Scalable and Programmable Neural Network Inference Accelerator Based on In-Memory Computing

Hongyang Jia, *Member, IEEE*, Murat Ozatay, *Graduate Student Member, IEEE*, Yinqi Tang, *Member, IEEE*, Hossein Valavi, *Student Member, IEEE*, Rakshit Pathak, *Student Member, IEEE*, Jinseok Lee, *Graduate Student Member, IEEE*, and Naveen Verma, *Senior Member, IEEE*

***Abstract*— This work demonstrates a programmable in-memory-computing (IMC) inference accelerator for scalable execution of neural network (NN) models, leveraging a high-signal-to-noise ratio (SNR) capacitor-based analog technology. IMC accelerates computations and reduces memory accessing for matrix-vector multiplies (MVMs), which dominate in NNs. The accelerator architecture focuses on scalable execution, addressing the overheads of state swapping and the challenges of maintaining high utilization across highly dense and parallel hardware. The architecture is based on a configurable on-chip network (OCN) and scalable array of cores, which integrate mixed-signal IMC with programmable near-memory single-instruction multiple-data (SIMD) digital computing, configurable buffering, and programmable control. The cores enable flexible NN execution mappings that exploit data- and pipeline-parallelism to address utilization and efficiency across models. A prototype is presented, incorporating a 4 × 4 array of cores demonstrated in 16 nm CMOS, achieving peak multiply-accumulate (MAC)-level throughput of 3 TOPS and peak MAC-level energy efficiency of 30 TOPS/W, both for 8-b operations. The measured results shows high accuracy of the analog computations, matching bit-true simulations. This enables the abstractions required for robust and scalable architectural and software integration. Developed software libraries and NN-mapping tools are used to demonstrate CIFAR-10 and ImageNet classification, with an 11-layer CNN and ResNet-50, respectively, achieving accuracy, throughput, and energy efficiency of 91.51% and 73.33%, 7815 and 581 image/s, 51.5 k and 3.0 k image/s/W, with 4-b weights and activations.**

***Index Terms*— Deep learning, hardware accelerators, in-memory computing (IMC), neural networks (NNs), scalable architecture.**

I. Introduction

DEEP learning based on neural networks (NNs) has enabled breakthroughs in a broad range of artificial-intelligence (AI) tasks, such as vision, language processing, and molecular discovery/diagnosis [1]–[9]. The increasing scale and computational complexity of NN models has pushed computing systems to their limits of energy efficiency and performance, especially in power/energy/space-constrained edge applications. This has motivated algorithmic and software solutions (e.g., model quantization/compression [10]–[14] and compiler optimizations [15], [16]), as well as hardware solutions in the form of accelerators [17]–[21].

In terms of accelerators, a key focus has been on matrix-vector multiplies (MVMs), which dominate NN inference computations, with data movement/accessing being a critical concern due to the high-dimensionality MVMs typically involved [22]. This has led to spatial architectures (e.g., systolic arrays), employing a 2-D structure of processing engines (PEs) providing storage and compute, to exploit the 2-D data reuse.

In-memory computing (IMC) is an emerging approach, aimed at taking the spatial architecture to an extreme, with high-density memory bit cells serving as the PEs. Recent implementations show that IMC can simultaneously achieve 10 × higher energy efficiency and compute density for MVMs compared to standard digital accelerators [23]–[29]. However, addressing full NN workloads requires incorporating MVMs into programmable architectures and to which the workloads can be scalably mapped, while preserving the efficiency and throughput benefits. Recent work [30] has begun to explore programmability, through a heterogeneous CPU-centric architecture. This provided a platform for building-up a software stack to deploy workloads, but without architectural optimizations to address efficiency and performance with scalable execution. This work focuses on such architectural design, together with the workload-mapping techniques, to maximize hardware utilization through flexible parallelism optimized for IMC. Architecture-software codesign is employed to address the drastically different physical attributes of IMC, compared to digital accelerators.

The primary contributions of this work are as follows.

1. We analyze the challenges for scalable NN mapping to IMC introduced by its physical attributes, namely high compute efficiency/density and large state-swapping overheads, and outline flexible mapping approaches for

Manuscript received April 30, 2021; revised July 27, 2021 and September 21, 2021; accepted September 27, 2021. Date of publication October 19, 2021; date of current version December 29, 2021. This article was approved by Associate Editor Jun Deguchi. This work was supported in part by Princeton University through the Generosity of William Addy '82. *(Corresponding author: Hongyang Jia.)*

The authors are with the Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ 08544 USA (e-mail: hjia@princeton.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2021.3119018.

Digital Object Identifier 10.1109/JSSC.2021.3119018

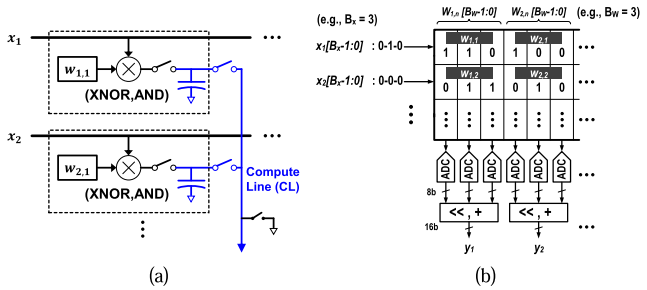


Fig. 1. Illustration of high-signal-to-noise ratio (SNR) IMC, showing (a) charge-domain IMC based on capacitors and (b) extension to MVMs with multi-bit input-vector and weight elements.

exploiting parallelism while mitigating associated overheads for IMC executions.

2. We propose a programmable array-of-cores IMC architecture designed to support flexible parallelism for optimally mapping different NNs and to enable architectural and execution scale-up with minimal overheads.
3. We demonstrate the programmable NN inference accelerator architecture in 16 nm CMOS, including a scalable on-chip network (OCN) and 4×4 array of IMC-based cores with integrated digital near-memory SIMD engines, reconfigurable buffering for flexible dataflow, and localized control.

In addition, accompanying software is developed for the prototype chip, including: 1) NN training libraries incorporated in standard deep-learning design frameworks (PyTorch, TensorFlow) to quantize NN models for execution on the chip and 2) a prototype NN mapping toolchain to deploy NNs onto the architecture. Multiple representative NN benchmarks executing on the chip are demonstrated, achieving accuracies equivalent to ideal digital computation.

The remainder of this article is organized as follows. Section II provides background on IMC technology and its challenges for efficient NN mapping. Section III provides an architectural overview of the NN inference accelerator, describing the key building blocks of the array-based architecture. Section IV describes the microarchitectural design within the IMC-based cores. Section V provides mapping illustrations for supporting different NN dataflows, and analyzes the mapping approaches and architectural supports. Section VI presents the prototype measurements, software toolkits, as well as NN demonstrations. Finally, Section VII concludes.

II. IMC Background and Challenges

A. High-SNR IMC

A critical consideration for architectural integration of IMC is forming a robust abstraction of its operation. This is challenged by the noise tradeoffs inherent in IMC architectures. In contrast to traditional digital architectures, which access stored data from memory one bit at a time by activating a single row, IMC derives its energy and throughput gains by accessing a compute result over multiple stored data by activating multiple rows at once. The row parallelism determines the energy and throughput gains, but also causes the dynamic range of the result to increase, thus reducing the SNR for a given readout architecture [31].

A dominant source of noise in IMC arises due to the need for analog operation to fit computation within the constrained bit cells. The SNR is thus limited by variations and nonlinearities typical in analog circuits. Recently, a high-SNR approach to IMC for binary input-vector and matrix (weight) elements was proposed in [27]. As illustrated in Fig. 1(a), this performs 1-b multiplication in the bit cells, corresponding to a logical XNOR or AND operation, and then makes use of capacitors to perform computation via analog charge accumulation for reduction of the column data. The capacitors are formed using the overlying metal wires. Thanks to lithographic precision in forming metal structures, as well as the intrinsic linearity and temperature stability of capacitors, this leads to high-precision computation, with analysis showing the potential to reach thousands of rows before capacitor nonidealities (random mismatch) limit the column-computation dynamic range. This enables the formation of robust abstractions, which are necessary for architectural integration.

TABLE I
PHYSICAL ATTRIBUTES OF IMC VERSUS DIGITAL HARDWARE

	IMC	Digital
Energy of 8-b MAC (pJ)	0.06	0.9
Energy of 8-b write (pJ)	1.8	1.8
Ratio of energy write/MAC	30	2
Area of 8-b MAC engine	0.4	9

*Numbers for 16nm CMOS.

B. Multi-Bit IMC

In [30], the capacitor-based binary MVM IMC approach was extended to multi-bit input-vector and matrix (weight) elements. As shown in Fig. 1(b), this was done via a bit-parallel/bit-serial (BPBS) scheme, where weight bits are stored in parallel columns, and input-vector bits are applied serially, thus preserving binary-operand computation in each column. The multi-bit-operand results were then formed after the column ADCs, by simply bit-shifting (binary weighting) and adding the column outputs. It is important to note that involving an ADC in this manner has quantization implications, which are also analyzed in detail [30]. Nonetheless, in terms of creating architectural abstractions, the robustness of capacitor-based analog computing leaves only such quantization effects to be modeled, which is standard practice in digital architectures.

