A Practical Low-Power Memristor-based Analog Neural Branch Predictor

Jianxing Wang[1], Yenni Tim[1], Weng-Fai Wong[1], and Hai (Helen) Li[2]

[1] School of Computing, National University of Singapore
[2] Swanson School of Engineering, University of Pittsburgh
{*wang1988, timyn, wongwf*}*@comp.nus.edu.sg, hal66@pitt.edu*

ABSTRACT

Recently, the discovery of memristor brought the promise of high density, low energy, and combined memory/arithmetic capability into computing. This paper demonstrates a practical neural branch predictor based on memristor. By using analog computation techniques, as well as exploiting the accuracy tolerance of branch prediction, our design is able to efficiently realize a neural prediction algorithm. Compared to the digital counterpart, our method achieves significant energy reduction while maintaining a better prediction accuracy and a higher IPC. Our approach also reduces the resource and energy required by an alternative design.

Keywords

Memristor, Branch Prediction, Neural Branch Predictor

1. INTRODUCTION

The concern about energy consumption in the current and future technology nodes [1] has driven researchers to examine alternative solutions. In particular, a number of next generation non-volatile random-access memory (NVRAM) technologies are being actively investigated as the candidates for both on-chip processor caches, and off-chip memory. Among them, the *memristor* [2] has emerged as a potential replacement for flash memory. As a nanoscale non-volatile device, it has been noted that the characteristics of memristor is highly similar to the dynamics of neural synapses, making it to be an ideal candidate in the construction of large-scale and energy-efficient neuromorphic network [3]. In this paper, we shall examine such a link through the use of memristor in neural branch predictor.

Due to the deep-pipelines of modern processors, recovery from branch misprediction is usually expensive. For instance, in the Sandy Bridge microarchitecture, a mispredicted micro-op causes the pipeline to stall for at least 15 cycles [4]. Such a penalty can significant degrade performance if enough mis-predicted branches occur along the critical path of execution. Hence, an accurate branch predictor remains a key component in improving single-thread performance.

After it was first proposed by Jiminez [5], neural branch predictor has drawn a significant amount of interest [6], [7]. Two out of five competitors participating in the latest Championship Branch Prediction [8] used neural-based algorithms. However, it is worth noting that while they are highly accurate, neural predictors are usually complex and difficult to implement. Hence most commercial processors still rely on simple and yet practical predictor like the two-level predictor [4], leaving a gap between what is ideal and what is feasible. Furthermore, existing implementations still requires conventional SRAM cells with a large silicon footprint, and incurs a significant amount of leakage energy. Our work tries to address these two issues through the use of the non-volatile memristor device in the design of a neural branch predictor.

A neural branch predictor using memristors was previously proposed in [9]. Unfortunately, no evaluation was made to show the effectiveness of such a design. In this paper, we will propose a simpler design that is as accurate, and yet consumes some 27% less resources and energy compared to the prior work. Specifically, this paper makes the following contributions:

- We present a practical low-power analog neural branch predictor design using the memristor devices.
- We investigate the energy issues in existing digital predictors, and demonstrate that our design is much more energy efficient.
- We compare the prediction accuracy and IPC rate between our predictor with several existing schemes, and show that our approach is more effective.

The rest of this paper is organized as follows: Section 2 presents some background information of the memristor device and neural branch predictor; Section 3 describes both the architectural-level and circuit-level design of our memristor-based neural predictor; Section 4 provides the experiment results and Section 5 concludes the paper.

2. BACKGROUND

2.1 Memristor

The so-called missing fourth circuit element, the memristor, was first demonstrated by Hewlett-Packard in 2008 [2]. It is a bipolar device that consists of a semiconductor thin film of thickness D sandwiched between two metal contacts. A movable boundary (domain wall) separates the mental oxide layer into two regions: a highly conductive doped region (width w), and a high resistive undoped region (width $D-w$). In HP's implementation, the undoped region contains oxygen-rich TiO_2 while the doped region is injected with auxiliary oxygen vacancies, and is thus positively charged. By applying a positive voltage to the electrode on the doped (undoped) side, the oxygen defects move towards the undoped (doped) region. As a consequence, the boundary front shifts, and the overall resistance of the device is reduced (increased). When the metal oxide layer becomes completely doped (undoped), the device enters a hysteresis state with lowest (highest) resistance. The process is reversible. By applying negative voltage to the doped side, the oxygen defects can be pushed back to the doped region and the overall device resistance would increase accordingly.

