A 65 nm General-Purpose Compute-in-Memory Processor Supporting Both General Programming and Deep Learning Tasks

Yuhao Ju, *Member, IEEE*, Yijie Wei, *Member, IEEE*, and Jie Gu, *Senior Member, IEEE*

***Abstract*— This work presents a special unified compute-in-memory (CIM) processor supporting both general-purpose computing and deep neural network (DNN) operations, referred to as the general-purpose CIM (GPCIM) processor. By implementing a unique CIM macro with two different bitcell arrays and a central compute unit (CCU), GPCIM can be reconfigured to a CIM DNN accelerator or a CIM vector central processing unit (CPU). By using special reconfigurability, dataflow, and support of a customized vector instruction set, GPCIM achieves SOTA performance for end-to-end deep learning tasks with enhanced CPU efficiency and data locality. A 65 nm test chip was fabricated demonstrating a 28.3 TOPS/W DNN macro efficiency and a best-in-class peak CPU efficiency of 802 GOPS/W. Benefit from a data locality flow, 37%–55% end-to-end latency reduction on artificial intelligence (AI)-related applications is achieved by eliminating inter-core data transfer in traditional heterogeneous system-on-chip (SoC). An averaged 17.8× CPU energy efficiency improvement is achieved compared with vector RISC-V CPUs in the existing machine learning (ML) SoCs.**

***Index Terms*— Compute-in-memory (CIM), deep neural network (DNN) accelerator, end-to-end performance, general-purpose computing, machine learning (ML), system-on-chip (SoC), vector central processing unit (CPU).**

I. Introduction

As THE size of the deep neural network (DNN) model rapidly grows for better accuracy and performance in machine learning (ML)/artificial intelligence (AI) applications, the demand for hardware energy efficiency is also quickly rising, especially for edge device applications. Various optimization techniques have been explored at different design hierarchies, such as quantization [1], [2], variable precisions [3], [4], [5], adaptive clocking [6], sparsity techniques [7], [8], and data compression [9]. One major bottleneck is the frequent access for the on-chip memories such as SRAM, scratch pads, and buffers limiting the further improvement of the performance and energy efficiency of DNN accelerators. As a result, compute-in-memory (CIM) has become one of the most popular circuit-level solutions to overcome the memory access bottleneck for DNN processing. CIM enhances computing efficiency by allowing the arithmetic processing tasks, i.e., multiplication and accumulation (MAC) operations to happen at the original data site, i.e., the SRAM bitcell arrays. Since the data movement is eliminated between the storage units and processing units, CIM can dramatically reduce the memory access power. Recent published CIM works [10], [11], [12], [13], [14], [15], [16], [17] have demonstrated CIM SRAM techniques can improve the ML/AI accelerator energy efficiency from 0.5–1 to 10–100 TOPS/W.

Despite the ongoing boom of CIM processors, one lack of consideration is CPU operations. As for end-to-end processing of AI-related tasks, a general-purpose computing unit, e.g., CPU, is necessary for pre/post-processing, data preparation for neural networks (NNs), accelerator configuration, and non-MAC tasks, especially for edge device applications which usually have higher proportion of pre-processing and data management workload to deal with the raw data from sensors or cameras. Such workload is usually performed by a CPU which contributes significant processing time. As shown in Fig. 1(a), a heterogeneous architecture consisting of both CPU and accelerator is commonly used for tasks involving ML/AI [18], [19], [20], [21], [22], [23]. This traditional architecture engages a CPU core, an accelerator, and a direct memory access (DMA) engine for data transfer.

Fig. 1(b) shows a detailed processing flow to briefly explain the design methodology of a heterogeneous AI SoC. In this SoC, the accelerator is a peripheral for AI acceleration which uses the CPU as a control processor. In the beginning, the CPU can decode the ML/AI models to instruction commands (CMD) to control/configure the accelerator and itself. In addition, for some end-to-end AI applications, the raw data needs some general-purpose pre-processing work which can be done by the CPU. After that, prepared input data, weights, and accelerator commands will be sent to the digital accelerator/CIM core for ML/AI processing by the DMA engine. After the results are generated for the current layer of an ML model, the DMA transfers the output data back to the CPU core doing some data preparation work for the next layer inference such as data reshuffling. After the inter-layer data processing work finishes, another round of data movement and ML inference

Manuscript received 1 March 2024; revised 17 June 2024 and 15 August 2024; accepted 20 August 2024. This article was approved by Associate Editor Ben Keller. This work was supported in part by the National Science Foundation under Grant CCF-2008906. *(Corresponding author: Yuhao Ju.)*

Yuhao Ju and Jie Gu are with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA (e-mail: yuhaoju2017@u.northwestern.edu; jgu@northwestern.edu).

Yijie Wei was with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208 USA. He is now with the Kilby Lab, Texas Instruments, Dallas, TX 75243 USA (e-mail: j-wei@ti.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2024.3453114.

Digital Object Identifier 10.1109/JSSC.2024.3453114

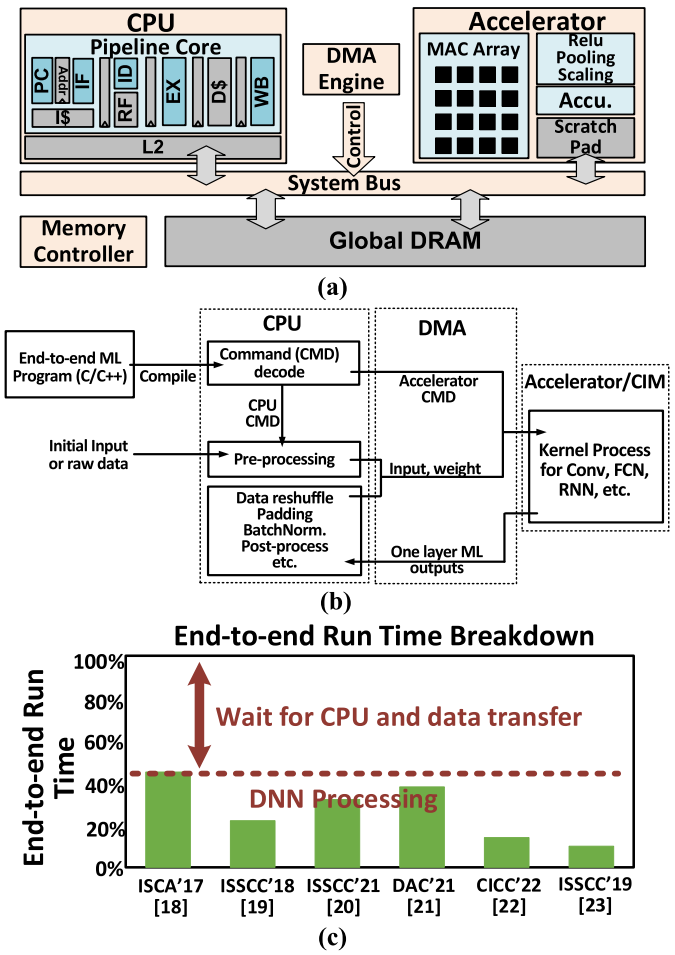


Fig. 1. (a) Conventional heterogeneous architecture for DNN SoC, (b) details of the processing flow for the heterogeneous architecture including a CPU and an accelerator, and (c) run time breakdown for the end-to-end processing of AI-related tasks from the prior DNN SoCs.

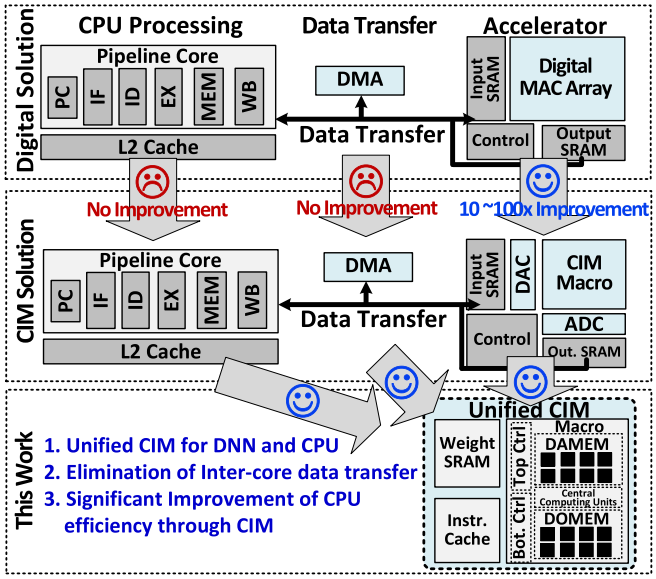


Fig. 2. Challenges of the end-to-end AI tasks using existing digital CIM processors and contributions of this work.

will be launched for the next layer of the model. Basically, the CPU is inserted to control the accelerator and data transfer as well as finish the general-purpose work that the accelerator cannot do in a complete end-to-end ML workload.