C. Challenges for Scalable NN Execution

Despite their increased efficiency and throughput for MVM computation compared to digital architectures, the ability for IMC architectures to achieve efficient and scalable execution for full NNs is challenged by their physical attributes, namely unimproved write costs and high hardware density. Generally, scalable execution requires maximizing computer hardware

TABLE II
CHALLENGES OF STATICALLY MAPPING NNS TO IMC

		NN Benchmark (8-b)						
		darknet19	**resnet18**	**resnet34**	**resnet50**	**resnet101**	**vgg19**	**alexnet**
Case I : Minimizing memory	**Utilization (without replication)**	0.01	0.11	0.10	0.19	0.18	0.02	0.44
Case II: Maximizing utilization	**Total bit cells (with replication)**	21981M	277M	579M	610M	1216M	3171M	178M

utilization in space and time, by scheduling and mapping different subgraphs of a computation onto the available hardware resources. This instates state-swapping overheads, e.g., writing NN weight data to the hardware.

Table I compares the pertinent physical attributes of IMC (measured from the presented prototype, as described in Section VI) and digital architectures (reported in [32] and scaled for 16 nm technology). Two aspects are of particular note. First, the IMC energy ratio of write operations to multiply-accumulate (MAC) operations is 30× (as compared to just 2× for digital architectures). The reason for this is that compute-line (CL) driving and bit-line (BL) driving, which determine the MAC and write energies, respectively, both scale directly with the number of rows, but CL driving corresponds to many MAC operations (set by the IMC row parallelism) while BL driving corresponds to a single-write operation. Second, IMC has much higher hardware density (22×), enabling greater hardware parallelism. But this also incurs larger amount of state data that needs to be written to achieve high hardware utilization.

One approach that has been suggested to eliminate the state-swapping (write) overheads is to statically store all NN weights in IMC hardware. However, this is typically infeasible, because IMC intrinsically couples storage and compute resources, which need to be optimized separately. Specifically, NN weights are generally involved in very different number of operations. So, on the one hand, allocating a single IMC bit cell for each weight bit, to minimize storage footprint, leads to very low utilization for bit cells involved in fewer operations. Table II, in the first row, shows this overall IMC bit-cell utilization for full execution of different NNs, defined as the ratio of bit-cell operations per second performed to those available in total. On the other hand, replicating weight data in IMC bit cells according to the number of operations, to maximize utilization, would require an infeasibly large number of bit cells, preventing the scale-up of NN models. Table II, in the second row, shows this overall bit-cell requirement.

Instead of static allocation, writing weights from separate memory enables independent optimization of the storage and compute resources, but incurs write overhead. Typically architectures aim to amortize this over many compute operations by exploiting data reuse. The actual number of compute operations per weight determines the maximum possible data reuse, and is set by the NN model and input batch size. For this work, we focus on optimizing for batch size of 1, as is typically required in latency-sensitive edge applications. In this case, the average computations per weight for ResNet-50, as an example, is 170. As visualized in the roofline plot in Fig. 2, the high compute energy efficiency in IMC requires much greater reuse to amortize the overheads from writing data into IMC bit cells (including data accessing from embedded memory and communicating over weight-loading (WL) network). This is exacerbated by the high hardware density of IMC, which necessitates compute parallelism to ensure high utilization. However, the typical approach of data-level parallelism (also known as replication) directly reduces data reuse below the maximum. Accordingly, the physical attributes of IMC make it challenging to adequately amortize write overheads.

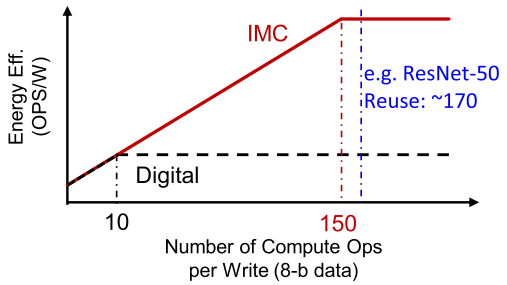


Fig. 2. Roofline plot showing IMC-write and IMC-compute bound regimes with respect to weight-data compute intensity.

This makes it necessary for an IMC architecture to support flexible mappings, specifically to address write overheads while ensuring high parallelism. In addition to data-level parallelism, another common form of parallelism is operation-level parallelism (similar to instruction-level parallelism in CPUs), where computations (e.g., NN layers) are pipelined. Both forms of parallelism incur important overheads of their own, e.g., state replication in data-level parallelism, and inter-stage buffering and latency in operation-level parallelism. Thus, flexible optimization motivates integrating proper architectural and microarchitectural supports for addressing these.

While the attributes (e.g., Table I) arise from general tradeoffs relative to digital hardware, the range of different IMC optimizations and designs previously reported [23]–[26], [28], [30], [33] will ultimately raise different architectural requirements. While this work employs an IMC macro similar to that in [30], the focus there was on enabling programmability to explore the deployment of a software stack for mapping applications, without explicit focus on the efficiency of full-scale NN executions. In contrast, this work focuses on flexible and efficient NN executions through different forms of parallelism.

III. CHIP OVERVIEW AND SCALABLE IMC ARCHITECTURE

This section presents an overview of the scalable NN inference accelerator and the building blocks of the array-of-cores

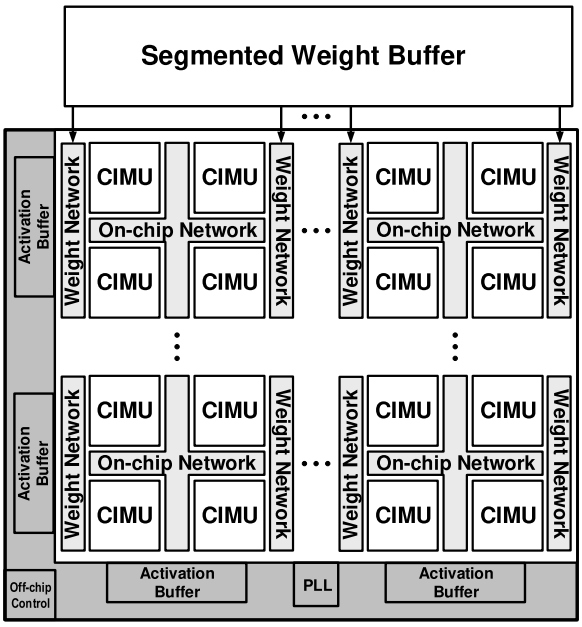


Fig. 3. Architectural block diagram of programmable and scalable IMC NN inference accelerator.

architecture. The scalable architecture is composed of an array of cores called compute-in-memory units (CIMUs), as shown in Fig. 3. The number of cores can be selected based on area and throughput requirements, with the presented prototype comprising a 4 × 4 array of CIMUs. The CIMUs employ a form of weight-stationary dataflow, by storing a subset of the NN weights for computation. The CIMUs are connected by a flexible OCN, which is used for transmitting input–output activation data between cores.

In addition, there is a dedicated WL network, for moving weights to the CIMUs from an SRAM-based weight buffer, which would ultimately be on chip but is not integrated in the prototype. The weight-buffer architecture assumes segments of 36 kB SRAM blocks, to provide weight data with high total bandwidth. The total size of the weight buffer is chosen to be 28 MB, to mitigate DRAM accesses for the NN models of interest, as is becoming typical for edge-inference processors, ranging from extreme-edge devices [34] to edge servers [35].

Finally, the test chip integrates top-level peripheral modules for testing purposes to interface with a host processor, including a 128 kB SRAM input–output buffer; configuration interfaces; and controllers. The capacity of the activation buffer is chosen to support the activations from most hidden layers for batch size of 1 processing of ImageNet data in a ResNet-50 model (note, few layers requiring larger memory exploit buffering within the cores). Though larger batch sizes can increase data reuse, batch-size of 1 is the focus due to latency-sensitive execution for edge platforms. As is typical, larger batches are then handled by providing images one at a time from the host processor.

A. Architectural Execution Model

The architecture implemented in the test chip supports weight-data parallelism and layer pipelining. Fig. 4 illustrates

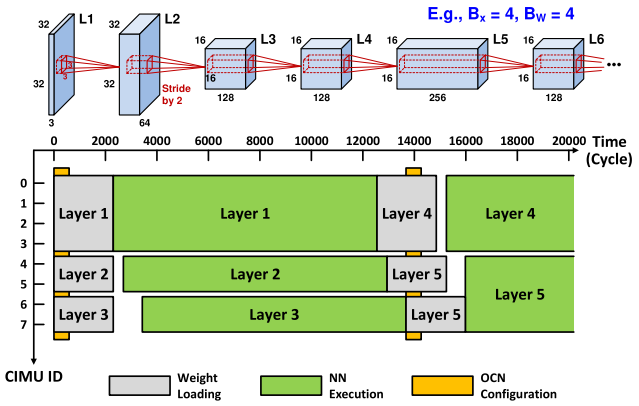


Fig. 4. Parallelized and pipelined execution of NN layers on the array-based architecture.

