An Energy-efficient Programmable Mixed-Signal Accelerator for Machine-Learning Algorithms

Mingu Kang[§]*, Prakalp Srivastava[†]*,
Vikram Adve[†], Nam Sung Kim[†], and Naresh R Shanbhag[†]
[†]University of Illinois, Urbana-Champaign; [§]IBM TJ Watson Research Center, Yorktown Heights;

Abstract—We propose PROMISE, the first end-to-end design of a PROgrammable MIxed-Signal accElerator from Instruction Set Architecture (ISA) to high-level language compiler for acceleration of diverse machine learning (ML) algorithms. We take advantage of the superior energy efficiency from analog/mixed-signal processing by exploiting the inherent error tolerance of ML algorithms. We first propose an ISA to support operations pervasive in ML algorithms with a PROMISE architecture based on silicon-validated components. Second, we develop a compiler that can take a high-level programming language (Julia) and generate PROMISE code with an IR design. Third, we show how the compiler can map an application-level error tolerance specification for neural networks down to low-level hardware parameters to minimize energy consumption. PROMISE achieves 2.3× delay and 4.5× energy savings, and 14% additional energy savings with compiler optimization, on average for diverse ML algorithms as compared to digital ASICs.

1 Introduction

Emerging applications such as in health care, surveillance/monitoring and others leverage the decision making capability based on machine learning (ML) algorithms. Those algorithms have demanded high computing capability to process large volume of data with limited energy budget. In order to improve energy efficiency, researchers have proposed analog/mixed-signal accelerators [1], [2]. These rely on small-signal computation which is less precise but more energy efficient than traditional large-signal computation in the digital domain (See Fig. 1). Therefore, they are suitable for ML inference where the application itself is tolerant to such imprecision.

However, these accelerators lack a programmable architecture, instruction sets, or compiler support necessary for supporting high-level programming languages. Moreover, the small-signal computations create energy vs. accuracy trade-offs that must be controlled from the application level in order to ensure that accuracy goals are met. Designing hardware architecture and instruction set support to expose the available hardware knobs to software in a controllable way require careful hardware, ISA and software design.

Tackling these challenges, we propose PROMISE, the first end-to-end design of a PROgrammable MIxed-Signal accElerator from Instruction Set Architecture (ISA) to high-level language compiler for acceleration of diverse ML algorithms. This article makes the following major contributions:

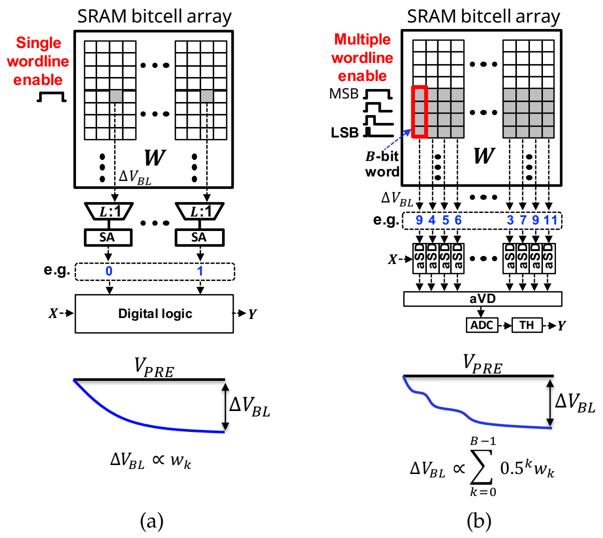

Fig. 1: Block diagrams and read operations with bitline swing (ΔV_{BL}): (a) conventional system, (b) Compute Memory (CM), with bit precision of scalar weight w, $B = 4$ and column mux ratio $L = 4$.

- We propose an ISA with a PROMISE architecture built with silicon-validated components.
- We develop a compiler that takes a high-level programming language (Julia) and generate PROMISE code.
- The compiler maps an application-level error tolerance specification down to low-level hardware parameters (swing voltages ΔV_{BL}) to minimize energy consumption.
- In this extended article, we show how to maximize the throughput of deep neural network (DNN) by optimally mapping the algorithm to a multi-bank structure.
- We also extend our results from [3] to enable

*First two authors contributed equally to the paper the convolutional neural network (CNN) based on PROMISE ISA and compiler infrastructure.

- Analog kernel data reuse is enabled by employing a charge-recycling mixed-signal multiplier [4].