As shown in Fig. 1(c) on the reported workload in various accelerators-enriched SoCs, DNN processing time often takes only 12%–45% of total run time, leaving the performance and energy efficiency bottlenecked by CPU processing and data transfer [18], [19], [20], [21], [22], [23]. For instance, as reported in the published SoC [22] from Meta on applications of augmented reality (AR) and virtual reality (VR), to finish the first layer of a convolutional neural network (CNN) model for eye gaze tracking, over 80% of the execution latency is consumed by initial data preparation and data movement. The accelerator only takes 0.18 ms for inference out of a 5.36 ms total latency. Another example is a CNN-based SoC for on-the-fly visual recognition and classification of insect blobs from Intel [19]. The data pre/post-processing, such as image/sensor capture, extraction, accelerator/camera configuration, and so on, takes over 70% of the run-time for the end-to-end procedure.

Unfortunately, considering the whole SoC operations and the end-to-end performance of ML/AI applications, even though the recent CIM developments [10], [11], [12], [13], [14], [15], [16], [17] can achieve orders of magnitude efficiency improvement for MAC operations, the CPU-related general-purpose computing and data transfer are still not addressed by the prior CIM works limiting the performance and energy efficiency of the whole SoCs. Previously, a general-purpose instruction CIM was delivered [16], however, missing the support of efficient DNN computing. Special neural CPU schemes were developed aiming at eliminating SoC-level core-to-core data movement by using reconfiguration techniques to convert a digital accelerator core to a CPU core [24], [25], [26]. However, pure digital implementation in the above works limit the energy efficiency of the operations compared with prior CIM-based designs [10], [11], [12], [13], [14], [15], [16], [17].

To overcome the aforementioned challenges, as shown in Fig. 2, this work proposes a unified general-purpose CIM (GPCIM) architecture. The main contributions of this work include: 1) compared with advanced DNN-based CIM design [10], [11], [12], [13], [14], [15], [16], [17], GPCIM is a unified digital CIM architecture to focus on the end-to-end ML workload bottlenecks, which is not only for efficient DNN inference but also supporting general-purpose vector CPU processing; 2) the state-of-the-art (SOTA) energy efficiency has been achieved on the CPU mode for general-purpose processing in end-to-end ML workload by exploiting the simpler pipeline, reduction of memory access and high data locality of CIM architecture; and 3) special dataflow and dedicated instruction sets are developed for seamless data sharing and smooth mode switching between CPU and DNN operations to overcome the inter-core data movement overhead in conventional ML SoC rendering significant improvement on end-to-end performance. A 65 nm test chip has been fabricated to demonstrate the benefits of the proposed GPCIM in end-to-end ML tasks.

The rest of the article is organized as follows. The overview of the top-level architecture of GPCIM and mode

reconfigurations are introduced in Section II. Section III illustrates the details of the CIM technique supporting CPU operations with efficiency improvement. Design details are provided for the central compute unit (CCU) core and the customized CIM vector instruction set. Section IV introduces the special dataflow to avoid expensive data movement. The implementation and measurement results obtained by the test chip are shown in Section V along with demonstrations on end-to-end ML use cases. Section VI draws the conclusion. This article is a detailed extension of the conference publication in [27].

II. Top-Level Architecture of GPCIM With Two Reconfiguration Modes

A. GPCIM Top-Level Architecture

Fig. 3 shows the chip top architecture. Four CIM cores are designed for parallel processing with independent instruction cache and weight SRAM for each core supporting both DNN inference and general-purpose computing. Each GPCIM core has special dataflow controllers to cooperate with CIM macro for different computing processes. Differing from traditional CIM macro [10], [11], [12], [13], [14], [15], [16], [17], one GPCIM macro contains two different bitcell arrays, i.e., data cache activation memory (DAMEM) and data cache output memory (DOMEM). Four reconfigurable CCUs are implemented between DAMEM and DOMEM in GPCIM macro to handle four 32b results accumulation in the DNN process and four 32b vector executions for general-purpose computing. Two pulse generators are attached bitcell arrays to control the read, write, and ML inference. Sense amplifiers, latches, and buffers for two bitcell arrays support the data read/write.

Each bank of DAMEM has 32 × 32 bitcells with a 9-transistor (9T) architecture for each bitcell. As the details shown in Fig. 4(a), a 3-transistor (3T) nand gate is inserted into a traditional 6-transistor (6T) SRAM bitcell architecture for 1b multiplication in each DAMEM bitcell for DNN operations. The multiplication result of each DAMEM bitcell can directly reach the CCU for accumulation through independent DNN read-out (DOUT) lines instead of reading through the bitline. Each bitcell has its own DOUT line for maximum throughput support. The DOMEM bitcell architecture is also presented in Fig. 4 (b). It is an 8-transistor (8T) bitcell with two bitlines and two wordlines supporting dual-port read/write. DOMEM performs two reads and one write within one clock cycle controlled by the CIM controller and the pulse generator. The DOUT line connection details and its routing layout are shown in Fig. 3. 32 DOUT lines are routed on the top layers of the bitcell layout using metal 4 and metal 6 of 65nm technology. This routing solution fully utilizes the bitcell width to avoid the area overhead of the DAMEM.

B. Reconfiguration Modes of GPCIM

Fig. 5 shows that GPCIM can be configured to two different CIM modes. Vector CPU mode supports parallel general-purpose workload in end-to-end ML applications while DNN mode focuses on the MAC operations of DNN inference. Both modes utilize CIM techniques for high energy efficiency. In DNN mode, DAMEM is configured as input memory which can realize the multiplication of inputs and weights inside of the bitcell array with the results sent to the CCU for accumulation. DOMEM is used for output memory to store the MAC results. In the vector CPU mode, DOMAM and DAMEM are reused as the data cache and register file (RF) providing the data for vector-based execution. As for the CCU used for DNN accumulation and CPU execution, the CCU can be reconfigured to four adder trees in DNN mode and four ALUs as near-bitcell execution units for CPU mode with 8.8% total overhead by logic reuse. Unused logic and SRAM banks are gated for power saving in both modes.

C. Control Sequence of GPCIM Vector CPU Mode

Control flow in the CPU mode is demonstrated in Fig. 6 including five operation phases in a single cycle, i.e., write-back, pre/dis-charge, latch update, and vector execution. The phases are determined by the control signals generated by the pulse generator which is designed using a customized tunable delay buffer chain. The write-back phase is for data writing from the previous cycle. Executable data reading is done by pre/discharging the bitlines of two bitcell arrays. In the latch update phase, the sense amplifier and latch receive the data from the bitcell arrays and send it to the CCU for vector-based execution. This control flow can also be used for reading the partial sum results from DOMEM for accumulation under DNN operation.

III. Detailed Implementation of Reconfiguration and Central Compute Units of GPCIM

A. Efficiency Improvement of Vector CPU

As mentioned in the previous sections, supporting vector general-purpose processing by CIM techniques brings significant benefits to the energy efficiency of end-to-end application processing. Fig. 7 delivers a detailed explanation of the power reduction of GPCIM. To perform a fair apple-to-apple comparison, a customized digital RISC-V vector pipeline core is also built as an equivalent counterpart with the same number of vector lanes, maximum vector length (MVL), RF size, and Level 1 (L1) cache size. The digital counterpart runs at the same clock speed as GPCIM to show the power reduction breakdown in Fig. 7(a). Fig. 7(b) shows the detailed microarchitecture of the digital RISC-V counterpart with a 5-stage pipeline. Compared with the counterpart, the shorter 2-stage pipeline of GPCIM leads to reduced flip-flops power by 7.6× which is one of the major power contributors in the digital RISC-V core. Vector RF is eliminated by integrating it into the CIM data cache with similar data read/write functionalities. All the execution units are integrated into CIM macro with lower data access costs than the case of L1 cache in the digital counterpart, achieving 1.9× cache power saving and 1.3× ALU power saving. Certain pipeline logic in the digital counterpart such as forwarding and other bypass logic is also removed with 2.1× logic power saving. In general, 4.6× total power reduction is achieved by GPCIM rendering energy efficiency improvement.

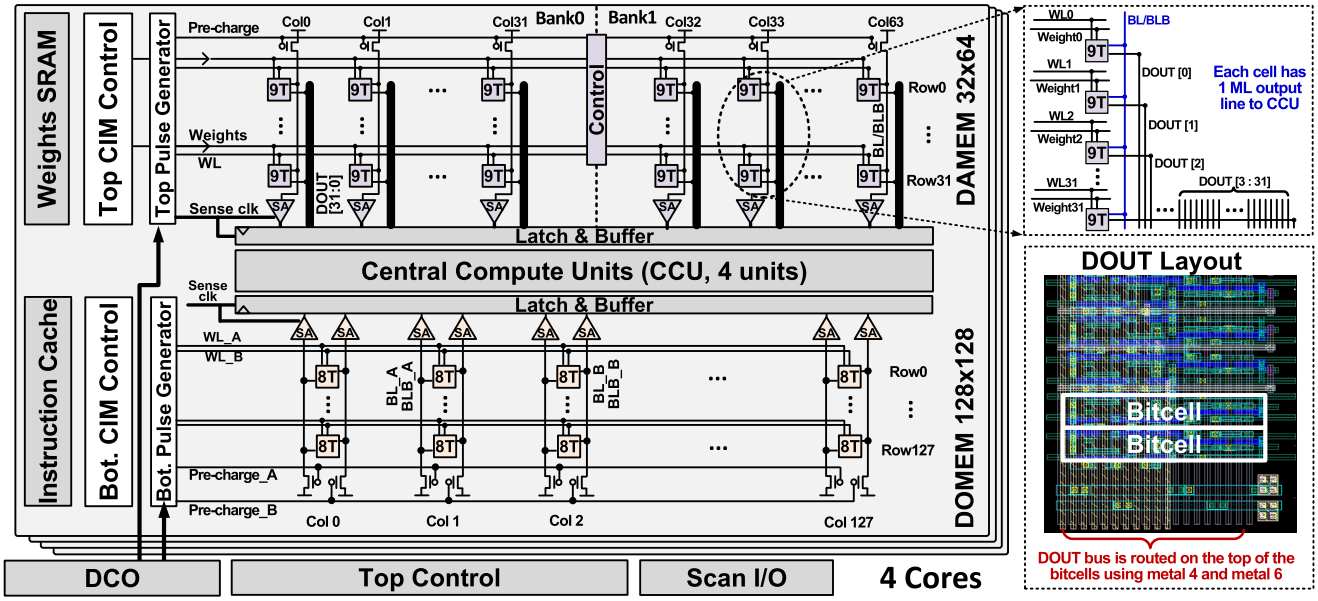


Fig. 3. Top-level architecture of the developed GPCIM.

