References

1. D.A. Jiménez and C. Lin, ''Dynamic Branch Prediction with Perceptrons,'' *Proc. 7th Int'l Symp. High Performance Computer Architecture* (HPCA 7), IEEE CS Press, 2001, pp. 197-206.
2. D.A. Jiménez and C. Lin, ''Neural Methods for Dynamic Branch Prediction,'' *ACM Trans. Computer Systems,* vol. 20, no. 4, Nov. 2002, pp. 369-397.
3. D.A. Jiménez and G.H. Loh, ''Controlling the Power and Area of Neural Branch Predictors for Practical Implementation in High-Performance Processors,'' *Proc. 18th Int'l Symp. Computer Architecture and High Performance Computing* (SBAC-PAD 06), IEEE CS Press, 2006, pp. 55-62.
4. R. Sethuram et al., ''A Neural Net Branch Predictor to Reduce Power,'' *VLSI Design, Proc. 20th Int'l Conf. VLSI Design* (VLSID), IEEE CS Press, 2007, pp. 679-684.
5. D.A. Jiménez, ''Fast Path-based Neural Branch Prediction,'' *Proc. 36th Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 36), IEEE CS Press, 2003, pp. 243-252.
6. A. Seznec, ''A 256 Kbits L-Tage Branch Predictor,'' *J. Instruction-Level Parallelism* (JILP), vol. 9, May 2007; http://www.jilp.org/vol9/v9paper6.pdf.
7. A.H. Kramer, ''Array-Based Analog Computation,'' *IEEE Micro,* vol. 16, no. 5, Oct. 1996, pp. 20-29.
8. J. Schemmel et al., ''A Mixed-Mode Analog Neural Network Using Current-Steering Synapses,'' *Analog Integrated Circuits and Signal Processing,* vol. 38, no. 2-3, Feb./Mar. 2004, pp. 233-244.

$h = 128$ bits, which is expanded from the 40 bits of H. The use of redundant history can improve prediction accuracy,[2] so this predictor replicates the 40 branch history bits to obtain the required 128 as described elsewhere.[3]

The weights vector is obtained by reading eight weights from each of 16 tables, and a single weight from a table of bias weights. The bias weights table has 2,048 entries. 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 other tables each have 256 entries to keep the total predictor state under 32 Kbytes. The tables 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. To generate an index for each table, a fraction of the **A** vector is XORed with the low-order bits of the branch PC.

When a branch's outcome becomes known, it's shifted into H. The lowest-order bit of the branch address is shifted into **A**. A high-accuracy implementation must keep speculative versions of H and **A** that are restored on a misprediction. If the prediction is incorrect, or if the magnitude of the predictor output is under a set threshold, 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.

