SiMul: An Algorithm-Driven Approximate Multiplier Design for Machine Learning

Zhenhong Liu
University of Illinois at Urbana-Champaign

Amir Yazdanbakhsh
Georgia Institute of Technology

Taejoon Park
Hanyang University

Hadi Esmaeilzadeh
University of California, San Diego

Nam Sung Kim
University of Illinois at Urbana-Champaign

The need to support various machine learning (ML) algorithms on energy-constrained computing devices has steadily grown. In this article, we propose an approximate multiplier, which is a key hardware component in various ML accelerators. Dubbed SiMul, our approximate multiplier features user-controlled precision that exploits the common characteristics of ML algorithms. SiMul supports a tradeoff between compute precision and energy consumption at runtime, reducing the energy consumption of the accelerator while satisfying a desired inference accuracy requirement. Compared with a precise multiplier, SiMul improves the energy efficiency of multiplication by 11.6x to 3.2x while achieving 81.7-percent to 98.5-percent precision for individual multiplication operations (96.0-, 97.8-, and 97.7-percent inference accuracy for three distinct applications, respectively, compared to the baseline inference accuracy of 98.3, 99.0, and 97.7 percent using precise multipliers). A neural accelerator implemented with our multiplier can provide 1.7x (up to 2.1x) higher energy efficiency over one implemented with the precise multiplier with a negligible impact on the accuracy of the output for various applications.

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

As we enter the era of dark silicon,[1] specialization and acceleration gain prevalence. Advances in machine learning (ML) are further fueling this resurgence of application-specific circuits, architecture, and even system design.[2] One of the common characteristics of ML algorithms is their robustness to noise, imprecision, and even error in data.[3] The timing combination of these two trends provides an unprecedented opportunity for rethinking many aspects of the computing stack. Although relaxing the long-held abstraction of "full accuracy" is attractive, it needs to be done while allowing the algorithms and application to decide to what extent the abstraction is relaxed. Precision flexibility is a must in many domains and, at the very least, provides an illusion of control over the added stochasticity from approximation.

In this article, we define such a "flexible" approximation at one of the lowest levels of the stack (circuits) by leveraging insights from one of the highest levels of the stack (algorithms). Such a cross-stack approach has the potential to offer a solution that significantly improves efficiency and performance of a rather larger domain of applications. Specifically, we exploit the following two insights related to ML inference algorithms: (1) Multiply-accumulate (MAC) is one of the most commonly used mathematical operations, and (2) one of the operands is often fixed after training. The first insight highlights that reducing the energy of MAC operations can have a significant effect on the overall efficiency of ML accelerators. The second insight hints that the opportunity can be exploited more effectively if the constant nature of the operand is exploited. The challenge is providing a design that is not specific to one application or algorithm, as the ML domain is volatile and the pace of innovation is fast.

To address this challenge and yet exploit the opportunity to apply approximation, we propose SiMul, an approximate multiplier with user-controlled precision and a modest area overhead. SiMul takes advantage of the fact that multiplication is iterative shift and add operations in which the shift amount depends on the position of "1s" in the binary encoding of one of the operands. Because one of the operands is fixed, it is possible to pre-process the operand and encode a limited number of shift amounts, which achieve the acceptable level of precision, instead of binary representation. Interestingly, with our technique, the value encoding becomes the knob that allows the higher-level application to tune up the level of approximation during execution. Such a circuitry can potentially further reduce energy of even the low-power accelerators that built on bit-flexibility in their architecture.[3,4] The rest of this article elaborates on the details and benefits of this novel approximation technique for ML inference applications.

CHARACTERISTICS OF ML ALGORITHMS

In this section, we identify three algorithmic characteristics common across various ML algorithms. First, some imprecision in each computation leads to a graceful degradation in inference accuracy of ML algorithms (such as in Reagen *et al.*[5]). Second, the key computations of these ML algorithms consist of multiplications of a given input vector and a pre-trained and fixed weight matrix. That is, one of the operands of each multiplication is constant. For example, an artificial neural network (ANN) consists of K inputs and L outputs. Let $u = u(t)$ denote the K-dimensional external input, $y = y(t)$ the L-dimensional output, and W the connection weight matrix of size $L \times K$ where $w_{i,j}$ is the weight between y_j and x_i. Then, y is given by $y = f(Wx + b)$ where f is a nonlinear function and b is a constant bias. We exploit these characteristics to design a far more energy-efficient multiplier with less precision loss than one that does not exploit them. Lastly, we also observe that each ML algorithm requires a very different degree of compute precision[3] for acceptable inference accuracy, although some imprecision in each computation is generally tolerable.