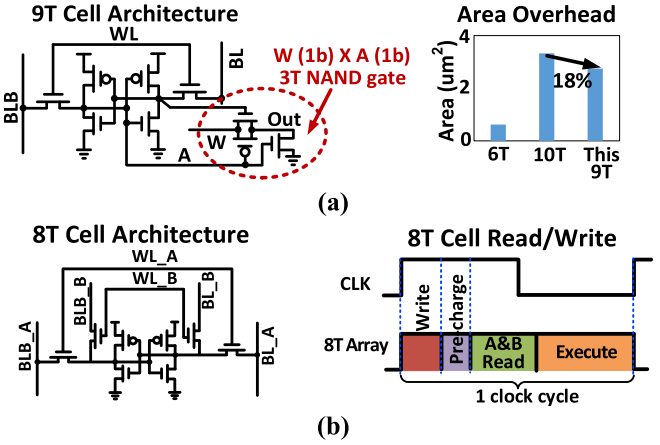


Fig. 4. (a) Detailed design of 9T cell in DAMEM and (b) detailed design of 8T in DOMEM.

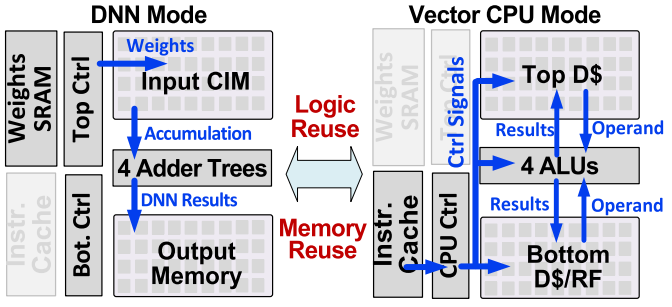


Fig. 5. Dataflows for two reconfiguration modes of GPCIM: DNN mode and vector CPU mode.

B. Logic Reuse of Central Compute Units

In Section II, CCU is introduced as a core processing functional unit for both DNN mode and CPU mode. Fig. 8 shows the details inside the CCU which is designed based on four 32b adder trees with additional vector CPU reconfiguration overhead. Obviously, because CIM techniques have stringent requirements on the area efficiency of the macro,

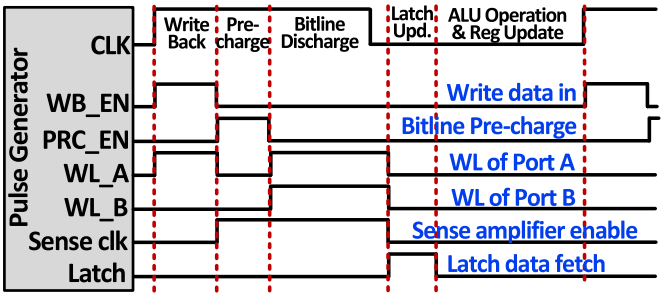


Fig. 6. Control sequence of vector CPU mode of GPCIM.

efficient logic reuse is implemented in the CCU to support both DNN mode and CPU mode. Examples are shown in Fig. 8 for three different CIM CPU instructions: vector multiplication instruction (VMUL), vector Boolean instruction (VBoolean), and vector shift instruction (VShift). For VMUL instruction, the majority of adders are fully reused to build a 32b multiplier to realize 32b multiplication in four cycles. VShift instruction exploits the reuse of shifters and registers which are also used for the partial sum of shift accumulation in the DNN mode. Certain overhead is paid for complete general-purpose processing such as small Boolean function modules for the VBoolean instruction. Unused logics in different modes are gated by the clock gating functions. As shown in Fig. 9, like [11], a customized 16T full adder design is integrated into the CCU with 28% power reduction and 32% area reduction compared with a conventional 28T full adder.

Thanks to the efforts of logic reuse of CCU, compared with a SOTA baseline CIM design which contains four 32b adder trees, GPCIM macro observes a total 8.8% area overhead to realize vector general-purpose computing, which is shown in Fig. 10. In addition, the impact of power consumption overhead for DNN processing caused by the additional logic is 5.3%.

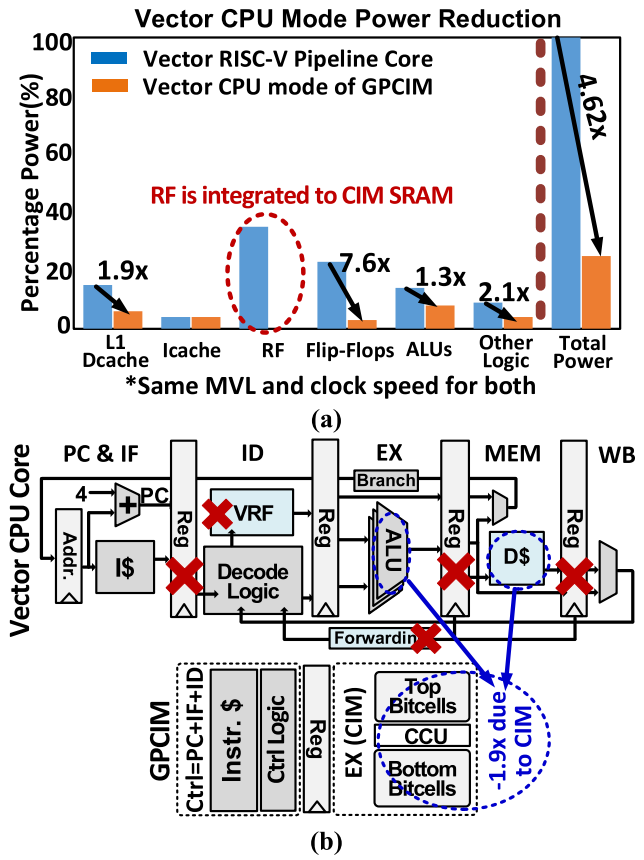


Fig. 7. (a) Power saving of GPCIM compared with a reference RISC-V design. (b) Detailed explanations of power reduction.

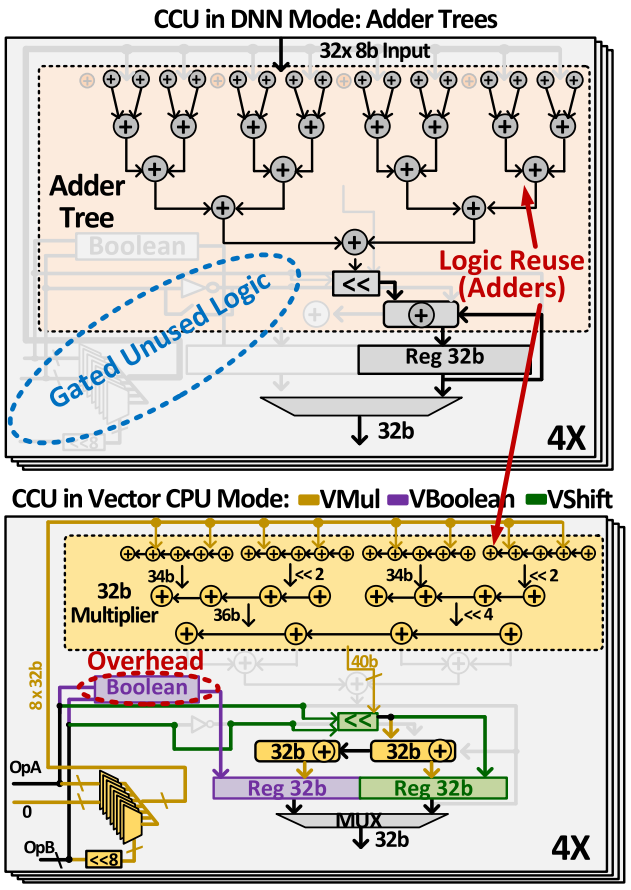


Fig. 8. Examples of logic reuse of CCU in vector CPU mode of GPCIM.