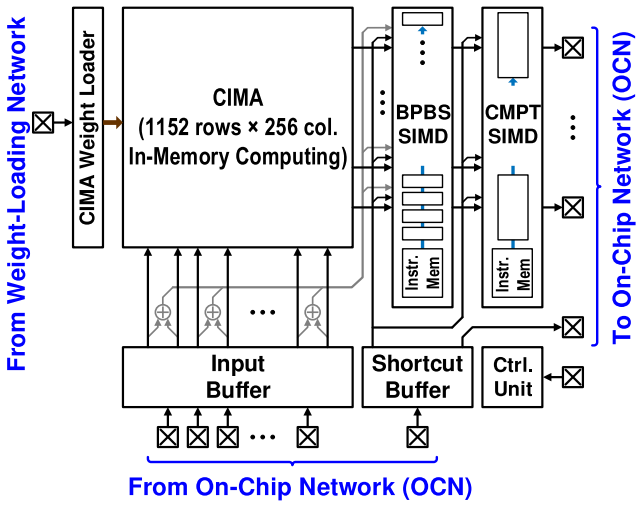


Fig. 5. Block diagram of the programmable CIMU core, supporting various NN dataflows.

scheduling for an example set of NN layers (assumes eight CIMUs for illustration). A subset of the layer weights are mapped to parallel CIMUs at a time (first layers 1–3, then layers 4–5), and pipelined execution proceeds. Leveraging different forms of parallelism enables optimization of energy efficiency and throughput, with consideration to the parallelism overheads.

B. CIMU Core

Fig. 5 shows a block diagram of the programmable CIMU core. It includes a mixed-signal compute-in-memory array (CIMA), input buffer, shortcut buffer, and near-memory-computing (NMC) SIMD datapaths. The central block is the CIMA, which includes a 1152 (row) × 256 (column), capacitor-based IMC bit-cell array. Using the BPBS approach described earlier, the CIMA can be configured to perform MVMs with multi-bit input-vector and matrix elements scalable from 1 to 8 bits. In addition to the energy and throughput versus SNR/quantization considerations discussed in Section II, the array size is chosen to balance NN-mapping granularity.

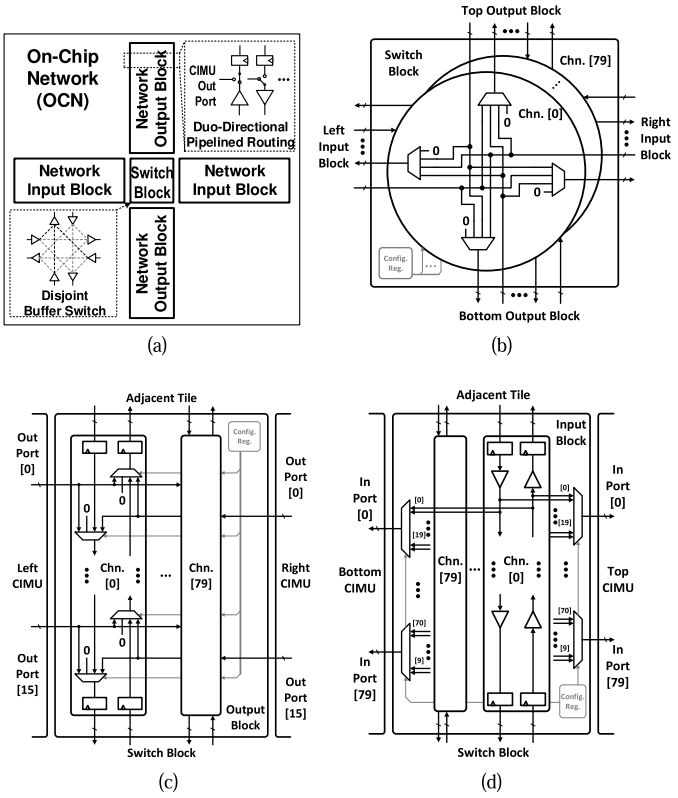


Fig. 6. Structure of OCN, including (a) block diagram of OCN in a 2×2 CIMU tile, (b) buffered switch block, (c) output block, for transmitting output activations from CIMU cores, and (d) input block, for feeding input activations to CIMU cores (rotated 90° clockwise).

At the inputs to the CIMA, a configurable input buffer receives and buffers input-activation data from the OCN. The input buffer provides a range of configurability for supporting different NN mappings and dataflows.

In parallel with the input buffer, there is an auxiliary buffer, called the shortcut buffer. While the input buffer feeds data to the CIMA, the shortcut buffer bypasses the CIMA, providing data directly to the following SIMD engines. This enables further configurability for NN compute graphs, for instance: supporting buffering needed for shortcut activation paths, such as residual connections; alignment between convergent activation paths; logical operations on activation paths, such as depth-wise shuffling.

Following the CIMA, the programmable near-memory SIMD engines receive the digitized column-computation results and support further digital operations. The first SIMD engine, the BPBS SIMD, is optimized for the shifting and addition required for BPBS reconstruction in multi-bit computations. The second SIMD engine, the compute (CMPT) SIMD, is optimized for element-wise operations, such as activation functions, activation scaling, and biasing, and cross-element operations, such as pooling and peep-hole operations. The final quantized computation results are then sent to the OCN.

C. On-Chip Network

The OCN enables flexible computation mappings in a manner similar to coarse-grained reconfigurable arrays (CGRAs) [36]–[38]. Fig. 6(a) shows a block diagram of the OCN in a tile grouping of 2×2 CIMU cores. Within a tile, the OCN is composed of three building blocks, all of which are fully synthesizable: a switch block, output blocks, and input blocks.

As shown in Fig. 6(b), the repeater buffered switch block employs a fully-disjoint architecture to connect ingoing wires flexibly to outgoing wires. The input and output blocks are implemented as duo-directional, pipelined routing segments, connected to the CIMU input buffers and connected from the CIMU CMPT SIMD outputs, respectively. Fig. 6(c) shows details of the output block. The 80 duo-directional channels are routed vertically, fitting between CIMUs on the left and right side. Each channel is pipelined by registers at the input/output from/to the adjacent tile and switch block. Similar to the output block, Fig. 6(d) shows details of an input block. With full connectivity at the CIMU outputs, each CIMU input port can be limited to connecting to only a subset (20 chosen for this design) of the duo-directional routing channels in the input block, providing adequate routing configurability.

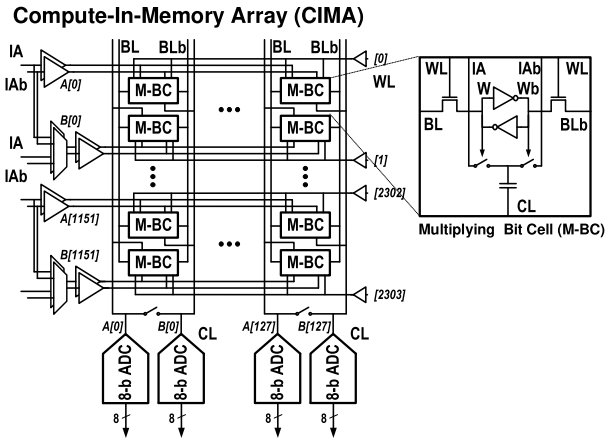


Fig. 7. Block diagram of CIMA.

IV. CIMU Microarchitectural Design

A. Compute-in-Memory Array

Fig. 7 shows details of the CIMA, which employs the capacitor-based multi-bit computation scheme mentioned in Section II. The precision of capacitor-based computation is exploited toward a high level of IMC parallelism. The CIMA has 1152 rows and 256 independent column CLs, which can also be configured to a 2304-by-128 array to extend input dimensionality. Each CL is followed by an 8-b SAR ADC. The ADC resolution is selected by balancing the energy/area overhead and the impact of quantization effects (as described in Section II-B). Several studies have explored the ADC resolution requirements [30], [39]. For instance, analysis performed in [30] shows that, with the column dimensionality adopted in this work, 8-b ADC resolution leads to SNR close to standard integer computation for the range of weight/activation precisions of interest in current quantized NNs. Accordingly, the 256 CIMA column ADCs occupy roughly 20% of the area and 29% of the energy of the CIMA.

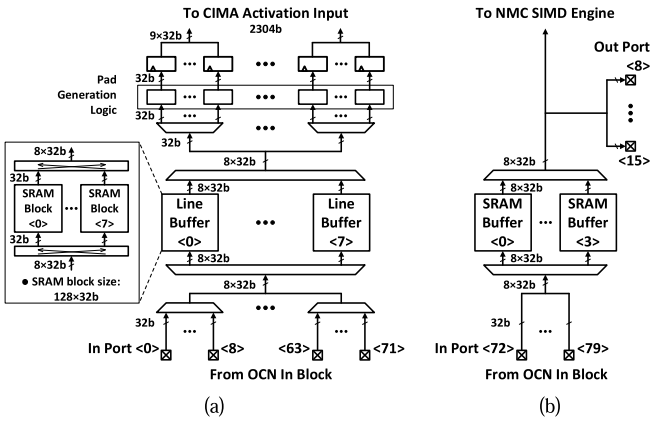


Fig. 8. Block diagrams of the CIMU buffers, including (a) input and (b) shortcut buffer.

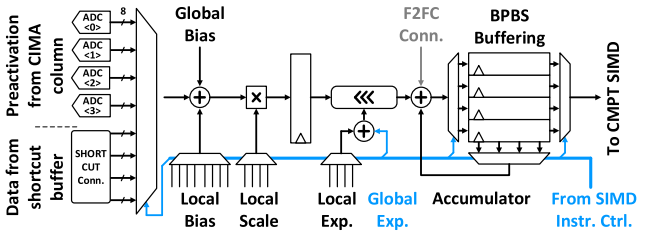


Fig. 9. Microarchitecture of BPBS SIMD datapath, receiving data from CIMA outputs and shortcut buffer.

