14.2 A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration

Jingcheng Wang, Xiaowei Wang, Charles Eckert, Arun Subramaniyan, Reetuparna Das, David Blaauw, Dennis Sylvester

University of Michigan, Ann Arbor, MI

Data movement and memory bandwidth are dominant factors in the energy and performance of both general purpose CPUs and GPUs. This has led to extensive research focused on in-memory computing, which moves computation to where the data is located. With this approach, computation is often performed on the memory bit-lines in the analog domain using current summing [1-3], which requires expensive analog-to-digital and digital-to-analog conversions at the array boundary. In addition, such analog computation is very sensitive to PVT variations, limiting precision. More recently, full-rail (digital) binary in-memory computing was proposed to avoid this conversion overhead and improve robustness [4, 5]. However, both prior in-memory approaches suffer from the same major limitations: they accelerate only one type of algorithm and are inherently restricted to a very specific application domain due to their limited and fixed bit-width precision and non-programmable architecture. Software algorithms, on the other hand, continue to evolve rapidly, especially in novel application domains, such as neural networks, vision and graph processing, making rigid accelerators of limited use. Furthermore, most available SRAM in today's chips is located in the caches of CPUs or GPUs. These large CPU and GPU SRAM stores present an opportunity for extensive in-memory computing and have, to date, remained largely untapped.

In this paper, we present a general purpose hybrid in-/near-memory compute SRAM (CRAM) that combines the efficiency of in-memory computation with the flexibility and programmability necessary for evolving software algorithms. CRAM augments conventional SRAM in a CPU with vector-based, bit-serial [6, 7] in-memory arithmetic. It can accommodate a wide range of bit-widths, from single to 32b or 64b, and operation types, including integer and floating point addition, multiplication and division. To maintain compatibility with CPU/GPU operation, CRAM writes/reads operands conventionally with horizontal word-lines and vertical bit-lines. Then, using a transposable bitcell [8], CRAM operates directly on the stored operands in memory with additional horizontal compute bit-lines. This enables the same bit position from two vectors elements to be simultaneously accessed on a single bit-line. Logic operations are performed on the bit-line (in-memory), while small additional in-column logic (near-memory with 4.5% SRAM bank area overhead) enables carry-propagation between successive bit-serial calculations, enabling multi-bit arithmetic operations in SIMD fashion across all vectors of elements. To maintain versatility, the memories can function either as traditional or compute memories. The approach was implemented in a small IoT processor in 28nm CMOS, consisting of a Cortex-M0 CPU and 8 CRAM banks of 16KB each (128KB total). The system achieves 475MHz operation and, with all CRAMs active, produces 30GOPS or 1.4GFLOPS on 32b operands for graph, neural, and DSP applications.

Figure 14.2.1 shows the overall organization of the IoT processor. The ARM core can access all 8 memory banks and load/store data using the horizontal word-lines and vertical bit-lines. Then, in-memory instructions can be streamed from one bank to one or more compute-configured banks, while the M0 simultaneously performs other processing with the remaining memory banks. Banks performing in-memory computing use the horizontal compute bit-lines (CBLs) and vertical compute word-lines (CWLs).

Figure 14.2.2 shows the architecture of the 128×256 CRAM sub-array, which is one quarter of a 16KB CRAM macro. An 8T transposable bitcell is used to provide bidirectional access. Fig. 14.2.2 shows an example operation of the data flow for a 1b addition performed in 1 cycle of the bit-serial computation. Here, we add the second bit positions of vector A (A_1=0) and vector B (B_1=1) with carry-in C (=1) from the previous cycle, and store the result back to vector D. First, the CRAM instruction decoder receives the ADD instruction and the 3 column addresses of bits A_1, B_1 and D_1. It activates the CWLs of A_1 and B_1 simultaneously to compute 'A AND B' on CBL and '$\overline{A}$ AND $\overline{B}$' on CBLB. Since A=0 and $\overline{B}$=1, both CBL and CBLB discharge. Then, after the dual sense amps, the results propagate to the near-memory logic located at the end of each CBL. The NOR gate generates 'A XOR B', which combined with C_{in} from the carry latch produces Sum=0 and C_{out} =1. Sum is then written back to D, and C_{out} is stored in the carry latch, which provides C_{in} for the next cycle, thus completing one full bit-serial addition in one clock cycle.