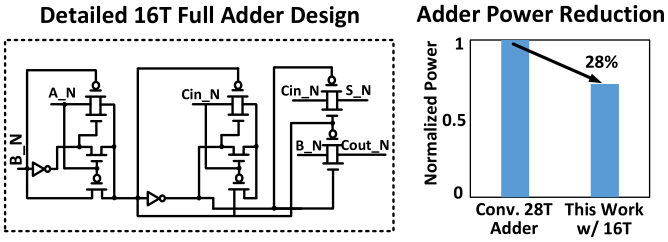


Fig. 9. Customized adder design in CCU and power saving.

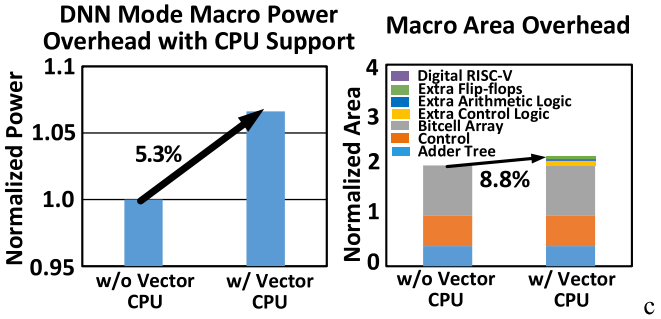


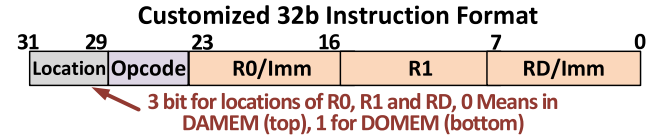
Fig. 10. GPCIM DNN processing power and area overhead for CPU reconfiguration support.

C. Customized CIM Vector Instruction Set

In order to support the special GPCIM-based vector execution, a customized 32b integer instruction set (ISA) architecture is developed and cooperated with the GPCIM architecture. As introduced in Fig. 11, the 32b GPCIM instruction format contains three special bits as the location pointers. The first two bits are used to indicate the source locations of two operands. The third bit is for the destination of the execution result from CCU. In the location pointer, 0 represents DAMEM and 1 represents DOMEM. Examples of the GPCIM instructions are listed in Fig. 11(b), which contains Boolean instructions, arithmetic instructions, shift instructions, data move instructions, branch instructions, and instructions for special functions of GPCIM. The first CCU unit and first 32 columns of the CIM array will be used as a scalar unit to provide scalar value if the instruction is needed. MVCSR instruction is designed to configure special control and status registers (CSRs) in GPCIM. SWITCH instruction is used to assist a smooth mode switching from CPU mode to DNN mode. PCS is used to store the program counter (PC) and pipeline status before the GPCIM switches to DNN mode to continue the general-purpose processing after DNN inference.

An example of mode switching from GPCIM CPU mode to DNN mode is illustrated in Fig. 12. Total configuration and switching latency are between 15 and 20 run cycles which includes 4–6 cycles for CSR configuration of the controllers and pulse generators, 6–8 cycles for the parameter set up of DNN processing, up to 4 cycles for the CSR configuration in the CCU and the execution of SWITCH instructions.

By implementing the customized ISA, GPCIM can achieve vector-based general-purpose processing. Fig. 13 demonstrates the power benefits across several instructions compared with the function-similar instructions running in the RISC-V digital



(a)

Type	Opcode	R0	R1	RD	# Cycles
Vector Boolean	VAND, VOR, VXOR, VNAND, VNOR, VXNOR	Vec/Sca/Imm	✓	✓	1
Vector Arithmetic	VADD	Vec/Sca/Imm	✓	✓	1
	VSUB	Vec/Sca/Imm	✓	✓	1
	VRSUB	Vec/Sca/Imm	✓	✓	1
	VOP	Vec/Sca	✓	✓	4
	VMIN/VMINU/ VMAX/VMAXU	Vec/Sca/Imm	✓	✓	1
	VZEXT/VSEXT		✓	✓	1
	VMUL	Vec/Sca/Imm	✓	✓	4
	VMULH	Vec/Sca/Imm	✓	✓	4
	VMERGE	Vec/Sca/Imm	✓	✓	2
Vector Shift	VSLL	Vec/Sca/Imm	✓	✓	1
	VSRL,VSRA	Vec/Sca/Imm	✓	✓	1
Vector Move	VMV	Vec/Sca		✓	1
	VMVI	Imm		✓	1
VCompare	>, <, =	Vec/Sca/Imm	✓	✓	1
Jump	JMP	Scalar		✓	1
Branch	>, <, =	Scalar	✓	Imm	1
Special Functions	MVCSR	Scalar		CSR	1
	SWITCH				2
	PCS	Imm		✓	1

* # Cycles is based on 32b operations
Vec/Sca/Imm: R0 can be in Vector RF, or Scalar RF or an immediate value
PCS: Stop and hold PC for DNN mode
MVCSR: Move data to control and status registers
SWITCH: Switch mode from vector CPU mode to DNN mode

(b)

Fig. 11. (a) 32b customized CIM vector instruction standard format. (b) Main instructions in the customized instruction set.

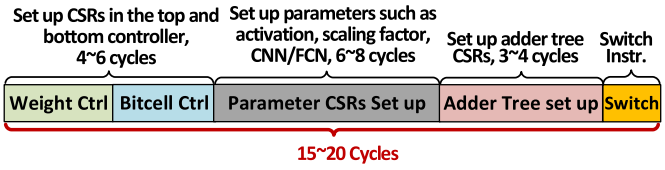


Fig. 12. Switching and configuration control from CPU mode to DNN mode.

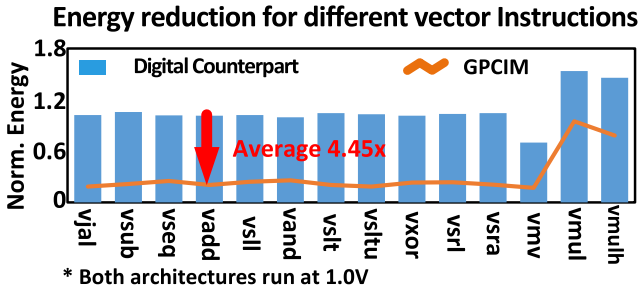


Fig. 13. Energy reduction from different GPCIM instructions.

counterpart mentioned in Section III-A. GPCIM achieves an averaged 4.45× power reduction for the sampled vector instructions.

IV. Special Dataflow for End-to-End ML Tasks With Data Locality

A. Special Dataflow With Smooth Mode Switching

Because of the reconfigurable macro architecture of GPCIM, a special data movement scheme is also developed to reduce the data movement as in the traditional heterogeneous SoC for end-to-end operations. The detailed dataflow is shown in Fig. 14. In most cases, GPCIM starts from the CPU mode performing crucial general-purpose processing work as preprocessing and data preparation for the DNN inference. Especially for some edge device applications [19], [22], the CPU is needed to deal with pre-processing workloads for the raw data from the camera, LiDAR, sensors, and so on. After the general-purpose computing of GPCIM in CPU mode, the input data for DNN inference is stored at DAMEM which serves as the data cache in CPU mode but an input CIM bitcell array for DNN mode. An instruction-controlled mode switching is launched after that which means the GPCIM in DNN mode can directly process the input data stored in the DAMEM for DNN inference, e.g., the first layer of a DNN model, without any data transfer as in the traditional architecture. After finishing the first layer of DNN inference, the GPCIM stores the results in DOMEM which is the output memory. Then the GPCIM switches back to CPU mode and directly uses the DNN results to perform general-purpose processing work, such as data alignment, batch normalization, and padding to prepare the data for next-layer DNN inference. Because DOMEM is also one of the data caches in GPCIM, expensive data movement can be eliminated. In addition, the vector CPU mode of GPCIM can perform post-processing work seamlessly after DNN inference. The data locality for DAMEM and DOMEM helps GPCIM achieve end-to-end performance improvement by avoiding data movement between two cores.

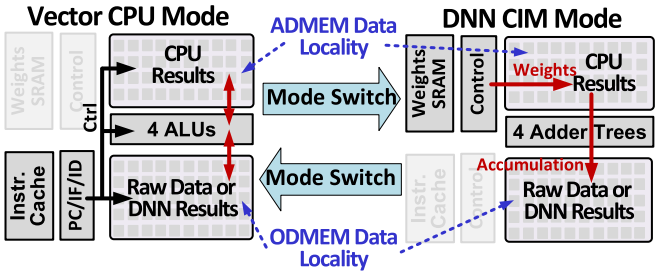


Fig. 14. Special dataflow in GPCIM for end-to-end processing with data locality.

B. Examples of Image Classification With CPU Workload

Fig. 15(a) shows examples of image classification tasks considering the CPU workload and data transfer. In these examples, the CPU is focusing on the inter-layer data preparation works, such as batch normalization, pooling, and data alignment. In the image classification tasks, GPCIM achieves 52%–56% end-to-end latency improvement for VGG16 and ResNet18 models on ImageNet and Cifar10 datasets compared with a representative heterogeneous SoC, i.e., Gemmini [21], which contains a RISC-V CPU pipeline core and an ML accelerator. The latency benefits are due to the elimination of data transfer and parallel computing acceleration on CPU mode for the inter-layer data management work. Fig. 15(b) shows the DNN inference accuracy on Cifar10 and ImageNet datasets with 8b integer quantization. Both VGG16 and ResNet18 models can achieve similar accuracy compared with prior 8b works [6], [7], [8].

