A Programmable Heterogeneous Microprocessor Based on Bit-Scalable In-Memory Computing

Hongyang Jia, *Student Member, IEEE*, Hossein Valavi, *Student Member, IEEE*, Yinqi Tang, *Student Member, IEEE*, Jintao Zhang, *Student Member, IEEE*, and Naveen Verma, *Member, IEEE*

***Abstract*— In-memory computing (IMC) addresses the cost of accessing data from memory in a manner that introduces a tradeoff between energy/throughput and computation signal-to-noise ratio (SNR). However, low SNR posed a primary restriction to integrating IMC in larger, heterogeneous architectures required for practical workloads due to the challenges with creating robust abstractions necessary for the hardware and software stack. This work exploits recent progress in high-SNR IMC to achieve a programmable heterogeneous microprocessor architecture implemented in 65-nm CMOS and corresponding interfaces to the software that enables mapping of application workloads. The architecture consists of a 590-Kb IMC accelerator, configurable digital near-memory-computing (NMC) accelerator, RISC-V CPU, and other peripherals. To enable programmability, microarchitectural design of the IMC accelerator provides the integration in the standard processor memory space, area- and energy-efficient analog-to-digital conversion for interfacing to NMC, bit-scalable computation (1–8 b), and input-vector sparsity-proportional energy consumption. The IMC accelerator demonstrates excellent matching between computed outputs and idealized software-modeled outputs, at 1b TOPS/W of 192|400 and 1b-TOPS/mm^2 of 0.60|0.24 for MAC hardware, at V_{DD} of 1.2|0.85 V, both of which scale directly with the bit precision of the input vector and matrix elements. Software libraries developed for application mapping are used to demonstrate CIFAR-10 image classification with a ten-layer CNN, achieving accuracy, throughput, and energy of 89.3%|92.4%, 176|23 images/s, and 5.31|105.2 μJ/image, for 1|4 b quantization levels.**

***Index Terms*— Charge-domain compute, deep learning, hardware accelerators, in-memory computing (IMC), neural networks (NNs).**

I. Introduction

MACHINE-LEARNING inference, particularly based on neural networks (NNs), has provided unprecedented capabilities in various cognitive tasks, such as vision and language processing. [1]–[5]. However, pervasive deployment, especially in edge applications, has been limited by the high computational requirements that are dominated by high-dimensionality matrix-vector multiplications (MVMs). To address this, many optimizations, focusing on both hardware specialization and model design (e.g., sparsity, compression, and pruning), have been researched [6]–[9]. Hardware acceleration has proven to be critical. However, traditional digital acceleration addresses computation and is, thus, restricted in the gains that it can provide for addressing the large amount of data movement involved in high-dimensionality MVMs [10]. For instance, embedded memory is often employed to exploit the many opportunities for data reuse in MVM. However, moving data from the point of storage to the point of computation outside the memory imposes substantial communication costs (both energy and delay), which scales with the size of memory and, thus, the amount of data stored, causing data movement to dominate in NN computations [11].

In-memory computing (IMC) overcomes this by exploiting both the structural alignment between 2-D matrix multiplication and a 2-D array of memory bit cells, as well as the dataflow alignment of computations in MVMs and the signaling provided by perpendicular word-lines and bit-lines. This enables accessing a computational result over many stored bits in a memory column, rather than accessing the raw bits one at a time, as done in standard memory. Doing so exploits the potentially high level of parallelism provided by high-density bit cells and amortizes the communication costs by a factor equal to the number of column bits involved in the computation, which is referred to as the row parallelism. However, the number of bits involved also sets the dynamic range of the computational result, such that accessing via memory bit lines and readout circuitry imposes a direct signal-to-noise ratio (SNR) tradeoff [12], [13]. Furthermore, the need to fit computation within area-constrained bit-cell circuitry motivates analog operation, whereby richer transistor functionality than that restricted to simple switch-based digital models can be leveraged. In this case, computational noise is often dominated by analog nonidealities (nonlinearity and variations).

This SNR tradeoff has limited IMC both in terms of scale, due to ensuing loss in accuracy or inefficiency of SNR recovery techniques, and in terms of integration in practical computing systems (i.e., larger architectures and associated software), due to inability to form robust functional abstractions of the computation. To manage this, previous IMC designs have reduced the amount of row parallelism [14], [15]. However, this directly impacts the energy and throughput benefits, especially when the overheads associated with IMC are considered (e.g., non-standard bit cells, multi-cycle computations, and necessary periphery). Previous IMC designs have also exploited statistical parameter training in

Manuscript received December 22, 2019; revised March 10, 2020; accepted March 30, 2020. Date of publication April 29, 2020; date of current version August 26, 2020. This article was approved by Associate Editor Vivek De. *(Corresponding author: Hongyang Jia.)*

Hongyang Jia, Hossein Valavi, Yinqi Tang, and Naveen Verma are with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: hjia@princeton.edu).

Jintao Zhang is with the IBM T. J. Watson Center, Ossining, NY 10562 USA.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2020.2987714

machine learning, to enable reduced SNR by incorporating models of the computational noise, in both chip-specific [16]–[18] and chip-generalized [19] training algorithms. This has shown promise, warranting the further research needed to transition such models toward generalized abstractions, suitable for application design and mapping across the range of hardware design parameters and operating conditions.

Recently, the approach of charge-domain analog computation has been introduced for NNs [20], [21], including specifically for IMC [20]. This enables high SNR by leveraging capacitors within the bit cells for analog computation. The capacitors are formed using metal-fringing structures above the bit cells, thus introducing no area overhead [20]. Importantly, high stability of capacitance value can be achieved (due to temperature stability and well-controlled geometry via lithographic precision). As a result, the SNR tradeoff instated by IMC can be leveraged for significant energy and throughput benefits. With regard to the limitations previously faced, charge-domain computation has enabled the greatest scale and technology-normalized energy efficiency achieved by IMC to date [13]. The purpose of this article is to present details of how charge-domain IMC can be integrated into a practical heterogeneous computing architecture with associated software, extending presentations in [12].

The primary contributions of this article are as follows.

1) We present and demonstrate a heterogeneous microprocessor in 65-nm CMOS, integrating a mixed-signal IMC accelerator for energy-efficient MVMs, a configurable digital near-memory-computing (NMC) accelerator for efficient localized element-wise computation on vectors, and an RISC-V CPU for general-purpose computation, along with required peripherals.
2) We extend the previous approach of charge-domain IMC [20], which is restricted to single-bit input-vector and matrix elements, to arbitrary input-vector- and matrix-element precisions. We analyze the quantization error that arises from this, fundamentally stemming from the SNR tradeoff instated by IMC. Due to high-SNR analog computation, the quantization error can be precisely modeled and integrated in the IMC abstraction required for architectural and software design.
3) We design and analyze, in detail, the microarchitecture for the IMC accelerator. With a primary objective of demonstrating programmability, interfaces are designed, which enables the integration of the accelerator in the standard processor memory space, yielding tight CPU coupling.

In addition, software libraries are developed integrating with standard NN design frameworks (Keras, TensorFlow) to enable application mapping to the architecture and training optimized to the quantized computation performed by the bit-scalable IMC accelerator. Several NNs are mapped to the chip and demonstrated, yielding computation equivalent to ideally modeled computation, thus illustrating the robust integration of IMC in the computing system.

The remainder of this article is organized as follows. Section II provides an overview of the architectural design of the heterogeneous microprocessor and extension of IMC from single- to multi-bit element computations. Section III describes and analyzes the microarchitectural design of the mixed-signal IMC accelerator, including the interface required between IMC and microprocessor architecture. Section IV presents the prototype measurements, software toolkits, and NN demonstrations. Finally, Section V concludes this article.

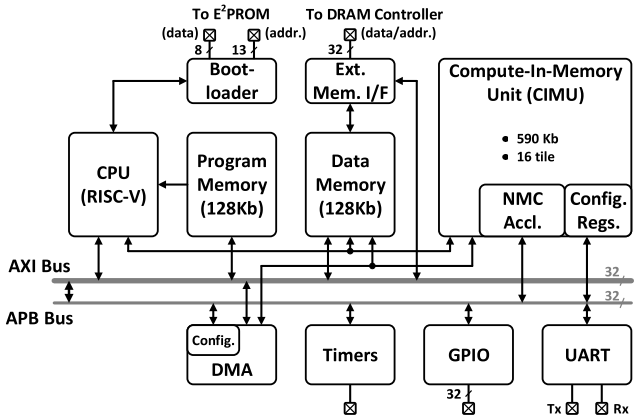


Fig. 1. Architectural block diagram of programmable heterogeneous IMC microprocessor.

