A 16nm Heterogeneous Accelerator for Energy-Efficient Sparse and Dense AI Computing

Gopikrishnan Raveendran Nair
Yu Cao
Department of ECE, University of Minnesota, Minneapolis, MN, USA

Fengyang Jiang
Jeff Zhang
School of ECEE, Arizona State University, Tempe, AZ, USA

ABSTRACT

Artificial intelligence (AI) has evolved from dense Deep Neural Networks (DNNs) toward a diverse set of models, such as sparse graph convolutional neural networks (GCNs). These new models differ in model size, processing flow, memory access patterns, and data/model sparsity. Hardware platforms optimized for dense DNNs with a regular data structure are inefficient to manage new unstructured, sparse workloads, such as GCNs. For instance, in-memory computing (IMC) units that is suitable for dense matrix/vector computation, but significantly underutilized for sparse data.

In this work, we propose a new reconfigurable heterogeneous accelerator, with the target to balance the computation needs and energy efficiency in diverse AI models. Based on representative DNNs and GCNs, we propose two types of processing elements (PEs): (1) A Latch-based digital IMCs (LIMC) for regular and dense computation, and (2) A digital SIMD array (SIMD) with fine-grained control for irregular and sparse workloads. To integrate both types of PEs and dynamically manage the data flow, we design reconfigurable modules of scatter/gather and buffers, supporting different types of memory access and compute patterns. The new heterogeneous accelerator has been designed and taped out at 16nm. Based on 16nm design data, it achieves an 11× improvement in latency compared to baseline homogeneous accelerators, and up to 2.1× and 20× improvement in TOPS/mm^2 and TOPS/W, respectively, as compared to state-of-the-art accelerators.

CCS CONCEPTS

• **Hardware → Application specific integrated circuits**.

KEYWORDS

GNNs, Sparsity, AI accelerator, Irregular data access.

ACM Reference Format:
Gopikrishnan Raveendran Nair, Yu Cao, Fengyang Jiang, and Jeff Zhang. 2024. A 16nm Heterogeneous Accelerator for Energy-Efficient Sparse and Dense AI Computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '24), August 5–7, 2024, Newport Beach, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665314.3670824

This work is supported by COCOSYS, one of six centers in JUMP2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

ISLPED '24, August 5–7, 2024, Newport Beach, CA, USA

ACM ISBN 979-8-4007-0688-2/24/08…$15.00
https://doi.org/10.1145/3665314.3670824

Table 1: GCN and SpMV workloads are dramatically sparse.

Dataset	Domain	Dimension	Sparsity
Cora	GCN	2708 × 2708	99.82%
Citeseer	GCN	3327 × 3327	99.89%
Consph	2D/3D Problem	83334 × 83334	99.9%
Ohne2	Semiconductor Device	181343 × 181343	99.9%

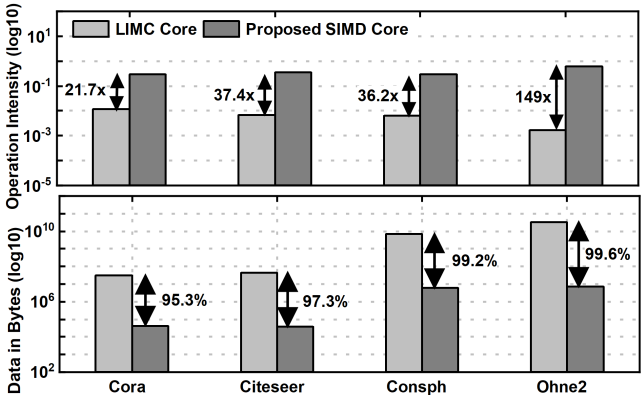


Figure 1: Mapping of sparse workloads on IMCs is inefficient, while our proposed SIMD cores effectively improve computational efficiency and data access.

1 INTRODUCTION

Dedicated accelerators have been proposed to achieve efficient CNN acceleration [1, 2]. With bigger models, having separate memory and compute units poses great challenges to these traditional accelerator architectures [2]. To address this, In-Memory Compute (IMC) was introduced. IMCs are efficient at dense, compute bound MAC operations with high operation intensity (i.e., the number of compute operations performed on each of the loaded data) [3]. Other computations such as sparse matrix vector multiplication (SpMV) and feature aggregation in GCNs are characteristically different than dense MACs. This reduces the emphasis on data movement cost and high compute efficiency that motivates IMC use [3].