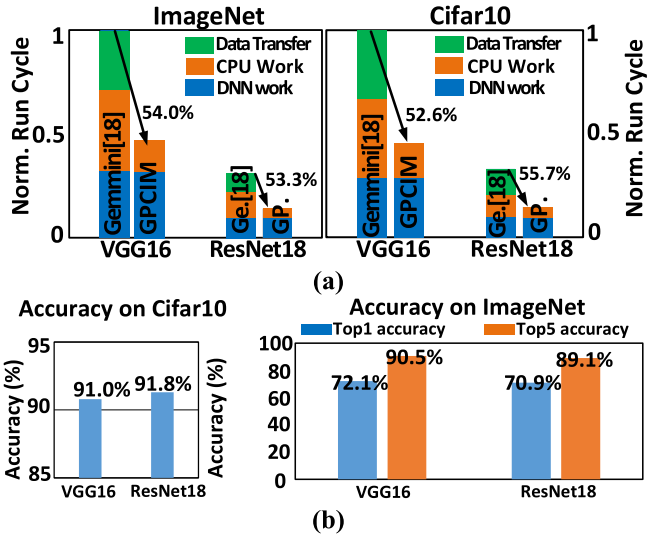
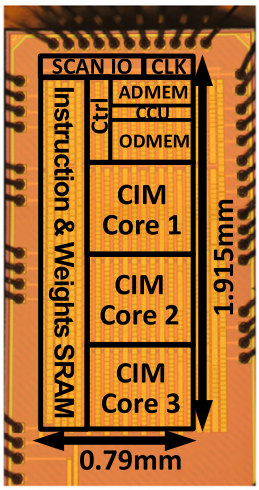


Fig. 15. Eng-to-end performance on image classification tasks compared with heterogeneous SoC Gemmini [21].



Technology	65nm CMOS
Chip Area	1.52mm²
Total Macro Size	9KB
Macro Area	0.24mm²
Total SRAM	41KB
Bit Precision	INT8, INT32
Max Freq.	245MHz
Supply Vdd	0.6 ~ 1V
CPU Power	4.89mW
DNN Power	8.4mW
CPU Perf. (GOPS)	3.92 (1V, 32b)
CPU Effi. (GOPS/W)	802 (1V, 32b)
DNN SoC Effi. (TOPS/W)	7.62 (8b, 1V) 16.5 (8b, 0.6V)
DNN Macro Effi. (TOPS/W)	14.8 (8b, 1V) 28.3 (8b, 0.6V)

Fig. 16. Chip micrograph and specifications.

V. Measurement Results and Evaluation for Test Case

A. Chip Implementation

A 4-core GPCIM processor was designed and fabricated using a 65 nm CMOS process. The chip micrograph and implementation details are shown in Fig. 16. The active die area is 1.52 mm^2 (0.79 mm $\times$ 1.915 mm) with 1.0 V nominal supply voltage and 245 MHz maximum operating frequency. Total CIM macro size is 9 KB and total SRAM size is 41 KB including the instruction caches and the weight SRAM banks. The chip can support 8b integer bit precision for DNN mode and 32b customized vector integer ISA for the CPU mode. The chip was tested with the supply voltage scaled down to 0.5 V. An FPGA board is used in the chip testing for data streaming in and out of the test chip through scan IO ports for verification and measurement.

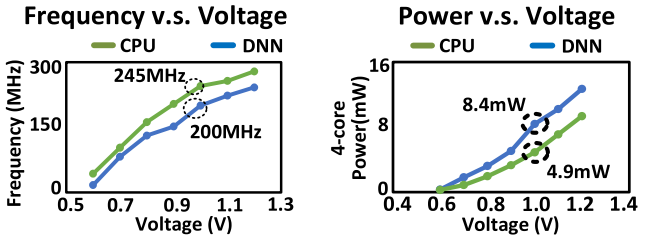


Fig. 17. Measured frequency and power with voltage scaling.

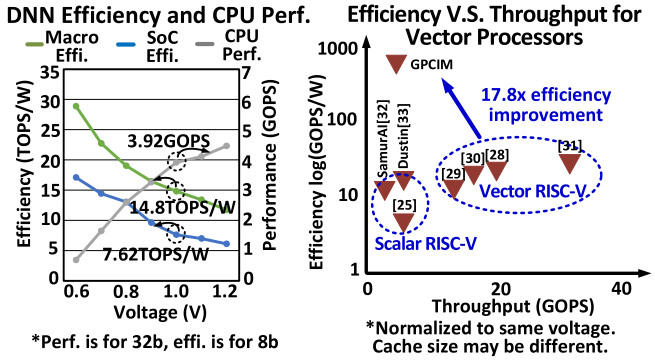


Fig. 18. Measured energy efficiency of DNN in GPCIM with voltage scaling. Throughput and efficiency of CPU in GPCIM in comparison with existing RISC-V SoCs.

B. Measurement Results

Fig. 17 shows the measured power and frequency with the supply voltage scaled down to 0.5 V. The nominal supply voltage for both DNN modes and CPU modes is at 1.0 V with 8.4 mW of DNN power at 200 MHz and 4.9 mW of CPU power at 245 MHz nominal frequency for four cores.

As for the DNN mode, Fig. 18 shows the energy efficiency for both the macro and the SoC with the supply voltage down to 0.5 V. The results are based on the INT8 bit precision. The macro-level energy efficiency for GPCIM in DNN mode is 14.8 TOPS/W at 1.0 V and 28.3 TOP/W at 0.6 V. The SoC-level energy efficiency for DNN operation achieves 7.62 TOPS/W at 1.0 V and 16.5 TOPS/W at 0.6 V. On the other hand, as for the CPU mode of GPCIM, the throughput is 3.92 GOPS at 1.0 V and 0.69 GOPS at 0.6 V with 32b integer precision. The peak energy efficiency for the CPU mode of GPCIM is 802 GOPS/W at 1.0 V and 1.75 TOPS/W at 0.6 V.

Fig. 18 also illustrates a comparison of GPCIM's CPU efficiency and throughput with existing published CPU SoCs with scalar and vector RISC-V processors [25], [28], [29], [30], [31], [32], [33]. The reported throughput and efficiency from existing SoCs are normalized to the same voltage as GPCIM. The throughput of GPCIM is less than many prior solutions due to the lower clock speed of GPCIM to support memory operations. However, GPCIM achieves a significant 17.8$\times$ energy efficiency improvement for general-purpose computing over existing RISC-V vector processor SoCs [28], [29], [30], [31].

In Fig. 19, six vectorized benchmarks are evaluated on GPCIM and the digital counterparts of RISC-V mentioned in Section III-A with the same SRAM size, MVL, and clock speed as GPCIM for fair comparison. Canneal, Stream Cluster, and the Black Scholes benchmark are from the

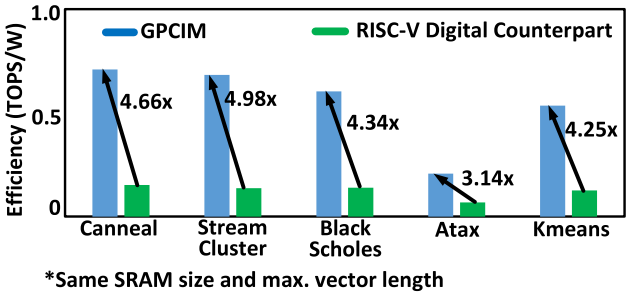


Fig. 19. Efficiency comparison with a digital RISC-V on different benchmarks [34], [35], [36].

PARVEC benchmark suite [34]. Canneal is an element-based engineering computing benchmark with data parallelism. Stream Cluster is also a benchmark that can be used for vector processors for the data mining application. Black Scholes is a benchmark from financial analysis applications. GPCIM achieves 4.66×, 4.98×, and 4.34× energy efficiency improvement over three benchmarks from PARVEC. One matrix operation-related vector benchmark is also evaluated on GPCIM and the digital counterpart, i.e., Atax from PolyBench [35]. GPCIM achieves 3.14× energy efficiency improvement. Because of the heavy workload of vector multiplication in Atax, the efficiency for both GPCIM and digital counterpart are not as high as other benchmarks. Another data mining benchmark is Kmeans from Rodinia [36] in which GPCIM can be 4.25× more energy efficient than the RISC-V digital counterpart.

Since the digital counterpart is not an open-source design and the processors in Fig. 18 have large systems supporting more complicated general-purpose computing. To make a fair comparison, GPCIM is compared with compact vector processors that support the RISC-V vector extension (RVV) in Table I. As Table I shows, GPCIM can achieve best-in-class CPU energy efficiency with an appropriate equivalent RVV subset support compared with Spatz [37], Arrow [38], and Vicuna [39] thanks to the CIM technique. The efficiency of Spatz [37] for MAC operations is close to the CPU mode of GPCIM because of an integrated group of MAC units (MACUs) in its vector ALU, which makes it similar to a small digital accelerator with higher efficiency.