This research was supported in part by the Singapore Ministry of Education Research Grant MOE2010-T2-1-075.

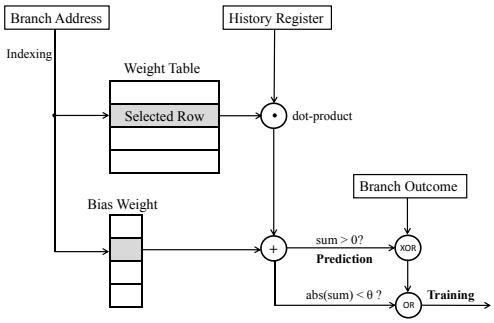


Figure 1: The perceptron predictor diagram

2.2 Neural Branch Predictor

The first implementable neural branch predictor is the *perceptron* predictor proposed by Jiminez [5]. It requires a two-dimension weight table with an extra column of bias weight to be stored in some fast SRAMs. Each row of the table consists of h n-bit signed integer weights, where h is a fixed global history length. To generate a prediction, first the branch PC address is used to index into one of the rows, and then a dot-product is performed between the global branch history vector and the selected row (excluding the bias weight). The prediction is made by examining the sign bit of the sum of the dot-product and the selected bias weight. Mathematically, the weighted sum S for row i is calculated by

$$S = W_{i0} + \sum_{k=1}^{h} (W_{ik} \times H_k) \tag{1}$$

where W_{i0} denotes the bias weight of row i, $(W_{i1}, \ldots, W_{ih})$ is the selected weight vector and $H = (H_1, \ldots, H_h)$ is the history register.

Training of the weights occurs when the prediction is wrong or the weighted sum S is less than a pre-defined threshold θ. The bias weight is increased (decreased) as the branch is actually taken (not-taken). Other weights are updated in relations to the branch outcome. If the result of a history branch i is positively (negatively) correlated to the current branch, its weight will be increased (decreased).

Compared to the conventional *gshare* [10] predictor, the main advantage of perceptron lies in its capability to capture specific relationship between a historical branch and the current one. The magnitude of a non-bias weight represents the strength between a historical branch and the current branch to be predicted, while the sign of the weight indicates the type of correlation (positive or negative). In addition, the bias weight assists the prediction for those branches with looping behavior. As a result, the perceptron predictor achieves a high prediction accuracy.

However, there is a serious drawback with the perceptron predictor: the dot-product calculation essentially determines how fast a prediction can be made. By using an adder tree, adding *h* integer numbers still requires $\log h$ operations. Although there are proposals to speed up the computation [6], they come at the expense of accuracy. Due to these issues, most neural-based schemes to date have been deemed impractical for actual implementation.

2.3 Analog-enabled Neural Predictor

The *scaled neural analog predictor* (SNAP) [7] is the first neural-based branch predictor that aims to be both accurate and feasible. It uses analog computation techniques that are commonly found in the modeling of neural network to calculate the compute-intensive dot-product. SNAP not only has a manageable prediction timing, but also has a better energy-efficiency than the digital counterpart. However, SNAP still assumes that the weight table has to be stored in SRAM, resulting in large leakage energy consumption.