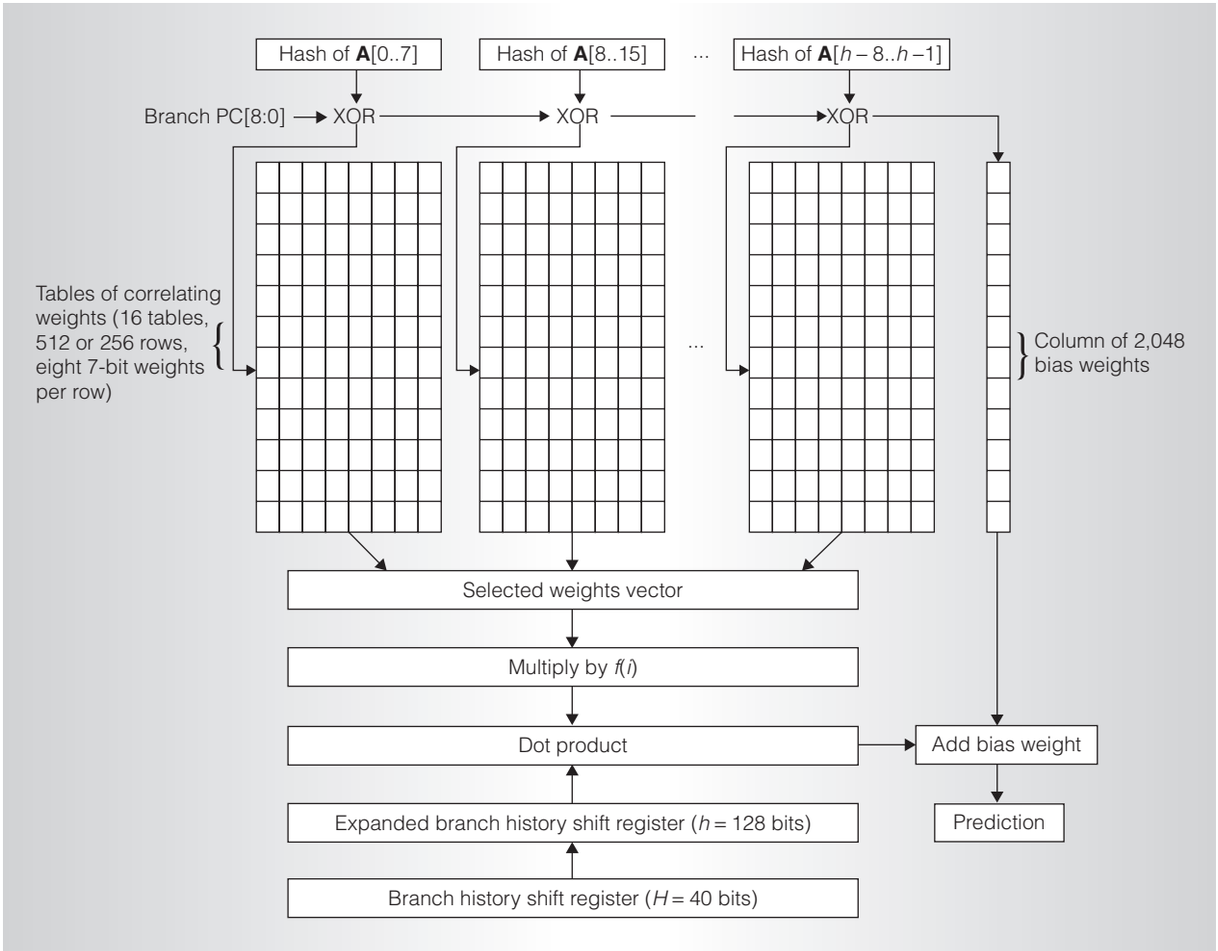

Figure 3. Prediction data path.

Analog circuit design

The SNAP design incorporates several blocks shown in the mixed-signal approximate computation model of Figure 1, namely DACs, ADCs, and an analog computation block. The analog computation block performs a previously infeasible dot-product computation, which offsets the power required for conversions. The SNAP circuit's 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 uses analog current-steering and summation techniques to execute the dot-product operation. Unlike the design in Figure 1, however, storage remains in the digital domain.

Design components

The circuit design in Figure 4 consists of four major components: current-steering DACs, current splitters, current-to-voltage converters, and comparators (effective ADCs).

Binary current-steering DACs. With digital weight storage, DACs must convert digital weight values to analog values that can be combined efficiently. Although the perceptron weights are seven bits, one bit represents the sign of the weight and this design requires only 6-bit DACs. We chose current-steering DACs for their high speed and simplicity. One DAC corresponds to each weight, each consisting of a current source and a bias transistor as well as one transistor for each bit in the weight.

