SP-IMC: A Sparsity Aware In-Memory-Computing Macro in 28nm CMOS with Configurable Sparse Representation for Highly Sparse DNN Workloads

Amitesh Sridharan[1], Fan Zhang[1], Jae-sun Seo[2], Deliang Fan[1]

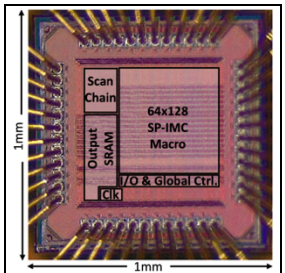
[1]Johns Hopkins University, Baltimore, MD.

[2]Cornell Tech, New York, NY.

Deep neural networks (DNNs) have experienced unprecedented success in a variety of cognitive tasks due to which there has been a move to deploy DNNs in edge devices. DNNs are usually comprised of multiply-and-accumulate (MAC) operations and are both data and compute intensive. In-memory computing (IMC) methodologies have shown significant energy efficiency and throughput benefits for DNN workloads by reducing data movement and eliminating memory reads. Weight pruning in DNNs can further improve the energy/throughput of DNN hardware through reduced storage and compute. Recent IMC works [1-3,6] have not explored such sparse compression techniques unlike ASIC counterparts to enable storage benefits and compute skipping. A recent work [4] attempted to exploit this by compressing weights using a binary map and a custom compression format. This is sub-optimal because the implementation requires a complex routing mechanism (butterfly routing), additional compute to decode compressed weights and has limited flexibility in supporting different sparse encodings. Fig. 1 illustrates our motivations and the challenges for implementing weight compression in digital IMC designs and the need for a new methodology to enable sparse compute directly on compressed weights. In this work, we present a novel sparsity-integrated IMC (SP-IMC) macro in 28nm CMOS which, for the first time, utilizes three popular sparse compression formats, i.e., coordinate representation (COO), run length encoding (RL) and N:M sparsity [7] all along the matrix column direction with tunable precisions. SP-IMC stores and directly processes the sparse compressed weights in the macro, achieving higher storage density, reduction in re-write operations to the macro and higher overall energy efficiency.

Fig. 2 shows the proposed SP-IMC macro which comprises of 64x128 bit-cells. The IMC macro consists of 16 column groups (CG), where each CG consists of 32 row groups (RG) and one accumulation logic (AL) block. Each RG has 16 bit-cells split into two 8b-rows by a multiply decode and compare (MDC) block. The two 8b-rows have four bits of 10T bit-cells for weight storage and another four bits of traditional 6T bit-cells to store the indices for decoding the compressed weights. The 10T bit-cell has four additional transistors T1-T4, besides the 6T. The transistor pairs "T1, T2" and "T3, T4" each perform AND operations in parallel with the streamed-in input activation (IA) and the stored weight to reduce the IA bottleneck in previous designs [1-6] which rely on a purely bit-serial approach to support large IA precision. Thus, each 10T bit-cell performs a 1b-W:2b-IA multiplication and the four 10T bit-cells at the top and bottom collectively perform two 4b-IA:2b-W partial multiplications. These partial-multiply outs serve as inputs to the MDC block which has shift accumulators/partial multipliers to complete the 4b2b multiplications. The 4b indices at the top and bottom of the MDC block serve as the input to the RL/COO decode block which calculates the RLC indices based on the index from the previous column or directly pass the indices from the bit-cell to the comparator blocks. Now the comparators compare it with indices generated either by a local counter or the spillover counter. If the comparison is successful, the partial products (PP) are sent to the adder tree and then to a shift accumulator to complete MACs. The adder trees are split into two 32-input trees, and the outputs of adder trees are shift-accumulated. The accumulator precision is chosen by the weight precision control (WPC) signal between 14b (for 8b-W) and 11b (for 4-b-W). Finally, a spillover accumulator is present to support edge cases of compressed weights, e.g. uneven sparsity across matrix columns.