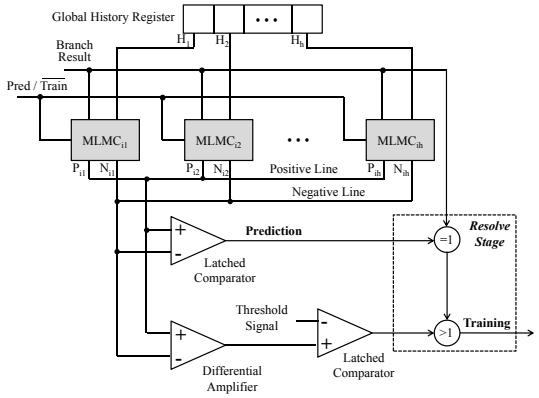


Figure 2: Design diagram of the memristor-based predictor

3. MEMRISTOR-BASED NEURAL PREDICTOR

3.1 Architecture Design

Figure 2 shows the architecture-level design of our memristor predictor. We chose to use perceptron as the underlaying prediction algorithm. However, instead of using conventional SRAM storage, each weight is now stored in a two-dimensional table of *multi-level memristor cell* (MLMC). The MLMC at position (i, j) has two analog outputs: P_{ij} and N_{ij}. The weight W_{ij} stored in this MLMC is effectively the relative difference between them, i.e., $W_{ij} = P_{ij} - N_{ij}$. By simply exchanging the roles of P_{ij} and N_{ij} inside a MLMC the weight can be negated. This is controlled by the history bit, H_j. Such a design allows us to perform multiplication by 1 and -1 in a faster manner than the digital counterpart in which a complement operation is needed.

In the prediction phase, the program counter is used to index a row of weights, $W_{i1}, \ldots, W_{ih}$. Each cell will output one of its analog current signal on a positive line and a negative line in accordance to the *global history register* $(H_1, \ldots, H_h)$ in the following way. If $H_k = 1$, P_{ik} will be put onto the positive line and N_{ik} will be output onto the negative line. If $H_k = 0$, the roles of N_{ik} and P_{ik} are reversed.

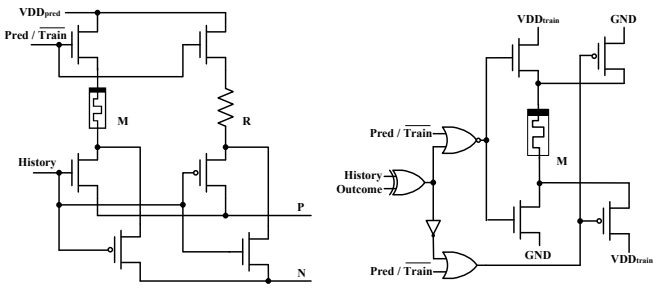
All the current signals on the shared positive (negative) line will be summed up naturally by Kirchoff's Current Law. The final prediction is made based on the relative current difference between the two lines, which is detected by an *latched comparator* [11]. If the total current on the positive line is equal or larger than that on the negative line, the branch is predicted to be 'taken'. Otherwise, it is predicted to be 'not-taken'. Note that the analog summation process is accomplished almost instantly once all the cell outputs stabilize. This provides a significant speed advantage over the digital adder tree approach.

The training signal is generated at the branch resolving stage of the pipeline. To avoid a second read of the perceptron table, we perform a threshold comparison at the prediction stage and propagate its result to the later stage of the pipeline where branches are resolved. The comparison is done by first

computing the difference between the analog signal on the positive and negative lines, followed by a latched comparison. This is done in parallel with the prediction. While training is required, all MLMCs in the selected row would be updated simultaneously given the branch result and history bits.

3.2 MLMC Circuit Design

Figure 3 gives the detailed design of a MLMC. The cell contains two sub-branches: a memristor branch, and a resistor branch. The value of R is chosen to be in the middle of the resistance range of M. When the prediction signal is activated, the upper CMOS gates saturate, and the currents going through M and R would be directed to output terminals *P* and *N* based on the history bit. The history input controls the two pairs of MOSFET below M and R. When the history is '1' (taken), M is connected to P and R is connected to N. If the history is '0' (not-taken), the roles of P and N are reversed which is equivalent to multiplying the weight by -1.



(a) Prediction circuit (b) Training circuit

Figure 3: Circuit design inside a MLMC

When a number of MLMCs are connected in the way shown in Figure 2, assuming all the history bits are set to '1' (memristor sub-branch), the relationship between the voltage on the shared positive line V_P and each MLMC is given by

$$(VDD_{pred} - V_P) \times (\frac{1}{R_{M1}} + \frac{1}{R_{M2}} + ... + \frac{1}{R_{Mh}}) = \frac{V_P}{R_L} \quad (2)$$

where R_{Mi} is the total resistance of the memristor including the two activated MOSFETs along the i-th sub-branch. R_L is the load resistor added to the input of the comparator. After simplification we get

$$V_P = VDD_{pred} / (\frac{1}{R_L \times \sum_{i=1}^{h} \frac{1}{R_{Mi}}} + 1) \quad (3)$$