C. Comparison With Prior Works

Table II illustrates a comparison table with prior tape-out works. Compared with digital CIM for DNN [10], [11], the GPCIM achieves similar SOTA energy efficiency on DNN inference. In CPU mode, GPCIM achieves the best-in-class energy efficiency of 802 GOPS/W. Compared with a prior instruction-supported CIM [16], GPCIM achieves a 7.3× efficiency improvement for the multiplication instruction. For apple-to-apple comparison, the efficiency numbers in [16] are scaled to 32b in the table. Compared with a digital reconfigurable design supporting DNN and CPU [25], GPCIM achieves 10× higher DNN efficiency and 118× higher CPU efficiency. Compared with digital heterogeneous SoC supporting DNN and CPU [29] [32], GPCIM achieves 19× higher DNN efficiency and 45× higher CPU efficiency.

TABLE I
COMPARISON TABLE WITH COMPACT VECTOR PROCESSORS

	Spatz [37]	Arrow [38]	Vicuna[39]	This Work
Platform	Backend Results	FPGA	FPGA	Tape-out Chip
Design Method	Digital	Digital	Digital	CIM
Area	20.1 mm^2	-	-	1.52 mm^2
Architecture	Vector Processor + MACUs	Vector Processor	Vector Processor	Vector Processor + Accelerator
ISA	RVV 1.0	RVV 1.0	RVV 1.0	Customized ISA
% of Full Integer RVV Support	58%	72%	100%	**78%
CPU Power	1.07W	0.297W	*0.2W	4.89mW
Max. Frequency	594MHz	112MHz	80MHz	245MHz
CPU Performance (GOPS)	***285 (32b)	*5.4 (32b)	10 (32b)	Add: 3.92 (1V, 32b) Mul: 0.98 (1V, 32b)
CPU Efficiency (GOPS/W)	***266 (32b)	*18 (32b)	*50 (32b)	Add: 802 (1V, 32b) Mul: 144.7 (1V, 32b)

*Estimation results and scale to 32b vector operations
**Equivalent results since this work uses customized ISA
***Values are for MAC operations

D. Evaluation on Test Cases

To demonstrate the benefit of the developed GPCIM, a detailed end-to-end case study is implemented on the chip for simultaneous localization and mapping (SLAM). In SLAM, a mobile agent such as a robot or a self-driving car constructs or updates a map of an unknown environment while simultaneously keeping track of its own position in the environment. For SLAM, CNN-SLAM [40] is one of the commonly used algorithms for DNN-based SLAM solutions on edge devices.

Fig. 20(a) illustrates the detailed sub-tasks in the CNN-SLAM algorithm. Except for DNN processing, there are four major tasks that contribute over 76% of the workload: camera pose estimation (12%), depth refinement (31%), key-frame creation (15%), and pose graph optimization (18%). The four tasks have complicated operations, such as vector division and exponentiation that require a CPU to handle. In addition, since DNN inference for depth prediction only happens on key frames, parallel general-purpose computing is highly demanded for preprocessing the raw data from the camera. CNN-SLAM algorithm is implemented on both GPCIM and a heterogeneous SoC Gemmini [21] for evaluation and comparison. As shown in the detailed processing flow in Fig. 20(b), the accelerator utilization in the traditional architecture is fairly low for the CNN-SLAM process because the accelerator needs to wait for the RISC-V core to finish the CPU workload and launch the data movement. GPCIM can eliminate expensive data transfer efforts to speed up the processing of CNN-SLAM algorithms. In addition, the CPU core in Gemmini is a single scalar pipeline core. GPCIM, on the other hand, provides more computing power with parallel processing to improve the latency of the frame-based data processing.

Fig. 21(a) shows the energy efficiency comparison for different sub-tasks between GPCIM and Gemmini. As for the general-purpose workload, the GPCIM achieves 7.7× efficiency improvement on camera pose estimation, 8.0× efficiency improvement on depth refinement, 7.6× efficiency improvement on the key frame creation, and 8.2× efficiency improvement on pose graph optimization. The energy efficiency for DNN inference of GPCIM is 35× higher than

TABLE II
COMPARISON TABLE WITH PRIOR WORKS

	ISSCC'22 [10]	ISSCC'21 [11]	ISSCC'19 [16]	ESSCIRC [29]	SamurAI [32]	ISSCC'22 [25]	This Work
Process	28nm	22nm	28nm	22nm	28nm	65nm	65nm
Area (mm2)	6.69	0.202 (macro)	2.55	16	4.5	4.47	1.52
Architecture	DNN	DNN	CPU	CPU+DNN	CPU+DNN	CPU+DNN	CPU+DNN
Type	Digital CIM	Digital CIM	Digital CIM	Digital	Digital	Digital	Digital CIM
CPU Power	-	-	-	-	24.5mW (1x)	589mW@1V 10 cores	4.89mW@1V 4 cores
DNN Power	69.4mW@1V	35mW@0.72V	-	-	96mW (total)	116mW@1V	8.4mW@1V
Bit Precision	INT8, INT16	INT4,INT8	Arbitrary	INT8, 64, FP64	INT8, 16, 32	INT8, INT32	INT8, INT32
Max. Frequency	220MHz	-	475MHz	961MHz	350MHz	400MHz	245MHz
Norm. Supply Vdd	1.0V	0.72V	1.1V	0.85V	0.9V	1.0V	1.0V
Macro Size	12KB	8KB	16KB	-	-	-	9KB
DNN SoC Efficiency (TOPS/W)	19.5 (1V, 8b)	-	-	0.11 (0.85V, 8b)	0.38 (0.9V, 8b)	0.66 (1V, 8b)	7.62 (1V, 8b)
DNN Macro Efficiency (TOPS/W)	30.8 (1V, 8b)	24.7 (0.72V, 8b)	Add: 2.2 (1V, 8b) Mul: 0.22 (1V, 8b)	-	-	-	14.8 (1V, 8b) 23.5 (0.7V, 8b)
CPU Performance (GOPS)	-	-	Add:27.6 (1V, 32b) Mul:0.75 (1V, 32b)	-	1.5 (1V, 32b)	Add: 4 (1V, 32b) Mul: 1 (1V, 32b)	Add: 3.92 (1V, 32b) Mul: 0.98 (1V, 32b)
CPU Efficiency (GOPS/W)	-	-	*Add:550 (1V, 32b) *Mul:19.7 (1V, 32b)	9.0 (0.85V, 64b)	7.44 (1V, 32b)	Add: 6.79 (1V, 32b) Mul: 2.35 (1V, 32b)	Add: 802 (1V,32b) Mul: 144.7 (1V, 32b)

*Scale up to 32b. Original data for 8b operation: 2.2TOPS/W (Add) and 220GOPS/W (Mul)

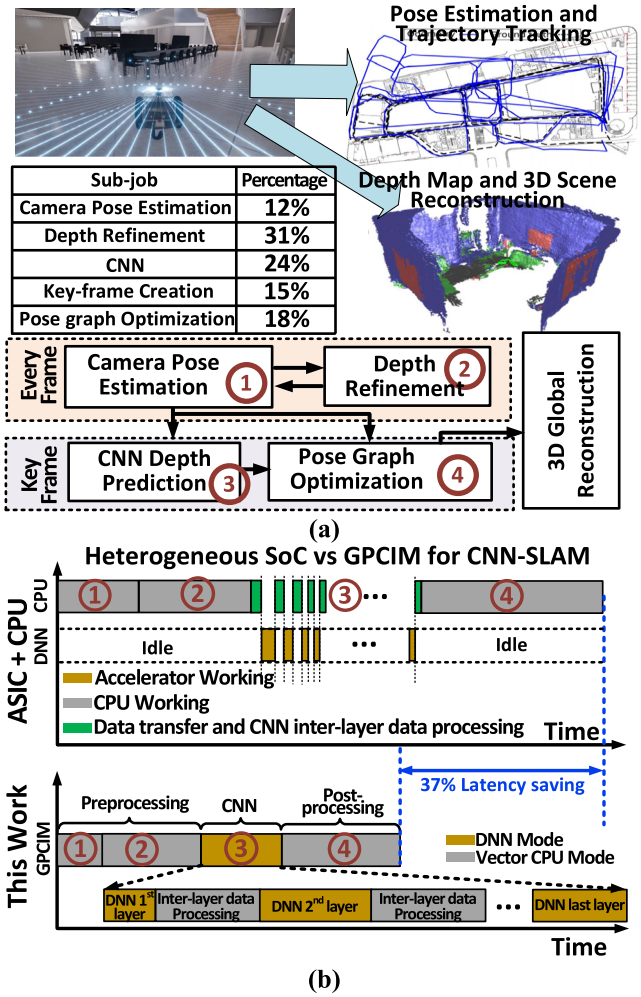


Fig. 20. (a) CNN-SLAM application [40]. (b) Difference in processing sequence between conventional SoC and GPCIM.