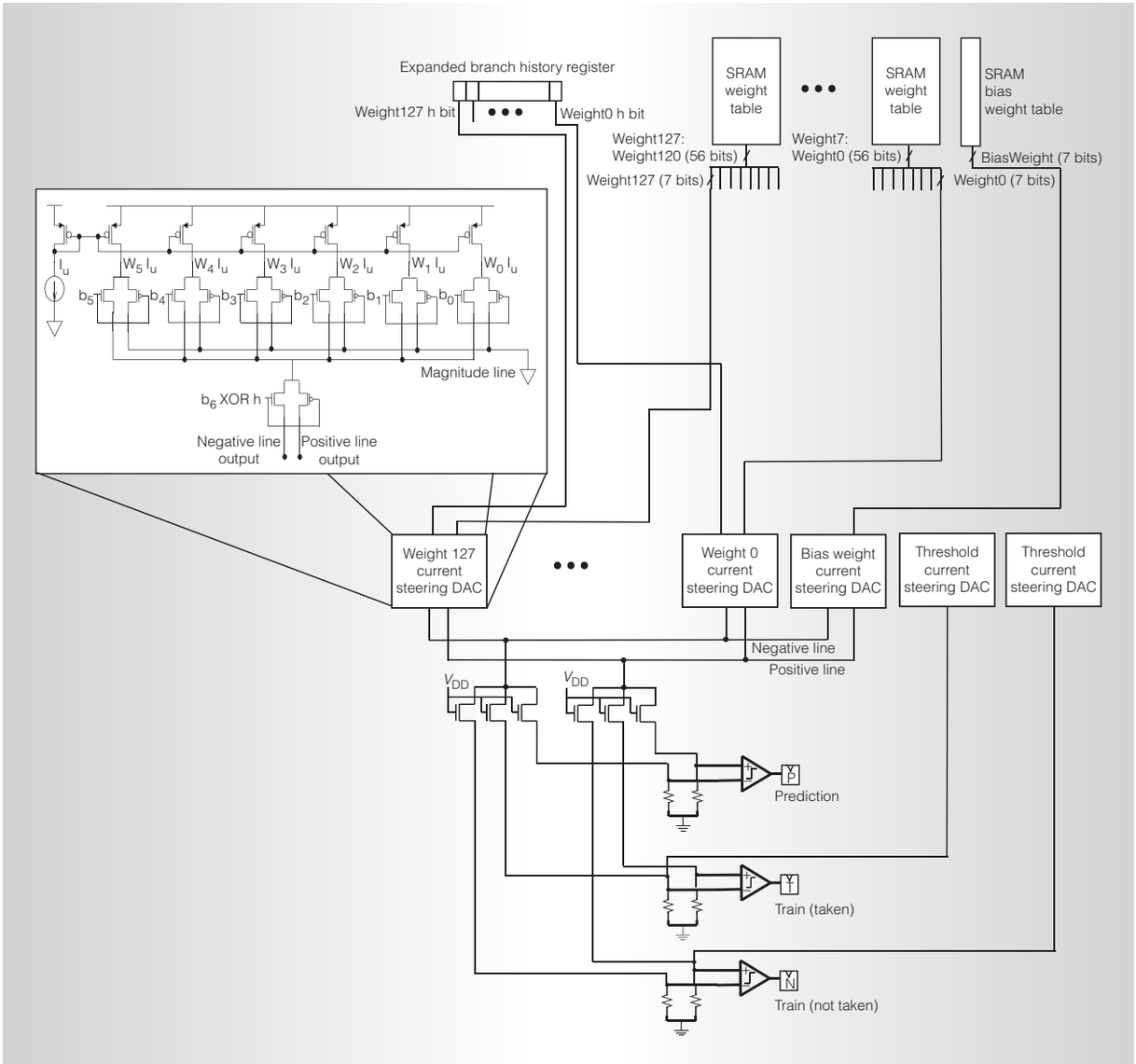

Figure 4. Top-level diagram of a scaled neural analog predictor.

The DAC transistors' source and gate nodes are tied together so they share the same gate voltage, V_G, and source voltage, V_S. A current source fixes the current through the bias transistor. 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 accordingly:[4]

$$I_D = \frac{\mu C_{ox}}{2}\frac{W}{L}(V_{GS} - V_t)^2.$$

This configuration, known as a *current mirror*, is a common building block in analog circuit design. The mirrored current is proportional to W/L, where W is the transistor's width and L is its length. 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 steers 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*. To convert a digital magnitude of 5, or 0101, for example, currents I and 4I are steered to the magnitude line, and 2I and 8I to ground. According to Kirchhoff's current law, the magnitude line contains the sum of the currents with weight bits that are 1 (5I in the example) and thus approximates the digitally stored weight.