GCNs and SpMVs involve computations on sparse matrices, and storing sparse matrices on IMCs is inefficient [4]. Since these workloads are 99% sparse, as shown in Table 1, transferring these zeros from memory and storing them in Latch-based IMCs (LIMCs) lead to significant bandwidth, energy wastage and underutilization of LIMCs. To address this, we propose the use of SIMD-based compute cores to effectively handle the sparse computations. Figure 1 bottom, shows the volume of data needed to do sparse computation using LIMC and SIMD cores for GCN and SpMV. For evaluation with LIMCs, we follow the same method as in [5], where the extremely sparse matrix is mapped entirely to the IMC crossbar. SIMD

cores operate on sparse data represented in compressed formats like COO, CSR, eliminating the need to move zeros from memory, achieving up to a 99% reduction in the volume of data fetched compared to IMCs. SIMD cores achieve up to 149x more operational intensity (i.e. the number of compute operations performed on each of the loaded data), than LIMCs as shown in Figure 1 top. Since the adjacency matrix is 99% sparse, LIMCs effectively compute only 1%, whereas with SIMD cores, only the non-zero entries of sparse matrices are fetched from memory. This results in higher operational intensity. Additionally, to exploit activation/input sparsity at the input of LIMCs we propose the use of operation skipping and booth multiplier to achieve coarse and fine grained operation skipping. Though IMCs are efficient for dense MACs [6], they still incur hardware virtualization overhead, such as data loading leading to significant performance loss [3, 7]. To this end, we propose the use of a ping-pong LIMC core to efficiently hide the weight update cost by overlapping it with the computation. Since CNN and GCN involve both dense and sparse computations, we chose them as representative heterogeneous AI workloads.

The major contributions of this work are:

- A heterogeneous accelerator, with a LIMC to manage the dense operations, and a SIMD core for the sparse operations.
- Operation skipping and booth multiplier to handle activation sparsity at the input of LIMC.
- Ping-Pong LIMC design to efficiently handle weight update overheads of IMC accelerators.
- 16*nm* implementation and tapeout of the proposed accelerator with comprehensive evaluation of the performance with state-of-the-art accelerator designs.

2 BACKGROUND AND MOTIVATION

2.1 Workload Analysis

CNNs use dense MAC convolutions to extract features from input data. GCNs perform feature transformation (FT) and feature aggregation (FA) operations, capturing the relationships among the nodes in the graph. FT is a dense MAC computation between the feature matrix of the graph and the weight matrix, while FA of each node involves the weighted sum of neighboring nodes, determined using the extremely sparse adjacency matrix [8]. Therefore, GCN accelerators must handle both sparse, non-MAC FA, and dense MAC FT. SpMV workloads have extremely sparse matrices as inputs. The sparsity of the adjacency matrix in GCNs and SpMV inputs, as shown in Table 1, is significantly larger than the weight or activation matrix in CNNs (10%-50%) [9]. Table 2 highlights the main differences in CNN, GCN and SpMV workloads. In CNNs, 99% of computations are MAC operations [10]. In GCNs, FT is a MAC operation, whereas FA and SpMV are sparse computations.

2.2 Limitations of Prior Works

IMC based CNN accelerators [11, 12] are efficient at MAC operations with high operational intensity. But, they are inefficient for sparse non-MAC aggregation in GCN and SpMV due to low operational intensity. [8] proposes a novel algorithm to accelerate feature aggregation in GCNs by exploiting the symmetry property of graphs and to reduce redundant computations. [5] proposes an IMC-based GCN accelerator. They store the extremely sparse adjacency matrix in the IMCs, leading to poor operational intensity and wasted memory bandwidth. These accelerators are optimized to exploit the characteristics and computational requirements of GCNs, rendering them inefficient for CNN acceleration. Therefore, we need an accelerator to balance the heterogeneity of AI models.

Table 2: Workload characteristics of CNNs, GCNs and SpMV.

Workload	Kernel	Type
Convolution	MAC	Dense/Sparse
Fully connected, MLP	MAC	Dense/Sparse
Feature transformation	MAC	Dense/Sparse
Feature Aggregation	Sparse matrix	Very Sparse
SpMV	Sparse matrix	Very Sparse

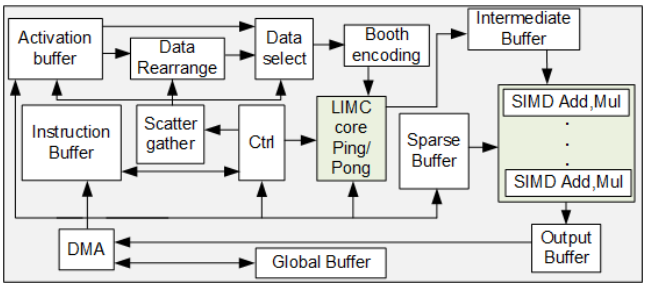


Figure 2: Architecture of the heterogeneous accelerator with ping-pong LIMC and SIMD cores.