II. Architecture Overview and Rationale

Analysis of machine-learning workloads for both training and inference has shown that high-dimensionality MVMs dominate, accounting for 70%–95% of the computation [22]. This is especially true for the CNNs of interest in target edge applications. IMC, which benefits MVMs, can, thus, have a significant impact, but the remaining computations must also be addressed. These include element-wise operations, such as activation functions, scaling, adding, and offset. [22], as well as other signal-processing operations required in audio, video, and so on, and pipelines [2], [23], [24]. Such computations are distinct from MVMs in that they benefit from significant data locality (i.e., a small number of operands is involved in the fundamental operations even though the operations might be parallelized). This implies that they can be well addressed by traditional digital acceleration, where computation energy, rather than data movement and/or memory accessing, is critical. On the other hand, the wide range of different computations involved makes configurability and programmability essential. Thus, it is preferable to delegate such computations to digital accelerators or CPU. However, especially for computations performed on the parallel elements of MVM output vectors, integrating such acceleration near IMC is beneficial. This mitigates the otherwise potentially high cost of data movement from the IMC accelerator, particularly given the physically large IMC area expected for enabling high row parallelism.

This motivates the heterogeneous microprocessor architecture shown in Fig. 1. The architecture includes: 1) a 590-Kb compute-in-memory unit (CIMU), which integrates IMC with configurable digital interfaces to the external microprocessor architecture and a localized configurable NMC accelerator

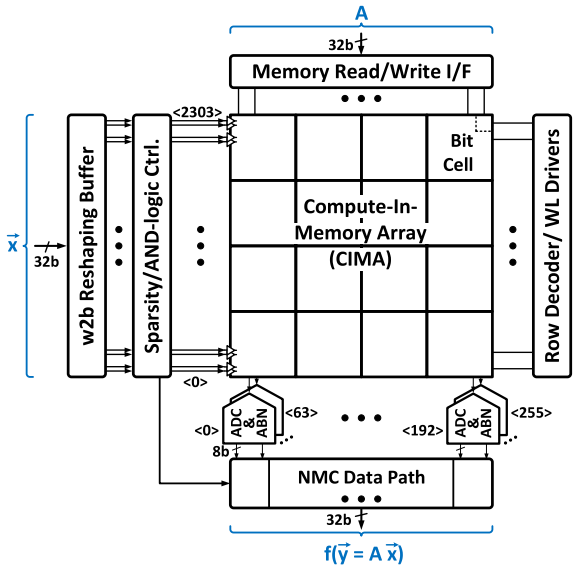


Fig. 2. Block diagram of CIMU.

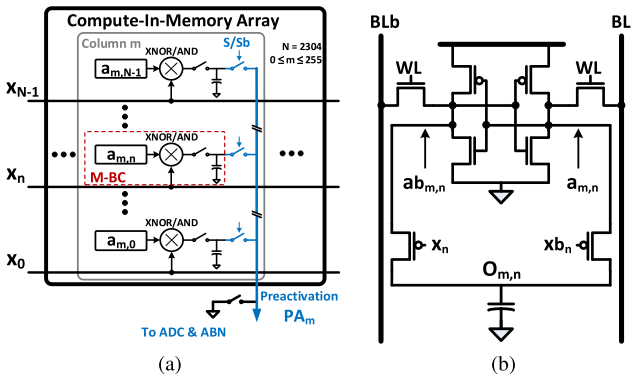


Fig. 3. Structure of CIMA. (a) Block diagram. (b) Details of the 8T M-BC.

for element-wise operations; 2) RISC-V CPU, based on the low-power design in [25]; 3) 128 KB of the standard program and data memory (P/DMEM), implemented using foundry-provided SRAM compilers; 4) two-channel DMA engine, each capable of 32-b transfers in roughly one clock cycle; and 5) peripherals for external interfacing, including external-memory control, bootloading, host-PC communication (UART), general-purpose IO (GPIO), and scheduling (timers).

The central block is the CIMU, whose microarchitecture is shown in Fig. 2. In addition to the interfacing and NMC blocks, the CIMU is comprised of a compute-in-memory array (CIMA), made up of 16 (4×4) tiles, which can be activity-gated for configuration into an IMC array of size up to 2304 rows $\times$ 256 columns. Following each column are two different analog-to-digital conversion options. The analog batch normalization (ABN) block is as in [20], providing batch normalization and activation binarization. For binarized activations, batch normalization simply reduces to the comparison of the preactivation (i.e., analog column-computation output) against a configurable analog reference voltage. The analog-to-digital converter (ADC) is for multi-bit quantized activations and employs a successive-approximation register (SAR) architecture providing an 8-b digital output to the NMC block for subsequent configurable element-wise digital computation. The choice of column dimension (2304) and ADC precision (8-b) is derived from considerations of the fundamental energy/throughput versus SNR tradeoff in IMC, where larger column dimension enables greater amortization of the communication/readout cost but increases the output dynamic range, thus increasing the readout complexity. This SNR-tradeoff consideration is quantitatively described in Section II-B.

A. Compute-In-Memory Array

The CIMA is based on the charge-domain computation shown in Fig. 3(a) and presented in [20], which implements an inner product operation (multiplication and accumulation) between input vectors and stored matrix, both having binary elements. This is achieved via the multiplying bit cell (M-BC) circuit shown in Fig. 3(b). M-BC operation starts by shorting together (via *S*/*Sb*) and discharging all of the local capacitors while holding x_n/xb_n high. Then, individual M-BC capacitors are decoupled, and multiplication is performed between binary input element data provided on x_n/xb_n and binary stored element data retained on $a_{m,n}/ab_{m,n}$, and the result is stored on the local capacitor. Finally, all capacitors in a column are shorted together to implement accumulation. The process then restarts by again discharging the shorted capacitors. The analysis in [26] shows that well over 10K IMC rows can be employed before computation SNR is limited by capacitor mismatch, enhancing the gains that can be achieved from IMC through row parallelism.

For 1-b elements, taken to have values of $+1$ and -1, M-BC multiplication reduces to a logical XNOR operation and is, thus, purely digital with binary output states ensuring perfect linearity. This work extends to multi-bit elements, where two different number-representation formats and associated computations are supported: 1) XNOR-compute, where bit values are taken to be $+1$ and -1, requiring logical XNOR in the M-BC and 2) AND-compute, where bit values are taken to be 0 and 1 (as in two's complement format), requiring logical AND in the M-BC. Logical AND is readily achieved in the M-BC simply by masking x_n to remain high. Again, both computations are purely digital and inherently linear.

In terms of density, the capacitor itself consumes no additional area, as it is formed using metal-fringing structures above the bit cell. Additional transistors are required, and while a variety of capacitor-based M-BC configurations were considered (e.g., connecting together top plates and driving bottom plates), the configuration shown was selected as it leads to a small number of additional transistors (8T). This incurs just 80% area overhead compared with a standard 6T bit cell, both using logic-design-rules, for a total cell area of 1.8 μm^2 (i.e., greater area overhead compared with a 6T bit cell using foundry push rules). In addition to the 8T structure, two additional transistors are also required per bit cell for the shorting switches *S*/*Sb*. These incur additional area but are laid out separately, combining together the switches for 3-b

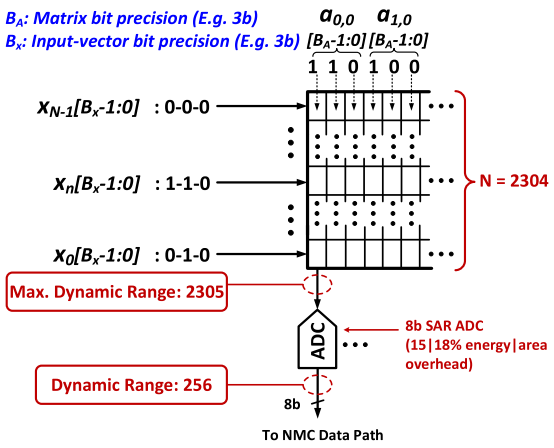


Fig. 4. Illustration of BPBS computation scheme for multi-bit IMC.

cells, to take advantage of opportunities for sharing layout features [20].

The logical CIMA columns are folded into three physical columns. The logical column length is chosen to be a multiple of 9 (3×3). As in [20], this is done to better match the trend of 3×3 kernels in CNNs [4]. Furthermore, it also enables layout feature sharing of the *S*/*Sb* switches and relaxes ADC layout constraints for pitch matching to columns.