A switch then steers the magnitude value to the *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, the positive line contains the sum of all positive values and the negative line the sum of all negative values.

Current splitter. The currents on the positive and negative lines are split roughly equally by three transistors to allow for three circuit outputs: a 1-bit prediction and two bits 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 increased power consumption.

Current-to-voltage converter. The currents from the splitters pass through resistors, creating voltages that become the 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.[4] The preamplifier requires only a small amount of circuitry—namely, a bias current, two transistors, and two resistors. We chose a track-and-latch comparator design because of its high-speed capability and simplicity.[4] 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 1-bit ADC and uses positive feedback to regenerate the analog signal into a digital one; thus, a heavy-weight analog-to-digital conversion isn't required. 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 1-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 is 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 design splits the absolute value comparison into two 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 comparator to produce prediction output *P*, which would increase the total circuit delay, the circuit performs all three comparisons 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 circuit adds the threshold value to the current on the negative line. If the prediction *P* is 1 (taken) and the *T* output is 0, meaning the negative line with the threshold value added is larger than the positive line, the difference between the positive and negative weights is less than the threshold value and the predictor should train.

Similarly, to produce *N*, the circuit adds the threshold value to the current on the positive line. If the prediction *P* is 0 (not taken) and the *N* output is 1, meaning the positive

line with the threshold value added is larger than the negative line, 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: $(C \oplus P) + P\bar{T} + \bar{P}N$.

Scaling weights by coefficients. To multiply weights by coefficients, the DACs utilize nontraditional transistor widths determined by the fitted inverse linear curve shown in Figure 2. As weight position increases and correlation with current branch outcome decreases, DAC transistor widths are scaled down, reducing the amount of current that a weight contributes to the positive or negative line. For example, rather than using traditional DAC widths to convert a digital base-2 value to an analog value (1, 2, 4, 8, 16, and 32), a weight in position 3 has DAC widths of 1, 1, 3, 5, 11, and 21, while a weight in position 128 has widths 0, 1, 1, 2, 4, and 8. Scaling DAC widths differently for each weight effectively multiplies each weight by a predetermined coefficient.

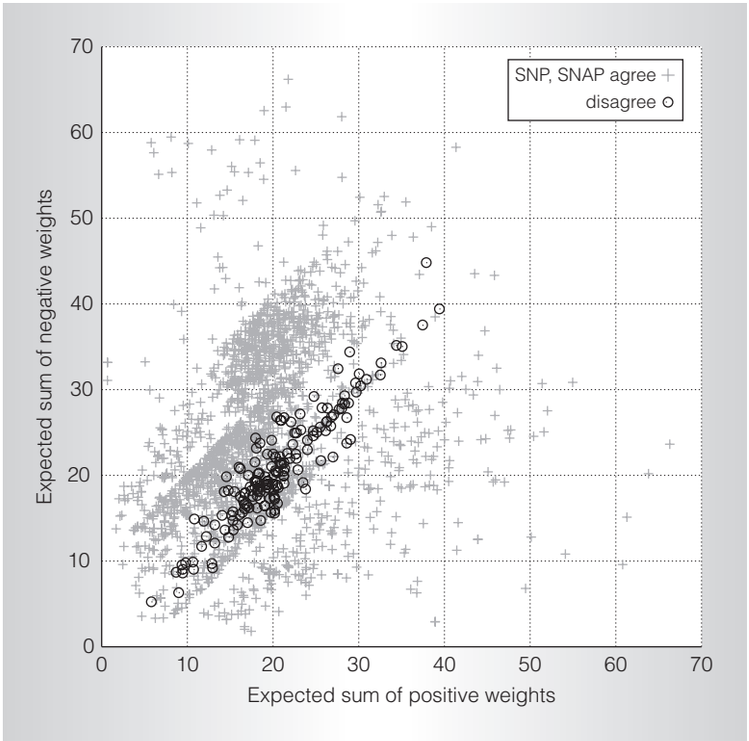

Figure 5. Analog prediction errors for sum combinations.