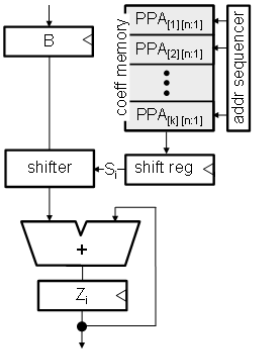
To evaluate the impact of multiplication imprecision on the final inference accuracy, we first take three ML applications: handwriting digit, isolated spoken digit, and face recognitions based on ANN, liquid state machine (LSM),[6] and support vector machine (SVM), respectively. The ANN is based on a modern multi-layer perceptron[7] trained with the Modified NIST (MNIST) dataset.[8] The LSM and SVMs are trained with Texas Instruments (TI) 46-word speech database[9] and a dataset.[10] Then, we reduce the accuracy of each multiplication by injecting a random value to the precise multiplication value such that the resulting value is up to 5, 10, 15, 20, and 25 percent larger or smaller than the precise value. For a similar level of inference accuracy (about 90

percent), LSM and SVM demand a much higher accuracy for multiplication than ANN. For example, to achieve 90-percent inference accuracy, LSM and SVM can only allow no more than 15 and 5 percent errors, respectively, while ANN can tolerate more than 25 percent errors for each multiplication. This provides a strong motivation for a multiplier with tunable compute precision to allow us to minimize energy consumption while achieving an acceptable inference accuracy.

APPROXIMATE MULTIPLIER SUPPORTING VARIABLE PRECISION AND ENERGY CONSUMPTION

A MAC operation is the most fundamental and critical computation element for nearly all ML algorithms. Although weight values of most ML algorithms are initially trained with floating-point arithmetic, they are converted to fixed-point numbers for inference due to the high delay, area, and power cost of supporting floating-point operations in hardware. We go a step further and present a significance-driven iterative approximate multiplier (SiMul), exploiting the three observations made previously.

Generally, we multiply the multiplicand and the multiplier values by repeatedly performing shift and add operations on the multiplicand value where the shift amounts are determined by the bit positions of 1s in the multiplier value. Figure 1(a) shows the block diagram of our SiMul, consisting of a K-bit shifter, adder, and shift register where K denotes the bit width of the multiplier. A and B denote given multiplier and multiplicand values, respectively. S_i and Z_i indicate a shift amount and output at i^{th} iteration, respectively. Because the trained weight (multiplier) values are determined and known before inference runs, we propose pre-processing each given weight value to determine the bit positions of "1s" in advance and storing the pre-processed weight value instead of the actual binary values in on-chip memory denoted by PPA[k][n:1]. PPA[k][i] represents the shift amount for the k^{th} weight value at i^{th} iteration (or the bit position of the i^{th} leading 1 for the k^{th} weight value).



(a) Multiplier

SiMul-LOD-PP:

$A = 01111101_2$ $S_1 = 110_2$

$B = 01011010_2$

$Z_1 = 0000000000000000_2$

$+ 0001011010000000_2$

SiMul-ELOD-PP:

$A = 01111101_2$ $S_1 = 111_2$

$B = 01011010_2$

$Z_1 = 0000000000000000_2$

$+ 0001011010000000_2$

(b) Examples of 1st iteration

```
A[1] = A;
for(i=1; i ≤ n; i++){
m = find_leading_one_bit_index(A[i]);
 // check the (n-i+1) bits starting from bit m
 if(A[i][m:m-(n-i)] == {(n-i+1){1'b1}}){
   // compensation for consecutive 1s
   S[i] = m+1; A[i+1] = 0;
  } else{
   S[i] = m;
   //clear current leading one bit
   A[i+1] = A[i]&(~(1 << S[i]));
  }
}
```

(c) ELOD-PP algorithm pseudo-code for n iterations

Figure 1. SiMul exploiting preprocessed coefficients.