If a history bit is '0' on the i-th branch, R_{Mi} is replaced by the fixed resistance on the resistor sub-branch. The value of V_N is calculated in a similar way, but the role of the history bit is reversed. The relative difference between V_P and V_N determines the final prediction. Together with the nonlinear behavior of memristor, the whole predictor is much less deterministic than its digital counterpart. However, the basic principle of perceptron predictor is still preserved under such an analog design: the increasing/decreasing of a weight (smaller/larger R_i), changes the weighted sum $(V_P - V_N)$ along a particular path, given their history correlation.

For training, auxiliary CMOS gates are added to control the programming current/voltage's direction into M (Figure 3b). When the branch history and outcome are positively correlated, the middle pair of MOSFETs are activated so that the current will push the domain wall towards the undoped side, and so decreases the resistance of M. On the other hand, if the branch history and outcome are different (negatively correlated), the right pair of MOSFETs are activated to enable a reverse current that pushes the domain wall to the doped region, thereby increasing the device resistance. Notice that the magnitude of read and write voltages are different. A small V_{DD} is chosen for prediction (read) such that the disturbance to the memristor's state is minimal, while a larger V_{DD} is required for a proper training (write) of the cell.

In theory, the memristor can be configured with arbitrary resistance levels to imitate any n-bit number. However, using more levels would result in lower noise margins that may render the difference in the output signals undetectable. For practical reasons, and after much consideration, we configured the memristor to have only 16 levels of resistances, equivalent to a digital design of 4-bit weights.

3.3 Alternative MLMC Design

The MLMC design of our predictor was inspired by Shi's work [9]. In that proposal, memristors were used in both of the branches. The major advantage of a double-memristor design is that it produces a wider output range, making the predictor more noise-tolerant. However, we noted that in the case of neural branch predictor, the exact value stored in each MLMC is not required, only the aggregated output is of concern. Hence, any error induced by the smaller differences is tolerable. Furthermore, for such scheme to work properly, both memristors inside the cell have to be programmed simultaneously during the training phase. This complicates the circuit design, and increases the programming and leakage power. As we will show in the next section, the double-memristor design does not yield better accuracy, but instead consumes more energy.

3.4 Implementation Issues

a) Analog noise and thermal fluctuation: Unlike digital circuits, analog systems, especially for those on-chip components, are more susceptible to signal noises. The current signals on the two shared lines can be disturbed by voltage bounces generated from MOSFET switchings which may affect the results of the analog computations. However, this can be mitigated through some mixed-signal IC design techniques such as the *guard rings* [12]. On the other hand, instead of using conventional SRAM storage as in SNAP [7], our design stores the perceptron weights in a purely analog form (resistance). The elimination of current-steering DACs potentially reduces the amount of noises coming from signal conversion, and thus enhances the overall robustness.

While the ion mobility inside a memristive material like TiO_2 may be affected by thermal fluctuations, there is no evidence currently to show that this issue is significant enough to have major impact on our design. As more and more materials are found to exhibit memristive property, it is likely that a commercialized memristor device will be as temperature-insensitive as the other on-chip CMOS components.

b) Memristor latency and state-drift: Since the memristor is essentially a variable-resistor, the read latency of a MLMC is dominated by the switching speed of the MOSFETs, and hence has very limited impact on the prediction latency. On the other hand, the original TiO_2-based memristor [13] has an infeasibly long programming latency under a typical V_{DD}. Such a limitation should eventually be addressed by technological advances such as the use of recent discovered

TaO-based materials [14], which has already demonstrated sub-nanosecond switching speed.