Circuit evaluation methodology

We composed a transistor-level implementation of the SNAP circuit in the Cadence Analog Design Environment using predictive technology models (PTMs, http://www.eas.asu.edu/~ptm) at 45 nm. These models are the standard for research and circuit design with immature technologies, and they account for the nonideal physical behavior of shrinking transistors. All transistors in the design use 45-nm bulk CMOS model cards that can be found on the PTM website; a description of the BSIM4 model parameters appears in the user's guide (see http://www-device.eecs.berkeley.edu/~bsim3/bsim4.html). We used Spectre transient analyses for all circuit simulations. We assumed a 1-V power supply and a 10 percent rise/fall time for each clock speed. To measure analog power, we multiplied the supply voltage by the average current drawn from the power supply.

Analog power, speed, and accuracy

The SNAP design presents a classic tradeoff between power, speed, and accuracy. The principal factor determining circuit delay is the size of the currents produced in the DACs. Larger currents drive outputs to stabilize sooner, thereby decreasing delay 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. When the currents on the positive and negative lines are similar, the currents require more time to stabilize before a winner can be determined.

Figure 5 shows analog prediction errors for various positive/negative sum combinations; unsurprisingly, disagreements between analog predictions and predictions given by a precise implementation arise when the two sums are closest in magnitude. These errors occur because the currents didn't have 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 because similar sums allow less room for errors in current values. Incorrect currents can result from nonlinearity in

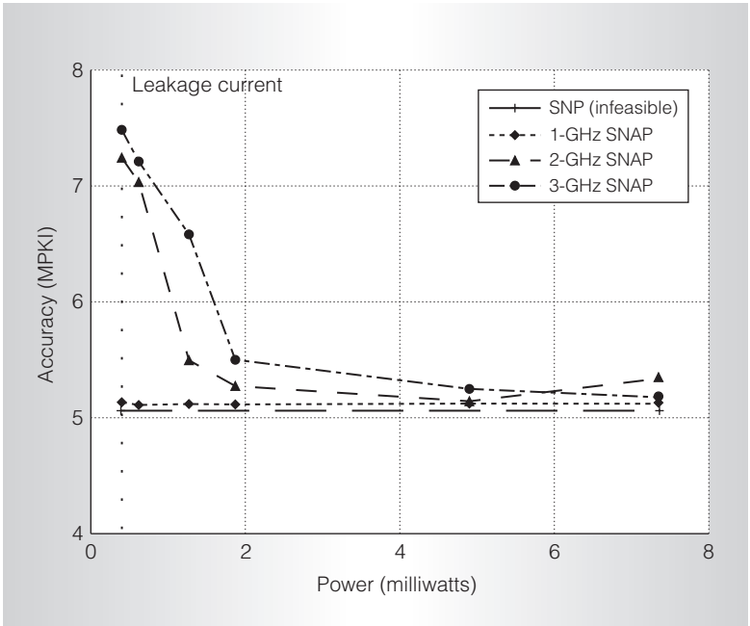

Figure 6. Trade-off between power, speed, and accuracy.

the DACs as well as process variation and noise. Fortunately, similar sums correspond to low prediction confidence because the dot-product result is close to zero and doesn't signify a strongly taken or not-taken prediction. Errors on low-confidence predictions mitigate the impact of errors on overall prediction accuracy.

The width of the error band in Figure 5 increases as clock speed increases or power decreases, causing a decrease in predictor accuracy. Figure 6 shows the relationship between power, speed, and accuracy. We adjusted the DAC bias currents to generate the multiple power levels. The lowest power shown (0.4 mW) corresponds to the DACs running strictly on leakage currents; this configuration approaches the power consumption floor at 45 nm. At 1 GHz, the predictor maintains high accuracy over the range of power levels simulated, where leakage current alone achieves an average accuracy of 5.13 MPKI (mispredictions per thousand instructions). The drop in accuracy from 4.9 to 7.4 mW at 2 GHz is a function of inaccuracy in the DAC currents coupled with the particular behavior of the traces simulated. At 3 GHz, 7.4 mW is required to achieve 5.18 MPKI.