For a given weight value A (= 01111101_2 (125_{10})) and the pre-determined number of iterations n (= 4), we store the first, second, third, and fourth leading 1s in A (= 110_2 (6_{10}), 101_2 (5_{10}), 100_2 (4_{10}), and 011_2 (3_{10})) in PPK[k][1], PPK[k][2], PPK[k][3], and PPK[k][4], respectively. In the first iteration illustrated in "SiMul-LOD-PP" of Figure 1(b), we take the bit position of the first leading 1 (110_2 (6_{10})) from PPK[k][1], left-shift B by that amount (such as 01011010_2 (90_{10}) << 110_2 (6_{10})), and accumulate the shifted value to Z_0 that is initialized to 0s before the first iteration begins; LOD-PP stands for leading 1 detection pre-processing. In this example, the result of the first iteration is $Z_1 = 0000101101000000_2$ (5760_{10}). In the second iteration, we repeat the same procedure as the first iteration but for PPK[k][2] (= 01011010_2 (90_{10}) << 101_2 (5_{10})) + Z_1), obtaining 8640_{10} ($5760_{10} + 2880_{10}$) or accomplishing 77-percent accuracy (= 8640_{10} / ($125_{10} \times 90_{10}$)). We may repeat the same procedure for the subsequent two leading 1s, providing 98-percent and about 100-percent accuracy after the third and fourth iterations, respectively.

Note that the shift register stores n shift amounts PPK[k] and left-shifts the stored value by $\log_2(K)$ bits (= 5 bits for a 32-bit multiplier) at every iteration to supply a necessary amount for shifting B at each iteration. That is, we need only 20 bits per PPA[k] for four iterations while we should provide 32 bits per entry for storing actual weight values in the on-chip memory. This also reduces on-chip data transfer energy, although it is not the key focus of this article.

More intelligently pre-processing the weight values to obtain the shift amounts can considerably improve accuracy of SiMul when we allow only one or two iterations for a multiplication. For instance, assume that A and B are 01111101_2 and 01011010_2, respectively, while limiting the number of iterations (denoted by n) to one. The first iteration in the previous example (SiMul-LOD-PP) gives 0001011010000000_2 (5760_{10}) or achieves 51-percent accuracy. This low accuracy after the first iteration is because we cannot consider the four consecutive 1s after the first leading 1 bit; these 1s represent a substantial percentage of the magnitude of the multiplier value (26 percent and 13 percent for the first and second consecutive 1s after the first leading 1, respectively). To improve the accuracy of the proposed SiMul-LOD-PP for small n, we propose a SiMul with an enhanced LOD-based pre-processing technique (denoted by SiMul-ELOD-PP in Figure 1(b)); we pre-process each weight value such that the S_i values minimize the absolute error ($1 - |Z_{approx} / Z_{accurate}|$ where Z_{approx} and $Z_{accurate}$ are the approximate and precise values, respectively) considering the maximum number of intended iterations (n). For example, in Figure 1(b), we can choose $S_1 = 111_2$ instead of 110_2 when only one iteration is intended for SiMul. With the S_1 determined by ELOD-PP, the result of the approximate multiplication is 0010110100000000_2 (11520_{10}), as shown in "SiMul-ELOD-PP" of Figure 1(d). This provides 98-percent accuracy, reducing the error from 49 percent to 2 percent. Figure 1(c) describes the ELOD-PP algorithm.

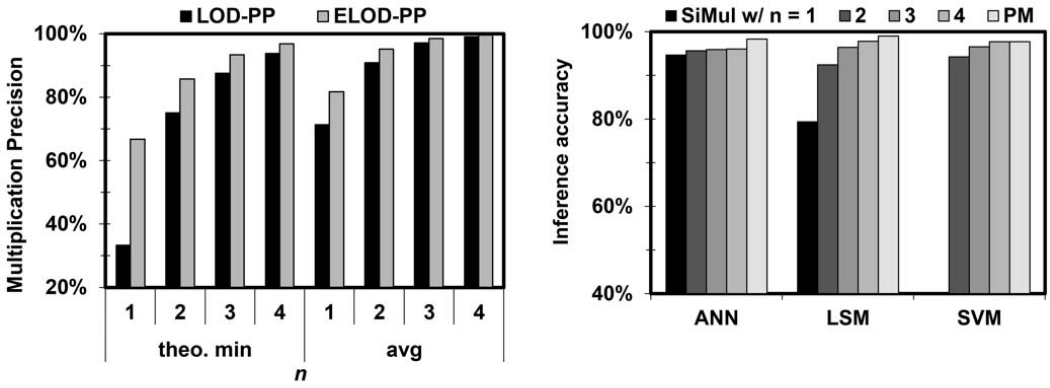
Prior work[6] detects 1s in a given multiplicand (input) value "on the fly" (either bit by bit or using one or more hardware-based LODs). That is, serializing multiplication operation hurts the throughput or providing hardware-based leading 1 detectors significantly diminishes energy efficiency; our study shows that the hardware-based leading 1 detectors for 32-bit operands is substantial. In contrast, SiMul-ELOD-PP "pre-processes" relatively unchanging multiplier (weight) values to determine the bit positions of leading 1s, eliminating the expensive hardware-based leading 1 detectors.