Figure 14.2.3, left, shows how two vectors of 2b numbers (A and B) are added bit-by-bit starting from the least significant bit (LSB). Note that while only one bit of a multi-bit operand is processed in each cycle, all compute bit-lines operate simultaneously, resulting in massive parallelism (2048 CBLs in our design). Subtraction is performed by first inverting B and then adding to A with C_{in} pre-set to 1. As shown in Fig. 14.2.3, multiplication is more complicated as it requires predication. For this, the tag latch (Fig. 14.2.2) is used to enable the write-back driver, resulting in a conditional copy/addition. First, 4 empty columns in the array are reserved for the product and initialized to zero. In the first cycle, the LSB of the multiplier is loaded to the tag latch. In cycles 2 and 3, the multiplicands are copied to product columns only if their tag is 1. In cycle 4, the second bit of the multiplier is loaded to the tag latch. In the next 2 cycles, for rows with tag = 1, the multiplicands are added to the *second* and *third* bits of the product, shifting the multiplicands by 1 to account for the multiplier bit position. Finally, we store C_{out} in the most significant bit (MSB) of the product to complete the multiplication. Note that partial products are implicitly shifted as they are added using appropriate bit addressing in the bit-serial operation and no explicit shift is performed. Division is conducted similarly by implicit shifting and subtraction from a partial result. Floating point arithmetic is implemented using repeated integer add/sub/mult/div with predication. Fig. 14.2.3 provides a list of supported computations and their performance, demonstrating both the versatility of CRAM and its high performance due to bit-line parallelism.

Figure 14.2.4 shows measurement results from the prototype chip fabricated in 28nm CMOS that contains 8 CRAM banks (128KB memory with 2048 computing rows) and a Cortex-M0 processor. The figure shows measured frequency and energy efficiency of 8b addition and multiplication across supply voltage. At 1.1V the maximum frequency of 475MHz results in 122GOPS for 8b addition and 9.4GOPS for 8b multiplication. The best energy efficiency is achieved at 0.6V and 114MHz, resulting in 0.56TOPS/W for 8b multiplication and 5.27TOPS/W for 8b addition. Fig. 14.2.4 shows measured frequency and leakage power distributions for 21 measured dies.

Figure 14.2.5 shows the performance of the test chip for diverse computationally intensive tasks ranging from neural networks to graph and signal processing. The total latency in cycles is compared with a baseline operation, where CRAMs are only used as data memories and the computation is entirely performed on the ARM CPU. The first benchmark is the 1st convolutional layer from Cuda-convnet and the second is the last fully connected layer from AlexNet. Due to their size, these layers must be executed in multiple smaller sub-sections. The third application consists of 512 simultaneous 32-tap FIR filters and the fourth application performs traversal of a directed graph represented by a 192×192 adjacency matrix. The workload breakdown shows the percentage of time spent on input loading and output loading vs. in-memory computation. Speedup, compared to executing the same workload with the ARM Cortex-M0, varies from 7.2-to-114×, with the greatest gains obtained when the operation is compute-heavy and low on input/output movement.

Figure 14.2.6 compares the proposed approach with other state-of-the-art in-memory accelerators. The proposed work is the only solution to provide a wide range of instructions and flexible bitwidth. It repurposes the memory storage already available in processors, thereby accelerating computation while maintaining programmability.

Acknowledgements:
We gratefully acknowledge TSMC University Shuttle Program for chip fabrication. This work was supported in part by ADA, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References:

[1] J. Zhang, et al., "In-Memory Computation of a Machine Learning Classifier in a Standard 6T SRAM Array," *IEEE JSSC*, vol. 52, no. 4, pp. 915-924, 2017.
[2] A. Biswas, et al., "Conv-RAM: An Energy-Efficient SRAM with Embedded Convolution Computation for Low-Power CNN-based Machine Learning Applications," *ISSCC*, pp. 488-489, 2018.
[3] S. Gonugondla, et al., "A 42pJ/Decision 3.12TOPS/W Robust In-Memory Machine Learning Classifier with On-Chip Training," *ISSCC*, pp. 490-491, 2018
[4] W. Khwa, et al., "A 65nm 4Kb Algorithm-Dependent Computing-in-Memory SRAM Unit-Macro with 2.3ns and 55.8TOPS/W Fully Parallel Product-Sum Operation for Binary DNN Edge Processors," *ISSCC*, pp 496-497, 2018.
[5] Y. Zhang, et al., "Recryptor: A Reconfigurable In-Memory Cryptographic Cortex-M0 Processor for IoT," *IEEE Symp. VLSI Circuits*, 2017.
[6] K. Batcher, "Bit-Serial Parallel Processing Systems," *IEEE Trans. on Computers*, vol. 31, no. 5, pp. 377-384, 1982.
[7] C. Eckert, et al., "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," *ACM/IEEE ISCA*, pp. 383-396, 2018.
[8] J. Seo, et al., "A 45nm CMOS Neuromorphic Chip with a Scalable Architecture for Learning in Networks of Spiking Neurons," *IEEE CICC*, 2011.