Predictor evaluation

We measured scaled neural predictor accuracy using a trace-driven simulator derived from the Championship Branch Prediction Competition (CBP-2) contest infrastructure (http://www.jilp.org/cbp). This contest let competing predictors use approximately 32 Kbytes of state to simulate implementable branch predictors. The SNP design restricts its hardware budget similarly so that its results can be easily compared to other published results. To evaluate predictor accuracy, we simulated a set of traces that includes the SPEC CPU2006 integer benchmarks. We generated analog accuracy numbers by characterizing the analog circuit behavior as a statistical error model and mapping it back to the CBP-2 simulation infrastructure.

We compared the SNP algorithm and SNAP implementation against two other predictors from the literature: the piecewise linear branch predictor[5] and L-Tage.[1] The piecewise linear branch predictor is one of the most accurate neural predictors but has a high implementation cost. L-Tage is a table-based predictor that uses partial matching and represents the most accurate implementable predictor in the literature. (Further details concerning methodology and predictor tuning appear elsewhere.[3])

As Figure 7 shows, the scaled neural predictor is the most accurate neural predictor measured to date, although it isn't competitive from a power perspective. The scaled neural analog predictor incurs a small loss in potential prediction accuracy due to the use of nonideal analog circuits, yet it maintains higher accuracy than any previous neural predictor. The analog version achieves an average accuracy of 5.18 MPKI compared to 5.06 for the digital version; thus, the analog circuit's imprecision results in only a 0.12 MPKI decrease in accuracy. Piecewise linear branch prediction, which is less feasible than the SNAP design but still arguably implementable, results in 5.4 MPKI. L-Tage achieves an average accuracy of 4.91 MPKI.

The total power consumed by the prediction step of the SNAP design includes table lookups in addition to the analog computation. Running at 3 GHz with a 1-V supply voltage, the average analog power required to obtain high prediction accuracy is 7.4 mW.