COMPUTE AND INFERENCE ACCURACIES

Multiplication Accuracy

Figure 2(a) plots the minimum and average accuracy of the 32-bit SiMul-LOD-PP and SiMul-ELOD-PP for varying n. As expected, the multiplication accuracy considerably increases with more iterations; the SiMul-LOD with one, two, three, and four iterations achieves 71-, 91-, 97-, and 99-percent average accuracy for 100,000 pairs of randomly generated operands. The accuracy improvement with the SiMul-ELOD-PP is notable for one and two iterations; the minimum and average accuracies of the SiMul-ELOD-PP are 67 and 86 percent. They are 33 and 10 percent higher than those of the SiMul-ELOD-PP for one iteration, and 10 and 5 percent higher for

two iterations. This demonstrates the effectiveness of the proposed ELOD-PP technique when n is limited to one or two. Considering the preliminary inference accuracy versus multiplication accuracy of the LSM-, ANN-, and SVM-based ML applications, we expect that an n that is fewer than four should be sufficient for most inference applications to achieve high enough inference accuracy.



(a) Multiplication accuracy (b) Inference accuracy

Figure 2. Multiplication and inference accuracies for different n values.

We can also compare the accuracy of the SiMul-ELOD-PP with that of two other approximate multipliers (in other words, an iterative approximate log multiplier (IALM)[11] and a precise multiplier with the least significant bits (LSBs) of the operands are masked to 0s). For one iteration, it was reported that the theoretical minimum and average accuracies of the IALM are 75 and 91 percent, respectively. Although the IALM is more accurate than the SiMul-ELOD-PP for one iteration, the energy consumption of IALM is much higher for two iterations due to its complexity. For a precise multiplier, masking some LSBs of A and B to 0s can reduce the dynamic energy consumption because the logic associated with the LSBs will not switch. However, we observe that masking even a small number of LSBs significantly reduces the accuracy, and thus it may not be able to provide sufficient compute precision for some ML algorithms; masking four, eight, and 12 LSBs reduces the average compute precision to 78, 48, and 15 percent, respectively. The reason is that for small fixed-point numbers where 1s are in the LSB side, simple truncation often leads to zero, incurring a huge "relative accuracy" loss when these values are multiplied. In contrast, our multiplier is very effective even for multiplications of small values because ours captures significant 1s whether they are in the MSB or LSB side.