B. Multi-Bit IMC

To extend the bit-wise IMC of [20] to multi-bit IMC, a bit-parallel/bit-serial (BPBS) scheme is used. Illustrated in Fig. 4 for the case of 3-b elements, BPBS stores matrix-element bits in parallel columns and provides input-vector element bits serially, in LSB-first order. This reduces each column computation to an inner product between binary vectors. For multi-bit computation, each column output is digitized, proper bit-weighting is applied via digital barrel shifting, and the digital summation is performed across the column outputs. For example, for the first input bit, the MSB column ADC output is shifted by two, the MSB-1 column ADC output is shifted by one, and the LSB column output is unshifted. For the next input bit, one additional shift is applied to all columns. Multi-bit extension for IMC has been considered previously [27] but must be analyzed in terms of the critical SNR tradeoffs inherent in IMC and the impacts of different number representation formats conducive for IMC. This is done next.

The two multi-bit number representations require slightly different computations. With XNOR-compute, a B-bit number is represented using $B+1$ bits as

$$x = \sum_{i=1}^{B-1} b_i \times 2^{i-1} + (b_{0+} + b_{0-}) \times 2^{-1} \tag{1}$$

where b_i's are taken to have values of $+1$ and -1 (represented as a logic-0 state in the digital circuits). In this case, a value of zero cannot be represented by one bit alone. Thus, to represent the LSB, two bits b_{0+} and b_{0-} are required, each with 2^{-1} weighting. Unlike conventional two's complement representation, the sign is determined by all the bits combined (e.g., sign extension requires setting the MSB to the desired sign, and all other extended bits as well as the original MSB to the opposite sign). The required conversions for XNOR-compute are made using simple logic circuitry preceding the CIMA. Then, following the CIMA, the ADC output is made bipolar, from -128 to 127, by applying a constant offset, before bit-weighting and summation across BPBS column computations. With AND-compute, a B-bit number is represented using standard two's-complement number representation. In this case, the ADC output is multiplied by -1 for the MSB column and again by -1 for all of the column computations involving the MSB input-vector bit. Then, bit-weighting and summation across BPBS column computations are performed.

While the BPBS scheme enables arbitrary bit precision for input-vector and matrix elements, rounding error is effectively introduced by the finite-precision ADC. This sets the effective signal-to-quantization noise ratio (SQNR) achievable and corresponds to the energy/throughput versus SNR tradeoff for IMC mentioned earlier.

The SQNR is, thus, analyzed and defined as follows:

$$\text{SQNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \frac{\sum_{m=1}^{M} y_m^2}{\sum_{m=1}^{M} (y_m - \hat{y}_m)^2} \tag{2}$$

where y_m denotes the inner-product of two vectors, $\vec{a}_m$ and $\vec{x}$, with full-precision (floating-point) elements, and $\hat{y}_m$ denotes the inner-product result computed via the BPBS scheme (a total of M inner products of length N are used). The computed inner-product incurs noise due to element quantization of $\vec{a}_m$ and $\vec{x}$, as with standard integer computation, but also due to rounding of the bit-wise column computations by the finite-precision ADC.

Specifically, with each M-BC providing a binary XNOR or AND result on its local capacitor, charge accumulation over N M-BCs (column dimensionality) results in a dynamic range of $N+1$ possible voltage levels. If such dynamic range is supported by the column ADC, integer computation is perfectly emulated. However, with a large N, preferred for larger IMC gains through row parallelism, the ADC cost can be excessive. In this design, for instance, the column dimensionality is up to $N = 2304$, and the ADC dynamic range is 8-b (256 levels). These values are chosen by considering the relative ADC overhead and impact on SQNR. Specifically, the 8-b ADC results in 15% area overhead and 18% energy overhead in each column.

The simulated SQNR, assuming perfect analog accumulation and digitization, is shown in Fig. 5 for both XNOR- and AND-computes (with uniformly distributed input-vector and matrix elements). To isolate the various quantization effects, the analysis considers different levels of quantization for input-vector elements (B_x) and matrix elements (B_A), as well as different levels of column dimensionality N. With $N = 255$, the dynamic range of the column computation matches that of the 8-bit ADC quantization, and the SQNR of integer computation is perfectly emulated, increasing directly with B_x and B_A. On the other hand, higher N would require an ADC precision of $\log_2(N+1)$ ($\sim$12 b for the maximum

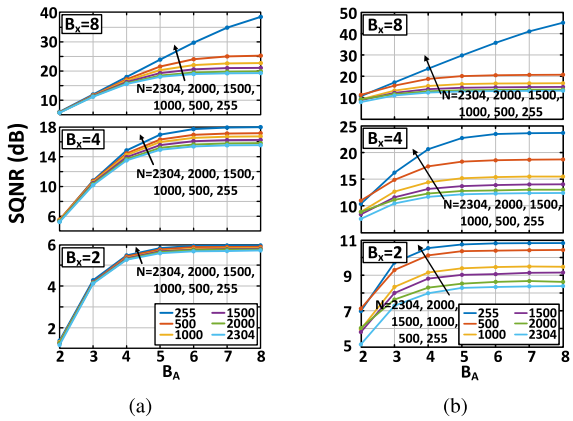


Fig. 5. Analysis of SQNR with respect to B_X, B_A, and N. (a) XNOR-compute and (b) AND-compute, where the $N = 255$ cases ideally emulate standard integer compute.

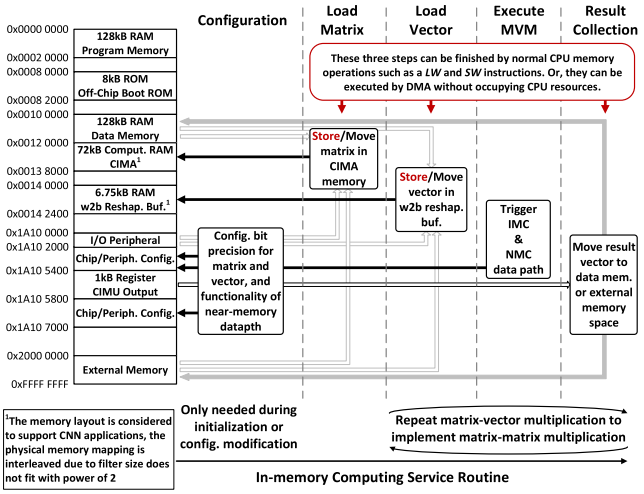


Fig. 6. Memory map and IMC MVM service routine.

$N = 2304$ in this design), ultimately leading to SQNR saturation as B_x and B_A are increased, due to ADC rounding effects. Nonetheless, for the B_x and B_A ranges of interest for low-power quantized NN applications (2-6 b) [28], [29], the selected design point leads to SQNR close to integer computation (i.e., the $N = 255$ curves). As seen, XNOR- and AND-computes lead to somewhat different SQNR curves due to both the different number representation formats and bit-wise computations involved, offering design alternatives for different application requirements.

However, more importantly than the resulting SQNR, we point out that the high capacitor-matching precision ensures that the column computation and subsequent ADC rounding effects can be accurately modeled even for large N. Consequently, the quantization effects can be robustly incorporated into parameter-training algorithms, as done for standard quantized NNs [28], [29]. This enables multi-bit IMC to be robustly employed for NN applications, as we demonstrate in this work through training libraries incorporated in Keras and TensorFlow (see Section IV-B). In addition, in this work, the column dimensionality can be configured to less than the maximum (2304) via column-gating switches at 64-row increments, as in [20]. This provides a knob for mitigating ADC rounding effects if needed in applications. Finally, activation sparsity also restricts the effective dynamic range of the column computation, thus enabling another way to mitigate ADC rounding effects. To exploit this, microarchitectural support is included in the CIMU, as described in the following.

III. CIMU Microarchitectural Design

This section presents microarchitectural design details of the CIMU blocks, along with analysis of the cycles required for computation and data movement to/from the CIMU to characterize its achievable utilization. As mentioned previously, the primary objective of the architecture is to demonstrate the integration of IMC in a programmable platform. Accordingly, the CIMU microarchitecture is designed for a tightly coupled accelerator architecture, where the CIMU is integrated into the standard processor memory space. While this imposes circuit overheads for proper interfacing, it enhances programmability by making the CIMU input/output buffers, configuration registers, and IMC array accessible from software as standard memory locations (as demonstrated in Section IV-B).