2 Background

In this section, we identify commonalities across various ML inference algorithms and describe a mixed-signal accelerator well-suited for ML algorithms.

2.1 ML Algorithms

$$y_j = f\left(D(W_j, X)\right) = f\left(\sum_{i=1}^{N} d(w[j][i], x[i])\right) \quad (1)$$

The ML algorithms involve repeated Vector Distance (VD) computations denoted by $D(W_j, X)$ between N-dimensional input vector X and weight vector W. Commonly used VD computations include the dot product, L1 distance, L2 distance, and Hamming distance.

These ML algorithms have the following three dataflow properties in common. **(P1)** A single VD is obtained by first computing N element-wise Scalar Distances (SDs) $(d(w[j][i], x[i]))$ followed by an aggregation step such as a sum generating the final scalar VD $D(W, X) = \sum_{i=1}^{N} d(w[j][i], x[i])$. **(P2)** The VD between a single query vector X and multiple (say N_o) weight vectors W_js ($j = 1, 2, ...N_o$) needs to be computed. **(P3)** The VD goes through a simple decision function $f()$ such as sigmoid or ReLu to generate the decision y_j.

2.2 Mixed-Signal ML Accelerator

The compute memory (CM) [1], [2] deeply embeds energy-efficient analog computations into the periphery of conventional bitcell array. More specifically, CM stores a B-bit word in a column-major format (i.e., a word is stored in B bitcells connected to the same bitline (BL) across B rows), as shown in the red box of Fig. 1(b)) whereas conventional memory does in a row-major format (Fig. 1(a)). After BLs are pre-charged to V_{PRE} for a read cycle, CM simultaneously asserts B_w wordlines (WLs). The durations of these asserted WLs are proportional to the binary weight values of the corresponding bit positions in a given B-bit word with a binary Pulse-Width Modulated (PWM) WL signaling scheme (Fig. 1(b)). Subsequently, each BL develops a voltage drop (ΔV_{BL}) proportional to a binary-weighted sum of B bits in the corresponding column, which constitutes the first processing stage of CM: **(S1)** analog Read (`aREAD`). `aREAD` can not only seamlessly convert digital values stored in memory into analog values for subsequent analog computation stages, but also fetches highly condensed B-bit information per BL, significantly improving energy efficiency and throughput.

For ML algorithms, CM stores pre-trained W_j in its bitcells, then it can serve as a very energy-efficient mixed-signal ML accelerator with the following stages: **(S2)** analog Scalar Distance (`aSD`) implementing scalar distance computations right next to the bitcell array; and subsequent analog Vector Distance (`aVD`) performing the aggregation ($\sum_{i=1}^{N}$ in (1)) by simply charge-sharing all the analog outputs from `aSD` blocks in one shot; **(S3)** Analog-to-Digital Conversion (`ADC`) converting the analog output of `aVD` into a digital word, and **(S4)** ThresHold (`TH`) generating a final decision from the digital word based on a given decision function $f()$ in (1). Note that the `aSD` stage can support scalar comparison, multiplication, subtraction, addition, and absolute computation using column pitch-matched analog circuitry, while the `ADC` and `TH` stages consume negligible portion of total energy as they operate infrequently (once after $\geq$128 `aSD` operations). It has been shown that the CM [1], [2] offers significantly lower energy consumption and delay than digital ML accelerators, at the expense of limited reconfigurability. Furthermore, the absence of an instruction set limits its use to short sequence of operations with a single computation kernel, single memory bank, and fixed parameters such as a vector length.

3 Instruction Set Architecture for Mixed-Signal Accelerators

In this section, we present PROMISE architecture, discuss challenges in developing ISA, and then propose ISA.

3.1 PROMISE Architecture

Single Bank Architecture: PROMISE is built on CM (Fig. 2(a)), where the standard SRAM read and write functionalities are preserved (at the bottom) for additional flexibility. Along with **(S1)** `aREAD`, **(S2)** `aSD` and `aVD`**,** **(S3)** `ADC`, and **(S4)** `TH` described in Section 2, PROMISE comprises `X-REG` and `CTRL` to transform CM into a programmable mixed-signal accelerator. The detailed specification is as follows.