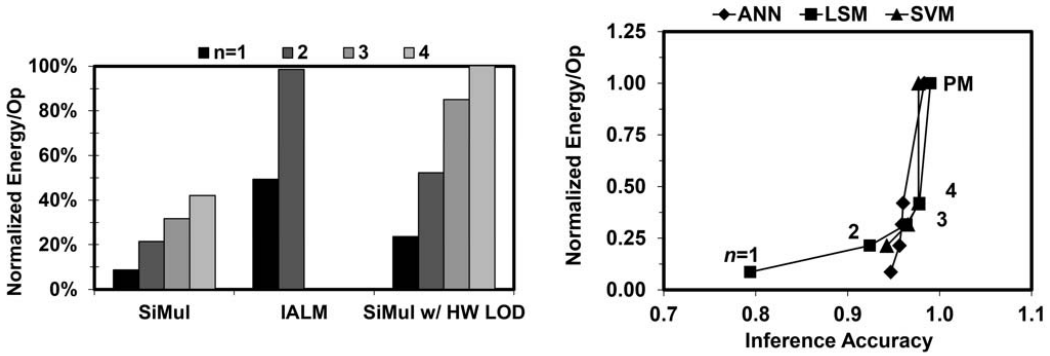
Inference Accuracy

Figure 2(b) plots the inference accuracy when the SiMul-ELOD-PP and a precise multiplier (PM in Figure 2(b)) are employed for matrix vector multiplications. For the handwriting digit recognition using ANN, one iteration can provide 95-percent recognition accuracy; a fused add-multiply can offer only 3.6-percent higher accuracy than the SiMul-ELOD-PP with one iteration; more iterations marginally improve the recognition accuracy. In contrast, the isolated spoken digit recognition using LSM, which exhibits very low accuracy with one iteration, needs at least three iterations to achieve its recognition accuracy close to the inference accuracy using a precise multiplier. Face recognition using SVM requires four iterations to achieve inference accuracy comparable to the inference accuracy using the precise multiplier. Note that the inference accuracy of SVM becomes nearly zero when compute accuracy is below 10 percent. Because the average compute accuracy of the SiMul-ELOD-PP with one iteration is below 10 percent, the accuracy bar for SVM with one iteration is now shown in Figure 2(b). This analysis confirms our earlier discussion that each ML algorithm requires a different degree of compute precision to achieve sufficient inference accuracy. We can optimize n such that the SiMul minimizes the energy consumption without notably impacting recognition accuracy.

ENERGY AND AREA EVALUATION

For energy and area evaluations, we synthesize 32-bit SiMul-ELOD-PP and precise multiplier units using Synopsys Design Compiler (32-nm PDK offering standard cells with high and low threshold voltage choices) and Power Compiler. We use Synopsys DesignWare library for 32-bit adder and 32-bit fused add-multiply for fixed-point arithmetic. For the synthesis, we take the standard library cells characterized with a typical process corner and set the target cycle time of all the multiplier implementations to 1.2 ns determined by the minimum delay of a 32-bit precise multiplier. To accurately measure the energy consumption of the multipliers, we first collect 10,000 pairs of operands, after running the isolated spoken digit recognition application implemented with C. Second, to capture the switching activities of the nodes in these multipliers, we perform gate-level logic simulations with the collected operand traces. Finally, we apply the captured switching activities to Synopsys Power Compiler to measure the dynamic and leakage power consumption.

Figure 3(a) plots the energy consumption per operation (or simply energy/op) of two iterative approximate multipliers (SiMul-ELOD-PP and IALM), normalized to that of the 32-bit precise multiplier. For each iteration, the SiMul-PP consumes about 10 percent of the energy of the precise multiplier, which consumes 5.82 pJ/op. Thus, the SiMul-PP consumes only 42 percent of the conventional multiplier even for four iterations (a more than 2x reduction in energy/op). The IALM consumes as much energy/op as the fused add-multiply for two iterations. Finally, the precise multiplier masking four, eight, and 12 LSBs to 0s consumes 4.7, 4.3, and 3.2 pJ/op, respectively.



(a) Normalized energy per multiplication (b) Inference accuracy vs energy per multiplication

Figure 3. Normalized energy per multiplication and inference accuracy versus energy per multiplication for different n values.

Figure 3(b) plots the energy/op of the SiMul-ELOD-PP versus inference accuracy of ANN, LSM, and SVM algorithms. Increasing n linearly increases energy/op while ANN's recognition accuracy is sufficiently high (94.7 percent) even with one iteration, where we can achieve nearly 12x improvement in energy efficiency at the cost of 3.6-percent degradation in recognition accuracy compared to the precise multiplier. In contrast, LSM's recognition accuracy is not sufficient with one iteration (79 percent). However, when n is increased from one to two, we observe a considerable increase in inference accuracy (92 percent), and, practically, the inference accuracy does not increase with n larger than 3. When we choose three iterations for LSM, SiMul-ELOD-PP can improve the energy efficiency by 3.2x at the cost of 2-percent degradation in inference accuracy, compared to the precise multiplier. Finally, SiMul-ELOD-PP requires two iterations for SVM to achieve the inference accuracy comparable to the accuracy that can be achieved by the precise multiplier, improving the energy efficiency by 4.7x.