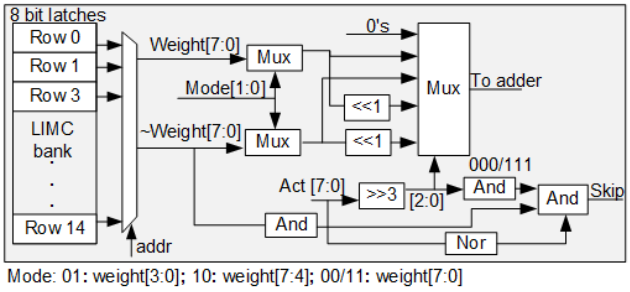


Figure 3: Microarchitecture of the proposed LIMC bank with the booth multiplier. Operations are skipped at the coarse level if the weights/activations are zero and at the fine level if the booth encoding bits are 000/111.

3 SYSTEM DESIGN AND ARCHITECTURE

3.1 Architecture Overview

To achieve a balanced operational intensity across dense and sparse workloads, we integrate a heterogeneous AI accelerator with LIMC and sparse SIMD core. Figure 2 shows the architecture of the SOC system. To address the weight update overhead we use ping-pong LIMC. Booth multiplier with operation skipping is used to address sparsity at the input of LIMCs. SIMD units with multiply, add and ReLU handles sparse computations and activation functions respectively as needed. Programmable scatter-gather and control logic handles the different memory access patterns and ensure efficient integration of the LIMC and SIMD cores.

3.2 Latch-based IMC (LIMC) Core

3.2.1 LIMC architecture. The building block of LIMCs are latches and is equivalent to the bitcell in SRAM IMCs. LIMC supports both 8-bit and 4-bit computations. To balance the area constraint and parallel MAC operations, we divide the LIMC macro into banks and computations happen parallelly across banks. Each bank is

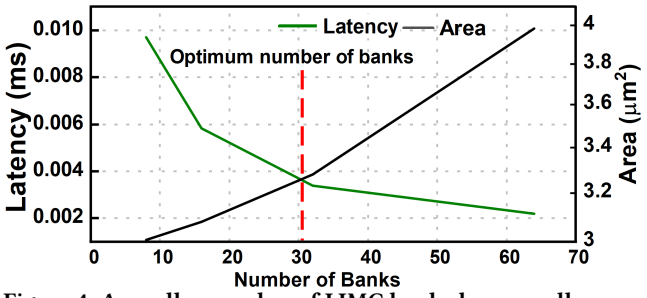


Figure 4: A smaller number of LIMC banks have smaller area due to less number of multipliers, but will also have more iterations (the size of the input/the number of banks) at the input of LIMC increasing latency.

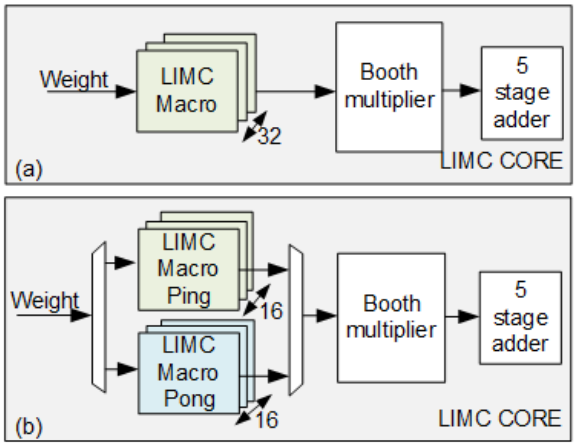


Figure 5: (a) Conventional IMC core where computation and weight updates are sequential. (b) Proposed Ping-Pong based LIMC core to efficiently hide the IMC weight update latency by overlapping the computation and IMC weight updates across the ping-pong cores.

connected to a modified booth multiplier. With booth the number of partial products is halved [13] as compared to bit-serial computing in SRAM or RRAM-based IMCs, resulting in faster computations.

The number of banks is optimized to achieve a balance between the area and the compute latency. As number of banks increase the number of booth multipliers required for parallel computation increases, resulting in an area increase. The compute latency decreases with number of banks as the number of iterations required is lower leading to lower latency. Number of iteration is defined as the size of the input at the LIMC divided by the number of banks. As illustrated in Figure 4 we select the number of banks to be 30, to achieve an optimum balance between area and latency. The micro-architecture of a LIMC bank with booth multiplier is as illustrated in Figure 3. Eight columns of 1-bit latches share one set of booth multiplier. To achieve maximum weight mapping efficiency in LIMC for the common convolution use cases, the number of rows in a bank is set to 15, a multiple of 3 and 5, the most common kernel sizes. Thus, with 30 banks containing 15 rows each, the total number of rows in the LIMC is 450. The system bus is 128 bits wide. Therefore, we set the number of columns in the LIMC to 128, which is equivalent to 16 columns of 8-bit latches. The output generated by the LIMC is stored in a banked intermediate buffer with 16 banks, facilitating parallel LIMC output writes.