Another issue related to any memristor-based design is the *state-drift* problem. Since the existing ion-drift models do not impose a programming threshold, even a minor electric charge may disturb the memristor's state. Fortunately, research [15] has demonstrated that the internal switching dynamics requires a fairly strong electric field for programming in practice. Thus it is highly unlikely that the state-drift phenomenon will create a serious issue on our predictor design.

c) Process variation: As the fabricating process continues to scale, process variation becomes critical, and can affect some key design parameters of nanoscale devices. For the memristor, fluctuations in the resistance range directly impacts the accuracy of the analog computation. Although a neural-based predictor is more tolerant to minor weight variations, these manufacturing fluctuations could conceivably lead to performance degradation in some cases. Rajendran et. al. [16] have analyzed the effect of process variation on a memristor-based threshold gate design, and proposed two algorithms to correct these variations. Their solutions can be essentially extended to other memristor-based applications including ours.

4. SIMULATION

4.1 Experiment Setup

To evaluate our design, 20 benchmarks from the SPEC2006 suite are simulated using the cycle-accurate simulator MARSSx86 [17]. The detailed parameters for the base machine are shown in Table I. As with previous works, the accuracy of the branch predictor is measured in *Mispredictions per kilo-instruction* (MPKI), which is a common metric for evaluating branch predictors. The following predictors were evaluated in our experiment:

- **GShare**. This is our baseline predictor. While considering 16-bit global history, 64K entries are required. A typical 2-bit saturating counter width is chosen.
- **Perceptron**. The perceptron weight width is fixed at 4-bit to provide a fair comparison with the memristor-based predictor. Various global history lengths are tested and the best-performed one is chosen (48-bit).
- **SNAP**. The scaling factors are calculated using the fitting formula from [7]. While the original design had tables of different sizes, we implemented only a uniform table size solution. The global history register is fixed at 48-bit as well.
- **Memristor**. The memristor model used is a simple linear ion drift model [18] that has been configured with 16 distinct resistance levels. The double-memristor predictor scheme proposed in [9] was also evaluated.

Parameters	Value
Processor Core	Single, out-of-order
Pipeline Width	4
Pipeline/Frontend Stages	14 / 4
Fetch/Issue Queue Size	48 / 64
ROB Size	128
Physical Register File Size	256
Load/Store Queue Size	48 / 32
I-TLB/D-TLB Size	64 / 64
In-flight Branches	24
BTB Size	4K, 4-way associative
RAS Size	24
Functional Units	4 ALU, 2 FPU, 2 LU, 1 SU
Memory/Cache System	Perfect cache, 3-cycle latency

TABLE I: Simulation platform.

Although the accuracy of a linear ion drift model is usually considered to be insufficient in modeling fabricated memristor device, it satisfies the basic memristive system equation and has the advantage of being computational efficient. On the other hand, while the TaO_2-based memristor has already demonstrated sub-nanosecond switching performance, the lack of a practical model makes the simulation difficult. Therefore, some parameters in the linear ion drift model are tuned manually so that the device can work according to a GHz-level clock frequency, assuming these physical properties of memristor are scalable. Although modern fetch units may withstand the penalty of multi-cycle prediction in certain situations, for the simplicity of discussion, we assume all predictors generate a prediction within one CPU cycle.

4.2 Power Consumption

a) Leakage power: The leakage power consumed by the weight table of the digital perceptron was estimated through the modeling of a tagless table. CACTI simulation shows that storing the perceptron weights in a typical 256-row SRAM table consumes 7.9mW of leakage power (memory cells only). For a predictor like *gshare* that requires a larger budget to be effective, the leakage power will be even higher. Moreover, such a leakage power number should be common among all SRAM table-based predictors (including SNAP).

On the analog side, we used LTSpice with 45nm PTM to monitor the current passes through each MLMC for measurement of the power consumption. Simulation showed that each MLMC (single-memristor) drains 5.5nW of standby power, and thus the same-sized weight table consumes 67.6μW of leakage power in total. For comparison, the double-memristor design [9] consumes a little bit more than 92μW due to the extra MOSFETs required for programming two memristors. Hence, our approach achieves a two orders of magnitude savings in leakage power compared to the conventional SRAM-based design without performance penalty. On the other hand, the significant leakage power of the digital predictors already dominates overall energy consumption, making both the perceptron and SNAP designs inefficient.

b) Dynamic energy: The dynamic energy consumed by a prediction includes the table lookups and the analog/digital computation. Given the same table dimensions, the lookup step consumes a similar amount of energy for both analog and digital storage. For digital SRAM, the energy required to read out a 24-byte block (48 history counters of 4-bit weight) was measured at 18.9pJ. At a 1 GHz clock with a 1V V_{DD}, our analog predictor requires only 0.3pJ to obtain stable outputs on both the positive and negative lines. The alternative double-memristor scheme consumed a slightly lower 0.27pJ of read energy. For comparison, SNAP required a larger 0.4pJ energy just for its analog computation.