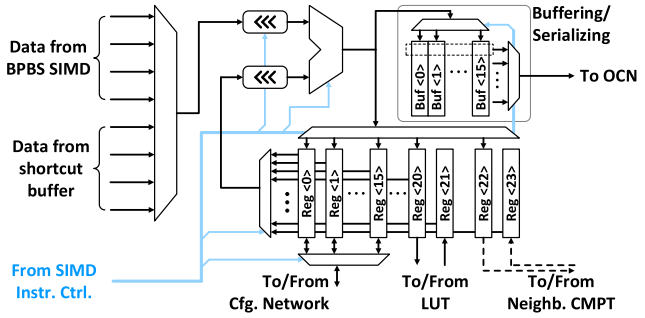


Fig. 10. Microarchitecture of CMPT SIMD datapath, supporting various element-wise and cross-element operations.

B. Input Buffer

Fig. 8(a) shows details of the input buffer, which can be configured to support different computation mappings, dataflows, and reuse patterns. The parallel input ports take OCN data at configurable bandwidth (e.g., depending on level of convolutional reuse relative to a stride-1 3×3 kernel) via multiplexers that are capable of running at up to $9\times$ the CIMA interface clock frequency.

The input buffer comprises eight SRAM-based line buffers, each of which is designed to buffer pixels for a convolutional window. The switching networks around the SRAM blocks enable flexible in-buffer arrangement of data, for ease of sequencing to the CIMA. This includes arranging data near the base of the CIMA to facilitate column gating for improved energy efficiency and SNR. Additionally, local padding logic is provided, to generate zero padding, as needed.

C. Shortcut Buffer

The auxiliary shortcut buffer shown in Fig. 8(b) has a similar, though simplified, structure as the input buffer. As data from the shortcut buffer is directly provided to the NMC SIMDs, the maximum input dimensionality is set to 256 corresponding to the number of CIMA columns.

In addition to sending activation data to the NMC SIMDs, the shortcut buffer can also bypass and directly send data to the CIMU output ports for transmission on the OCN. Such bypassing enables efficient support for NN dataflow, such as depth-wise shuffling (by configuring CIMU input-/output-port connections to the OCN) and shortcut paths (by providing path latency through buffering for proper reconvergence). Direct input-/output-port connectivity, isolated from other CIMU datapath blocks, enables the shortcut buffer to be a flexible independent resource for implementing NN operations.

D. BPBS SIMD Datapath

The BPBS SIMD constructs digitized outputs from the column computations, corresponding to inner products between binary vector elements, into two's compliment results across column computations, corresponding to inner products between multi-bit vector elements. Fig. 9 shows the microarchitecture of the BPBS SIMD datapath, which has similarities with the near-memory datapath employed in [30]. The instruction controller for BPBS SIMD issues one instruction per clock cycle from a 128-entry instruction buffer. The SIMD instruction set has both fused instructions for BPBS computation and a no-operation instruction, to flexibly align the pipeline with CIMA and CMPT SIMDs. To match the layout pitch and throughput of the CIMA columns, each datapath is multiplexed across four columns. This optimizes for 4-b weights, with minimal-overhead extension to 8-b supported in the CMPT datapath (described next). The datapath includes multiple pipelined computation stages, which receive operands from local registers and SIMD instructions. A shift-and-scale stage, composed of integer adder and multiplier, is provided for ADC gain and offset correction and bias-shift and scaling for batch normalization. A barrel-shifting stage is provided for multiplying with local base-two exponential terms, and applying global binary weighting for BPBS construction. Finally, buffers and an accumulator are provided for serial BPBS operations and for interfacing to the CMPT SIMD.

Two features of the datapath are as follows. First, in addition to ADC outputs from the CIMA, the datapath can also receive shortcut-buffer data, to execute programmable computations on shortcut-path activations. Second, a face-to-face connection (F2FC) input is provided at the accumulator for summing partial inner-product results from an adjacent CIMU core, through dedicated interfaces for efficient MVM inner-dimension extension.

E. CMPT SIMD Datapath

Fig. 10 shows the microarchitecture of the CMPT SIMD datapath module, which enables support for different NN dataflows through arithmetic following MVMs. The CIMU

core includes 16 CMPT modules, with each multiplexed across four BPBS SIMD datapaths. This is motivated by two considerations. First, multiplexing again enables layout-pitch and throughput matching, where supporting diverse computations in the CMPT SIMD leads to larger datapath area and optimizing for 4/8-b weights leads to considerable bandwidth reduction from the SIMD datapaths. Second, this allows CMPT-datapath input locality across $4 \times 4 = 16$ CIMA columns, enabling low-overhead cross-element operations (e.g., as required by LSTMs) and support for weight precision larger than 4 b. In addition to the BPBS SIMD datapath, the CMPT SIMD datapath can take inputs from the shortcut buffer for more flexible activation path merging (e.g., as needed in GRU [40]).

The CMPT SIMD datapath comprises four main blocks: 1) arithmetic-logic unit (ALU); 2) register file for intermediate storage; 3) shared look-up table (LUT) across all the CMPT datapath modules; and 4) an output buffer for sending computation results to the OCN. The ALU takes two inputs from the BPBS SIMD, shortcut buffer, or local register file, both of which can be barrel shifted for multiplication with base-two exponent terms. The ALU supports standard integer arithmetic computations, such as ADD/SUB/MULT, and specialized operations for NN dataflows, such as ReLU activation function, average pooling, and max pooling. The computed data are either pushed back to the local register file, or forwarded to the output buffer. The register file has 16 general-purpose entries for storing the intermediate result, which is adequate support for NNs of interest. These register file entries can also be preloaded with constant numbers, or read out for debugging purposes, through the CIMU configuration network. Two special-purpose register-file entries are provided for exchanging data with neighboring datapath modules to better support cross-element operations. The nonlinear activation functions, such as sigmoid and *tanh*, are implemented via a shared LUT, which is controlled through two additional special-purpose registers (one for addressing, while the other one for reading LUT data), without intervention of CMPT SIMD instructions. The output buffer stores the final quantized output activations, and reshapes them into bit streams of multiple parallel output activations.

V. Programmable and Scalable NN Mapping

This section illustrates how the architecture and microachitecture support programmable and scalable mapping of NNs by flexible supports for parallelism. First, several example mappings of different NN dataflows are described. Then, methods for extending inner and outer MVM dimensions for NN layer scalability are described. Finally, the impact of such supports are analyzed, particularly on the scalability of the architecture.

A. Layer Mappings

Considering some standard NN layer types, Fig. 11 shows example mappings of convolutional, shortcut/residual-connection, fully-connected, and memory-augmented layers. Fig. 11(a) illustrates mapping of a convolutional layer (e.g., 3×3 kernel, stride of 1), where the convolutional kernel is flattened, and weights are mapped into the CIMA along the column dimension while inputs/outputs correspond to input/output activations. The flattened kernel weights are densely loaded at the bottom of the CIMA column at run time to enable gating of unused rows for enhancing energy efficiency and SNR.

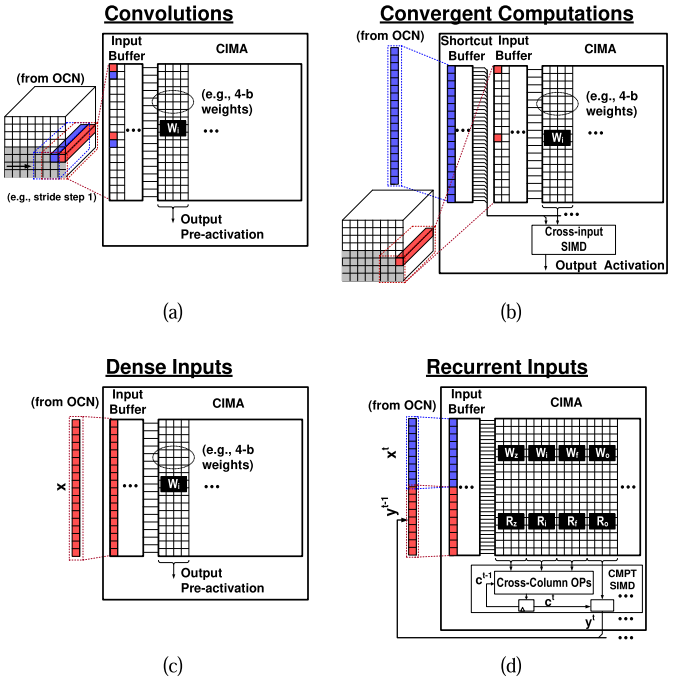


Fig. 11. Examples of various dataflow mappings for different NN-layer types, including (a) convolutional layer, (b) convergent shortcut/residual connection between two activation paths, (c) dense layers, and (d) memory augmented layers (e.g., LSTM).

Fig. 11(b) illustrates mapping of a shortcut/residual connection between two activation paths, supported through the input buffer together with the shortcut buffer. The input buffer receives and sequences activation data to the CIMA, as described above. The shortcut buffer also receives activation data from the OCN (e.g., residual path), however, passes it directly to the NMC SIMD engines for merging computation. The shortcut buffer also provides synchronization as a FIFO with latency configurable to match with the layer-pipeline latency.