Fig. 6 shows the resulting memory map and programming model for executing MVMs. First, the CIMU is configured, for the desired input-vector and matrix bit precisions, input-vector and matrix dimensionalities, near-memory computing operations, and so on. This is typically done once or infrequently in an application. Second, matrix elements are loaded, from one of four sources: 1) directly from the CPU, as a result of general-purpose computations performed there; 2) from data memory; 3) from I/O; and 4) from the external-memory interface. The last three sources can exploit the DMA module for efficient data transfers. This is also typically done infrequently in an application, though it must be a focus of application-mapping optimizations (for instance, weight loading in NNs). Third, input vectors are loaded, also from the same four sources as matrix elements. Forth, MVM is performed. This is done by the CPU writing to a CIMU control register, and waiting for an interrupt, triggered upon completion (alternatively, the CPU can also poll a status bit in the CIMU control register). Finally, the computed output vector, stored in a CIMU peripheral buffer, is then moved, again to any of the original four sources.

Sections III-A–III-E describe the details of the key microarchitectural blocks of the CIMU and then provide performance and utilization analysis.

A. Word-to-Bit Reshaping Buffer

The purpose of the word-to-bit (w2b) Reshaping Buffer is to efficiently interface the external 32-b processor architecture with the configurable high-dimensional bit-wise operations performed within the CIMA. Fig. 7(a) shows the w2b Reshaping Buffer that takes input-vector data at near the maximum bandwidth of the external 32-b buses and sequences binary vectors of configurable dimensionality (up to $N = 2304$) to the CIMA. Up to eight binary vectors can be sequenced,

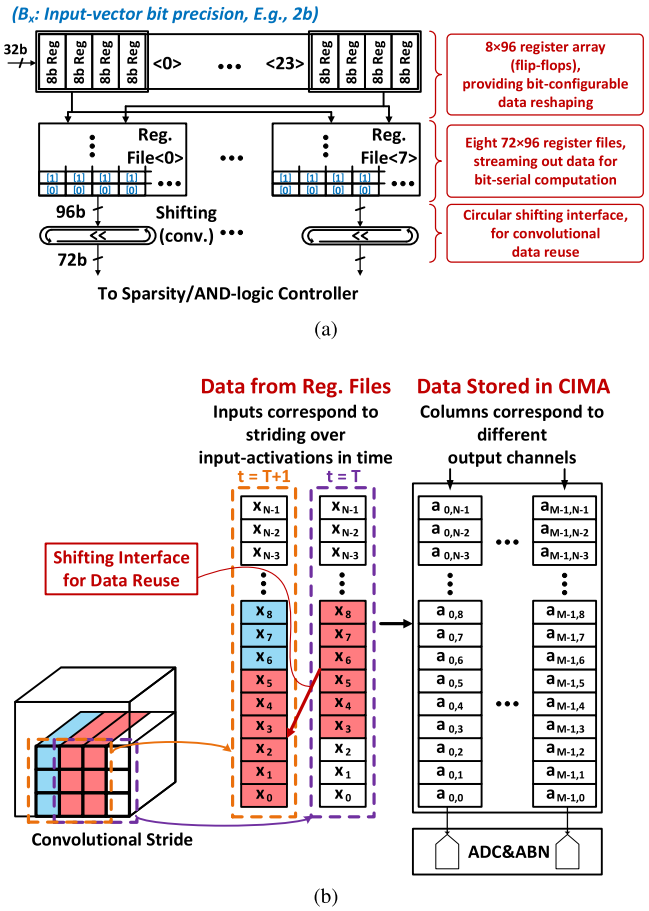


Fig. 7. Interface from processor architecture to CIMA, showing (a) block diagram of w2b Reshaping Buffer and (b) illustration of CIMA-stored matrix data and register-file-provided input-vector data, from shifting interface for input-activation reuse in convolutions.

corresponding to the configurable bit-serial computation. The w2b Reshaping Buffer also supports convolutional reuse of the data, optimized for 3×3 CNN kernels with a stride of 1 (i.e., the case with maximum reuse). This is motivated by the trend toward modular CNN design, where other kernel sizes, such as 5×5 and 11×11, can be composed of 3×3 kernels [4], [11]. For strides larger than 1 (i.e., cases with less reuse), convolutional reuse is not supported, in favor of reducing buffering complexity (the analysis in Section III-E shows minimal impact on throughput for pipelined input-activation transfers).

The w2b Reshaping Buffer operates by first loading 32-b input words into a set of four 8-b registers, filling up 24 such registers with successive 32-b data. Then, input data are moved to eight register files, segmented in this manner to ensure adequate bandwidth of configurable transfers to the register files and transfers from the register files. Namely, for 1-b input-vector elements, all data bits from all of the 24×4 8-b registers are moved in parallel (i.e., as 768 bits) to the eight register files. For 8-b input-vector elements, one data bit from each of the 24×4 8-b registers is moved in parallel (i.e., as 96 bits) to one of the register files, until all eight data bits are loaded. Then, loading proceeds to the next register file, until all eight register files are loaded. This places input-vector element bits in consecutive locations of the register file columns, for bit-serial sequencing. Similar loading is employed for the other bit configurations, with the number of cycles for the reshaping operation proportional to the input-vector bit-precision B_x. The reshaping operation is only required once for every 24 transfers of 32-b input data. Thus, the maximum utilization of the external 32-b buses is $24/(1+24) = 96\%$ and $24/(8+24) = 75\%$ for 1-b and 8-b input-vector elements, respectively (corresponding to the number of cycles required for reshaping in each case).

Each of the eight register files has 96 columns, with enough bits per column to store eight 8-b input-vector elements (i.e., 64 bits), plus a sparsity-mask bit for each input-vector element (i.e., 8 bits, used as described in the following). This allows four input-vector elements of bit precision up to 8-b to be stored and also double buffered so that loading can be pipelined with readout to the CIMA. The readout is performed of 72 bits at a time, from 3/4th of the columns. For the case of convolutional reuse, this enables processing of data from three of the columns, while a fourth column is simultaneously being loaded. A shifting interface at the output implements convolutional striding, selecting three columns from a set of four. While full convolutional reuse in 3×3 kernels requires only 1/9th of the data to be loaded, this would substantially increase the buffering requirements to at least three full rows or columns of input feature maps. Instead, loading 1/3rd of the data reduces buffering to only the input activations being processed and has minimal impact on throughput (as analyzed in Section III-E). To illustrate the input-activation reuse scheme, Fig. 7(b) shows the matrix elements (weight data) stored in the CIMA and the input-vector elements (input-activation data) provided from the register files through the shifting interface, during two CIMA operations. Finally, the full binary input vector, with dimensionality up to $N = 2304$, is provided to the CIMA in four cycles, by serially accessing bits from the four input-vector elements stored in each column. Each serial access is input to one row of CIMA tiles shown in Fig. 2, with fewer accesses required in the case of smaller input-vector dimensionalities (and corresponding tile gating).

B. Sparsity/AND-Logic Controller

The purpose of the Sparsity/AND-logic Controller is to provide input-vector sparsity-proportional energy savings and SQNR gains, as well as proper AND-logic functionality from the M-BCs. The energy savings arise from gating broadcast of the input-vector elements x_n/xb_n and, thereby, also preventing the charging of the local M-BC capacitors. Together, these two energy sources account for roughly 66% of the CIMA energy. The SQNR gains arise from a resulting reduction in the maximum dynamic range of the column computation, as discussed in Section II-B. The logic functionality for AND-compute is achieved by simply masking x_n to remain high, thereby preventing charging up of the M-BC capacitor via the corresponding PMOS transistor.

Fig. 8 shows the Sparsity/AND-logic Controller. We point out that with AND-compute, a zero-valued input-vector

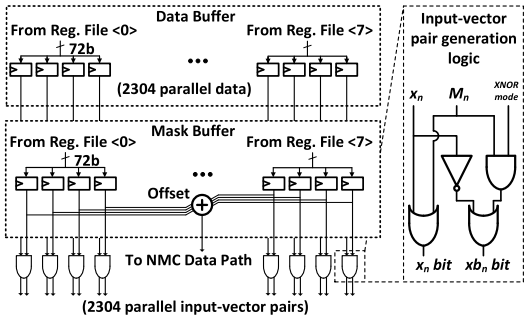


Fig. 8. Block diagram of Sparsity/AND-logic Controller.