Note that the energy/op increases dramatically for marginal improvement in inference accuracy, as shown in Figure 3(b), and thus it is critical to tune compute precision before the improvement in inference accuracy quickly diminishes to achieve the maximum energy efficiency. The synthesis results show that the area of the SiMul-ELOD-PP (1,622 μm^2) and IALM (4,657 μm^2) is 26

and 75 percent that of the precise multiplier unit (6,234 μm^2), respectively. Compared to a 32-bit floating-point multiplier, SiMul-ELOD-PP requires only 15 and 3 to 12 percent of the floating-point multiplier area and energy per multiplication. Finally, implementing hardware-based LOD similar to prior work[4] almost doubles the area of SiMul-ELOD-PP and loses the energy reduction benefit when n is increased from three to four, as shown in "SiMul w/ HW LOD" of Figure 3(b); prior work evaluated a 16-bit multiplier, while this work considered a 32-bit multiplier where the hardware-based leading 1 detector overhead is more significant.

Lastly, we implemented general-purpose neural accelerators[12] with SiMul-ELOD-PP multipliers, fixed-point precise multiplier (FxP PM), and floating-point precise multiplier (FP PM) running the AxBench suite[13] and its dataset for general-purpose approximate computing. This is an example of an ANN application that needs 32-bit multiplications to provide high compute accuracies.[12] Figure 4(a) plots the whole application energy consumption of the neural accelerators, normalized to that of the neural accelerator implemented with floating-point precise multipliers. We fixed the number of iterations of shift-and-add in SiMul-ELOD-PP to three/four to keep the output quality loss to an acceptable level. On average, SiMul-ELOD-PP in neural accelerators yields 1.7x lower energy than floating-point precise multiplier across the evaluated applications. Figure 4(b) shows the output quality loss of the evaluated applications using a neural accelerator with different multipliers. The quality loss is compared with that of the original precise execution of the application on a baseline CPU with no acceleration. Using a fixed-point precise multiplier has virtually no impact on the output quality loss compared to using a floating-point precise multiplier. These results are commensurate with previous work,[6] which uses lower precision for the implementation of the neural accelerators. On average, using SiMul increases the output quality by only about 2 percent compared to using a floating-point precise multiplier and fixed-point precise multiplier.

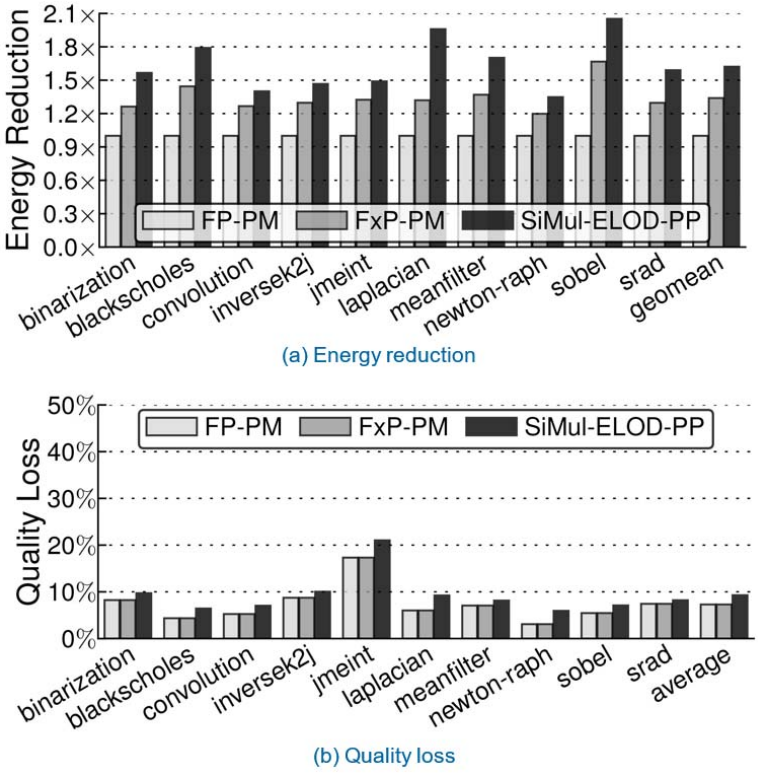


Figure 4. Comparison of general-purpose neural accelerators with a 32-bit SiMul-ELOD-PP multiplier, FxP PM, and FP PM.