Fig. 11(c) illustrates mapping of dense and other no-reuse (e.g., 1×1 convolution) layers. The input buffer receives activations with high bandwidth from the OCN, and directly sequences data to the CIMA.

Fig. 11(d) illustrates mapping of a recurrent/memory-augmented layer. The OCN is configured to provide output-to-input routing. The input buffer then concatenates the new state input x^t with preceding hidden state y^{t-1}. Using LSTMs as an example, the weight matrices preceding the four LSTM gates are interleaved across the CIMA columns, so that the element-wise and cross-element operations after MVMs can be performed within neighboring NMC SIMD datapaths, for output-data locality. The register file within the CMPT SIMD

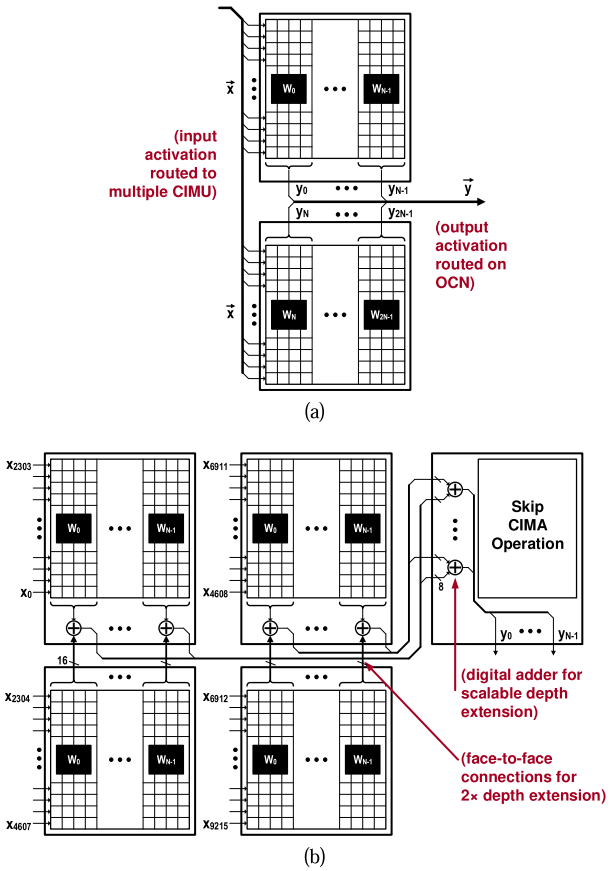


Fig. 12. Examples of mappings for layer scalability, including extension of (a) MVM outer dimensionality and (b) MVM inner dimensionality.

datapath also stores the previous cell state C^{t-1} locally for computing the current cell state C^t.

Weight and activation sparsity, which has been studied for digital accelerators [12], also impacts dataflow. The CIMA natively supports energy savings in proportion to activation sparsity, simply by masking zero-valued data within the input drivers [30], which dominant CIMA energy consumption. Though such masking does not increase throughput, like with digital zero-skipping mechanisms, some recent IMC designs have added additional supports at the cost of corresponding overheads [25], [41]. Weight sparsity is not a focus of optimization in this work, due to limitations observed at reduced parameter quantization levels; however, weight-sparsity at kernel-level granularity is natively exploited.

B. Layer Scalability

NN layer scalability requires efficiently extending MVM inner and outer dimensionalities across CIMU cores. Fig. 12(a) shows how extension of output channels is handled. The OCN broadcasts input activations to multiple CIMU cores operating in parallel. Then, the outputs from parallel CIMUs are regrouped on the OCN for feeding to subsequent NN layers. Fig. 12(b) shows how extension of input channels is supported. At compile time, kernels are segmented for mapping to parallel CIMU cores, and during execution the parallel CIMU outputs are reduced. This is done in two ways.

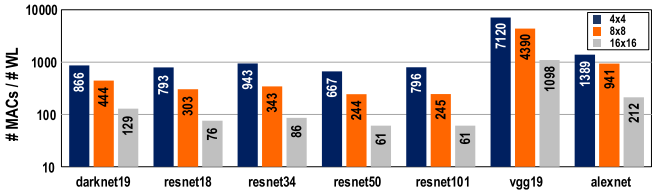


Fig. 13. Ratio of MAC operations to WL operations with architecture scale-up, with hardware parallelism used to map MVM execution loops.

First, as mentioned in Section IV-B, computed partial inner products from one CIMU can be fed and summed with those of a second CIMU, extending number of input channels by two. Second, for further arbitrary extension, computed activations from two CIMUs can be fed to a third CIMU for summation via element-wise adders following the input buffer (and then passed directly to the OCN or NMC SIMD for further element-wise operations).

C. Analysis of Mappings Exploiting Flexible Parallelism

As mentioned in Section II-C, the physical attributes of IMC introduce important challenges in exploiting parallelism toward efficient and scalable execution of NNs. These make it necessary to optimize flexibly across different forms of hardware parallelism, motivating the accelerator architecture described. This section illustrates the execution gains of such optimization, relative to a baseline approach based on data parallelism, commonly used in digital NN accelerators.

For quantitative illustration of the challenges, especially as the IMC architecture is scaled (from a 4×4 CIMU array to a 16×16 CIMU array), Fig. 13 shows the ratio of MAC operations to WL operations for different NN benchmarks. This analysis assumes that hardware parallelism is primarily used for mapping the MVM execution loops. As seen, the number of MAC operations decreases significantly with architectural scale-up, often falling below the expected compute-bound regime (as visualized in Fig. 2). This indicates that alternate approaches are required to exploit the high parallelism resulting from IMC hardware.

The architectural supports for flexible parallelism, spanning data-level (replication) and operation-level (pipeline) parallelism, allow compiler optimizations to efficiently achieve scalable execution on IMC hardware, i.e., to ensure high hardware utilization while minimizing state-loading overheads. Specifically, instead of relying just on replication across hardware, jointly exploiting data and pipeline parallelism enables scale-up while addressing WL operations. Fig. 14 shows the resulting reduction in WL operations, when optimizing across the parallelism supports for the different NN benchmarks. The reductions increase with architectural scale-up.

Fig. 15 shows the overall throughput gains through such optimization. Throughput gains are seen in many cases, where operation-level parallelism has modest overheads (e.g., pipeline buffering and latency). However, in several cases, the approach of data-level parallelism remains optimal. It is noted that this has strong correspondence with the ratio of MAC operations to WL operations (i.e., level of weight reuse)

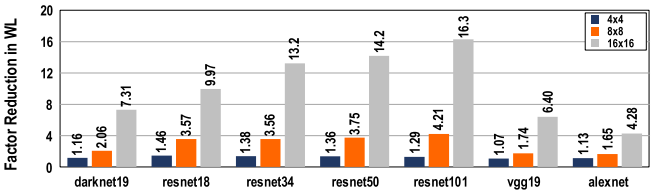


Fig. 14. Possible reduction of WL operations, when optimizing across supported parallelism approaches.

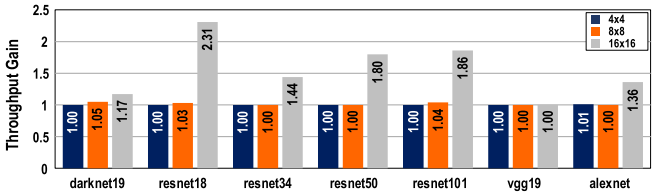


Fig. 15. Overall throughput gains by optimizing across supported parallelism.

analyzed in Fig. 13, but now with the overheads of the parallelism supports also having impact.

VI. Prototype Measurements

The IMC-based inference accelerator is prototyped in a 16 nm CMOS technology, integrating a 4×4 array of CIMU cores. Fig. 16(a) shows the die photograph, with top-level architectural blocks labeled. Fig. 16(b) shows the detailed layout view of a CIMU core, with CIMA, SIMD engines, and buffering resources marked. Fig. 16(c) and (d) show the photograph and block diagram of the test setup for measurement and demonstration, including a custom PCB implemented as an FMC daughter card with BNC power connectors for monitoring the power of key blocks separately, a controller FPGA to communicate with a host PC through PCIe link [42], and microcontrollers for start-up (PLL) control.

A. Block-Level Measurements

Table III provides a summary of the overall chip- and block-level measurements. Measurements are taken at the nominal voltage of 0.8 V. While timing closure targeted 500 MHz operation, testing was performed with CIMU digital logic running at 200 MHz and CIMA ADC outputs provided at 20 MS/s, where the ultimate frequency was limited by power deliver in the wirebond prototype. The primary focus of this design being on the architectural and microarchitectural supports required for efficient scalable execution, the frequency can be increased with further logic-path timing optimizations. The energy breakdown for execution of a representative CNN kernel of 3×3 window size and 128 input-channel depth is also shown. The energies for the CIMA, input buffer, and SIMD engines are the total numbers, measured for 4-b weight and 4-b activation configuration, assuming full utilization of the CIMA. The CIMA write energy corresponds to the standard SRAM write operation for loading weights into IMC memory. The OCN energy is reported for sending all four bits of one output activation through one OCN block.