A PROMISE bank consists of 256 (= N_{COL}) columns. An 8-bit (= B) word is distributed across four consecutive rows and two neighboring columns which store 4-bit MSB and 4-bit LSB to enhances linearity through a sub-ranged read technique [1]. That is, `aREAD` reads out a 128-element vector of digital values and seamlessly converts it to that of analog values. Furthermore, `aREAD` can simultaneously perform element-wise addition or subtraction with X, a 128-element vector representing the input operand for inference in (1). `aSD` and `aVD` are architected to perform operations on 128 analog values. `ADC` consists of eight 8-bit ADCs which operate in parallel. Note that the `aVD` output of each bank is digitized by these ADCs to prevent the noise from analog operations accumulating over the iterations. This digitization also enables a multiple-bank architecture, where reliable data transfers between banks are required. One of the ADCs operates just once per `aVD`operation for an 128-element vector computation thereby amortizing the ADC energy across multiple arithmetic operations. `TH` implements non-linear operations such as sigmoid not only to compute the decision functions $f()$ in (1) but also to aggregate intermediate computed values when the vector length N is larger than 128.

Lastly, `X-REG` is a digital block similar to a vector register file, holding eight 128-element vectors representing eight X values. `CTRL` is a controller to generate enable signals

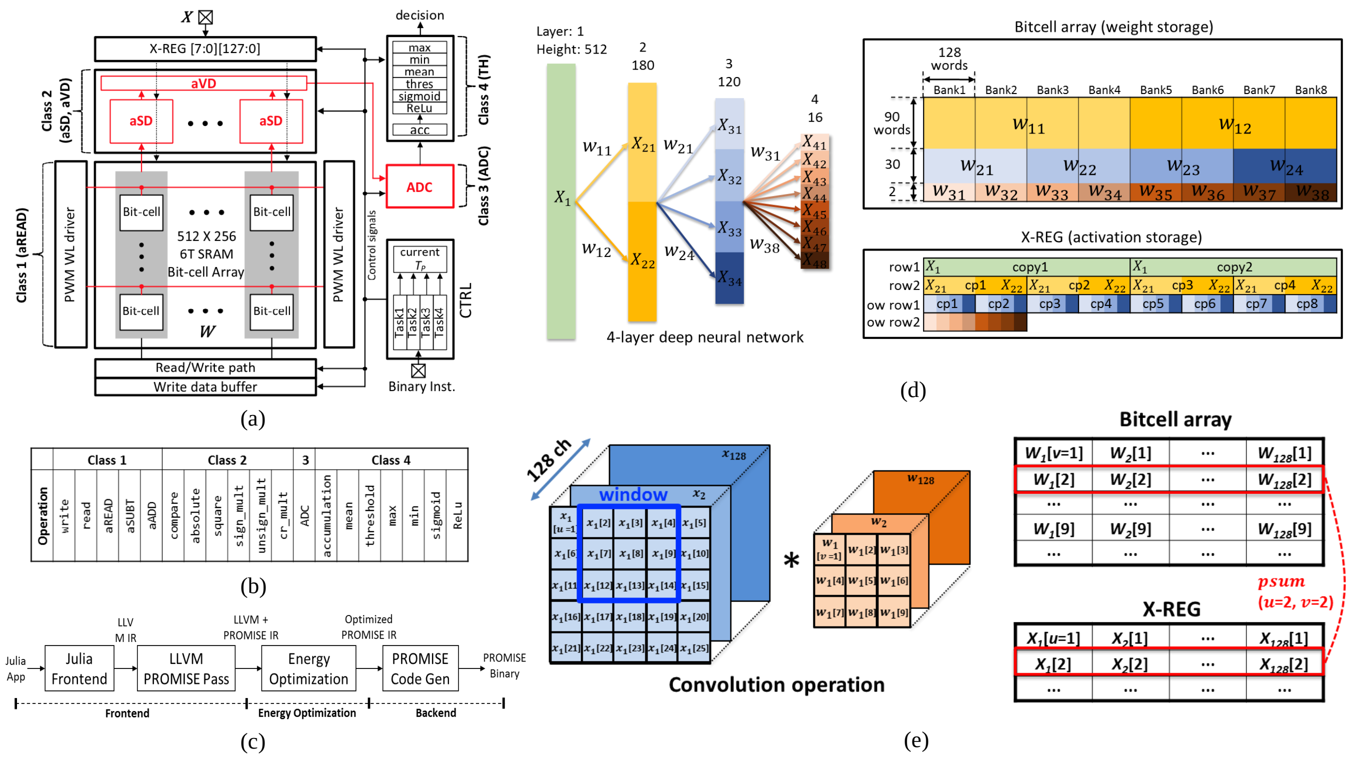

Fig. 2: PROMISE instruction set architecture [3]: (a) a single-bank architecture, (b) instruction set with available operations of each Class, (c) compiler pipeline, and algorithm mapping of (d) deep neural network (multi-layer perceptron), (e) convolutional neural network (CNN).

for the aforementioned components based on a given *instruction* and make CM function as a programmable mixed-signal accelerator. PROMISE can be extended to a multi-bank structure, which has multiple (up to eight in this work) PAGEs, each of which includes four banks. Thus, long (>128) vectors can be distributed across multiple banks for parallel processing.