The SP-IMC supports three representative compression formats in the column direction for element-wise weight sparsity, namely RL, COO, and N:M sparse encoding. RL and COO formats have different dataflows to support the decode of their respective indices, as shown in Fig. 3. N:M sparse encoding follows the dataflow of COO. In COO mode, each column of the memory array generates an index (through local counters) every cycle that pertains to the index of stored weights in their respective columns and these indices are used to gate accumulations of PP generated in each RG using the comparators. The accumulated PPs are then sent to the shift accumulator block for IA precision compensation semi-bit serially (2-bits/cycle). The dataflow is similar in RL mode as well, but the index generator (counters) now generates the zero count (ZC) between two non-zero weights, and RL compression incurs additional decode hardware in the row direction to specify the non-zero weight position. The index stored in the neighboring CG[n-1] is streamed and is added with the indices in the current CG[n]. The spillover dataflow exists to support corner cases, for e.g., all elements of a matrix row are not always mapped to the same CG, (Fig. 4 COO-CSC mapping) and can "spillover" to neighboring CGs. This arises out of uneven sparsity across matrix columns and is the case for RL, COO and N:M sparsity. It is greatly reduced for N:M sparsity due to fixed M. Fig. 3 shows the pipeline diagram of 4b-IA:4b-W MACs from a CG in the SP-IMC macro. It also shows the priority queue for index handling and the parallelism achieved in a SP-IMC to process sparse compressed MACs. The compression and IMC mapping methodology is elaborated in Fig. 4. Uncompressed mapping is done for convolutions by first flattening the 4D kernels to a 2D weight matrix and is transposed and stored onto the IMC array such that the kernel dimensions and input channel (R, S, C) fall into columns with adder trees and the output channel is mapped in the row direction to support parallel multiplications. We employ a similar approach when it comes to mapping compressed weights. Encoding in column direction is more IMC friendly because it retains column structure while breaking row structures, i.e., breaks accumulations and retains multiplications. Compressed sparse row (CSR) is not very IMC friendly and incurs additional hardware overhead i.e., IA reordering, additional accumulate and WB operations, hence not implemented in this macro. RL mapping is similar to COO, the indices are replaced with ZC. In RL to denote the end of each matrix column, its length is fixed, and the last element of all matrix columns are stored regardless of magnitude. Mapping matrices that have unequal non-zero weight distribution in every column will lead to utilization issues in the IMC. This can be alleviated during training by employing a fine-grained N:M sparsity structure as employed in Nvidia GPUs. Through this method the SP-IMC macro can also achieve a significant speedup by limiting the indices/column. Both RL and COO supports index/ZC precision of 4b and 8b which translates to 255 indices (COO) or 255 zeros (RL) between two non-zero elements.



Die micrograph.

The SP-IMC chip is implemented in 28nm CMOS. We measure the chips at 25^{0}C, between 0.57 and 1.2V supply. SP-IMC achieves 8.4-36.6 TOPS/W for 25% input toggle rate (TR) for fully non-sparse (i.e., only one index per column activating all bit-cells and adder trees nodes) 4b-IA:4b-W MACs and 7.5-115.3 TOPS/W for a pruning ratio of 1:16 with the same TR. Energy efficiency increases by 10% on average when the TR is decreased by 25%. Fig. 5 shows the measurement results. SP-IMC time-multiplexes sparsity. The adder tree activity factors decrease as the number of indices per column (i.e., sparsity) increases, because fewer adders are assigned to each index. SP-IMC shows 3-40× decrease in area when compared to a non-sparse baseline implemented without decode hardware. Fig. 6 shows the comparison with other works, and SP-IMC achieves the highest throughput regardless of sparsity due to dual IA parallelism. This work focuses on sparse compressed storage and so it vastly reduces the number of writes when compared to other works, this also affects the system latency. For the proposed FoM of (TOPS/W × TOPS/mm^2 × (# of weights stored per kb)), our work achieves up to 5.9× higher when compared to the best prior work. In summary, this work introduces a fully digital sparsity integrated IMC macro capable of directly processing COO, RL and N:M encoded sparse representations along with different bit- precisions for both IA (2b, 4b, 8b) and weight (4b, 8b), and with scalable sparsity (4b, 8b) for a variety of DNN workloads.