For training (i.e., updates), our single-memristor design consumes around 0.82pJ while the double-memristor scheme consumes more than 1.87pJ. Note that the digital perceptron consumes similar energy for both reads and updates. Figure 4 shows the total average energy breakdown for the two memristor-based schemes, averaged from all the benchmarks tested. Our single-memristor design outperforms the double-memristor approach in both leakage and dynamic energy (prediction + training). The data also shows that the double-memristor design results in a slightly better prediction energy, but requires more energy for training. In total, our design reduces energy consumption by 27.3%.

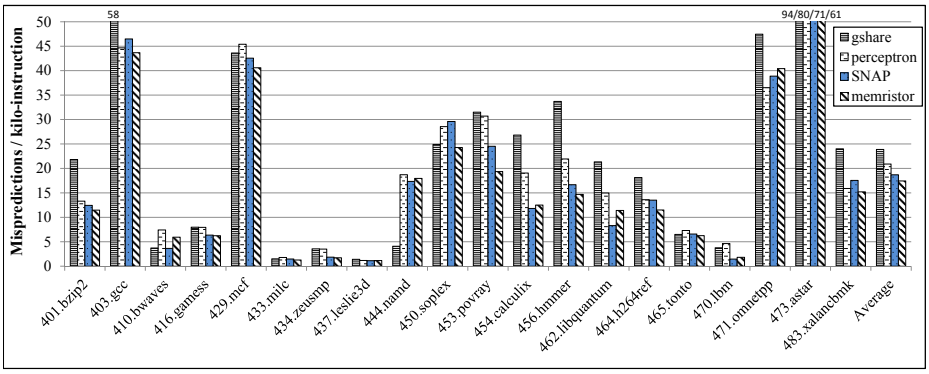


Figure 5: Mispredictions per kilo-instruction (MPKI).

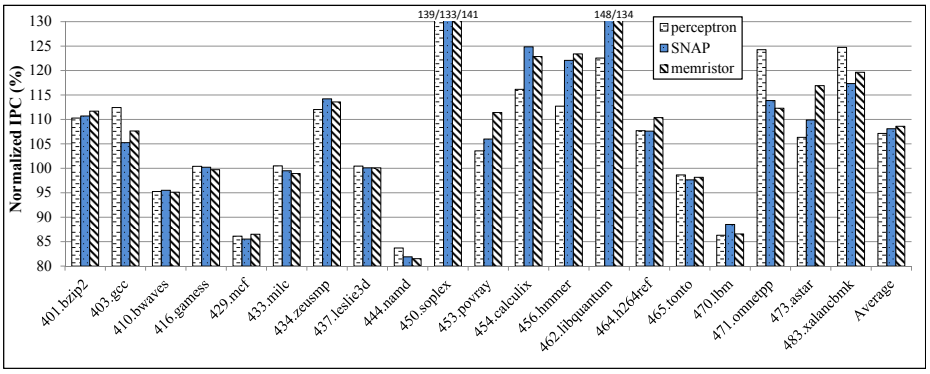


Figure 6: Normalized IPC against `gshare`.

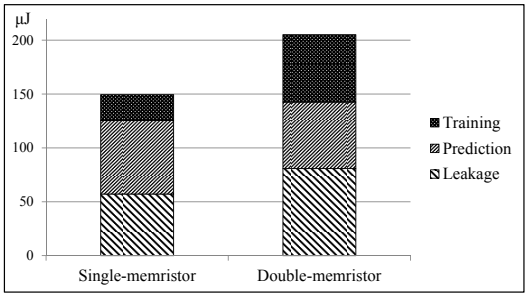


Figure 4: Energy breakdown for the memristor predictors.

