# A General-Purpose Compute-in-Memory Processor Combining CPU and Deep Learning with State-of-the-art CPU Efficiency and Enhanced Data Locality

Yuhao Ju, Yijie Wei, Xi Chen, Jie Gu
Northwestern University, Evanston, IL, USA

**Abstract**

This work presents a general-purpose compute-in-memory (GPCIM) processor combining DNN operations and vector CPU. Utilizing special reconfigurability, dataflow and instruction set, the 65nm test chip demonstrates a 28.5 TOPS/W DNN macro efficiency and a best-in-class peak CPU efficiency of 802GOPS/W. Benefit from a data locality flow, 37% to 55% end-to-end latency improvement on AI-related applications is achieved by eliminating inter-core data transfer.

**Introduction**

While tremendous progresses have been made for compute-in-memory (CIM) techniques, for end-to-end operations of AI-related tasks, a general-purpose computing unit, e.g. CPU, is not only mandatory but also often dominate the total latency due to significant pre/post-processing, data movement/alignment and versatile non-MAC tasks. As shown in Fig.1, the conventional architecture which engages the CPU core, ASIC/CIM accelerator, and a DMA engine for data transfer suffer from processor stall and underutilization issues. DNN computing often takes only 12%-50% of total run time leaving performance bottlenecked by CPU processing and data transfer [1]. For instance, 83% of run time was spent by CPU for data movement and preparation from the recent AR/VR SoC [2]. Unfortunately, the recent CIM developments do not address the need for improvements from CPU related processing and data transfer. In this work, we propose a unified general purpose CIM (GPCIM) architecture which obtain high efficiency for both DNN and vector instruction-based CPU. The highlights of this work include: (1) A unified digital CIM architecture has been developed for both vector CPU and DNN operations; (2) Best-in-class energy efficiency has been achieved on the vector CPU by exploiting the simpler pipeline, removal of cache access and data locality of CIM architecture; (3) To overcome the inter-core data transfer overhead in conventional architecture, special dataflow and dedicated instruction set are constructed for seamless data sharing between CPU and DNN operations rendering significant improvement on end-to-end performance. A 65nm test chip is developed to demonstrate the state-of-the-art energy efficiency from the GPCIM processor for both DNN (23.5TOPS/W) and CPU (802GOPS/W) tasks in end-to-end real-time applications.

**GPCIM Design and Instruction Set**

Fig. 2 shows the CIM macro which contains Data Cache Activation Memory (DAMEM) and Data Cache Output Memory (DOMEM) with the central computing units (CCU). The DAMEM is a 32bit 9T bitcell array which supports both regular SRAM function and 1b multiplication for DNN by appending a 3T NAND gate to the 6T SRAM. DOMEM is an 8T bitcell array with 2 bitlines which performs 2 read and 1 write within one clock cycle. Extra instruction cache and weight SRAM are added to support DNN and CPU functions. As shown in Fig. 2, in DNN mode, the CIM macro uses DAMEM as input memory for digital CIM MAC operation with stationary input and DOMEM as output memory. In CPU mode, DAMEM is used as a data cache (Dcache) and DOMEM acts as both register file (RF) and Dcache. Data movement between Dcache, RF and pipeline stages in traditional CPU has been significantly reduced or eliminated in GPCIM design. As shown in Fig. 2, a five-phase single-cycle operation including write-back, pre/dis-charge, latch update and vector execution, is performed for the CPU/DNN operation from DOMEM. Fig. 3 highlights the significant power benefits of GPCIM by exploiting CIM's concise dataflow in comparison with an equivalent digital counterpart of vector RISC-V pipeline core with L1 cache and RF. Shorter 2-stage pipeline in GPCIM leads to reduced flip-flops by 7.6X. Vector RF is eliminated by integrating it into CIM Dcache. ALUs are merged into CIM macro eliminating data cache access with 1.9X cache power saving and 1.3X ALU power saving. Certain pipeline logic in RISC-V pipeline core such as forwarding is also removed with 2.1X logic power saving. A 4.62X total power reduction is achieved for GPCIM. Fig. 4 illustrates logic reuse in CCU inside CIM macro. In the DNN mode, the CCU is configured as four adder trees to perform 8bit MAC utilizing the 1b results from DAMEM. In CPU mode, logics are reused to support different instructions. Extra logic beyond adder trees is also added for complete ALU functions. Input and clock gating are performed on unused logic in different modes. A customized 32b instruction set architecture (ISA) is designed to support integer vector CPU function. As in Fig. 5, 5b opcode defines different instruction types with the first 3 bits designated for locations of the operands and results. Special instructions such as "MVCSR", "SWITCH" and "PCS", are added to configure the control and status registers (CSR) for smooth mode switching between CPU and CNN. Reconfiguration costs 8.8% area overhead on CIM macro to support vector CPU operations. Fig. 6 shows a data movement scheme facilitating end-to-end operation for both CPU and CNN. After vector CPU operation for preprocessing, the CPU stores the CNN input data to DAMEM so that GPCIM can directly process the first layer of CNN without any data transfer in conventional architecture. After CNN processing, the data preparation such as data alignment, batch normalization and padding between different layers are performed seamlessly by CPU by configuring DOMEM to Dcache avoiding further data movement. As shown in Fig.6, GPCIM achieves 52~56% end-to-end latency improvement for CNN tasks due to elimination of data transfer and parallel vector processing compared with Gemmini [1] using a scalar RISC-V CPU and an accelerator.