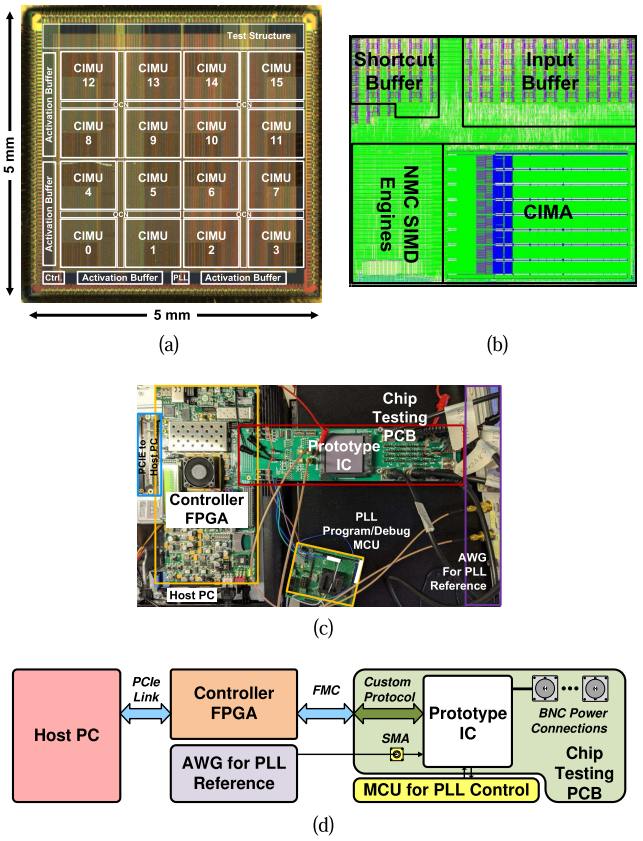


Fig. 16. Prototype system used for testing, including (a) die photograph of the test chip in 16-nm CMOS, (b) layout view of the CIMU core, (c) photograph of the test setup for interfacing with host processor, and (d) block diagram of the test setup.

TABLE III
Summary of Chip Energy Measurements

Technology (nm)	16	Digital frequency (MHz)	200
V_{DD} (V)	0.8	Total area (mm^2)	25
Energy breakdown ($3 \times 3 \times 128$ CNN kernel with 4-b act. and weight)			
CIMA (pJ/output act.)	19.08	BPBS SIMD (pJ/output act.)	3.89
CIMA Write (pJ/bit)	0.23	CMPT SIMD (pJ/output act.)	0.60
Input buf. (pJ/output act.)	8.60	OCN (pJ/output act./seg.)	0.27

As shown in Table III, the CIMA dominates energy, with the NMC SIMD engines introducing only small overhead. Thus, the overall architecture substantially leverages IMC. It is noted that the next dominant energy corresponds to the input buffer, which is primarily responsible for configurability in scalable dataflow mappings, exploiting hardware parallelism.

Fig. 17 presents detailed characterization of the basic CIMU operations, showing the transfer function of each CIMA column. For this, all 1's are loaded for the matrix data stored in the CIMA, and the number of 1's versus 0's in the input vector is swept. This ideally generates a ramp voltage at the ADC input. The data shown are after the 8-b ADC (following offset calibration), thus corresponding to the fundamental CIMA-transfer function from digital inputs to digital outputs.

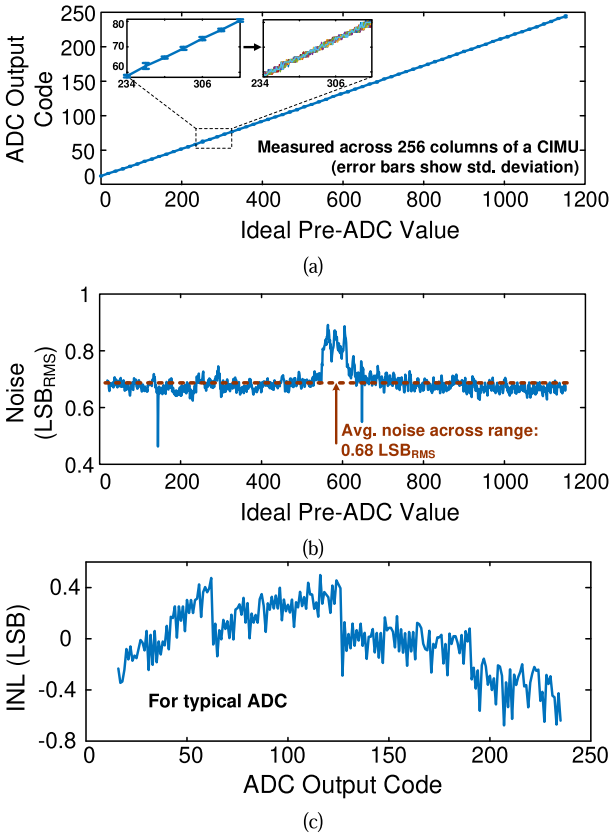


Fig. 17. Basic CIMA-column measurements, including (a) overall transfer functions, (b) noise standard deviation, and (c) INL.

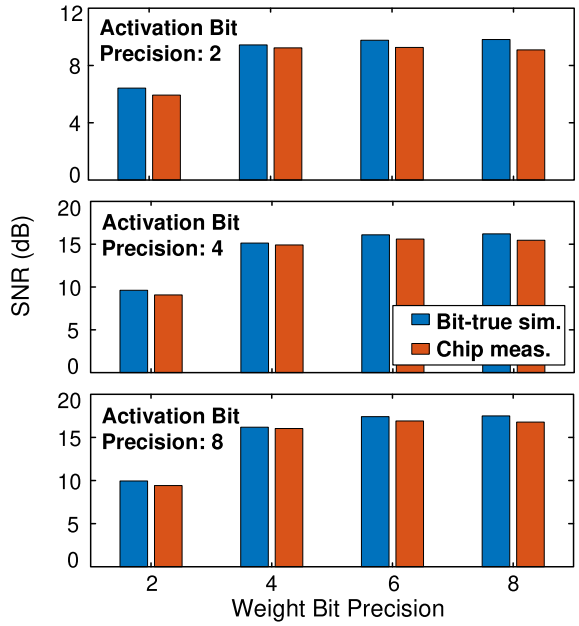


Fig. 18. Measured and bit-true-simulated SNR of multi-bit MVMs.

It is noted that the transfer function thus characterizes all nonidealities, through input driving, capacitor-based column computation, and ADC digitization.

Fig. 17(a) plots data from all columns in one CIMU core, and data from all CIMU cores are observed to be highly similar. The transfer function exhibits excellent linearity and stability, with tight error bars showing the standard deviation across the 256 columns (shown in detail and individual overlay, in the inset).

Fig. 17(b) shows the noise standard deviation observed in the ADC outputs across 20 repeated measurements. The average standard deviation is 0.68 LSB (shown as the orange dotted line).

Fig. 17(c) shows the overall integral nonlinearity (INL) of the transfer function from CIMA-input to ADC-output. The CIMA column, together with the following ADC, achieves nonlinearity less than 1 LSB. The periodic INL steps indicate mismatch in the SAR-ADC capacitors.

Moving now from column transfer functions to MVM operations, Fig. 18 shows the overall computation SNR after BPBS reconstruction with different bit precisions. For this, uniformly-distributed input data are generated in floating-point precision, and then quantized to integer for both input-vector and matrix elements. The blue bars show the ideal SNR (relative to full floating-point computation), assuming only quantization effects in data generation and ADC [i.e., representing ideal signal-to-quantization-noise ratio (SQNR)]. While the red bars show the final measured SNR including all physical analog effects.

As seen, at 4–8 b the SNR saturates, set by ADC quantization effects (as also predicted by analysis in [30]), while lower SNR is observed at 2 b, set by input-data quantization. In all cases, the measured SNR closely matches the ideal SQNR, indicating that quantization is the only significant noise effect in the computation. Since quantization can be modeled using standard digital approaches, a robust abstraction of the computation can be developed for architectural and software integration.