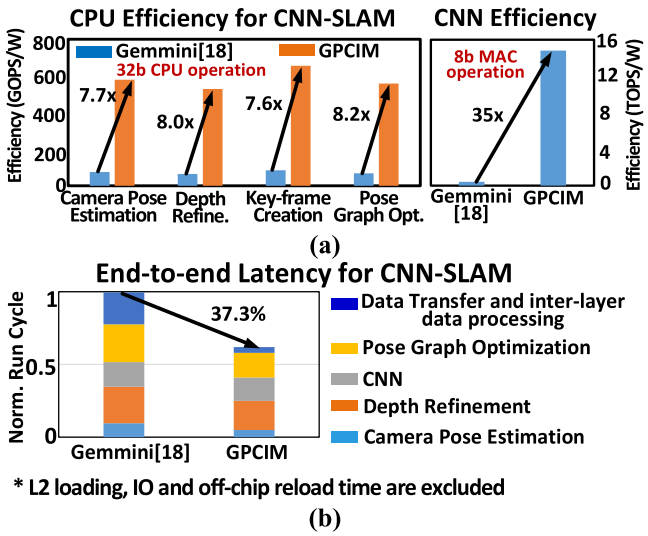


Fig. 21. (a) Efficiency and (b) run cycle comparison between Gemmini [21] and GPCIM when running CNN-SLAM.

Gemmini thanks to the CIM techniques. Note that Gemmini contains a larger SRAM size for both the CPU core and accelerator core than GPCIM in this work. Fig. 21(b) shows the end-to-end latency improvement of GPCIM for the CNN-SLAM algorithm. GPCIM achieves an overall 37.3% run cycle reduction than Gemmini due to vector-based parallel processing and reduction of data transfer efforts.

Another demonstration is built to showcase the benefit of GPCIM in human motion detection as shown in Fig. 22(a). A binary neural network (BNN) model is trained to perform the hand gesture classification by using the recorded surface electromyography (sEMG) signal and accelerometer sensors based on the Ninapro database [41]. 87.9% of the computing task comes from the heavy feature extraction workload being performed by the CPU as shown in Fig. 22(b). The feature extraction jobs include the extraction of four time-domain features from six channels of the sEMG signals including mean, histogram, variance, and slope sign change [42].

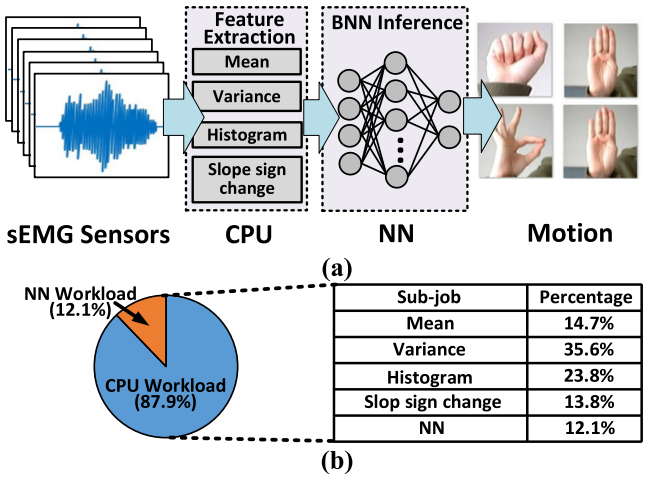


Fig. 22. (a) Human motion detection with NN-based edge computing. (b) Workload breakdown for the end-to-end motion detection task.

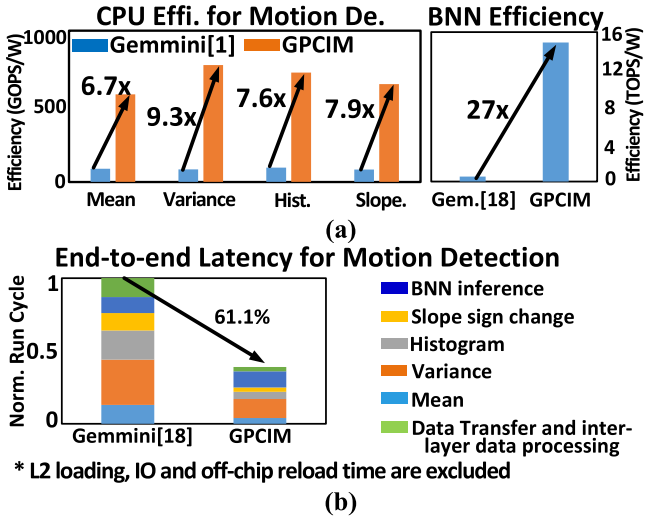


Fig. 23. (a) Efficiency and (b) run cycle comparison between Gemmini [21] and GPCIM when running motion detection test case.

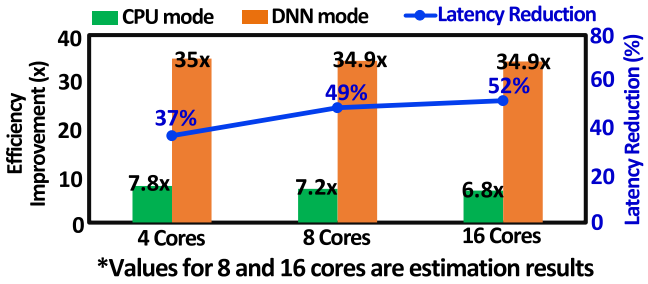


Fig. 24. GPCIM scalability estimation with 4, 8, and 16 cores running the SLAM testcase compared with Gemmini [21] from Fig. 20.

Compared with the efficiency of Gemmini CPU in Fig. 23(a), GPCIM achieves 6.7×, 9.3×, 7.6×, and 7.9× improvements over four feature extraction operations for each channel. The energy efficiency of GPCIM BNN inference is improved by 27× compared with the accelerator in Gemmini due to the use of the CIM technique. Since the majority of CPU workload for the motion detection is channel independent and has less data dependency than the preprocessing in the CNN-SLAM case. Compared to the SLAM case, the motion detection test case can fit better on the vector processor with parallel computing benefits. The GPCIM can achieve a 61.1% latency reduction compared with Gemmini mainly due to the vector CPU's parallelism. The latency benefit from data transfer is smaller in this case because the BNN model is relatively small with only four fully connected layers, requiring less data transferring than the DNN operation in the previous test case. Although the motion detection test case has a more vectorized CPU workload than SLAM, its preprocessing workload cannot be fully vectorized. 45%–55% of the CPU workload still has data dependency which means using Gemmini as a reference is still reasonable since this is the best complete open-source design of CPU + accelerator architecture we can find for the analysis of latency breakdown in the VLSI design flow running different programs and test cases.

E. GPCIM Scalability Discussion

To illustrate the scalability of GPCIM, a design-level experiment with estimations is shown in Fig. 24 with 4, 8, and 16 cores of GPCIM. Since the multi-core system requires some inter-core connections, a more complicated digital system bus is designed for 8 and 16 GPCIM cores which consumes extra power, especially in CPU mode. In that case, the CPU efficiency benefit drops from 7.8× to 6.8× compared with the fixed Gemmini [21] design which is used in SLAM. The DNN efficiency benefit is stable at ∼35× because this bus is not frequently used during DNN inference which can be gated. When the system bus is used in DNN mode, some arbiter functions can also be gated since the data loading in DNN inference is uniform and predictable which leads to less external power consumption over 4 cores. Considering the data dependency of the CPU workload which cannot be efficiently accelerated by adding more GPCIM cores, the latency reduction for 8 and 16 cores is only increased to 49% and 52% compared with the Gemmini [21] used in Fig. 20.

VI. Conclusion

This work proposes a unified GPCIM processor supporting both vector general-purpose computing and DNN operations. GPCIM macro contains two bitcell arrays (DAMEM and DOMEM) and CCU which can support smooth mode switching. Cooperated with special customized ISA, logic reuse, and special dataflow, GPCIM realizes high data locality for reduction of data transfer and 37%–55% end-to-end performance improvement for different test cases. Thanks to the CIM techniques with vector CPU support, GPCIM achieves 7.6–17.8× energy efficiency improvement compared with existing ML SoCs. The 65 nm test chip shows a maximum of 28.3 TOPS/W for DNN macro efficiency, 16.5 TOPS/W for DNN SoC efficiency, and a best-in-class peak CPU efficiency of 802 GOPS/W.

References

[1] Y. Li, S. Xu, X. Cao, X. Sun, and B. Zhang, "Q-DM: An efficient low-bit quantized diffusion model," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, New Orleans, LO, USA, Oct. 2023, pp. 76680–76691.

[2] K. Ueyoshi et al., "QUEST: A 7.49 TOPS multi-purpose log-quantized DNN inference engine stacked on 96 MB 3D SRAM using inductive-coupling technology in 40 nm CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 216–218.

[3] R. Guo et al., "A 5.99-to-691.1 TOPS/W tensor-train in-memory-computing processor using bit-level-sparsity-based optimization and variable-precision quantization," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 242–244.

[4] B. Keller et al., "A 17–95.6 TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI Technol. Circuits)*, Jun. 2022, pp. 16–17.

[5] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, Jan. 2019.

[6] T. Jia, Y. Ju, and J. Gu, "A compute-adaptive elastic clock-chain technique with dynamic timing enhancement for 2D PE-array-based accelerators," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2020, pp. 482–484.