3.2 PROMISE Instruction Set

The combination of algorithmic diversity and mixed-signal operations in PROMISE creates many challenges for ISA design choices. To support a broad range of ML algorithms, each stage needs to support diverse programmable operations. The analog processing also imposes intrinsic sequentiality in a chain of processing stages. Furthermore, two consecutive stages need to be physically closely placed to avoid substantial degradation in analog voltage from one stage to the next. We explore an ISA considering these constraints.

Instruction Format: We propose a wide-word macro instruction format, which is referred to as Task. Akin to a Very-Large Instruction Word (VLIW), a single Task consists of multiple operations, *except that the operations are sequential and not parallel as in VLIW architectures*. As depicted in Fig. 2(b), the four Class fields specify four operations for four pipelined stages of PROMISE, while the three other fields, OP_PARAM, RPT_NUM and MULTI_BANK configure all or specific Class operations. More specific descriptions of these seven fields are as follows.

Operating Parameter Field: OP_PARAM configures operating parameters of Class operations in a given Task, facilitating *flexible programmability*. This field includes the source and destination addresses for Class-1/2/4, and SWING parameter, which controls BL swing ΔV_{BL}, e.g., 111 allows 30 mV/LSB whereas 001 allows 5 mV/LSB. This parameter is a key knob to control the trade-off between energy and accuracy under software control.

Class Fields: Class-1 defines five possible memory operations. READ, WRITE, or aREAD makes CM perform a digital read, digital write, or analog read operation to a compute-memory address specified by OP_PARAM. aADD or aSUB fuses an analog read and an element-wise analog addition or subtraction into a single operation where two vector operands come from compute-memory and X-REG, respectively, whose addresses are specified by OP_PARAM.

Class-2 specifies a composition of one of six possible aSD operations with aVD operation for aggregation. Specifically, aSD operating on a computed value from Class-1 supports three unary operations: compare, absolute, and square and three binary operations: sign_mult, unsign_mult, and cr_mult where the other operand comes from an X-REG address specified by OP_PARAM.

Class-3 and Class-4 control whether an ADC should be performed or not and specify one of seven possible TH operations, respectively, as listed in Fig. 2(b).

Loop and Multi-Bank Control Field: RPT_NUM specifies how many times the Task should be executed to process multiple W_js. The CM and X-REG addresses are incremented sequentially every iteration. MULTI_BANK comprises 2 bits and specifies the number of banks used to distribute long (>128) vectors for parallel processing.

3.3 Algorithm Mapping

A ML algorithm sometimes requires several `Tasks` for different distance metrics. We present three examples: 1) template matching, 2) DNN (multi-layer perceptron), 3) CNN, and 4) kernel data reuse technique in analog domain for energy saving in neural networks.

3.3.1 Template Matching

Template matching employs L1 distance kernel to find the closest N-pixel image out of many candidate images (W_j) to input query image (X) by processing across multiple banks in parallel. In this example, $N = 512$, 127 candidate images, and four banks are chosen. The template matching is mathematically defined as:

$$j_{opt} = \arg\min_j \sum_{i=1}^{512} |x[i] - w[j,i]| \quad (2)$$

The corresponding `Task` instruction consists of: `RPT_NUM` = 127 specifying the number of candidate images; `MULTI_BANK` = 4 to distribute 512 pixels into four banks (128 pixels per bank) for parallel processing; `Class-1: aSUBT` to perform element-wise subtraction of X with W_j; `Class-2: absolute` with aggregation; and `Class-3: ADC` followed by a digital-domain `Class-4: min` to compute $f() = \arg\min_j$.