Acknowledgement:

This work is supported in part by the NSF under Grant No. 2314591, No. 2342726, No. 2349802, No. 2414603, and SRC/DARPA JUMP 2.0 CoCoSys Center.

References:

2024 IEEE Custom Integrated Circuits Conference (CICC) | 979-8-3503-9406-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/CICC60959.2024.10529009

979-8-3503-9406-1/24/$31.00 ©2024 IEEE

[1] D. Wang *et al.*, ISSCC, 2022. [2] H. Mori et al., ISSCC, 2023. [3] J. Oh *et al.*, ESSCIRC, 2023. [4] J. Yue *et al.*, ISSCC, 2023. [5] S. Liu *et al.*, ISSCC, 2023.

[6] H. Fujiwara *et al.*, ISSCC, 2022. [7] A. Zhou *et al.*, ICLR, 2021. [8] L. Yang *et al.*, NeurIPS'22.

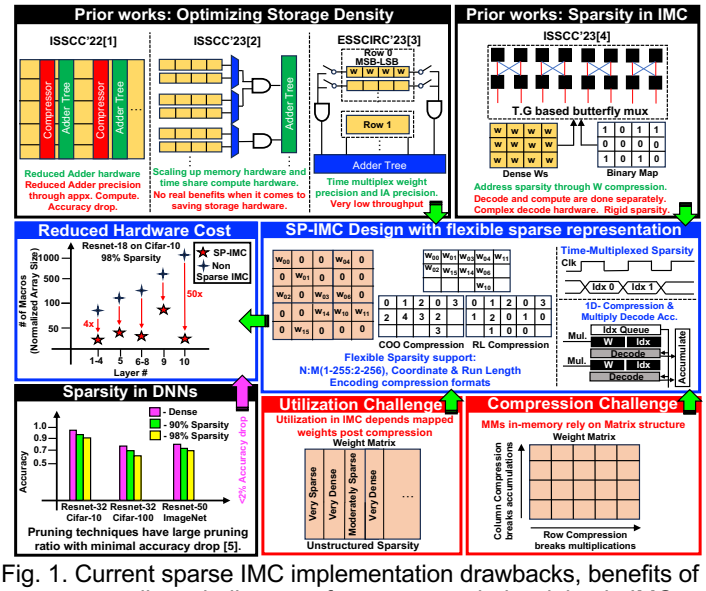


Fig. 1. Current sparse IMC implementation drawbacks, benefits of sparse encoding, challenges of sparse encoded weights in IMC.

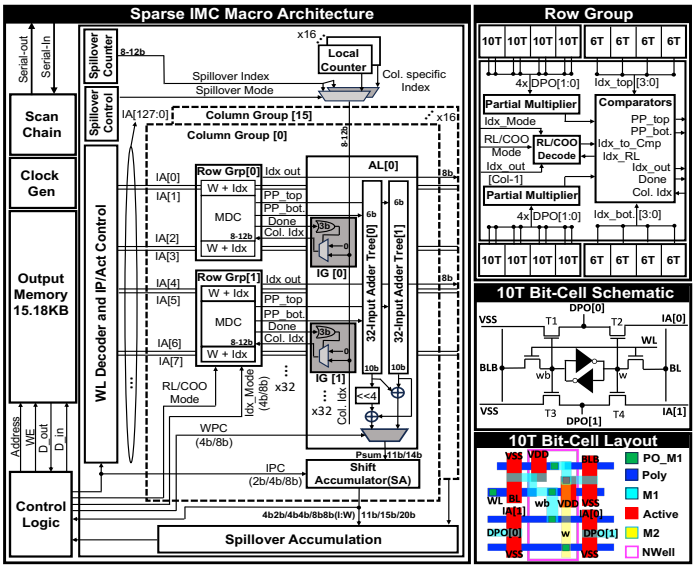


Fig. 2. Overall architecture of SP-IMC Macro, bit-cell schematic, layout, and micro-architecture of in-memory decode hardware.