4.3 Prediction Accuracy

Figure 5 shows the prediction accuracy of gshare, digital perceptron, SNAP, and our single memristor-based predictor. In certain cases like `410.bwaves`, `444.namd` and `470.lbm`, more independent looping-behavior branches are present. The interference by unrelated historical branches makes gshare more effective than the perceptron predictor. However, on average, gshare performs the worst among all the predictors. The perceptron predictor generally improves the low accuracy of gshare. In particular, for integer benchmarks like `401.bzip2`, `456.hmmer` and `471.omnetpp` that usually contain lots of complex dependent branches, larger reductions in MPKI were recorded because the perceptron algorithm can capture longer branch relationship than gshare.

When compared to digital perceptron, the analog counterparts turns out to be slightly more accurate. SNAP improves on the perceptron's weakness in predicting simple uncorrelated looping behavior branches by reducing the interference by unrelated history branches. Our memristor-based scheme, on the other hand, behaves more nonlinearly than either digital perceptron or SNAP, due to memristor's characteristic and its analog computation. This allows it to resolve certain irregular dependent branches. For example, compared to gshare, our predictor reduces by 35% the MPKI for `473.astar` which is the most difficult benchmark to predict. Another case with many irregular branches is `403.gcc`, where the memristor predictor reduces the MPKI by 24%.

4.4 Performance Analysis

The normalized IPC rates against the baseline are shown in Figure 6. For most benchmarks, a lower MPKI usually increases the IPC rate. However, all three predictors suffered larger than 13% IPC drops compared to gshare in the case of `429.mcf`. The digital perceptron only increased MPKI by a marginal amount, and the other two had lower MPKIs. Similar results were observed in `470.lbm` where the IPC for SNAP and memristor predictor decrease by more than 11% even with better MPKI numbers. Such a phenomenon shows that not all mispredicted branches share the same performance penalty. Different predictors may predict the same branch differently, even if they produce similar MPKI numbers. In certain cases, while the neural-based predictors predict more branches correctly, they also mispredicted more critical branches than gshare. Fortunately, those critical mispredicted branches usually have a limited impact on the overall performance. On average, digital perceptron, SNAP, and our memristor predictor boost the baseline IPC rate by 7.1%, 8.1% and 8.6%, respectively.

Compared with the double-memristor proposal in [9], our predictor reduces the MPKI by more than 10% with a 3% increase in IPC, under the same perceptron table dimension. Therefore, combined with the significant improvement in energy consumption, our single-memristor approach is superior.

	Perceptron	Memristor	Normalized Difference
401.bzip2	15.3	11.7	23.5%
403.gcc	76.2	31.8	58.3%
410.bwaves	8.9	6.5	26.4%
416.gamess	10.2	7.9	22.7%
429.mcf	52.5	34.4	34.6%
433.milc	2.4	1.4	44.2%
434.zeusmp	2.4	4.1	-72.4%
437.leslie3d	1.4	1.5	-2.9%
444.namd	73.8	12.5	83.1%
450.soplex	35.5	22.5	36.6%
453.povray	33.5	21.6	35.6%
454.calculix	16.1	16.2	-1.0%
456.hmmer	14.5	24.2	-67.1%
462.libquantum	12.1	14.3	-17.4%
464.h264ref	20.9	10.9	47.9%
465.tonto	11.4	4.5	60.7%
471.omnetpp	92.6	25.8	72.1%
473.astar	54.6	51.5	5.7%
483.xalanbmk	34.9	11.7	66.4%
Average	30.0	16.6	24.1%

TABLE II: Extra trainings per kilo-instruction (ETKI).

4.5 Prediction Confidence

Recall that the perceptron training process is invoked either after a misprediction has occurred, or the absolute value of the weighted sum is less than a threshold, indicating that the prediction has not yet been properly trained. MPKI does not provide information such as the confidence level of a predictor. In order to measure that, we shall analysis the training frequency of the neural predictors.

The data showed in Table II are the number of *Extra trainings per kilo-instruction* (ETKI) excluding those incurred by mispredictions. It essentially demonstrates how many predictions are essentially correct but are considered weak because of an under-trained predictor. As expected, the number of ETKI is a little more than MPKI since a neural-based predictor requires some tuning efforts after switching to another branch behavior. However, for the digital perceptron predictor, the number of weak-but-correct predictions is much larger than the mispredictions in certain cases like `444.namd` and `471.omnetpp`.