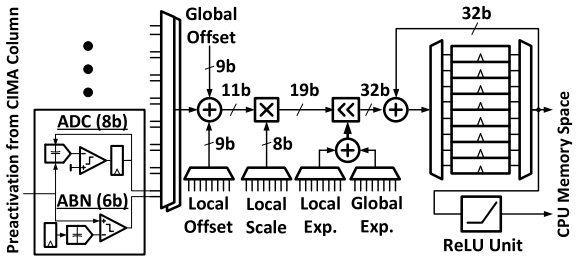


Fig. 9. Eight-way multiplexed NMC data path.

element is represented by successive logic 0's, which intrinsically avoids active switching, for both broadcast of x_n/xb_n and charging of local M-BC capacitors. However, with XNOR-compute, a zero-valued input-vector element is represented by $+1$ and -1 in successive bit positions, which leads to charging up of the M-BC capacitors during bit-wise XNOR multiplication. Thus, in the w2b Reshaping Buffer, a mask bit is derived for each input-vector element during its transfer to a register file, corresponding to a zero-valued element, during XNOR-compute, or any-valued element, during AND-compute. The mask bit is stored in a register file location corresponding to the input-vector element (i.e., requiring a total of 4 bits in each register-file column for each input vector). Then, both the input-vector-element bits (for bit-serial computation) and their corresponding mask bits are read out from the register files in four cycles (i.e., 72 bits from each of eight register files in parallel, for total input-vector dimensionality up to $N = 2304$) and stored in data and mask buffers, respectively. The mask buffer bits are fed to an adder to derive a count of the zero-valued elements. For XNOR-compute, such a count is required to properly offset the output of each column computation, which otherwise treats uncharged local capacitors as a value of -1, rather than 0. Thus, the count is provided to the NMC block to apply this offset in the digital domain, following the ADC. The data buffer bits are then simply fed to the CIMA, after proper gating from the mask bits, to drive the x_n/xb_n signals.

C. Analog-to-Digital Converter and Analog Batch Normalization

The ADC has an 8-b SAR architecture. It employs standard circuits and topologies using a 1.2-fF unit capacitor for its capacitive digital-to-analog converter (DAC) in the feedback path. The major consideration is fitting the ADC layout in the pitch of one CIMA column. As mentioned earlier, the CIMA columns are folded into three physical columns to relax the ADC layout pitch. Furthermore, the control logic required in the SAR architecture to sequence bit-cycling is shared across all the column ADCs, requiring only the feedback DAC, comparator, and output-code register for each column ADC.

The ABN offers an alternate mode implemented in a previous CIMA design [20] to provide energy-efficient batch normalization and binarization in binary-activation NNs. This functionality is readily supported within the NMC block and is only included for testing purposes, making it possible to remove the ABN for future area savings.

D. Near-Memory-Computing Block

The purpose of the NMC block is to provide both the digital operations required for BPBS computation and the parallel element-wise operations required on MVM output-vector elements. While the dominance of MVM computations in NNs leads to significant energy and throughput leverage of IMC, it is also necessary to accelerate element-wise operations and to do so close by to IMC. This ensures that element-wise operations continue to account for a small proportion of the energy and delay and maximizes the benefits of IMC by mitigating the cost of output data movement, especially when element-wise operations reduce the output data activity, as in the case of activation sparsity. However, the wide range of different element-wise operations (activation functions, multiplication, addition, shift, and so on) raises the need for configurability and programmability of the computations. Thus, digital accelerator architectures are preferred, motivating the NMC block.

Fig. 9 shows the data path in the NMC block. The NMC multiplexes across eight CIMA columns, both to relax the layout pitch requirements and to ensure that the relatively compact digital hardware is able to operate at a significantly higher frequency than the highly parallel CIMA hardware (as discussed further in the following). The NMC data path consists of several hardware computation stages, which can take operands from local registers, or global registers shared across all NMC blocks. An adder is provided to apply offset values, where, for example, a local register can be used to negate ADC offset from calibration or apply the offset required for batch normalization, or the global register can be used to apply the offset required for XNOR-compute sparsity control. A multiplier is provided to apply scaling values, for example, to negate ADC gain error or apply the scaling required for batch normalization. A barrel shifter is provided to apply multiplication by base-2 exponential terms, for example, for BPBS bit-weighting. Local scratchpad registers are provided to spatially and temporally store intermediate variables for BPBS and other element-wise operations. An activation function is provided, for application of nonlinear activation functions (currently, only ReLU is supported, but the extension to other nonlinear functions has been analyzed and can be provided

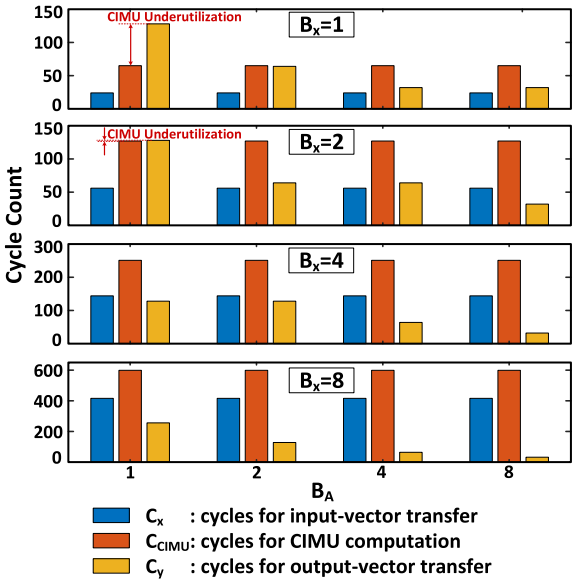


Fig. 10. Analysis of CIMU utilization across different bit precisions, evaluating data-transfer latency to/from CIMU against latency of CIMU (C_{CIMU} for the $B_x = 1$ case corresponds to the number of clock cycles required for the basic CIMU operation).

through a local lookup table). The control signals required for the NMC block are provided through a global control block, configured and sequenced through CIMU configuration registers. This yields an architecture similar to an SIMD processor.

E. Microarchitectural Analysis

In the programming model described in Section III, incoming input vectors are multiplied by a matrix (or portion of the matrix) loaded in the CIMA to perform MVM via IMC. However, for a tightly coupled accelerator architecture, an important concern is ensuring that input- and output-vector transfers in the 32-b processor do not limit the throughput of the highly parallelized CIMU computations. This section analyzes the respective latencies, where input-vector transfers, CIMU computations, and output-vector transfers are pipelined. For full application mapping, compiler design must also consider the latency of loading matrix elements in the CIMA, in order to optimize how computations are scheduled. To move toward such application-mapping optimizations, which are a focus of on-going research, matrix-element loading latencies are characterized here.

Fig. 10 shows the clock cycles required for input-vector transfers C_x, CIMU computations C_{CIMU}, and output-vector transfers C_y, for the implemented hardware operating in CNN mode with input-activation reuse (i.e., targeted design point for optimization). C_x assumes the convolutional reuse scheme of $2/3^{\text{rds}}$ within a 3×3 kernel, as the target for design optimization. The cycles required depend on the selected input-vector precision B_x and matrix-element precision B_A and are given for the maximum input-vector dimensionality, of $N = 2304$, and maximum output-vector dimensionality, of $M = 256/B_A$, which represents the worst case latency of transfers. Furthermore, the output-vector elements are assumed to have the full precision (B_y) supported by BPBS and NMC computation, allowing for diverse precision requirements of subsequent computations (e.g., by the CPU). C_x's dependence on B_x is due to the number of bits per element, which must be transferred. C_{CIMU}'s dependence on B_x is due to bit-serial compute, where the $B_x = 1$ case shows the number of cycles for single CIMA operation. C_y's dependence on B_x and B_A is due to the number of output-vector bits per element and the output-vector dimensionality, respectively, which must be transferred. All transfers are assumed to occur via the on-chip DMA module, avoiding CPU intervention to maximize bandwidth.

As seen, C_{CIMU} dominates for nearly all of the cases, except for $B_x = 1/B_A = 1$ case and slightly for $B_x = 2/B_A = 1$. This is because these cases represent the highest number of output-vector elements. Nonetheless, the analysis indicates the potential for high CIMU utilization, not significantly bottlenecked by input/output-vector transfers. However, we point out that the current CIMA design point has been optimized for energy efficiency and not throughput; considerable potential exists for increasing throughput, which is likely to ultimately make input/output-vector transfers an important concern, necessitating architectural optimizations for interfacing. Finally, we point out that other factors, such as the utilization of M-BCs within the CIMA, also play a role in setting the overall utilization and must be also optimized during application mapping.