**Measurement Results and End-to-end Test Case**

A 65nm test chip was fabricated with a nominal supply of 1.0V. As shown in Fig.7, for DNN mode, GPCIM achieves a 7.62~17.8TOPS/W 8-bit system energy efficiency and a 14.8~28.5TOPS/W macro energy efficiency matching prior CIM CNN performance [3, 4]. GPCIM achieves the highest CPU efficiency, more than 17.8X improvement compared with prior 4 vector RISC-V CPU and 3 scalar RSIC-V CPU despite a lower throughput due to slower operating frequency and less vectors (scalable) being implemented [7-10]. A comparison table with prior CIM and RISC-V CPUs is also shown in Fig. 7. Compared with a prior instruction supported CIM [5], this CIM achieves 7.3X efficiency improvement for 32b MUL

instruction. Compared with a recent reconfigurable ASIC+CPU digital design [6], GPCIM achieves 10X higher DNN efficiency and 118X higher CPU efficiency. Fig. 8 further demonstrates a detailed end-to-end case study from GPCIM using a popular CNN-based Simultaneous Localization and Mapping (SLAM) task for mobile robots. 76% operations including preprocessing (camera pose estimation, depth refinement) and post-processing (key-frame creation, graph pose optimization) need to be performed by CPU due to the non-CNN operations such as division and exponentiation which are challenging for conventional CIM+CPU architecture. For the SLAM task, the GPCIM achieves 35X CNN efficiency improvement, 7.9X CPU efficiency improvement, 37% end-to-end latency improvement, compared with Gemmini [1].

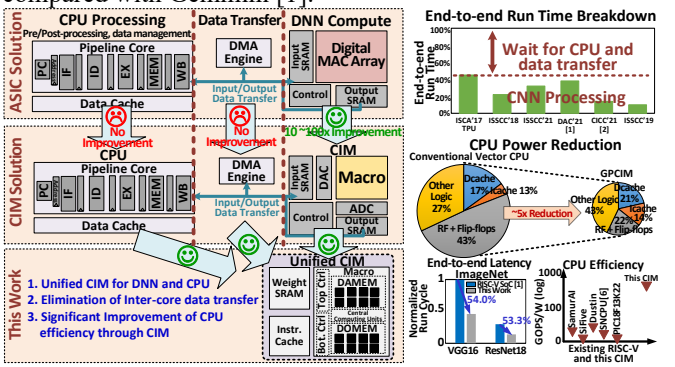



Fig. 1 Challenges of the end-to-end AI tasks using DNN/CIM techniques with CPU being bottleneck and contributions of this work.

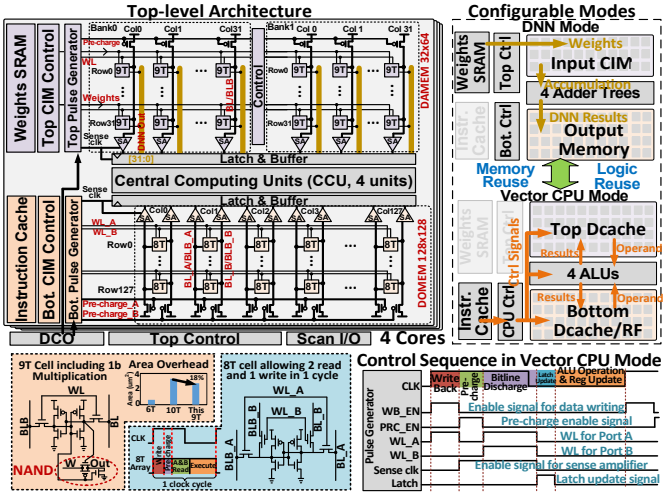



Fig. 2 Chip architecture, reconfiguration modes and CIM cells.

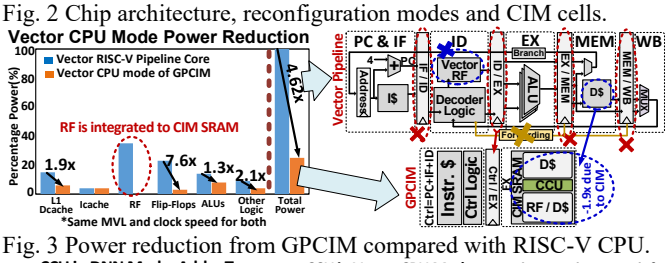



Fig. 3 Power reduction from GPCIM compared with RISC-V CPU.