3.3.2 Deep Neural Network (Multi-layer Perceptron)

Multi-layer perceptron (MLP) consists of multiplications between vector (activation X) and matrix (weight kernels W_js). The W_js do not change during inference whereas Xs change over streamed inputs. Therefore, weights and activations are stored in SRAM bitcell array and `X-REG`, respectively. For the vector multiplication between X and W_j to compute the j-th pixel of output activation, `Class-1:aREADs`, `Class-2:sign_mult`, `Class-3:ADC`, and `Class4:sigmoid` are chosen with `RPT_NUM` to be the number of pixels in the output layer. To maximize the parallelism, multi-bank can be exploited as an example shown in Fig. 2(d), where MLP with four layers (512-180-120-16) is processed across eight banks, e.g., the first (X_{21}) and second (X_{22}) halves of activations in the second layer are computed in parallel from bank1-4 (where $X_{21} = X_1 \cdot W_{11}$) and bank5-8 ($X_{22} = X_1 \cdot W_{12}$), respectively, by having two sets of X_1s (copy1 and 2). Similarly, activations in the following layers are also stored redundantly. To support this, `Class-4` should be able to have multiple destination addresses by `OP_PARAM`. The optimized bank-level parallelism achieves more than 7 times better throughput in this example as compared to naive mapping, where weights are stored sequentially in bank1-8.

3.3.3 Convolutional Neural Network (CNN)

A convolution operation to compute an output activation can be transformed as a matrix multiplication as shown in (3) and (4):

$$psum[u,v] = \sum_{i=1}^{128} w_{ij}[v]x_i[u] \quad (3)$$

where $w_{ij}[v]$ is the v-th element in the kernel from i-th ($i = 1-128$ in this example) input channel to j-th output channel as shown in Fig. 2(e), where pixel index of kernel $v = 1-9$ for 3×3 kernel size in this example. Similarly, $x_i[u]$ is the u-th pixel of the input feature map in the i-th channel. Figure 2(e) omits output channel index j for simplicity. For the first `Task`, `Class-1: aREADs`, `Class-2: sign_mult`, `Class-3: ADC`, `Class4: no operation` are chosen to enable convolution. Here, 128 operands $w_{i=1:128,j}[v]$ are stored in the same row of bitcell array whereas $x_{i=1:128}[u]$ are stored in the row of `X-REG`. The $psum[u,v]$ with all combinations $[u,v]$ are computed. Then, the p-th pixel of the j-th output feature map $y_j[p]$ is computed as follows:

$$y_j[p] = \text{ReLU}\left(\sum_{u,v\in wid} psum[u,v]\right) \quad (4)$$

where $u,v \in wid$ defines the coordinates in input feature map (u) and weight kernel (v) within the current convolution window, e.g., $(u,v) \in wid : (2,1),(3,2),...,(14,9)$, which are nine pairs covered by blue box in Fig. 2(e). For 3×3 kernel size, the $\sum_{u,v\in wid}$ processes the accumulation over nine $psum$s from (4) requiring `RPT_NUM = 9` and `Class4: accumulation`, `Class1-3: no operation` for the second `Task`, where the addresses of $psum$s in `X-REG` are specified by `OP_PARAM`. For the third `Task`, `Class-4: ReLu` is chosen. For sub-sampling layers, `Class-4: mean` or `max` are used for average or max pooling, respectively.

3.3.4 Analog Kernel Reuse in Neural Networks

Kernel reuse is widely used in digital accelerators for neural network algorithms to minimize the cost from data movement. For example, the fetched weight kernel W_j from bitcell array can be stored in the register and reused over multiple streamed-in input Xs without re-accessing the bitcell array in Section 3.3.2, which saves the memory access energy at the cost of latency and higher register capacity requirement. The kernel reuse can be also applied in CNNs, as the fetched $w_{ij}[v]$ from bitcell array can be reused for the multiplications with many operands, i.e., $x_i[u = 1,2,...,25]$ to compute all of $psum[u,v]$s in (3).

However, the CM needs to repeat `Class-1:aREAD` stage every time as the charge-sharing mechanism in the subsequent `Class-2:sign_mult` stage destructs the fetched analog value of $w_{ij}[v]$. To enable the kernel reuse in analog domain, the charge-recycling multiplier in [4] is employed (with `Class-2:cr_mult`), where the fetched analog level is sampled in the column pitch-matched capacitor and reused over multiple streamed-in operands without being destructed. Measured results in [4] shows that the sampled value can be reused up to 200 times until the analog level drops by 10% due to leakage. On the other hand, the multiplier requires additional training to compensate the offset α as it generates the output $(w + \alpha) * x$ instead of $w * x$.

4 Compiler

This section describes how PROMISE compiler translates a given ML algorithm described in a high-level language into the PROMISE ISA.