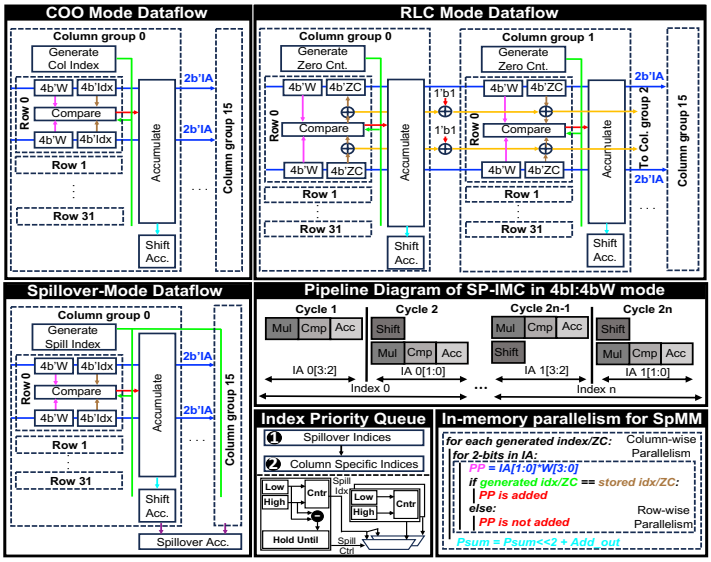


Fig. 3. Dataflow of various modes in SP-IMC, pipeline diagram, index priority queue, SpMM parallelism in memory.

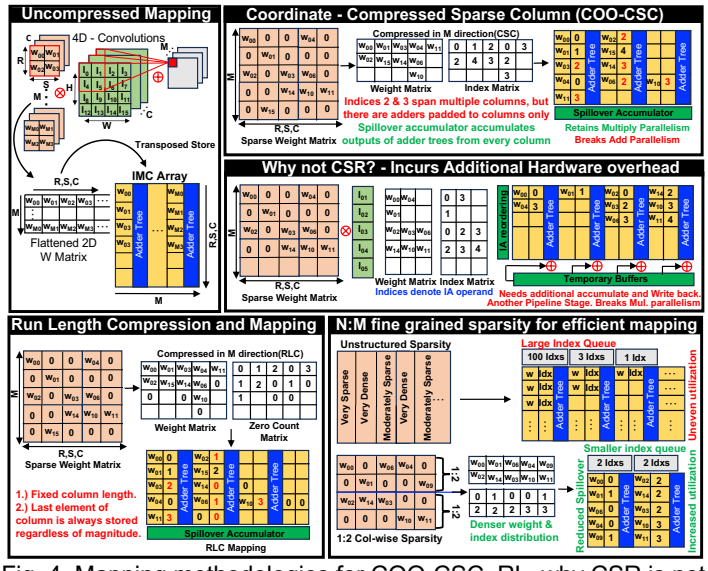
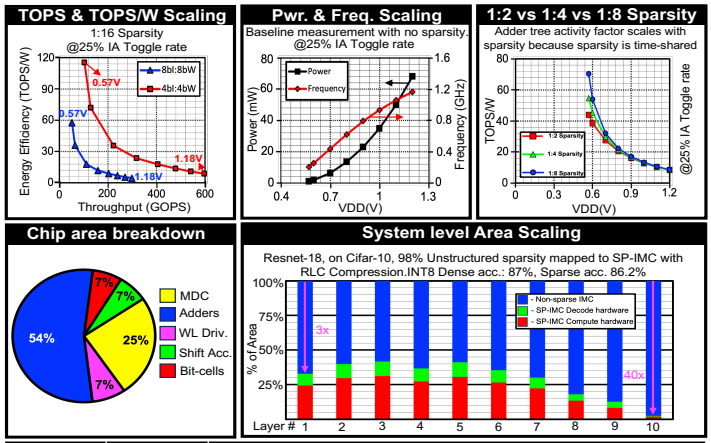


Fig. 4. Mapping methodologies for COO-CSC, RL, why CSR is not IMC friendly and the benefits of N:M sparse encoding.