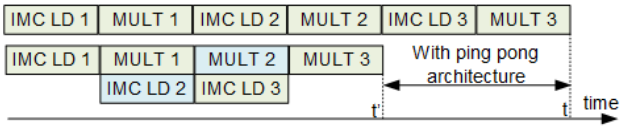


Figure 6: Proposed ping-pong IMC hiding data movement latency by overlapping weight load and computation.

Table 3: Improvement in compute latency achieved from the proposed ping-pong IMC architecture.

Mat A	Mat B	Clock Cycles w/ ping-pong	Clock Cycles w/o ping-pong	Reduction in Clock Cycles
(10,1433)	(1433,16)	3794	6320	39.9%
(100,1433)	(1433,16)	33494	44570	24.8%
(400,1433)	(1433,16)	132494	172070	23%
(1000,1433)	(1433,16)	339102	422820	19%

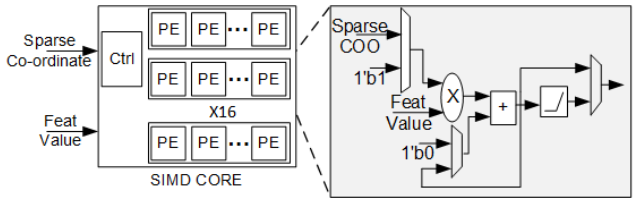


Figure 7: Micro-architecture of SIMD core.

3.2.2 Activation Sparsity. LIMC core dynamically handles activation sparsity at the input by using operation skipping and booth multiplier. This ensures sparsity handling at a coarse and fine granularity. At the coarse level, if all the activations at the LIMC bank input are zero the computations will be skipped. At finer granularity, if the booth encoding is either 000 or 111 the computations will be skipped as shown in Figure 3.

3.2.3 Ping-Pong IMC. For efficient integration of IMC-based accelerators the overhead of weight updates should be addressed. Updating IMC weights between computations leads to significant performance loss [7]. Existing works on IMC-based accelerators such as [5, 11, 14] do not address this overhead. To this end, we proposed the use of a ping-pong LIMC core to efficiently hide the weight update cost by overlapping it with the computation. Figure 6 shows an illustrative toy example with three independent workloads. With only one LIMC core as in Figure 5(a), weight writes and execution can only happen sequentially. To hide this overhead, we use ping-pong LIMC architecture Figure 5(b), where one core will be performing MAC computations while the other will be loading the new set of weights for subsequent computation.

Table 3 shows the improvement from the proposed ping-pong LIMC for matrix multiplication workloads. We achieved up to 39.9% reduction in clock cycles with the proposed ping-pong architecture. Matrix B is mapped to the LIMC, and its size is set to be larger than the number of rows in the LIMC to ensure ping-pong operations. When Matrix A is smaller, writes to the LIMCs dominate the latency, maximizing the benefits of the ping-pong solution. However, as the size of Matrix A increases, execution time becomes the dominant factor, reducing the gains of the ping-pong architecture.

3.2.4 SIMD Core and Sparse Buffer. SIMD cores compute the sparse workloads. Figure 7 illustrates the architecture of a SIMD core and the microarchitecture of a Processing Element (PE) within the core. The core comprises of PE rows with N columns, where each PE has its own scratch pad (SPAD) for storing partial results for reuse.

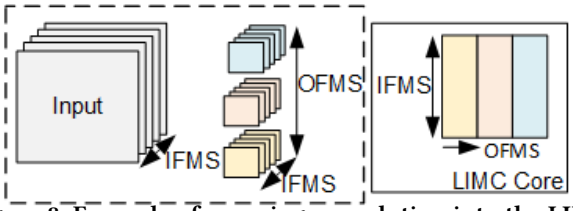


Figure 8: Example of mapping convolution into the LIMC core. The IFMs are mapped across the rows within a LIMC macro and the OFMs are mapped across the macros.