TABLE 1: Benchmarks for PROMISE Simulations.

Algorithm	Application	Database	Data size (N)	Problem size	Instructions (`Class 1,2,3,4`)	W	X	Comments	Precisions (W, X) for CONV-OPT
Matched filtering	Event (gun-shot) Detection	Gun-shot mono sound	(8-bit) 256 512 1024	100 test vectors	`aREAD` `sign_mult` `ADC` `threshold`	Filter weights	Test samples		5-bits
Template matching (L1 / L2)	Face Recognition	MIT-CBCL	(8-bit) 16×16 22×23 32×33	256 Candidates	`aSUBT` `L1:abs/L2:sq` `ADC` `min`	Candidate faces	Test samples	Nearest candidate based on either L1 or L2 distance	6-bits
Linear SVM	Face Detection	MIT-CBCL	(8-bit) 16×16	2 categories, 2000 training samples, 858 test samples	`aREAD` `sign_mult` `ADC` `threshold`	Weights	Test samples	Face data converted into a vector, linear SVM applied on it	6-bits
k-NN (L1 / L2)	Hand-written character recognition	MNIST	(8-bit) 16×16 22×23 32×33	10 categories, 54210 training samples, 200 test samples	`aSUBT` `L1:abs/L2:sq` `ADC` `min`	Training samples	Test samples	Sorting is done in external processor after processing on MATI	6-bits
Feature extraction (PCA)	Face Detection	MIT-CBCL	(8-bit) 16×16	2000 samples	`aREAD` `sign_mult` `ADC`	Weights	Samples	Four features used for face detection based on PCA	6-bits
Linear regression	Modeling linear predictor	Synthetic data	(8-bit) 2 dim.	8192 samples	`aREAD` `sq/sign_mult` `ADC` `accumulation`	T1: U T2: V T3: U T4: U	T4: V	2-D linear regression : $slope = \frac{\mathbb{E}[(u-\bar{u})(v-\bar{v})]}{\mathbb{E}[(u-\bar{u})^2]}$ Reformulated as : $slope = \frac{\Sigma uv - \Sigma\bar{u}\bar{v}}{\Sigma u^2 - \Sigma\bar{u}^2}$ $y\text{-}intercept = \bar{v} - slope \cdot \bar{u}$	6-bits
DNN (Multilayer Perceptron)	Hand-written character recognition	MNIST	(8-bit) 22×23	10 categories, 60000 training samples, 10000 test samples	`aREAD` `sign_mult` `ADC` `acc/sigmoid`	Weights	Test samples	5-layer DNN w/ nodes: 784-512-256-128-10	6-bits
CNN	Hand-written character recognition	MNIST	(8-bit) 32×32	10 categories, 60000 training samples, 10000 test samples	`aREAD` `sign_mult` `ADC` `acc/sigm/mean`	Weights	Test samples	C1:6@28×28, S2 C3:16@14×14, S4 C5:120@14×14, F6:10	5-bits

4.1 Code Generation

Figure 2(c) shows the PROMISE compiler pipeline for Julia. There are three parts: (1) a front end to map Julia applications to the IR, (2) energy optimizations on the IR, and (3) a back end to translate the IR to the PROMISE ISA. The IR, energy optimizations and back end are all designed to be independent of the source-level language, to make it easily extendable to other languages or DSLs.

Frontend (Julia program to PROMISE compiler IR): We chose Julia as the source language because it enables us to easily identify patterns of computations that can be offloaded to PROMISE (e.g., matrix multiplication) and Julia also supports several ML libraries.

The Julia frontend translates applications to LLVM IR. The PROMISE pass runs over each LLVM function and uses pattern matching to identify computations that can be offloaded to PROMISE.

Backend (compiler IR to PROMISE ISA): The backend of the compiler translates the LLVM IR to the PROMISE ISA by mapping each computation to an appropriate `Task`. This involves two parts: (1) compile time code generation for `Class1-4`, and (2) computing the `OP_PARAM`, `RPT_NUM`, `MULTI_BANK` fields at runtime and passing them to the PROMISE run-time, which runs on the host and launches the `Task`.

4.2 Energy Optimization

Many ML applications often tolerate lower accuracy. In our work, we allow the source-level programmer to express the tolerable application-level accuracy degradation p_m. As deterministic errors can be tolerated easily by re-training the parameters in ML algorithms, we focus on spatial random errors across bitcells from process variations in this section.