Dataset	Model	Accuracy(INT8)				
		Dense	N:M Pruning			Unstructured
			1:2	1:4	1:8	90% Sparsity
Cifar-10	Resnet18	93.8%	93.37%	92.49%	91.44%	93.2%
Cifar-100	Resnet18	75.9%	74.64%	74.63%	71.99%	73.9%

Fig. 5. Chip measurement results, accuracy results of pruned DNNs, area breakdown in macro and system level.

Work	ISSCC'22 [1]	ISSCC' 23 [4]	ISSCC'22 [6]	ISSCC'23 [2]	ESSCIRC'23 [3]	**This Work**
Technology	28nm	28nm	5nm	4nm	28nm	28nm
MAC Implementation	Digital	Digital	Digital	Digital	Digital	Digital
IMC Sparsity Support	X	Rigid	X	X	X	RL/COO/N:M
Supply Voltage (V)	0.45-1.10	0.64-1.03	0.5-0.9	0.32-1.1	0.9-1.1	0.57-1.18
Macro Area (mm^2)	0.049	NA	0.0133	0.0172	0.0159	0.24
Clock Frequency (MHz)	250	20-320	360-1440	1490	30-360	201-1160
Bitcell Transistors	8T	8T(55%) 10T(45%)	12T	8T x 2bit +OAI	6T+0.5T	6T+4T(50%) 6T(50%)
Array Size(b)	16K	1.15M	64K	54K	16K	4K(Weights) + 4K(Index)
Bit Precision	IA:1-4b W:1b	INT8	IA: 1-8b W:4b	IA: 8/12/16 W: 8/12	IA: 1-8 W: 8	IP:2b/4b/8b W:4b/8b
Full output precision	No	Yes	Yes	Yes	Yes	Yes
Performance(GOPS)[1,2,7]	62.5*	22.9*	104.735	127.15	0.95-11.6	41.29-238.86
Peak Energy Efficiency[2,7] (TOPS/W)	9.6-15.5	15.6[4]/70.37[5] (System)	17.5-63	87.4[8] 41.3[9]	22.4-60.4	4.38-57.67[6]
Compute Density[2,3,7] TOPS/mm^2	2.59	0.85	0.44-1.76	0.27-1.01	0.12-1.46	0.21-1.2
FoM[2,7,3]	3.182K	2.97K[4] 3.25K[5]	3.942K	3.219K	0.9K-4.1K	11K-24K(1:16) 5.5K-12K(1:8)

[1] Normalized to 8Kb. [2] One operation is either 8b multiplication or addition. [3] Normalized quadratically to 28nm. *Estimated from previous works. [4] 75% Sparsity. [5] 92% Sparsity. [6] 93.75% Sparsity (1:16 Sparsity). GOPS Calculation (8b8b/4b4b): 32 (or) 64 (Rows) x 16 (Columns) x 2 (MAC)/Latency. [7]Excludes write energy/latency otherwise incurred by other works for a scaled-up matrix that fits in SP-IMC and not in other works. [8] @ 12.5% TR. [9] @ 50% TR [10] @25% TR.

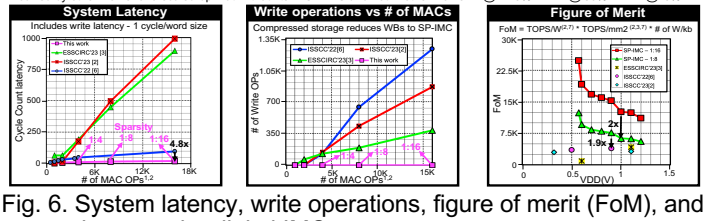


Fig. 6. System latency, write operations, figure of merit (FoM), and comparison to prior digital IMCs.