Table IV compares this work with the recently demonstrated state-of-the-art NN accelerators, including both digital and IMC-based architectures. The peak MAC-level throughput and peak MAC-level energy efficiency are measured for both 4- and 8-b weights/activations. As described in Section II-B, the number of CIMA cycles for BPBS computation scales linearly with both weight and activation precision, and accordingly so does the throughput and energy efficiency. The reported MAC-level energy includes that from the CIMA (with ADCs) and BPBS SIMD for BPBS reconstruction (other operations, such as input buffering and activation functions, are performed in peripheral blocks within the architecture as previously described to optimize end-to-end NN executions, but not included in the MAC-level energy). This work achieves orders-of-magnitude gain in MAC-level energy efficiency compared with digital implementations and highest MAC compute density. Comparing with other IMC demonstrations, this work achieves the highest energy efficiency, throughput, and compute density. There is only one exception [26], due to different target model/task complexities and compute precisions. This is also the only IMC work that programmably supports a range of model types, optimized for execution scalability. In addition to the MAC-level efficiency and throughput, the accelerator-level

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

	Not In-memory Computing			In-memory Computing					
	[19] Chen, JSSC'17	[20] Bankman, JSSC'19	[21] Jiao, ISSCC'20	[23] Guo, VLSI'19	[24] Wang, JSSC'20	[25] Yue, ISSCC'20	[26] Sinangil, JSSC'20	[30] Jia, JSSC'20	This work
Technology	65nm	28nm	12nm	65nm	28nm	65nm	7nm	65nm	16nm
Area (mm^2)	16	6	709	6.2	2.7	9	0.0032	8.56	25
V_{DD} (V)	0.8 – 1.2	0.6 | 0.8	0.85	0.9 – 1.1	0.9 – 1.1	0.9 – 1.05	0.8 | 1.0	0.85 | 1.2	0.8
On-chip Memory	864 Kb	2.6 Mb	1536 Mb	64 Kb[1]	1 Mb[1]	16 Kb[1]	4 Kb[1]	576 Kb[1]	4.5 Mb[1]
Bit Precision	16 b	1 b	8 b | 16 b	1 | 1 | 3 b	Arbitrary	2 b – 8 b	4 b	1 – 8 b	1 – 8 b
Optimized for Scalability	YES	NO	YES	NO	NO	NO	NO	NO	YES
Peak MAC-level Throughput (GOPS) 4-bit[2]	1.34K	4.5 | 30	3.3 M	38.4	131	36.15	372 | 455	54.6 | 136.6[4]	11.8 K[4]
Peak MAC-level Throughput (GOPS) 8-bit[3]	336	1.1 | 7.5	825 K	9.6	32.7	9.04	93 | 114	13.7 | 34.2[4]	3.0 K[4]
Peak MAC-level Energy Eff. (TOPS/W) 4-bit[2]	1.33	48.3 | 33.4	11.8	0.73	4.0	79.4[5]	611 | 436	25 | 12[4]	121[4]
Peak MAC-level Energy Eff. (TOPS/W) 8-bit[3]	0.33	12.1 | 8.4	3.0	0.18	1.0	19.8[5]	153 | 109	6 | 3[4]	30[4]
MAC Comp. Density (TOPS/mm^2)[2]	0.192	0.0024 | 0.016	4.65	0.014	0.16	0.066	116 | 142	0.015 | 0.038[4]	2.67[4]
Application	CNN	CNN	CNN	RNN	Vector OPs	CNN, MLP	Matrix-vec. mult.	Matrix-vec. mult.	CNN, RNN, MLP, etc.

[1]Only consider memory conduct in-memory computing [2]Numbers normalized to implemented/projected 4-bit compute
[3]Numbers normalized to implemented/projected 8-bit compute [4]Scales with number of bits used for input activation and weight [5]Number by exploring weight/input sparsity

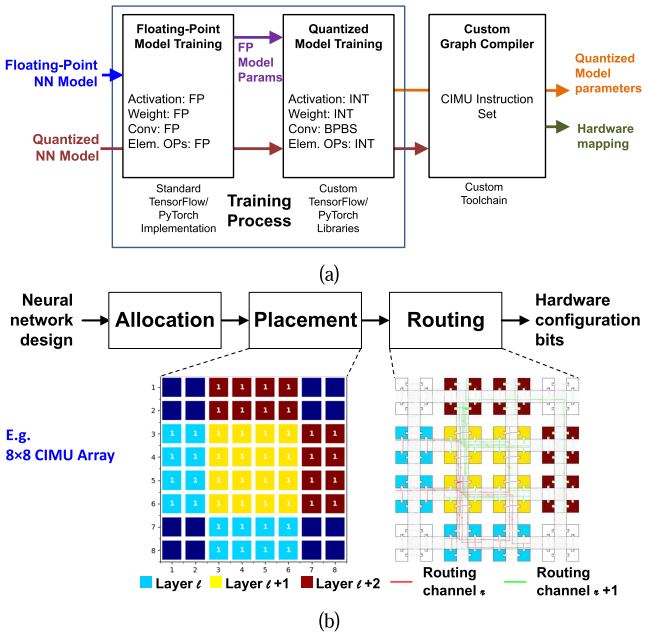


Fig. 19. Developed toolchains and compilers for (a) quantized NN training and (b) NN mapping and on-chip deployment.

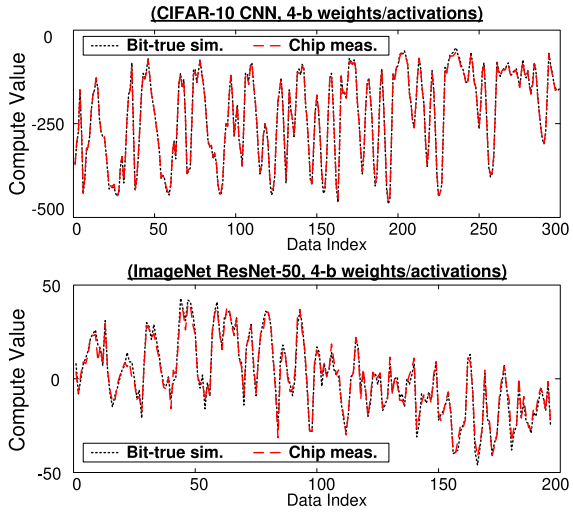


Fig. 20. Measured and bit-true-simulated pre-activations for benchmark NNs.

efficiency and throughput are important and characterized next by considering specific workload executions.

B. Software Mapping Tools and System Demonstrations

The accelerator architecture is co-designed with software mapping algorithms and tools. To better outline the model-deployment flow and system-demonstration configurations, Fig. 19 shows the developed NN training flow and software-mapping toolchain.

Fig. 19(a) shows a block diagram of the NN training flow, with software libraries integrated in TensorFlow and PyTorch. First, an NN model is trained with floating-point parameters. Then, the floating-point model is provided to the custom quantized-NN training flow, which models the BPBS and SIMD engines' quantization effects. This generates the integer NN model, which is then fed to a prototype graph compiler and mapping tools for deploying on to the prototype IC. It is noted that the trainer only models quantization effects, with no analog nonidealities necessary to consider.

Fig. 19(b) shows a block diagram of the toolchain for mapping and deploying NNs on the architecture. The prototype graph compiler performs allocation and scheduling of CIMU resources to maximize hardware utilization while executing the NN compute graph. Following allocation, physical placement is performed, employing a simulated-annealing-based algorithm, to minimize communication between CIMU cores. Finally, a heuristic optimization algorithm leveraging integer-linear programming is used to configure the OCN routing resources for communication between CIMU

TABLE V
NN DEMONSTRATIONS

Dataset	CIFAR-10	ImageNet
Neural-network style	VGG	ResNet-50
Bit precision	4b/4b weight/activation	4b/4b weight/activation[1]
Accuracy of chip (vs sim.)	91.51% (vs. 91.68%)	73.33% (vs. 73.89%)
Frame per second[2]	7815 images/sec.	581 images/sec.
Avg. throughput[2]	9.8 TOPS	3.4 TOPS
Accelerator energy eff.[2]	51.5K image/sec/W	3.0K image/sec/W
IMC bit-cell utilization[2]	83%	29%
Neural-network topology	L1: CONV 3×3 128 L2: CONV 3×3 128 L3: CONV 3×3 128 L4: CONV 3×3 128 POOL L5: CONV 3×3 256 L6: CONV 3×3 256 POOL L7: CONV 3×3 256 L8: CONV 3×3 256 POOL L9: DENSE 1024 L10: DENSE 1024 L11: DENSE 1024	Stack 1 { CONV 1×1 64; CONV 3×3 64; CONV 1×1 256 } ×3 Stride 2 Stack 2 { CONV 1×1 128; CONV 3×3 128; CONV 1×1 512 } ×4 Stride 2 Stack 3 { CONV 1×1 256; CONV 3×3 256; CONV 1×1 1024 } ×6 Stride 2 Stack 4 { CONV 1×1 512; CONV 3×3 512; CONV 1×1 2048 } ×3 DENSE 1000

[1]Shortcut projection in bottleneck layers employ 8-b weights, as typically required (accounting for <10% of total MACs).

[2]Energy, throughput and utilization are for array-based accelerator core, assuming 28MB weight buffer. The energy and cycles for embedded L2 weight buffers is not included.

cores, similar to CGRA and FPGA routing approaches [43], [44]. The toolchain then generates configuration data necessary for end-to-end execution of NN models on the architecture.

Using this training and mapping flow, multiple NNs have been demonstrated on the prototype. As two examples, Fig. 20 shows both measured and bit-true-simulated ideal pre-activation data for two benchmarks. The top data correspond to a VGG-style network performing CIFAR-10 image classification, while the bottom data correspond to a ResNet-50 network performing ImageNet classification, both with 4-b weights and activations. The measured pre-activation data, shown as red lines, matches closely with the simulated data, shown as black lines. Table V summarizes the results of these two benchmarks. The VGG-style NN is fully mapped on chip, while all four stacks of the ResNet-50 model are mapped, including 48 convolutional layers (three shortcut projections between stacks require 8-b precision, and are computed off-chip as this eases mapping of all other 4-b layers).