Compared to the digital counterpart, our memristor-based analog predictor significantly reduces the number of extra trainings while preserving a similar or even improving MPKI and IPC. Such an increased prediction confidence can be attributed to the inherent neural properties of memristors. Notice that the dynamics of memristor on its resistance change is a continuous, and highly nonlinear process rather than the simple discrete $+1$ or -1 arithmetic. In addition, the analog summation not only involves the memristor and resistor, but is also affected by other CMOS transistors. Even under a simplified model, the MLMCs together form a much more complicated relationship with the analog output than a simple summation of weights in the digital domain, thereby behaving more closely to a real-life neural network.

5. CONCLUSION

Neural-based branch predictors have been proven to perform very well in terms of accuracy. However, they face significant implementation challenges especially with the stringent demands of today's microprocessors. In this paper we described a practical and energy-efficient predictor design based on a promising device, namely the memristor. It mitigates the compute-intensive parts of the prediction algorithm by utilizing analog computation techniques. The use of the memristor as both a storage component as well as a compute element achieved significant leakage energy savings while maintaining the same level of prediction accuracy and IPC performance. In addition, the inherent neural property of the memristor increases prediction confidence with less predictor trainings, further reducing the dynamic energy consumed which is particular useful for resource constrained systems. As the technologies continue to evolve, there seem to be a role for analog and approximate computing in processor architectures [19]. The memristor, being CMOS compatible, opens up a number of applications that are expensive to be implemented in a purely digital environment.

REFERENCES

[1] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 365–376, 2011.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

[3] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[4] A. Fog, "The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers." [Online]. Available: http://www.agner.org/optimize/

[5] D. A. Jimenez and C. Lin, "Dynamic branch prediction with perceptrons," in *the 7th International Symposium on High-Performance Computer Architecture*, 2001, pp. 197–206.

[6] D. A. Jimenez, "Fast path-based neural branch prediction," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 243–252.

[7] R. St Amant, D. A. Jimenez, and D. Burger, "Low-power, high-performance analog neural branch prediction," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 447–458.

[8] "2nd JILP Workshop on Computer Architecture Competitions (JWAC-2): Championship Branch Prediction." [Online]. Available: http://www.jilp.org/jwac-2/

[9] S. Guangyu, "Constructing neural branch prediction with memristive device," Master's thesis, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, May 2011.

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

[11] P. Figueiredo and J. Vital, "Kickback noise reduction techniques for cmos latched comparators," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 7, pp. 541–545, 2006.

[12] D. Su, M. Loinaz, S. Masui, and B. 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, 1993.

[13] J. Yang, M. Pickett, X. Li, D. Ohlberg, D. Stewart, and R. Williams, "Memristive switching mechanism for metal/oxide/metal nanodevices," *Nature nanotechnology*, vol. 3, no. 7, pp. 429–433, 2008.

[14] F. Miao, W. Yi, I. Goldfarb, J. Yang, M. Zhang, M. Pickett, J. Strachan, G. Medeiros-Ribeiro, and R. Williams, "Continuous electrical tuning of the chemical composition of TaOx-based memristors," *ACS nano*, vol. 6, no. 3, pp. 2312–2318, 2012.

[15] M. Pickett, D. Strukov, J. Borghetti, J. Yang, G. Snider, D. Stewart, and R. Williams, "Switching dynamics in titanium dioxide memristive devices," *Journal of Applied Physics*, vol. 106, no. 7, 2009.

[16] J. Rajendran, H. Maenm, R. Karri, and G. Rose, "An approach to tolerate process related variations in memristor-based applications," in *VLSI Design*, 2011, pp. 18–23.

[17] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: a full system simulator for multicore x86 CPUs," in *DAC*, 2011, pp. 1050–1055.

[18] Z. Biolek, D. Biolek, and V. Biolková, "Spice model of memristor with nonlinear dopant drift," *Radioengineering*, vol. 18, no. 2, pp. 210–214, 2009.

[19] L. Boxun, S. Yi, H. Miao, W. Yu, C. Yiran, and Y. Huazhong, "Memristor-based approximated computation," preprint.