The energy optimization in the compiler pipeline in Fig. 2(c) assigns the `SWING` field of each `Task` in the application that would ensure that end-to-end error tolerance is met. Mapping a high-level parameter like p_m directly to a suitable swing voltage is challenging for algorithms such as neural networks that have multiple `Task`s.

We solve this problem by breaking it down into two parts: (a) determining a minimum bit precisions (B) for the activations to meet the given target accuracy, which can be supported by methodologies such as [5]; and (b) mapping the required bit precision to the hardware swing voltage. To achieve B-bit precision in the final output, the magnitude of error introduced must be less than $1/2^{B+1}$. The error magnitude of analog chain in PROMISE is inversely proportional to the `SWING` parameter. Therefore, the minimum `SWING` value is chosen to maximize the energy efficiency while maintaining the error magnitude is less than $1/2^{B+1}$.

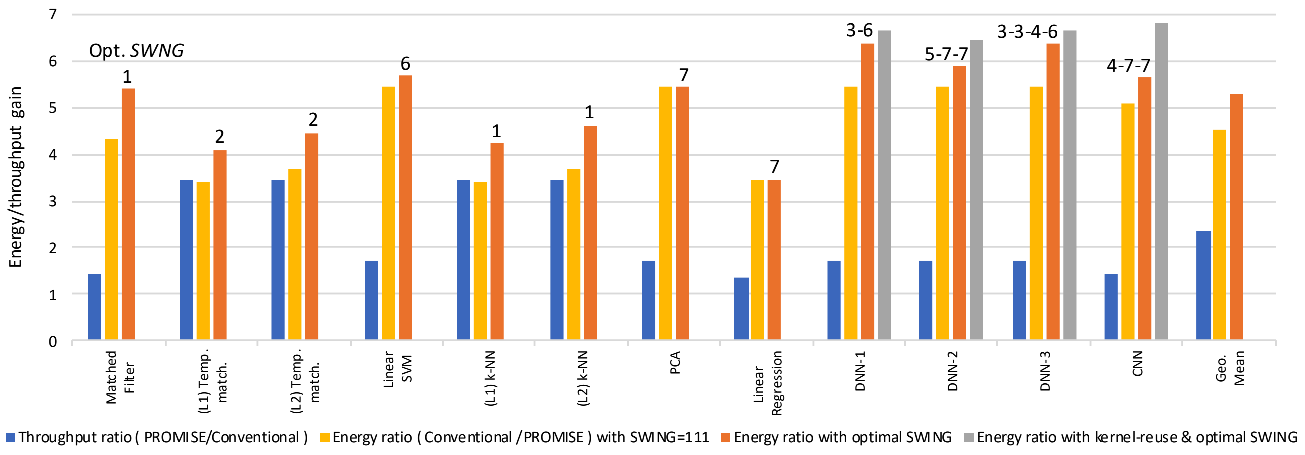

Fig. 3: PROMISE gains: speed-up and energy saving compared to conventional digital implementation, and additional energy savings by kernel data reuse and compiler-directed energy optimization.

5 Evaluation

In this section, we present the gains of PROMISE as compared to reference designs, and the additional gains by compiler-based energy optimization.

5.1 Validation Methodology

To evaluate PROMISE, we take modeling approaches described in [3] based on a TSMC 65 nm GP process and eight ML benchmarks listed in Table 1. The following two ASICs are considered as our comparison baselines: (a) **CONV-OPT**: We build the baseline digital architecture with the conventional SRAM and computing logic with the minimum precision as listed in Table 1. The computational logic is synthesized individually for each specific benchmark algorithm using the same process technology, (b) **CM**: The programmability overhead is estimated via the comparison to the measured results of CM [1]. The layout of conventional system was generated in a 65 nm process, which indicates that PROMISE has a 10% area overhead.

The compiler-based `SWING` optimization is analyzed by permitting an accuracy degradation $p_m < 1\%$. Energy gains with analog kernel reuse are also estimated for DNNs and CNN based on the silicon-measured energy and accuracy models of charge-recycling multiplier [4]. It is assumed that CONV-OPT also employs kernel reuse for fair comparison and that both PROMISE and CONV-OPT reuse the fetched data 50 times. The first layer of CNN requires short vector computations, e.g.,$\leq$ 25 elements per vector. AS PROMISE is aimed for large-scale vector/matrix computations with $128 \geq$ elements per vector, we assume that the first layer is processed externally, but heavy computations in the following layers are processed by PROMISE.