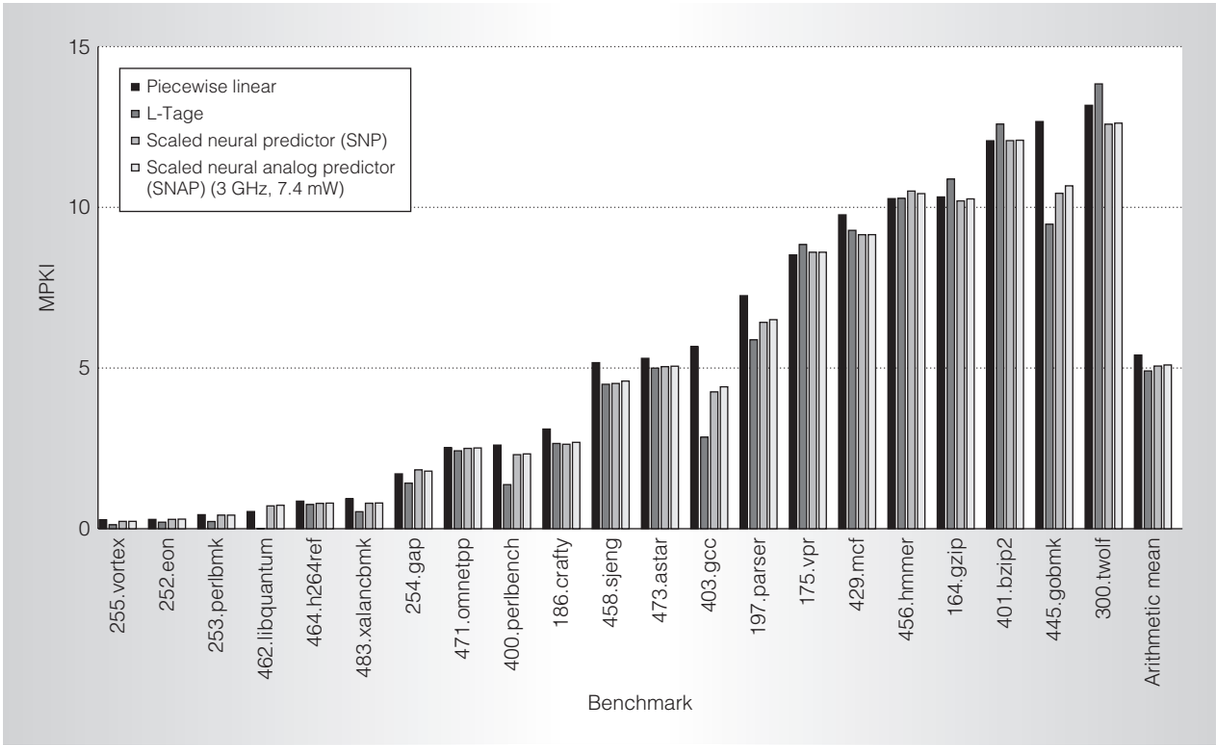

Figure 7. Accuracy of digital versus analog versions of the scaled neural predictor.

At slower clock speeds, however, the predictor achieves high accuracy at lower power levels—for example, at 1 GHz, the 0.4-mW power configuration achieves high accuracy.

We used CACTI 4.2[6] with 45-nm technology files to measure the dynamic power of the digital portion of the design. We modeled the 16 weight tables as tagless memories with 8-byte lines and estimated the total dynamic read power at maximum frequency at 117 mW. For comparison, we estimated the power consumed by L-Tage's various memory structures at 112 mW. Thus, the two predictors' memory components use comparable dynamic power, and the analog dot-product computation is a small fraction of the total power consumed by the neural predictor.

Implementation issues

To keep mixed-signal computational inaccuracy from reaching an unacceptable level, periodic digital feedback is necessary to limit the accrual of noise in the analog domain. Process variation and substrate noise are two potential causes of inaccuracy.

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, an incorrect table read or variations in the DAC transistors could lead to unexpected currents. A neural predictor training algorithm 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 predictor increases the weight. It will also adjust the nonfaulting weights, thereby quickly correcting for the faulting weight's inaccuracy. Analog neural predictors that adjust a weights vector based on a current summation are therefore more robust against process variation than table-based designs that predict based on the high bit of a digital counter value.

Noise tolerance

On-chip analog circuits are susceptible to substrate noise caused by digital switching. When numerous digital circuits are switching, they discharge currents into the substrate that cause the substrate voltage to bounce. Traditional techniques, such as guard rings, can help mitigate substrate noise's impact on mixed-signal circuits. A guard ring separates analog and digital circuits and creates a large capacitance that provides noise currents a low-impedance path to ground.

Testing

Manufacturing testing contributes significantly to production cost and time to market. In the past 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. The SNAP design can use existing digital testing techniques because it includes a DAC and a 1-bit ADC with only a small amount of analog circuitry in between.

One applicable technique uses scan chains to gain access to internal circuit components; this method scans testing data into a chain of registers from chip input pins, pulses the clock, and captures the results in registers and scans them to chip output pins. For the SNAP circuit, various input weights can be scanned in and a 1-bit prediction can be scanned out. The circuit component is determined to be good if its predictions are consistent with the expected prediction for an acceptable percentage of the time. Weight inputs and expected outputs could also be stored in memory with a pass/fail signal routed off chip.

For finer-grained failure detection, internal analog nodes can be accessed using analog scan chains, in which currents are shifted through scan chains using current mirrors and switches.[7] Miura proposed a BIST circuit to detect unexpected changes in the power supply current.[8] This technique could also detect unexpected currents through the transistors in the analog design.

We described a highly accurate neural prediction 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. By enabling more complex computation in the prediction loop, analog techniques might open the door to more aggressive neural prediction algorithms, such as the use of back propagation to compute nonlinear functions of correlation. Whether conventional, multitable-based predictors (such as L-Tage) or neural predictors will eventually prove the most accurate remains an open question.

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. As designs become predominantly constrained by power limitations and variability, we expect to see more applications of analog techniques assisting 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.