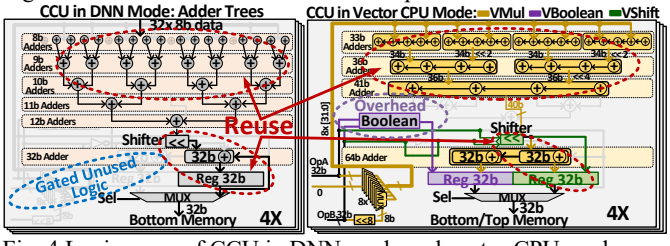



Fig. 4 Logic reuse of CCU in DNN mode and vector CPU mode.

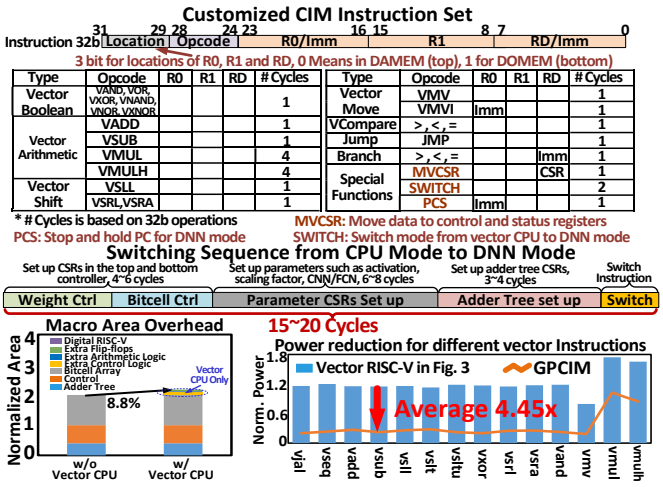



Fig. 5 Customized instruction sets, mode switching sequence, area overhead and instruction energy reduction from GPCIM.

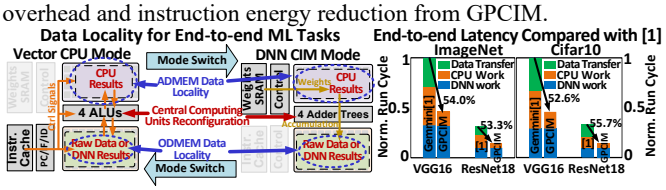



Fig. 6 Data locality flow and end-to-end latency improvement for CNN image classification tasks.

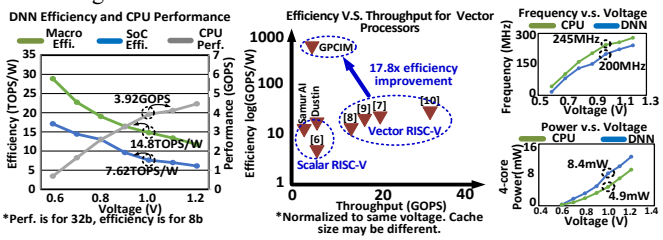
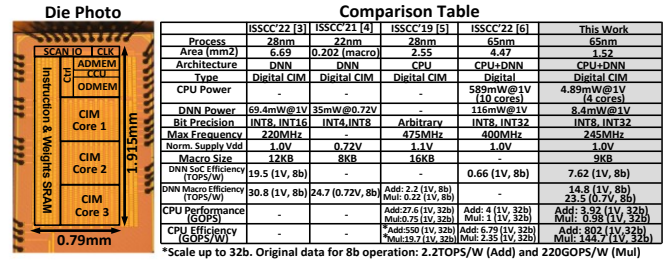



Fig. 7 Measurement results, comparison table and die photo.



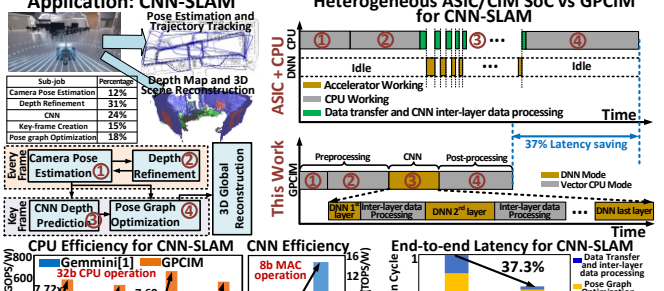




Fig. 8 A detailed case study on the SLAM application from GPCIM.

**Acknowledgements** This work is supported in part by NSF grant CCF-2008906.

## References

[1] H. Genc et al., *DAC, 2021*. [2] H. E. Sumbul et al., *CICC,2022*. [3] F. Tu et al., *ISSCC, 2022*. [4] Y. -D. Chih et al., *ISSCC, 2021*. [5] J. Wang et al., *ISSCC, 2019*. [6] Y. Ju et al., *ISSCC, 2022*. [10] B. Zimmer et al., *JSSC, 2016*. [8] A. Gonzalez et al., *ESSCIRC, 2021*. [9] K. Patsidis et al., *ISCAS, 2020*. [10] M. Perotti et al., *ASAP, 2022*.