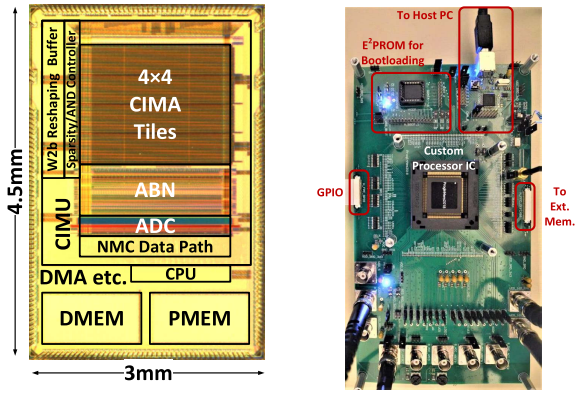
The number of clock cycles required for loading matrix elements C_{LOAD} is set by the number of bits stored in the CIMA and its physical dimensions. Although the CIMA logically has 2304 rows and 256 columns, each logical column is folded into three physical columns, yielding CIMA SRAM dimensionality of 768×768 bits. Thus, 24 DMA transfers of 32 bits are required for each row, and 768 row-wise writes are required in total. With each DMA transfer taking one clock cycle, while each row-wise write requiring 20 clock cycles, $C_{\text{LOAD}} = 33K$ cycles are required. The row-wise write cycles can be hidden with proper pipelining in the future design, giving a potentially lower $C_{\text{LOAD}} = 18K$ cycles.

IV. Prototype Measurements

The microprocessor is prototyped in a 65-nm CMOS process. Fig. 11(a) shows the die photograph, with major architectural blocks labeled. For testing and applications' development, a custom PCB is designed, as shown in Fig. 11(b), which provides off-chip bootloading via a E^2PROM, interfacing to a host PC, breakout header connections for general purpose IO (e.g., to interface with off-chip sensors), interfacing to off-chip DRAM controller, and BNC power connections for monitoring the power consumption of different on-chip voltage domains. Sections IV-A and IV-B describe the block-level testing of the chip and NN application demonstrations.

A. Block-Level Measurements

Table I provides a summary of the overall chip and block-level measurements. Measurements are taken at two voltage



(a) (b)

Fig. 11. Prototype system. (a) Die photograph of the microprocessor in 65-nm CMOS. (b) PCB for chip testing and application demonstration.

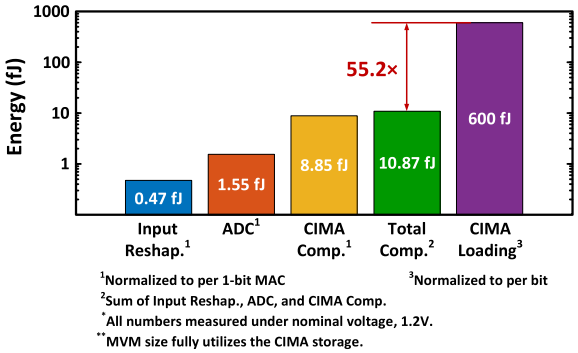


Fig. 12. Energy breakdown and comparison for MVMs within CIMU.

TABLE I

SUMMARY OF CHIP MEASUREMENT AND ENERGY BREAKDOWN

Technology (nm)	65	F_{CLK} (MHz)	100 | 40
V_{DD} (V)	1.2 | 0.7/0.85	Total area (mm^2)	13.5
Energy Breakdown @ VDD = 1.2V | 0.7V (P/DMEM, Reshap. Buf.), 0.85V (rest)			
CPU (pJ/instr.)	52 | 26	CIMA[1] (pJ/column)	20.4 | 9.7
P/DMEM (pJ/32b-access)	96 | 33	ADC[1] (pJ/column)	3.56 | 1.79
DMA (pJ/32b-transfer)	13.5 | 7.0	ABN[1] (pJ/column)	9.78 | 4.92
Reshap. Buf.[1] (pJ/32b-input)	35 | 12	NMC Data Path[1] (pJ/output)	14.7 | 8.3
CIMA Load[1] (pJ/32b-data)	19.2 | 9.6		

[1] **Breakdown within CIMU Accelerator**

conditions and corresponding clock frequencies: 1) the nominal voltage of 1.2 V, with a clock frequency of 100 MHz and 2) lower energy voltages of 0.7 V for the standard SRAMs and 0.85 V for the rest of the chip, with a clock frequency of 40 MHz. The energy for each block at each of these voltages is shown, noting that the CIMU computation energy (CIMA, ADC, and ABN) are shown for each CIMA column, with the exception of the NMC block, whose energy is shown per output, thus corresponding to 1-8 CIMA columns, depending on the matrix-element precision selected. The other energies are shown for operations on 32-b words. The weight loading energy in CIMA is also normalized to 32-b words for comparison with other data transfer costs.

To further analyze the CIMU, Fig. 12 shows a detailed energy breakdown of MVM-related operations, including input-vector reshaping, CIMA column computation, and ADC (all normalized for 1-b MACs) as well as CIMA loading (last bar). The CIMA computation and ADC energies correspond directly to values in Table I, divided by the column dimensionality (2304), while the input-vector reshaping energy corresponds to the value in Table I, divided by both the column dimensionality and the 32-b word length (normalizing for a single CIMA bit). Similarly, the CIMA-loading energy is normalized to a single bit (since 1-b MAC requires loading single CIMA bit). As shown, CIMA loading consumes only 55.2× more energy than MVM computation, indicating that even modest weight reuse in NNs ensures that the CIMU operation is not limited by weight loading (e.g., this is achieved for CNN output feature maps larger than 8×8, even with a batch size of 1).

Fig. 13 shows the detailed measurements of the basic CIMU operation, in the form of the column transfer function, for both ADC output and ABN output. The measurements correspond to both XNOR- and AND-compute due to the digital multiplication performed within the M-BC, leaving only the range of pre-ADC values to be interpreted differently mathematically (i.e., $[-2304, 2304]$ with step by 2 for XNOR-compute, while $[0, 2304]$ with a step of 1 for AND-compute).

Fig. 13(a) shows the basic ADC-output column transfer function across all 256 columns, where all 1's are loaded in the CIMA as 1-b matrix elements. Then, input vectors are applied with 1-b elements, where the number of 1's is swept, nominally yielding a ramp with 2305 uniformly spaced levels. The 8-b ADC then quantizes this to 256 levels. The result following offset calibration is shown, with error bars (visible in insets) corresponding to the standard deviation across the 256 columns. Fig. 13(b) shows ten ADC-output column transfer-function curves for one column, measured by sweeping the number of 1's in the input vector, but where 1's appear at different randomized element locations. This indicates the ideality of the computation against data patterns. Fig. 13(c) shows the ADC-output column transfer-function curve represented as an integral nonlinearity (INL). Note that the nonlinearity, observed to be less than 4 LSBs, corresponds to the entire column computation, not just the ADC. The dominant source of nonlinearity is charge-injection switching noise from the M-BC operation and ADC sampling, both occurring most prominently near mid-rail voltages. Fig. 13(d) shows the standard deviation observed in ADC output for each nominal value of the pre-ADC column computation, with the computation repeated 100 times. The average standard deviation is 0.37 LSB (shown as the yellow line), with periodic spikes smaller than 1 LSB observed at pre-ADC computation values that are close to the critical switching thresholds of the ADC.

Fig. 13(e) shows the ABN-output column transfer function, where matrix elements and input vectors are set as earlier; but for each case, the ABN code (DAC output) that causes an output transition is plotted. Again, high linearity and small variation across columns (standard deviation shown as error bars) are observed.

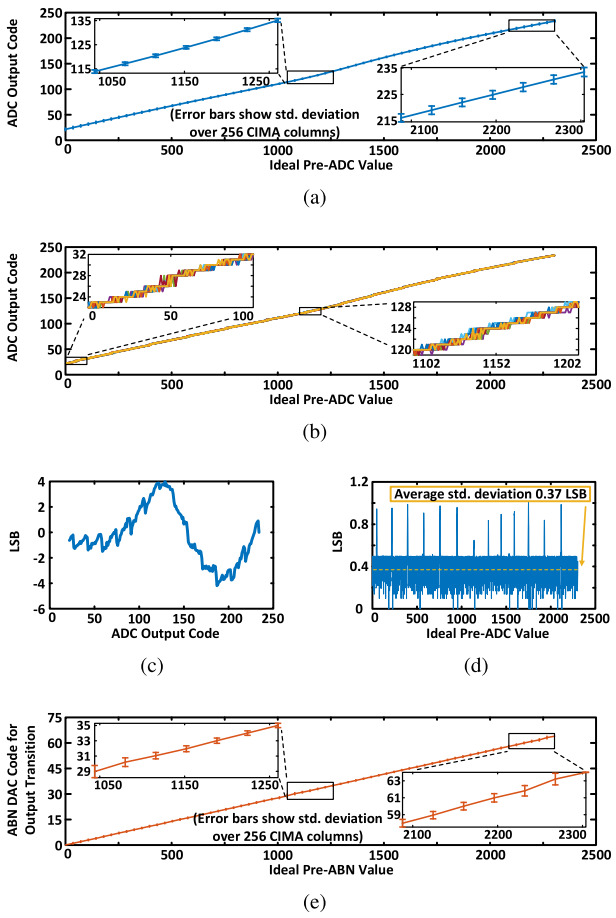


Fig. 13. CIMA-column measurements. (a) ADC output transfer function. (b) Drawing of ten different transfer function curves of one ADC output with random input ordering. (c) INL of one ADC. (d) Standard deviation of one ADC readout over 100 times repeated measurements. (e) ABN output transfer function.