RELATED WORK

Our work lies at the intersection of approximate computing and ML research. We are fundamentally different from the prior work in two major ways: (1) Instead of using fixed-precision for multiplication, we devise a novel multiplier of which we can dynamically trade off between compute precision and energy consumption, and (2) we leverage the insight that ML applications have generally a set of pre-learned parameters, which enables us to significantly simplify the design of the proposed approximate multiplier.

There are a handful of articles on architecture for neural acceleration.[4,12,14-16] These proposals either use precise fixed-point multipliers or perform the multiplication in serial with costly online hardware-based checking mechanisms to decide the termination point. Using a serial multiplier hurts throughput, and online hardware-checking diminished the energy reduction benefit from performing serial multiplication. However, SiMul leverages the availability of the pre-trained parameters in many ML applications to simplify the design of the multiplier. Kulkarni *et al*.[17] and Gupta *et al*.[18] proposed under-designed multiplier and adder designs in which the minimum/maximum precision is fixed at design time. In contrast, our design varies precision (and energy consumption) through iterations. Babića *et al*.[11] proposed an ILAM that can support variable precision through iterations, but our SiMul-ELOD-PP is far more energy-efficient with a smaller area than the ILAM. This is because we exploit fixed coefficient values in one of the multiplier's operands and pre-process them to maximize compute precision for n. This allows us to support a finer-grain tradeoff between precision and energy consumption than the ILAM. Various fixed-coefficient multipliers were proposed in the DSP domain.[19] Such multipliers require look-up tables per unique coefficient (such as 16 12-bit entries per coefficient for an 8×8 multiplier[19]), while our design only requires one 20-bit entry per coefficient. Finally, our work is orthogonal to circuit-level scaling techniques,[14] and it can be incorporated with their algorithm- and architecture-level scaling techniques.

CONCLUSION

In this article, we first identified three common characteristics of various ML algorithms: (1) tolerance to compute imprecision, (2) weight values fixed after a training phase, and (3) a different multiplication precision requirement per application for acceptable inference accuracy. Second, exploiting these characteristics, we proposed SiMul, an approximate multiplier with controlled precision, which is used to implement a neural accelerator for ML applications. SiMul supports a runtime tradeoff between multiplication precision and energy consumption, achieving 82- to 99-percent multiplication accuracy with 11.6x to about 3.2x less energy per multiplication than an optimized precise multiplier. Third, SiMul enables neural accelerators to minimize energy consumption while satisfying a minimum inference accuracy requirement, achieving 96.0-, 97.8-, and 97.7-percent inference accuracy compared to the inference accuracy of 98.3, 99.0, and 97.7 percent using precise multiplier for running ANN, LSM, and SVM-based applications, with 2.9x to about 7.8x higher energy efficiency than the one with the precise multiplier. Finally, we evaluated SiMul in a neural accelerator setting. Compared to an accelerator with fixed-point and floating-point multipliers, the proposed approximate multiplier reduces the energy consumption by 1.3x and 1.7x, respectively, with marginal variation in the output quality across a wide range of applications in AxBench.

ACKNOWLEDGMENTS

This work is supported in part by Samsung Electronics and the NSF (CNS-1705047).

REFERENCES

1. H. Esmaeilzadeh et al., "Dark Silicon and the End of Multicore Scaling," *Proceedings of the 38th International Symposium on Computer Architecture* (ISCA), 2011.