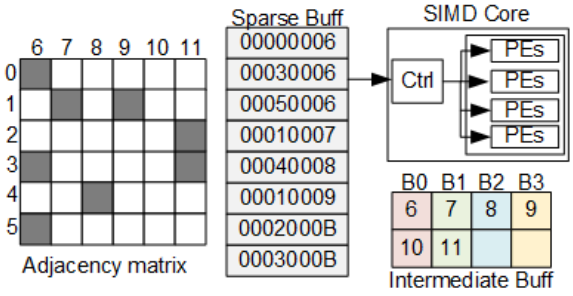
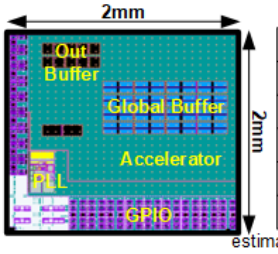


Figure 9: Mapping of a tile adjacency input matrix to the compute units for aggregation. Sparse buff stores the edges in COO format. The control fetches and assigns each edge to a PE row. Banked Intermediate buffer stores the features of the node to be accessed parallelly by the SIMD cores.

The SIMD core has both intra-PE row and inter-PE row parallelism. The adjacency matrix is partitioned into smaller tiles to fit into on-chip memory [8]. The sparse buffer holds the edges of the graph or the entries of the sparse matrix in COO format. For aggregation, based on the edge information, the control logic initiates read to the intermediate buffer to fetch the features of the neighboring nodes and then issues the aggregation to one of the PE rows. During SpMV the control logic fetches the vector elements from the intermediate buffer based on the sparse matrix entries in COO format. The size of the edge buffer is set to 16KB. The graph features or the vector elements are stored in the intermediate buffer. With 16 banks for the intermediate buffer, 16 reads can be done in parallel. Therefore, the control logic can issue 16 edges or 16 rows of sparse matrix in a cycle. Thus, the number of PE rows is set to 16. For each SIMD core, the number of columns in a PE is set to 128, supporting the feature dimensions of typical graph datasets. In cases where the dimension exceeds 128, it will be executed iteratively. The SIMD core can also be used as a ReLU activation unit for convolution operation, where the output from LIMC can be fed to SIMD core through the intermediate buffer.

3.2.5 Data Delivery Modules. The DMA, scatter-gather (SG), data rearrange, data select and activation buffer (AB), and a 512KB global buffer make up the data delivery modules. We have designed a dataflow with multiple hierarchies of memories to ensure pipelined data delivery at the input of SIMD core. SG will generate the address based on the workload data access patterns. For convolution, the data is fetched from AB and then stored in the line buffers (LB) inside the data rearrange module. Based on the stride and kernel size, the data will be selected from the LB. The data select module will choose the data directly from the activation buffer for matrix multiplication or from the data rearrange logic for convolution, and write it into the shift register (SR) inside the booth encoding block.



Technology	Intel 16nm
Area	4mm^2
Voltage	1 V
Clock	200Mhz
IMC capacity	115Kb
TOPS/W	36.3*

*estimation from post apr for 8bit MAC ops

Figure 10: Post layout of the proposed accelerator.

3.3 Mapping of Workloads to LIMC

3.3.1 Matmul Operations. To multiply two matrices A and B, matrix B will be mapped to the LIMC core crossbar. With 450 (30 banks and 15 rows per bank) entries in each macro, a maximum of 450 rows of matrix B can be mapped to the LIMC. If bigger, then it is computed iteratively. Matrix A stored in the global buffer, is fed to the input banks of LIMC and computation happens tile by tile.

3.3.2 Convolution. Mapping convolutions to the LIMC requires high crossbar mapping efficiency to achieve optimal performance. The weight's IFMs are mapped across the rows within a macro, and the weight's OFMS are mapped across the columns of the LIMC as shown in Figure 8. The activations are provided at the input banks of the LIMC for computation. To achieve maximum weight mapping efficiency in LIMC, the number of rows in a bank is set to 15, a multiple of 3 and 5. A single macro can support 50 and 30 IFMs for 3×3 and 5×5 kernels, respectively at 8-bit precision. Our design is specifically tuned to achieve maximum mapping efficiency for the kernel sizes of 3×3 and 5×5. We can support kernel sizes up to 15, with the mapping efficiency less than 100%.

3.4 Mapping of Aggregation to SIMD

Aggregation can happen in two ways: either FT followed by FA where the output of LIMC will be directly consumed by the SIMD core from the intermediate buffer (IB), or standalone FA where the DMA will populate the IB with node features. The aggregation of each node is statically assigned to a unique PE row, and the partial results of that node are stored locally in the scratch pad to avoid inter-core communication. The target node id modulo the number of SIMD cores 16, gives a unique PE row for that node. The edges fed into the sparse buffer are arranged in the ascending order of neighboring nodes as shown in Figure 9. This introduces both temporal and spatial locality within the aggregation flow, and thus helps in generating predictable memory accesses. To support this memory access pattern, the neighboring nodes are arranged such that it fall into consecutive banks of intermediate buffer, enabling parallel read access. Offline pre-processing of graph data to further improve the regularity in memory access and data reuse in the aggregation phase is well studied in [8] and is beyond the scope of this work. These offline pre-processing methods can be used in conjunction with our hardware to further improve GCN acceleration.