[7] S. Kim, S. Kim, S. Hong, S. Kim, D. Han, and H.-J. Yoo, "C-DNN: A 24.5–85.8 TOPS/W complementary-deep-neural-network processor with heterogeneous CNN/SNN core architecture and forward-gradient-based sparsity generation," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 334–336.

[8] Y. Wang et al., "A 28 nm 27.5 TOPS/W approximate-computing-based transformer processor with asymptotic sparsity speculating and out-of-order computing," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[9] C. Kim, S. Kang, D. Shin, S. Choi, Y. Kim, and H.-J. Yoo, "A 2.1TFLOPS/W mobile deep RL accelerator with transposable PE array and experience compression," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 136–138.

[10] F. Tu et al., "A 28 nm 29.2 TFLOPS/W BF16 and 36.5 TOPS/W INT8 reconfigurable digital CIM processor with unified FP/INT pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[11] Y.-D. Chih et al., "An 89TOPS/W and 16.3 TOPS/mm^2 all-digital SRAM-based full-precision compute-in memory macro in 22 nm for machine-learning edge applications," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 252–254.

[12] H. Mori et al., "A 4 nm 6163-TOPS/W/b 4790-TOPS/mm^2/b SRAM based digital-computing-in-memory macro supporting bit-width flexibility and simultaneous MAC and weight update," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 132–134.

[13] P.-C. Wu et al., "A 22 nm 832 Kb hybrid-domain floating-point SRAM in-memory-compute macro with 16.2–70.2 TFLOPS/W for high-accuracy AI-edge devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 126–128.

[14] B. Zhang et al., "PIMCA: A programmable in-memory computing accelerator for energy-efficient DNN inference," *IEEE J. Solid-State Circuits*, vol. 58, no. 5, pp. 1436–1449, May 2023.

[15] X. Zhang and A. Basu, "A 915–1220 TOPS/W, 976–1301 GOPS hybrid in-memory computing based always-on image processing for neuromorphic vision sensors," *IEEE J. Solid-State Circuits*, vol. 58, no. 3, pp. 589–599, Mar. 2023.

[16] J. Wang et al., "A compute SRAM with bit-serial integer/floating-point operations for programmable in-memory vector acceleration," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 224–226.

[17] Z. Chen, X. Chen, and J. Gu, "A 65 nm 3T dynamic analog RAM-based computing-in-memory macro and CNN accelerator with retention enhancement, adaptive analog sparsity and 44TOPS/W system energy efficiency," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 240–242.

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

[19] T. Karnik et al., "A cm-scale self-powered intelligent and secure IoT edge mote featuring an ultra-low-power SoC in 14 nm tri-gate CMOS," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 46–48.

[20] R. Eki et al., "A 1/2.3 inch 12.3 Mpixel with on-chip 4.97 TOPS/W CNN processor back-illuminated stacked CMOS image sensor," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 154–156.

[21] H. Genc et al., "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 769–774.

[22] H. E. Sumbul et al., "System-level design and integration of a prototype AR/VR hardware featuring a custom low-power DNN accelerator chip in 7 nm technology for codec avatars," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2022, pp. 01–08.

[23] V. Honkote et al., "A distributed autonomous and collaborative multi-robot system featuring a low-power robot SoC in 22 nm CMOS for integrated battery-powered minibots," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 48–50.

[24] T. Jia, Y. Ju, R. Joseph, and J. Gu, "NCPU: An embedded neural CPU architecture on resource-constrained low power devices for real-time end-to-end performance," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 1097–1109.

[25] Y. Ju and J. Gu, "A 65 nm systolic neural CPU processor for combined deep learning and general-purpose computing with 95% PE utilization, high data locality and enhanced end-to-end performance," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[26] Y. Ju and J. Gu, "A systolic neural CPU processor combining deep learning and general-purpose computing with enhanced data locality and end-to-end performance," *IEEE J. Solid-State Circuits*, vol. 58, no. 1, pp. 216–226, Jan. 2023.

[27] Y. Ju, Y. Wei, X. Chen, and J. Gu, "A general-purpose compute-in-memory processor combining CPU and deep learning with elevated CPU efficiency and enhanced data locality," in *Proc. IEEE Symp. VLSI Technol. Circuits (VLSI)*, Jun. 2023, pp. 1–2.

[28] B. Zimmer et al., "A RISC-V vector processor with simultaneous-switching switched-capacitor DC–DC converters in 28 nm FDSOI," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 930–942, Apr. 2016.

[29] A. Gonzalez et al., "A 16 mm^2 106.1 GOPS/W heterogeneous RISC-V multi-core multi-accelerator SoC in low-power 22 nm FinFET," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Grenoble, France, Sep. 2021, pp. 259–262.

[30] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis, and G. Dimitrakopoulos, "RISC-v2: A scalable RISC-V vector processor," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Seville, Spain, Oct. 2020, pp. 1–5.

[31] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, "A 'new ara' for vector computing: An open source highly efficient RISC-V V 1.0 vector processor design," in *Proc. IEEE 33rd Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2022, pp. 43–51.

[32] I. Miro-Panades et al., "SamurAI: A 1.7 MOPS-36 GOPS adaptive versatile IoT node with 15,000× peak-to-idle power reduction, 207 ns wake-up time and 1.3 TOPS/W ML efficiency," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.

[33] A. Garofalo et al., "A 1.15 TOPS/W, 16-cores parallel ultra-low power cluster with 2 b-to-32 b fully flexible bit-precision and vector lockstep execution mode," in *Proc. IEEE 47th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2021, pp. 267–270.

[34] J. M. Cebrian, M. Jahre, and L. Natvig, "ParVec: Vectorizing the PARSEC benchmark suite," *Computing*, vol. 97, no. 11, pp. 1077–1100, Nov. 2015, doi: 10.1007/S00607-015-0444-Y.

[35] J. Karimov, T. Rabl, and V. Markl, "PolyBench: The first benchmark for polystores," in *Performance Evaluation and Benchmarking for the Era of Artificial Intelligence (TPCTC)*, vol. 11135, R. Nambiar and M. Poess, Eds., Cham, Switzerland: Springer, 2018, pp. 24–41.

[36] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.

[37] M. Cavalcante, D. Wüthrich, M. Perotti, S. Riedel, and L. Benini, "Spatz: A compact vector processing unit for high-performance and energy-efficient shared-L1 clusters," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2022, pp. 1–9.

[38] I. Al Assir, M. El Iskandarani, H. R. Al Sandid, and M. A. R. Saghir, "Arrow: A RISC-V vector accelerator for machine learning inference," 2021, *arXiv:2107.07169*.

[39] P. Michael and P. Pete, "Vicuna: A timing-predictable RISC-V vector coprocessor for scalable parallel computation," in *Proc. 33rd Euromicro Conf. Real-Time Syst. (ECRTS)*, vol. 196, Jun. 2021, pp. 1:1–1:18.

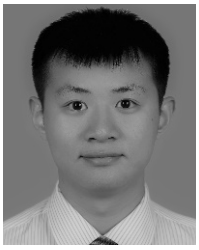
[40] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6565–6574.

[41] M. Atzori et al., "Building the ninapro database: A resource for the biorobotics community," in *Proc. 4th IEEE RAS EMBS Int. Conf. Biomed. Robot. Biomechatron. (BioRob)*, Jun. 2012, pp. 1258–1265.

[42] K. Otseidu, T. Jia, J. Bryne, L. Hargrove, and J. Gu, "Design and optimization of edge computing distributed neural processor for biomedical rehabilitation with sensor fusion," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

Yuhao Ju (Member, IEEE) received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2017, the M.S. degree from Northwestern University, Evanston, IL, USA, in 2019, where he is currently pursuing the Ph.D. degree in computer engineering.

His current research interests include computer architecture and machine learning accelerator design.



Yijie Wei (Member, IEEE) received the B.E. degree in electrical engineering from the University of Mississippi and North China University of Technology, Beijing, China, in 2017, and the M.S. degree in computer engineering and the Ph.D. degree from Northwestern University, Evanston, IL, USA, in 2019 and 2023, respectively.

He is currently a System Engineer with Kilby Labs, Texas Instruments, Dallas, TX, USA, focusing on research and development in embedded processing products. His research interests include low-power circuit design, analog amplifiers, and mixed-signal integrated circuits.

Jie Gu (Senior Member, IEEE) received the B.S. degree from Tsinghua University, Beijing, China, in 2001, the M.S. degree from Texas A&M University, College Station, TX, USA, in 2003, and the Ph.D. degree from the University of Minnesota, Minneapolis, MN, USA, in 2008.

From 2008 to 2010, he worked as an IC Design Engineer with Texas Instruments, Dallas, TX, USA, focusing on ultralow-voltage mobile processor design and integrated power management techniques. From 2011 to 2014, he was a Senior Staff Engineer with Maxlinear, Inc., Carlsbad, CA, USA, focusing on low-power mixed-signal broadband SoC design. He is currently an Associate Professor with Northwestern University, Evanston, IL, USA. He has served as program committees and conference co-chairs for numerous low-power design conferences and journals, such as ISPLED, DAC, ICCAD, and ICCD. His research interests include novel circuits and architectures for emerging applications.

Dr. Gu was a recipient of the NSF CAREER Award.