The 4-b VGG-style network for CIFAR-10 image classification achieves energy efficiency of 51.5 k image/s/W, and throughput of 7815 images/s, with testing accuracy of 91.51%. While the 4-b ResNet-50 network for ImageNet dataset shows 3.0 k image/s/W energy efficiency and 581 image/s throughput, with 73.33% testing accuracy. The achieved testing accuracies for both CIFAR-10 and ImageNet datasets are within the known range of the state-of-the-art quantized NNs [11], [45], particularly for fully-integer models, as used in this work [46]. As shown, the final accuracies from chip measurements match with the reference accuracies corresponding to ideal software-based computations. The VGG-style network achieves effective IMC bit-cell hardware utilization of 83%, while ResNet-50 achieves 29%. The utilization is primarily limited by mapping granularity, due to the chosen IMC array dimensions exceeding the workload MVM inner dimensions, as mentioned in Section III-B (e.g., ResNet-50 is particularly limited by 1×1 convolutions).

VII. CONCLUSION

IMC accelerates MVM operations, which dominate NN applications, by addressing both computation and data-movement costs. However, scalable end-to-end executions are challenged by the proportionally high cost of writing data to and extremely high parallelism of IMC hardware. To address these challenges, flexible parallelization strategies for mapping NNs are required, with associated architectural and microarchitectural support for minimizing the associated overheads. A prototype NN inference accelerator is demonstrated in 16 nm CMOS based on a scalable architecture, including a 4×4 array of IMC-based cores. The accelerator achieves 30 TOPS/W peak MAC-level energy efficiency and 3 TOPS peak MAC-level throughput, for 8-b computations. Using developed training and mapping toolchains, several NN benchmarks are demonstrated. At the accelerator level, an 11-layer CNN with 4-b weights and activations, performing CIFAR-10 classification, achieves test accuracy of 91.51% (matching ideal digital computation), with energy of 51.5 k image/s/W and throughput of 7815 images/s. A ResNet-50 network with 4-b weights and activations, performing ImageNet classification, achieves test accuracy of 73.33% (matching ideal digital computation), with energy efficiency of 3.0 k image/s/W and throughput of 581 images/s.

ACKNOWLEDGMENT

The authors would like to thank Julian Puscar at the University of California, San Diego (UCSD), La Jolla, CA, USA, Prof. Ian Galton, UCSD, Austin Rovinski at the University of Michigan (UMich), Ann Arbor, MI, USA, and Prof. Ronald Dreslinski, UMich, for their help with the PLL.

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] H. Yin *et al.*, "Dreaming to distill: Data-free knowledge transfer via deep inversion," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8712–8721.
[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[7] 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.
[8] A. W. Senior *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.
[9] B. Schmauch *et al.*, "A deep learning model to predict RNA-seq expression of tumours from whole slide images," *Nature Commun.*, vol. 11, no. 1, pp. 1–15, Dec. 2020.
[10] 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.

[11] J. Choi, Z. Wang, S. Venkataramani, P. I-Jen 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

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

[13] 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

[14] 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.

[15] R. Wei, L. Schwartz, and V. Adve, "DLVM: A modern compiler infrastructure for deep learning systems," 2017, *arXiv:1711.03016*. [Online]. Available: http://arxiv.org/abs/1711.03016

[16] M. Sivathanu, T. Chugh, S. S. Singapuram, and L. Zhou, "Astra: Exploiting predictability to optimize deep learning," in *Proc. 24th Int. Conf. Architectural Support Program. Lang. Oper. Syst. (ASPLOS)*, Apr. 2019, pp. 909–923.

[17] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Dec. 2014, pp. 609–622.

[18] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/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.

[19] 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.

[20] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8 μ J/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.

[21] Y. Jiao *et al.*, "A 12 nm programmable convolution-efficient neural-processing-unit chip achieving 825TOPS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 136–140.

[22] 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.

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

[24] 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.

[25] J. Yue *et al.*, "A 65 nm computing-in-memory-based CNN processor with 2.9-to-35.8TOPS/W system energy efficiency using dynamic-sparsity performance-scaling architecture and energy-efficient inter/intra-macro data reuse," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 234–236.

[26] M. E. Sinangil *et al.*, "A 7-nm compute-in-memory SRAM macro supporting multi-bit input, weight and output and achieving 351 TOPS/W and 372.4 GOPS," *IEEE J. Solid-State Circuits*, vol. 56, no. 1, pp. 188–198, Jan. 2021.

[27] 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.

[28] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42 pJ/decision 3.12TOPS/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.

[29] 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.

[30] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

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

[32] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Architecture (ISCA)*, Jun. 2017, pp. 1–12.

[33] 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.

[34] Flex Logix Technologies. (Oct. 2020). *Flex Logix Announces Working Silicon of Fastest and Most Efficient AI Edge Inference Chip*. [Online]. Available: https://flex-logix.com/wp-content/uploads/2020/10/2020-10-20-X1-chip-launch-FINAL.pdf

[35] Qualcomm Technologies. (Sep. 2020). *Qualcomm Cloud AI 100 Announcement*. [Online]. Available: https://www.qualcomm.com/media/documents/files/qualcomm-cloud-ai-100-announcement.pdf

[36] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2003, pp. 296–301.

[37] S. Yin, X. Yao, T. Lu, L. Liu, and S. Wei, "Joint loop mapping and data placement for coarse-grained reconfigurable architecture with multi-bank memory," in *Proc. 35th Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–8.

[38] T. Nowatzki, N. Ardalani, K. Sankaralingam, and J. Weng, "Hybrid optimization/heuristic instruction scheduling for programmable accelerator codesign," in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn. (PACT)*, 2018, pp. 1–15.

[39] S. K. Gonugondla, C. Sakr, H. Dbouk, and N. R. Shanbhag, "Fundamental limits on the precision of in-memory architectures," in *Proc. 39th Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: http://arxiv.org/abs/1412.3555

[41] J.-H. Kim, J. Lee, J. Lee, J. Heo, and J.-Y. Kim, "Z-PIM: A sparsity-aware processing-in-memory architecture with fully variable weight bit-precision for energy-efficient deep neural networks," *IEEE J. Solid-State Circuits*, vol. 56, no. 4, pp. 1093–1104, Apr. 2021.

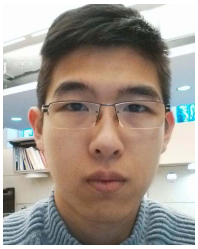
[42] M. Jacobsen and R. Kastner, "RIFFA 2.0: A reusable integration framework for FPGA accelerators," in *Proc. 23rd Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2013, pp. 1–8.

[43] S. A. Chin and J. H. Anderson, "An architecture-agnostic integer linear programming approach to CGRA mapping," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6.

[44] C. H. Hoo, A. Kumar, and Y. Ha, "ParaLaR: A parallel FPGA router based on Lagrangian relaxation," in *Proc. 25th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2015, pp. 1–6.

[45] C. Louizos, M. Reisser, T. Blankevoort, E. Gavves, and M. Welling, "Relaxed quantization for discretized neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–15.

[46] X. Zhao, Y. Wang, X. Cai, C. Liu, and L. Zhang, "Linear symmetric quantization of neural networks for low-precision integer hardware," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–16.



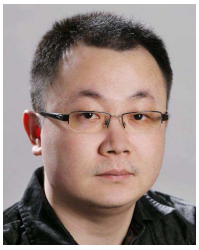
Hongyang Jia (Member, IEEE) received the B.Eng. degree in microelectronics from Tsinghua University, Beijing, China, in 2014, and the M.A. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2016 and 2021, respectively.

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

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

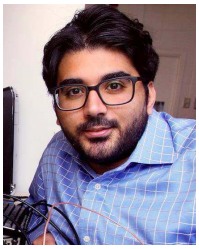
Murat Ozatay (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 2015, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2017, where he is currently pursuing the Ph.D. degree.

His research focuses on bringing together algorithms and insights for learning with technologies and systems for advanced sensing. His primary research interests include machine learning, artificial intelligence, the Internet-of-Things, and the design of very-large-scale integration systems.



Yinqi Tang (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.

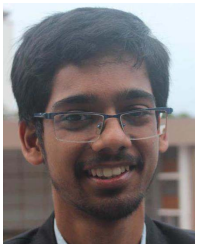
His current research interests include energy-efficient hardware systems for machine learning and deep learning applications, in both algorithm and hardware design aspects.



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.S. and Ph.D. degrees in electrical engineering from Princeton University, Princeton, NJ, USA, in 2015 and 2020, respectively.

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

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



Rakshit Pathak (Student Member, IEEE) received the B.Tech. degree in electronics and electrical communication engineering from IIT Kharagpur, Kharagpur, India, in 2018, and the M.A. degree in electrical engineering from Princeton University, Princeton, NJ, USA, in 2020, where he is currently pursuing the Ph.D. degree under the guidance of Prof. Naveen Verma.

His research focuses on algorithm-hardware co-design of machine learning platforms. His research includes design of heterogeneous computing and low-energy in-memory computing systems.

Jinseok Lee (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree with Princeton University, Princeton, NJ, USA.

Naveen Verma (Senior Member, IEEE) received the B.A.Sc. degree in electrical and computer engineering from The University of British Columbia, Vancouver, BC, Canada, in 2003, and the M.S. and Ph.D. degrees in electrical engineering from 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 TRANSACTIONS ON COMPONENTS, PACKAGING AND MANUFACTURING TECHNOLOGY Best Paper Award. He also serves on the technical program committees for ISSCC, VLSI Symposium DATE, and the IEEE Signal-Processing Society (DISPS). He has served as a Distinguished Lecturer for the IEEE Solid-State Circuits Society.