4 EXPERIMENTAL RESULTS

To benchmark the performance of our programmable heterogeneous architecture, we implemented the entire design in RTL, synthesized and completed the tapeout process using Intel's 16nm for 200 MHz. The post-layout is shown in Figure 10, and full SOC level simulations of CNN and GCN workloads were performed. For baseline comparison, the SIMD cores in the heterogeneous design were

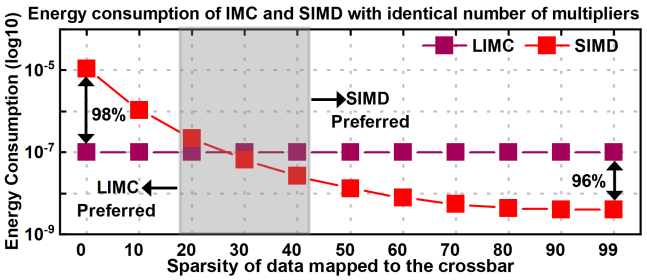


Figure 11: At lower level of sparsity LIMC achieves better energy efficiency whereas at higher sparsity SIMD achieves better energy efficiency for identical workload.

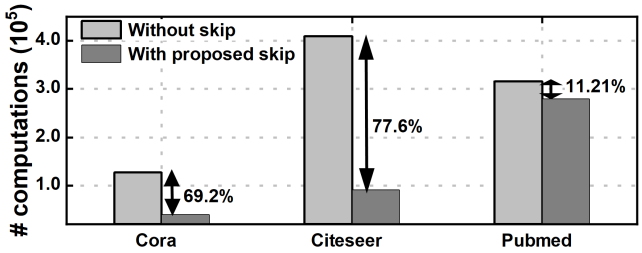


Figure 12: Reduction in the number of computations by exploiting sparsity at the input banks of LIMC by using operation skipping and booth multiplier.

substituted with LIMC cores having identical compute resources to create a homogeneous accelerator.

4.1 Workload Mapping: LIMC vs. SIMD

In this experiment, we make the LIMC and SIMD units identical, with the same number of multipliers, and perform a matrix multiplication operation. Identical inputs are fed to both LIMC and SIMD. The sparsity of the input matrix mapped to the LIMC crossbar is varied from 0 to 99%, while the sparsity of the other input matrix remains fixed at 0%. At sparsity levels below 20%, LIMC achieves up to 98% better energy efficiency compared to SIMD. Between 20% and 40% sparsity, both LIMC and SIMD achieve similar energy efficiency. For sparsity levels above 40%, SIMD achieves up to 96% improvement in energy efficiency. Regardless of the sparsity of the input mapped to the LIMC crossbar, LIMC always takes the same number of clock cycles to perform computations. In contrast, with SIMD, using compressed data representations allows us to operate solely on non-zero elements, thereby achieving lower latency. This results in lower energy consumption. Therefore, based on the sparsity of the workload being mapped to the LIMC crossbar, we can determine to which compute unit it should be mapped, as illustrated in Figure 11.

4.2 Improvement of LIMC from Input Skipping and Booth Multiplier

Figure 12 shows the reduction in the number of computations achieved by exploiting the sparsity at the input banks of LIMC using booth multiplier and operation skipping. We evaluated the first layer of feature transformation of GCN on the graph datasets Cora, Citeseer, and Pubmed. The dense weights with zero sparsity are mapped to the LIMC core and graph feature matrix is the input to the LIMC. We achieved up to a 77% reduction in the number

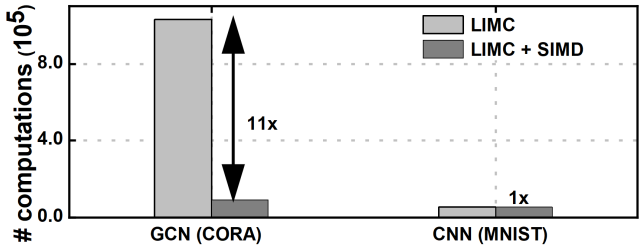


Figure 13: End-to-end GCN and CNN inference performance comparison between the LIMC-only homogeneous accelerator and heterogeneous architecture.