2. J. Park et al., "Scale-out Acceleration for Machine Learning," *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), 2017.
3. H. Sharman et al., "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Networks," *Proceedings of the 45th International Symposium on Computer Architecture*, 2018.
4. J. Albericio et al., "Bit-Pragmatic Deep Neural Network Computing," *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), 2017.
5. B. Reagen et al., "Minerva: Enabling Low-Power, Highly Accurate Deep Neural Network Accelerators," *Proceedings of the 43rd International Symposium on Computer Architecture* (ISCA), 2016.
6. D. Verstraeten et al., "Isolated Word Recognition with the Liquid State Machine: A Case Study," *Information Processing Letters*, vol. 95, no. 6, 30 September 2005, p. 521-528.
7. A. Damien, *Tensorflow Examples*; https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/multilayer_perceptron.py.
8. Y. LeCun et al., "The MNIST database of handwritten digits"; www.yann.lecun.com/exdb/mnist.
9. M. Liberman et al., "The TI 46-word speech database"; https://catalog.ldc.upenn.edu/ldc93s9.
10. O. Sakhi, "Face Detection using Support Vector Machine (SVM)"; www.mathworks.com/matlabcentral/fileexchange/29834-face-detection-using-support-vector-machine-svm.
11. Z. Babić, A. Avramović, and P. Bulić, "An Iterative Logarithmic Multiplier," *Microprocessors and Microsystems*, vol. 35, no. 1, February 2011, pp. 23–33.
12. H. Esmaeilzadeh et al., "Neural Acceleration for General-Purpose Approximate Programs," *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO), 2012.
13. A. Yazdanbakhsh et al., "AxBench: A Multi-Platform Benchmark Suite for Approximate Computing," *IEEE Design & Test*, vol. 34, no. 2, April 2017, pp. 60–68.
14. V. K. Chippa et al., "Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency," *Proceedings of the 47th Design Automation Conference* (DAC), 2010.
15. A. Yazdanbakhsh et al., "Neural Acceleration for GPU Throughput Processors," *Proceedings of the 48th International Symposium on Microarchitecture*, 2015.
16. A Yazdanbakhsh et al., "GANAX: A Unified SIMD-MIMD Acceleration for Generative Adversarial Network," *Proceedings of the 45th International Symposium on Computer Architecture*, 2018.
17. P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading Accuracy for Power with an Underdesigned Multiplier Architecture," *Proceedings of the 24th International Conference on VLSI Design*, 2011.
18. V. Gupta et al., "IMPACT: IMPrecise adders for low-power approximate computing," *International Symposium on Low Power Electronics and Design* (ISLPED), 2011.
19. K. Chapman, "Constant Coefficient Multipliers for the XC4000E," 1996.

ABOUT THE AUTHORS

Zhenhong Liu is a PhD student in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. His research interests include approximate computing and GPU architecture. He has a master's degree in electrical and computer engineering from the University of Wisconsin-Madison. Contact him at zliu118@illinois.edu.

Amir Yazdanbakhsh has a PhD in computer science from the Georgia Institute of Technology, where he worked as a research assistant in the Alternative Computing Technologies (ACT) Lab. His research interests include computer architecture, approximate general-purpose computing, programming language for hardware design, and deep reinforcement learning. He is also interested in exploring the interplay between machine-learning techniques and efficient computing system design. His work has been recognized by multiple fellow-

ships, including Qualcomm Innovation Fellowship and Microsoft Graduate Research Fellowship. Yazdanbakhsh is a member of the IEEE and ACM. Contact him at a.yazdanbakhsh@gatech.edu.

Taejoon Park is a professor at Hanyang University, where he chairs the Department of Robotics Engineering and directs the Collaborative AI-Robotics in Engineering (CARE) Center. He has a PhD in electrical engineering and computer science from the University of Michigan, Ann Arbor. He previously studied electrical engineering at Korea Advanced Institute of Science and Technology (KAIST) and Hongik University. His research interests are in cyber-physical systems (CPS) and AI with emphasis on deep learning and their applications in robots, vehicles, and factories. He has authored or coauthored 130+ papers/patents, including essential patents for the DVD standard, six of which were cited 100+ times. He is a member of the IEEE and ACM. Contact him at taejoon@hanyang.ac.kr.

Hadi Esmaeilzadeh is an associate professor in the Computer Science and Engineering Department at the University of San Diego, California, where he received early tenure. His research interests lie at the intersection of compute architecture, AI, robotics, and programming language. He is the founding director of the ACT Lab and is the recipient of the IEEE Technical Committee on Computer Architecture (TCCA) Young Computer Architect Award. He is a member of the International Symposium on Computer Architecture (ISCA) Hall of Fame. He has a PhD in computer science and engineering from the University of Washington, where his dissertation was recognized by the William Chan Memorial Dissertation Award. Contact him at hadi@eng.ucsd.edu.

Nam Sung Kim is a professor in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. His research incorporates device, circuit, architecture, and software for power-efficient computing. Kim has a PhD in computer engineering and science from the University of Michigan, Ann Arbor. He has received many awards, including from the IEEE/ACM Design Automation Conference, IEEE/ACM International Symposium on Microarchitecture (MICRO), NSF, University of Wisconsin, and IBM. He is a member of the IEEE International Symposium on High-Performance Computer Architecture and MICRO halls of fame. He is a Fellow of the IEEE. Contact him at nskim@illinois.edu.