5.2 Performance and Energy

Figure 3 shows that PROMISE (`SWING=111`) provides a speed-up of $1.4 - 3.4\times$ and $3.4 - 5.5\times$ energy savings compared to CONV-OPT across the benchmarks leading to an energy-delay-product (EDP) improvements of $4.7 - 12.6\times$ compared to CONV-OPT. The key reasons for PROMISE's superior throughput and energy efficiency are its column-wise parallelism and low-voltage swing mixed signal computations in aREAD (`Class-1`) and aSD/aVD (`Class-2`) operations.

Figure 3 also shows further energy savings by the compiler-directed `SWING` optimization in the energy optimization pass. Feature Extraction and Linear Regression are omitted from this evaluation as they are not classification algorithms. Three benchmarks, DNN-1(784-128-10), DNN-2(784-256-128-10), and DNN-3(784-512-256-128-10) are variants of MLP with 3, 4, and 5 layers respectively. We analyze the energy benefit with optimal `SWING` values, e.g., DNN-1(3,6), DNN-2(5,7,7), DNN-3(3,3,4,6), and CNN(4,7,7). Overall the benefits of the optimization range from 4%-20% with geometric mean of 14%.

Note that CONV-OPT also employs kernel reuse for fair comparison. Thus, the energy gains with kernel reuse are slightly higher than those without reuse. This is because the energy consumption of charge-recycling multiplier is lower than the previously used analog multiplier in [3]. In spite of the increased complexity of CTRL to support the programmability, the energy of CTRL takes minor (<10%) portion maintaining the programability overhead of PROMISE to be negligible as compared to CM.

PROMISE also achieves $1.3\times$ ($1.1\times$) EDP reduction with L1 (L2) distance as compared to k-NN accelerator [6], and $22\times$ EDP reduction as compared to DNN accelerator [7].

6 Conclusion

This paper presents PROMISE, the first end-to-end design of a programmable mixed-signal accelerator for diverse ML algorithms. PROMISE accomplishes a high level of programmability without losing the benefits of mixed-signal accelerators for a class of ML algorithms over digital ASICs.

References

[1] M. Kang et al., "A multi-functional in-memory inference processor using a standard 6T SRAM array," *IEEE Journal of Solid-State Circuits*, 2018.

[2] A. Biswas et al., “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications,” in *IEEE International Solid-State Circuits Conference*, 2018.
[3] P. Srivastava et al., “PROMISE: an end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms,” in *International Symposium on Computer Architecture*, 2018.
[4] M. Kang et al., “An In-Memory VLSI Architecture for Convolutional Neural Networks,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018.
[5] C. Sakr et al., “Analytical guarantees on numerical precision of deep neural networks,” in *International Conference on Machine Learning*, 2017.
[6] H. Kaul et al., “A 21.5 M-query-vectors/s 3.37 nJ/vector reconfigurable k-nearest-neighbor accelerator with adaptive precision in 14nm tri-gate CMOS,” in *IEEE International Solid-State Circuits Conference*, 2016.
[7] P. N. Whatmough et al., “A 28nm SoC with a 1.2 GHz 568 nJ/prediction sparse deep-neural-network engine with >0.1 timing error rate tolerance for IoT applications,” in *IEEE International Solid-State Circuits Conference*, 2017.

ABOUT THE AUTHORS

Mingu Kang (mingu.kang@ibm.com) is a research staff member in the IBM Thomas J. Watson Research Center, Yorktown Heights, where he develops machine learning accelerator architecture.

Prakalp Srivastava (psrivas2@illinois.edu) earned his doctorate from University of Illinois at Urbana–Champaign in 2019. Currently, he is a software engineer at Google Brain and is part of the TensorFlow team.

Nam Sung Kim (nskim@illinois.edu) is a professor at the University of Illinois at Urbana–Champaign. His research focuses on devices, circuits, and architectures for energy-efficient computing. He is an IEEE Fellow.

Vikram Adve (vadve@illinois.edu) is the Donald B. Gillies Professor Computer Science at the University of Illinois at Urbana–Champaign. His research is broadly in compiler techniques, and their use for parallel computing and for performance. He is a Fellow of the ACM.

Naresh Shanbhag (shanbhag@illinois.edu) is a professor at the University of Illinois at Urbana–Champaign. His research focuses on realizing integrated circuits and systems for machine learning, signal processing, and communications. He is an IEEE Fellow.