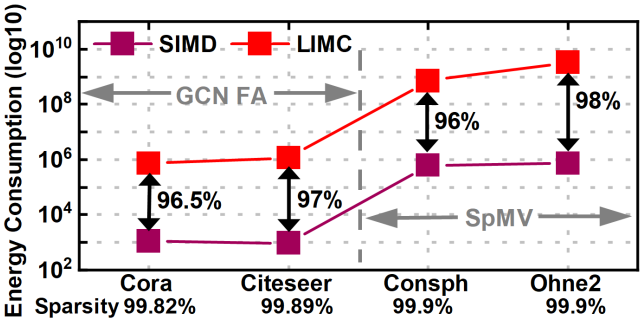


Figure 14: LIMC vs SIMD energy consumption for GCN FA and SpMV. Proposed SIMD achieves high energy efficiency for sparse GCN FA and SpMV workload.

of computations because the initial feature matrix of the graph is sparse. This holds true for CNN as well, where the activation of later layers contain more zeros due to ReLU.

4.3 Comparison with LIMC-only Accelerators

To benchmark the design, we compared our proposed heterogeneous accelerator against the baseline homogeneous accelerator, which consists solely of LIMC cores, using GCN (Cora) and CNN (MNIST) as the workloads. In the case of GCN using the homogeneous accelerator, both FT and FA are mapped to LIMC. In contrast, with the heterogeneous accelerator, dense FT is mapped to LIMC, and sparse FA is mapped to SIMD. For CNN, since convolution weights are dense, they are mapped only to LIMC for both the heterogeneous and homogeneous accelerators. The number of cycles required for GCN with a homogeneous accelerator is 11x more than with the proposed heterogeneous architecture, as illustrated in Figure 13. The poor operational intensity of LIMC for the sparse FA of GCN contributes to the overall latency of GCN inference when using a homogeneous accelerator. For convolution, both the heterogeneous and homogeneous architectures have the same performance since convolutions are only mapped to LIMCs.

4.4 Energy Inefficiency of LIMC with Sparsity

In Figure 14, the energy consumption of LIMC and SIMD cores for sparse GCN FA and SpMV workload is depicted. For sparse workloads, SIMD achieved up to a 98% improvement in energy efficiency as compared to LIMC cores.

4.5 Comparison with Prior Works

Table 4 shows the comparison of our accelerator with the state-of-the-art IMC based accelerators. The neural networks (NN) used to evaluate the performance are the same as in [16]. [15] is primarily

Table 4: Comparison with state-of-the-art CNN and GCN accelerators.

	[15]	[7]	[16]	[11]	This work	[5]	[8]
Workload	**CNN**				**CNN, GCN**	**GCN**	**GCN**
Technology	IMC 16nm	IMC 65nm	SIMC 65nm	DIMC 65nm	LIMC 16nm	IMC 32nm	FPGA
Size	$25mm^2$	$12mm^2$	$9.4mm^2$	$20.52mm^2$	$4mm^2$	$17.43mm^2$	N/A
IMC capacity	4.5Mb	65Kb	520Kb	300Kb	115kb	7.8Mb	N/A
Frequency	200MHz	100MHz	4MHz	10Mhz	200MHz	1GHz	220MHz
Precision	8b Fixed Pt	8b Fixed Pt	8b Fixed Pt	4bFixed Pt	8b Fixed pt	8bt Fixed pt	8b Fixed Pt
CNN performance (MNIST)	N/A	N/A	5.85ms/fr	2.25ms/fr[b]	0.023ms/fr*	N/A	N/A
CNN performance (CIFAR10)	0.13ms/fr	6.51ms/fr	11.5ms/fr	N/A	1.39ms/fr*	N/A	N/A
GCN (Cora)	N/A	N/A	N/A	N/A	3.3ms[c]	0.6ms	0.06ms
TOPS/W	1.8	2.3	7.96	14.07	36.3	N/A	N/A
TOPS/mm^2	2.67	0.037	N/A	0.014	5.615	N/A	N/A

*Used the same network as [16],[c] Used the same network as [5]

an IMC based accelerator which can support CNNs. This works achieves better performance on CIFAR10, which can be attributed to larger scale design with 400x more IMC compute capacity. Since they use IMC for compute, it will suffer from low operation intensity and energy efficiency for sparse workloads. They have also utilized a SIMD core primarily for element-wise activation function operations. [7], also uses ping-pong for weight updates by using alternate banks as ping and pong with 16 rows in each bank with a total of 128 rows. But here the size of the ping and pong array is extremely small with 64 rows each thus leading to negligible update overhead when compared to execution time. Thus for small array sizes the improvement from ping-pong is not significant. We achieved 5x and 2.3x performance improvement over [16] with frequency scaled down to 100MHz. [16] is a Stochastic computing (SC) based IMC and SC suffers from high cost of binary to stochastic number conversion and compute error.