Fig. 14 shows the detailed measurements of the multi-bit CIMU operation, using XNOR-compute as an example. Fig. 14(a) shows ideal SQNR measurements with uniformly distributed input-vector and matrix elements, for different bit precisions B_x and B_A, and two different dimensionalities $N = 1152$ and $N = 1728$. The measured results (square markers) show very good agreement with the expected SQNR (dashed lines), with some errors observed for B_A at 6 and 8 bits. This indicates that circuit non-idealities are small, and the overall computations closely match the modeled quantization effects. Fig. 14(b) shows the representative outputs actually obtained from a segment of the SQNR testing, with blue lines corresponding to measurements and orange lines corresponding to bit-true simulations. As seen, very good agreement is observed.

Table II provides a comparison with state-of-the-art NN accelerators that have recently been demonstrated, especially targeting energy efficiency. Both digital and IMC accelerator-based designs are shown. While IMC designs tend to achieve higher energy efficiencies, their overall throughputs tend to be limited due to the reduced scale that has been achieved for the analog computations involved. One exception is the

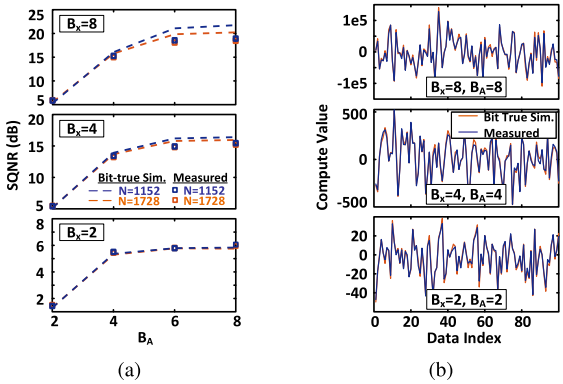


Fig. 14. Measurements of multi-bit MVMs. (a) SQNR. (b) Output waveform.

design in [20], which moves to the charge-domain computation based on capacitors. This work extends this approach further to provide bit-scalable IMC and integration in a programmable, heterogeneous architecture, with configurable NMC and fully programmable CPU. The degradations in energy efficiency and throughput compared with [20] mainly come from introducing the ADCs and reducing the column dimensionality by a factor of 2 for multi-bit SQNR tradeoffs, while overheads caused by the interfacing stay modest. The energy-efficiency and throughput numbers shown for this works are given for 1-bit compute, while the numbers for other bit-precisions scale linearly with B_x and B_A, respectively.

B. System Demonstrations and Software

To map NN applications and demonstrate the feasibility of integrating IMC in full systems, software libraries are developed. These follow the programming model described in Section III and mapping of input activations (input-vector elements) and weights (matrix elements), as shown in Fig. 7(b). As shown in Fig. 15, two types of libraries are developed: 1) training libraries for BPBS quantized NNs, integrated into TensorFlow and Keras and 2) inference-system libraries that allow the chip to be used as either a co-processor attached to a host processor or as a standalone platform. Specialized training libraries are developed because the BPBS computation introduces different rounding errors than standard integer computation. However, the high-SNR charge-domain analog computation allows BPBS rounding to be accurately modeled and incorporated in the loss function, as done for the standard quantized NN training [28], [29]. The training libraries implement a superclass for each type of NN layer in TensorFlow and Keras, with arguments for the bit precision and a flag to set the quantization type to either "integer" or "BPBS," enabling validation that the equivalent performance is achieved across a broad range of networks. The inference-system libraries provide an emulation mode, where BPBS CIMU computation is modeled in software, strictly for development purposes without requiring the chip. For inference-system deployment, libraries are developed for Python and MATLAB, whereby top-level NN control can be rapidly prototyped on a host system, with function calls made to the chip over the UART interface for energy-intensive (e.g., MVM) operations. Libraries are also

TABLE II
COMPARISON WITH STATE-OF-THE-ART NN ACCELERATORS

	Not In-memory Computing				In-memory Computing						
	[6] Chen, JSSC'17	[30] Moons, ISSCC'17	[31] Ando, JSSC'18	[21] Bank., JSSC'18	[32] Khwa, ISSCC'18	[16] Gon., ISSCC'18	[33] Jiang, VLSI'18	[34] Si, JSSC'19	[35] Guo, VLSI'19	[20] Valavi, JSSC'19	This work
Technology	65nm	28nm	65nm	28nm	65nm	65nm	65nm	55nm	65nm	65nm	65nm
Area (mm^2)	16	1.87	12	6	Not Avail.	0.81	0.11	0.037	6.2	12.6	8.56
V_{DD} (V)	0.8 – 1.2	1.0	0.55 – 1.0	0.8 | 0.6	1.0	1.2	1.0	1.0	0.9 – 1.1	0.7, 1.2	1.2 | 0.85
On-chip Memory	108 KB	128 KB	100 KB	328 KB	1 KB[1]	16 KB[1]	2 KB[1]	0.47 KB[1]	8 KB[1]	288 KB[1]	72 KB[1]
Bit Precision	16 b	4 – 16 b	1 – 1.5 b	1 b	1 b	8 b	1 b	1-4|2-5|3-7 b	1 | 1 | 3 b	1 b	1 – 8 b
Peak Throughput (GOPS)	120	400	1264[2]	400 | 60[2]	Not Avail.	8.2	60[2]	21.2 @4|5|7b	614.4[2]	18,876[2]	2185 | 874[2,3]
Peak Energy Eff. (TOPS/W)	0.083	10 @4b	6[2]	535 | 772[2]	55.8[2]	3.125	140[2]	18.37@4|5|7b	11.7[2]	886[2]	192 | 400[2,3]
Comp. Density (TOPS/mm^2)[4]	0.012	0.42	0.37[2]	0.25 | 0.038[2]	Not Avail.	0.013	0.53[2]	0.56@4|5|7b	0.22[2]	1.5[2]	0.60 | 0.24[2,3]
Configurability (dimensionalities, bits)	Dim.	Dim./bits	Bits	–	–	–	–	Bits	Dim.	Dim.	Dim./bits

[1]Amount is for in-memory computing only
[2]Given for 1-bit compute [3]Scales with number of bits in input-vector and matrix elements
[4]For MAC hardware (some areas measured from die photos)

Current-domain (limited scale, configurability, accuracy) | Charge-domain (limited configurability)

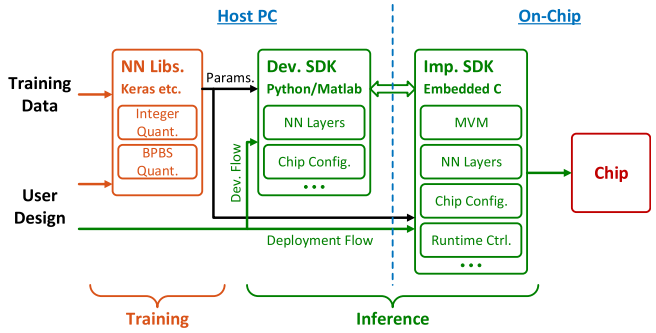


Fig. 15. Application mapping flow of the developed software and firmware stack.

TABLE III
NN DEMONSTRATION

	Network A (4/4-bit activations/weights)	Network B (1/1-bit activations/weights)
Accuracy of chip (vs ideal.)	92.4% (vs. 92.7%)	89.3% (vs. 89.8%)
Energy/10-way Class.[1]	105.2 µJ	5.31 µJ
Throughput[1]	23 images/sec.	176 images/sec.
Neural Network Topology	L1: 128 CONV3 – Batch norm. L2: 128 CONV3 – POLL – Batch norm. L3: 256 CONV3 – Batch norm. L4: 256 CONV3 – POLL – Batch norm. L5: 256 CONV3 – Batch norm. L6: 256 CONV3 – POLL – Batch norm. L7: 1024 FC – Batch norm. L8: 1024 FC – Batch norm. L9: 10 FC – Batch norm.	L1: 128 CONV3 – Batch norm. L2: 128 CONV3 – POLL – Batch norm. L3: 256 CONV3 – Batch norm. L4: 256 CONV3 – POLL – Batch norm. L5: 256 CONV3 – Batch norm. L6: 256 CONV3 – POLL – Batch norm. L7: 1024 FC – Batch norm. L8: 1024 FC – Batch norm. L9: 10 FC – Batch norm.