Whereas this project applies mixed-signal approximate computation to neural branch prediction, we're actively examining a broad class of applications that can tolerate imprecision. In addition to computation that relies on neural networks, candidates include graphics and media, machine learning, and convergent scientific applications.

Additionally, we're addressing opportunities to further reduce power. Whereas we presented an analog design for low-power computation, both the table lookups and the training steps require conventional amounts of power. Future solutions should

reduce the lookup and update power by pushing these functions into the analog portion of the design as well. Rather than performing a lightweight digital-to-analog conversion on the computation parameters, weight training, reading, and storage should be part of the analog domain.

In the long term, improving program performance while maintaining a firm digital abstraction might prove infeasible due to power constraints. Although digital and analog special-purpose designs have shown the ability to accelerate specific workloads in a power-efficient manner, each solution requires a focused design effort. Programmable analog general-purpose processing might prove the most cost-effective way to accelerate a range of applications. This acceleration will be possible only if the computations are insensitive to accreted noise and errors (such as convergent numerical algorithms, human-computer interfaces, and media) and/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. MICRO

Acknowledgments

Renée St. Amant is supported by a US National Science Foundation graduate research fellowship and Daniel A. Jiménez by NSF grants 0751138 and 0545898.

References

1. A. Seznec, ''A 256 Kbits L-Tage Branch Predictor,'' *J. Instruction-Level Parallelism* (JILP), vol. 9, May 2007; http://www.jilp.org/vol9/v9paper6.pdf.
2. A. Seznec, ''Redundant History Skewed Perceptron Predictors: Pushing Limits on Global History Branch Predictors,'' IRISA tech. report 1554, 2003.
3. R. St. Amant, D.A. Jiménez, and D. Burger, ''Low-Power, High-Performance Analog Neural Branch Prediction,'' *Proc. 41st Ann. IEEE/ACM Int'l Symp. Microarchitecture* (Micro 41), IEEE CS Press, 2008, pp. 447-458.
4. D.A. Johns and K. Martin, *Analog Integrated Circuit Design,* John Wiley & Sons, 1997.
5. D.A. Jiménez, ''Piecewise Linear Branch Prediction,'' *Proc. 32nd Ann. Int'l Symp. Computer Architecture* (ISCA 32), IEEE CS Press, 2005, pp. 382-393.
6. D. Tarjan, S. Thoziyoor, and N.P. Jouppi, ''Cacti 4.0,'' HP Laboratories Palo Alto, tech. report HPL-2006-86, 2006.
7. S. Wey and C.-L. Krishman, ''Built-in Self-test (BIST) Structures for Analog Circuit Fault Diagnosis with Current Test Data,'' *IEEE Trans. Instrumentation and Measurement,* vol. 41, no. 4, Aug. 1992, pp. 535-539.
8. Y. Miura, ''Real-Time Current Testing for a/d Converters,'' *IEEE Design & Test,* vol. 13, no. 2, Mar./Apr. 1996, pp. 34-41.

Renée St. Amant is a PhD student in the Department of Computer Sciences at the University of Texas at Austin. Her research interests include low-power microarchitectures and new computing technologies. St. Amant has a BS in electrical and computer engineering from the University of Texas at Austin.

Daniel Angel Jiménez is an associate professor in the Department of Computer Science at the University of Texas at San Antonio. His research interests include microarchitecture and low-level compiler optimizations. Jiménez has a PhD in computer sciences from the University of Texas at Austin. He is a member of the ACM.

Doug Burger is a principal researcher at Microsoft Research. His research interests include power-efficient, high-performance architectures, and approximate, general-purpose mixed-signal design. Burger has a PhD in computer sciences from the University of Wisconsin-Madison. He is a senior member of the IEEE and a distinguished scientist of the ACM.

Direct questions and comments about this article to Renée St. Amant, Univ. of Texas at Austin, Dept. of Computer Sciences, 1 University Station C0500, Taylor Hall 2.12, Austin, TX 78712-0233; stamant@cs.utexas.edu.