Our architecture achieves 9x better performance than [11] with the operation frequency of our design scaled down to 10 MHz. [11] is an all-IMC design, making it inefficient for sparse workloads. Despite having 67x more resources and 1GHz frequency, our accelerator achieves comparable performance with [5] for a two-layer end to end GCN inference latency. Full IMC based GCN accelerators like [5] are extremely inefficient in executing sparse FA aggregations, due to low operational intensity and wasted memory bandwidth from moving zeros. IMCs-based accelerators can also employ sparse representations while transferring data from memory to conserve bandwidth, albeit with the additional hardware overhead of decoding and accurately mapping it to the crossbars. However the operational intensity still remains low. [8], a FPGA implementation achieves better performance since it deploys symmetry based offline pre-processing to improve locality in graph. It also uses HBM with 512GB/s bandwidth which reduces the data movement latency. We achieve a 2.1x improvement in TOPS/mm^2 over [15] at 16nm and upto 20x improvement in TOPS/W over others.

5 CONCLUSION

IMCs are inefficient in accelerating extremely sparse workloads, due to their low operation intensity and energy efficiency. In this work, we propose a scalable heterogeneous accelerator with LIMC and SIMD cores, to balance the computation needs and the efficiency of CNN, GCNs and SpMVs. We design an architecture which efficiently integrates a ping-pong based LIMCs for MAC operations and the SIMD cores for sparse operations. We also define a workload mapping strategy to determine which core to utilize based on the sparsity of the workloads. We implement our design with Intel 16nm PDK at 200 MHz and successfully taped out a complete SOC accelerator based on the new design. For the CNN, GCN and SpMV benchmark workloads we achieve up to 11× improvement in performance over homogeneous accelerators and 2.1× improvement in TOPS/mm^2 over [15] at 16nm and 20× improvement in TOPS/W, as compared to state-of-the-art accelerators. Since AI models can be expressed as a combination for dense and sparse computations, the design strategy and analysis in this work can be extended to accelerate more diverse AI workloads.

REFERENCES

[1] Anupreetham Anupreetham et al. End-to-end fpga-based object detection using pipelined cnn and non-maximum suppression. In *2021 FPL*, 2021.
[2] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IJSSCircuits*, 52(1), 2017.
[3] Naveen Verma et al. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine*, 11(3):43–55, 2019.
[4] Biresh Kumar Joardar et al. Heterogeneous manycore architectures enabled by processing-in-memory for deep learning: From cnns to gnns:. In *ICCAD*, 2021.
[5] Sumit K. Mandal et al. Coin: Communication-aware in-memory acceleration for graph convolutional networks. *IEEE JETCAS*, 12(2):472–485, 2022.
[6] Zhenyu Wang et al. Ai computing in light of 2.5d interconnect roadmap: Big-little chiplets for in-memory acceleration. In *IEDM*, 2022.
[7] Jinshan Yue et al. A 2.75-to-75.9tops/w computing-in-memory nn processor supporting set-associate block-wise zero skipping and ping-pong cim with simultaneous computation and weight updating. *ISSCC*, 2021.
[8] Gopikrishnan Raveendran Nair et al. Fpga acceleration of gcn in light of the symmetry of graph adjacency matrix. In *DATE*, pages 1–6, 2023.
[9] Bingyi Zhang et al. Boostgcn: A framework for optimizing GCN inference on FPGA. In *IEEE FCCM*, 2021.
[10] En-Yu Yang et al. Flexacc: A programmable accelerator with application-specific isa for flexible deep neural network inference. In *ASAP*, 2021.
[11] Gokul Krishnan et al. 3d-isc: A 65nm 3d compatible in-sensor computing accelerator with reconfigurable tile architecture for real-time dvs data compression. In *A-SSCC*, 2023.
[12] Gopikrishnan R. Nair et al. 3d in-sensor computing for real-time dvs data compression: 65nm hardware-algorithm co-design. pages 1–1, 2024.
[13] Divya Govekar et al. Design and implementation of high speed modified booth multiplier using hybrid adder. In *(ICCMC 2017)*, pages 138–143, 2017.
[14] Yu-Der Chih et al. 16.4 an 89tops/w and 16.3tops/mm2 all-digital sram-based full-precision compute-in memory macro. In *ISSCC 2021*.
[15] Hongyang Jia et al. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. In *ISSCC*, 2021.
[16] Jiyue Yang et al. 65nm 8-bit all-digital stochastic-compute-in-memory deep learning processor. In *A-SSCC*, 2022.