[1]At VDD = 0.7V (P/DMEM, Reshap. Buf.), 0.85V (rest)

developed for C, whereby the prototyped NNs can be deployed entirely on the chip.

Multiple NNs have been deployed and evaluated on the chip using the software mapping flow. As a representative benchmark, Table III shows two networks for CIFAR-10 image classification, employing 4-b weight and activation precision, as well as 1-b weight and activation precision. The demonstrated systems achieve energy efficiencies of 105.2 and 5.31 μJ/image and throughputs of 23 and 176 images/s, respectively. The chip measurements, including all sources of analog noise (variation, nonlinearity, and so on), show equivalent testing accuracies as those from ideal software models. The 4-b network achieves 92.4% accuracy, while the 1-b network achieves 89.3% accuracy.

V. CONCLUSION

High-dimensionality MVM operations dominate in many critical workloads, such as NNs. Given the high cost of data movement and memory accessing, traditional digital acceleration is limited in the gains (energy efficiency and throughput) that it can provide, ultimately being limited by the overheads in bringing data to the accelerator. This motivates IMC. While analog operation, resulting in degraded computation SNR, has limited the integration of IMC within computing systems, recently, the charge-domain IMC has been proposed for high SNR. This work leverages charge-domain IMC to demonstrate integration in a fully programmable, heterogeneous architecture and software stack. The architecture extends charge-domain IMC to enable MVM with multi-bit (1-8 b) input-vector and matrix elements, highly parallel configurable NMC digital acceleration, and interfaces for IMC integration in the standard processor memory space, resulting in a tightly coupled architecture for enhanced programmability. A prototype in 65-nm CMOS demonstrates computation with the excellent matching between measurements and the software abstractions while achieving energy efficiency of 192|400 1b-TOPS/W and throughput of 2185|874 1b-GOPS at V_{DD} of 1.2|0.85 V (both scaling with the number of input-vector- and matrix-element bits). The developed software libraries are used to map NNs for CIFAR-10 image classification with 4- and 1-b weight and activation precisions, achieving 105.2 and 5.31 μJ/image at 23 and 176 images/s, respectively, with accuracies of 92.4% and 89.3%, matching ideal software implementation.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 779–788.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: http://arxiv.org/abs/1409.1556

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[7] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 243–254.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*. [Online]. Available: http://arxiv.org/abs/1510.00149

[9] X. Dai, H. Yin, and N. K. Jha, "Grow and prune compact, fast, and accurate LSTMs," *IEEE Trans. Comput.*, vol. 69, no. 3, pp. 441–452, Mar. 2020, doi: 10.1109/tc.2019.2954495.

[10] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[11] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[12] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable embedded microprocessor for bit-scalable in-memory computing," in *Proc. IEEE Hot Chips 31 Symp. (HCS)*, Aug. 2019, pp. 1–29.

[13] N. Verma *et al.*, "In-memory computing: Advances and prospects," *IEEE Solid StateCircuits Mag.*, vol. 11, no. 3, pp. 43–55, Sum. 2019.

[14] J. Wang *et al.*, "A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 76–86, Jan. 2020.

[15] P. Srivastava *et al.*, "PROMISE: An end-to-end design of a programmable mixed-signal accelerator for machine-learning algorithms," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 43–56.

[16] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42 pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 490–492.

[17] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017.

[18] J. Zhang and N. Verma, "An in-memory-computing DNN achieving 700 TOPS/W and 6 TOPS/mm^2 in 130-nm CMOS," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 358–366, Jun. 2019.

[19] B. Zhang, L.-Y. Chen, and N. Verma, "Stochastic data-driven hardware resilience to efficiently train inference models for stochastic hardware implementations," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 1388–1392.

[20] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-Mb in-memory-computing CNN accelerator employing charge-domain compute," *IEEE J. Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, Jun. 2019.

[21] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-On 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28-nm CMOS," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, Jan. 2019.

[22] S. Shukla *et al.*, "A scalable multi-TeraOPS core for AI training and inference," *IEEE Solid-State Circuits Lett.*, vol. 1, no. 12, pp. 217–220, Dec. 2018.

[23] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in English and Mandarin," in *Proc. 33rd Int. Conf. Mach. Learn. (ICML)*, vol. 48, Jun. 2016, pp. 173–182.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805

[25] P. Davide Schiavone *et al.*, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *Proc. 27th Int. Symp. Power Timing Modeling, Optim. Simulation (PATMOS)*, Sep. 2017, pp. 1–8.

[26] H. Omran, H. Alahmadi, and K. N. Salama, "Matching properties of femtofarad and sub-femtofarad MOM capacitors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 6, pp. 763–772, Jun. 2016.

[27] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with *in-situ* analog arithmetic in crossbars," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 14–26.

[28] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, 2017.

[29] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*. [Online]. Available: http://arxiv.org/abs/1805.06085

[30] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10 TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28 nm FDSOI," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2017, pp. 246–247.

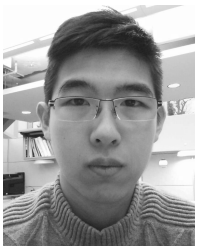
[31] K. Ando *et al.*, "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994, Apr. 2018.

[32] W.-S. Khwa *et al.*, "A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 496–498.

[33] Z. Jiang, S. Yin, M. Seok, and J. Seo, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," in *IEEE Symp. VLSI Technol. Dig. Tech. Papers*, Jun. 2018, pp. 173–174.

[34] X. Si *et al.*, "A twin-8T SRAM computation-in-memory unit-macro for multibit CNN-based AI edge processors," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, Jan. 2020.

[35] R. Guo *et al.*, "A 5.1pJ/Neuron 127.3 μs/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65 nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C120–C121.



Hongyang Jia (Student Member, IEEE) received the B.Eng. degree in microelectronics from Tsinghua University, Beijing, China, in 2014, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree.

His research focuses on ultra-low energy system design for inference applications. His primary research interests are CMOS IC design that leverages the approximate computing technique for model complexity reduction and mixed-signal computing for energy-efficient machine learning applications.

Mr. Jia received the Analog Devices Outstanding Student Designer Award in 2017.

Hossein Valavi (Student Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 2013, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2015, where he is currently pursuing the Ph.D. degree.

His research focuses on ultra-low-energy system design for signal processing and machine learning applications.

Mr. Valavi was a recipient of the Analog Devices Outstanding Student Designer Award in 2016.

Yinqi Tang (Student Member, IEEE) received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2014, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016, where he is currently pursuing the Ph.D. degree.

His current research interests include energy-efficient hardware systems for machine learning and deep learning applications, in both algorithm and hardware design aspects.

Jintao Zhang (Student Member, IEEE) received the B.S. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 2012, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2014 and 2019, respectively.

He is currently a Research Staff Member with the IBM T. J. Watson Center, Ossining, NY, USA, where he has been working on AI accelerator hardware design. His research interests include the realization of machine learning algorithms in mixed-signal/digital hardware, low-energy mixed-signal ASIC design for machine learning applications, and algorithms designed for such embedded systems.

Naveen Verma (Member, IEEE) received the B.A.Sc. degree in electrical and computer engineering from The University of British Columbia (UBC), Vancouver, BC, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2005 and 2009, respectively.

Since July 2009, he has been with Princeton University, Princeton, NJ, USA, where he is currently the Director of the Keller Center for Education in Innovation and Entrepreneurship and a Professor of electrical engineering. His research focuses on advanced sensing systems, exploring how systems for learning, inference, and action planning can be enhanced by algorithms that exploit new sensing and computing technologies. This includes research on large-area, flexible sensors, energy-efficient statistical-computing architectures and circuits, and machine-learning and statistical-signal-processing algorithms.

Prof. Verma was a recipient or a co-recipient of the 2006 DAC/ISSCC Student Design Contest Award, the 2008 ISSCC Jack Kilby Paper Award, the 2012 Alfred Rheinstein Junior Faculty Award, the 2013 NSF CAREER Award, the 2013 Intel Early Career Award, the 2013 Walter C. Johnson Prize for Teaching Excellence, the 2013 VLSI Symposium Best Student Paper Award, the 2014 AFOSR Young Investigator Award, the 2015 Princeton Engineering Council Excellence in Teaching Award, and the 2015 IEEE CPMT Best Paper Award. He has served as a Distinguished Lecturer for the IEEE Solid-State Circuits Society. He also serves on the technical program committees for ISSCC, VLSI Symposium, DATE, and the IEEE Signal Processing Society (DISPS).