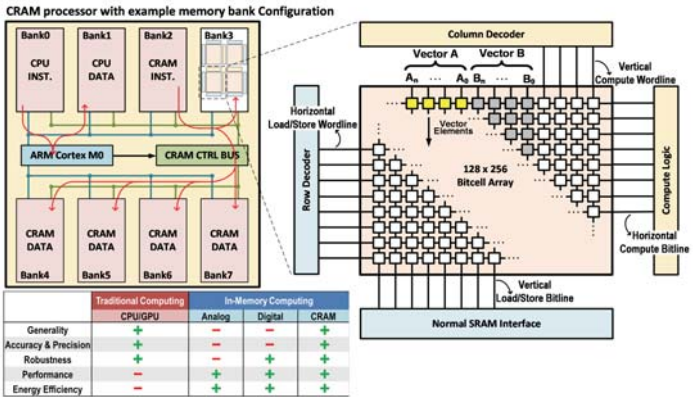


Figure 14.2.1: Chip architecture and storage and computation of data in transposable memory array.

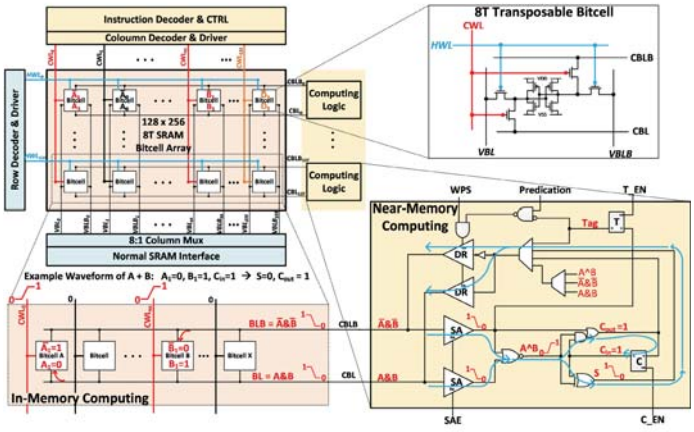
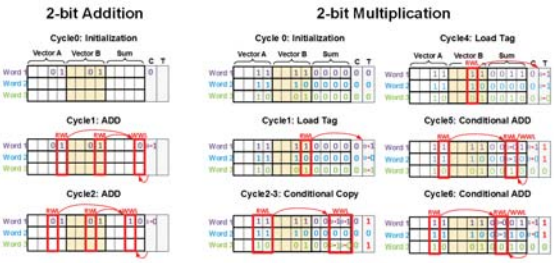


Figure 14.2.2: CRAM array architecture (top-left), 8T transposable bitcell (top-right), In-memory computing part (bottom-left) and near-memory computing part (bottom-right) of 1-bit addition. Addition of near-memory logic increased array size by 4.5%.



Type	Operation	# Cycles	32-bit Performance[2]	8-bit Performance[2]
Logic	AND	N[2]	30.4GOPS	122 GOPS
	NOR	N		
	XOR	N		
	NAND	N		
	OR	N		
	XNOR	N		
Integer	Add	N	30.4 GOPS	122 GOPS
	Sub	2N	15.2 GOPS	60.8 GOPS
	Mult	$N^2 + 5N - 2$	0.83 GOPS	9.40 GOPS
	UDiv	$1.5N^2 + 5.5N$	0.57 GOPS	6.97 GOPS
32-bit Float Point	Add/Sub	497B	0.20 GFLOPS	/
	Mult	67I	1.43 GFLOPS	/
	Div	69I	1.40 GFLOPS	/
Comparison	Equal	2N+1	14.9 GOPS	57 GOPS
	Greater/Less	2N+1		
	Search	N	30.4 GOPS	122 GOPS

1. N is the bit-width of data
2. Measured performance on 128KB CRAM test chip at 475MHz

Figure 14.2.3: 2-bit addition cycle-by-cycle demonstration (top-left), 2-bit multiplication cycle-by-cycle demonstration (top-mid & right), and list of CRAM instructions and its performance (bottom).

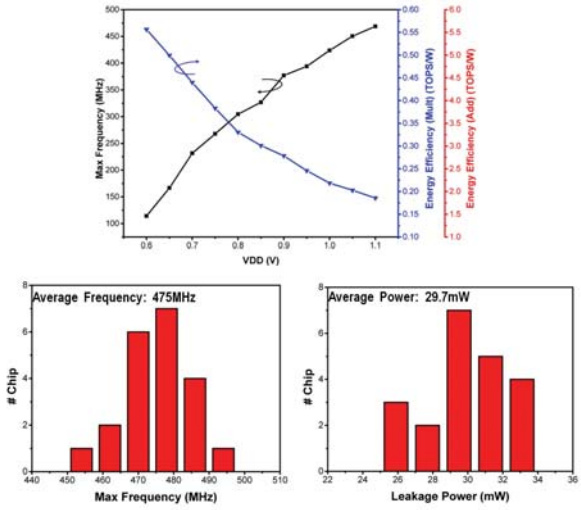


Figure 14.2.4: Frequency and energy efficiency of 8-bit multiplication and addition at different VDD (top), maximum frequency and leakage power distribution of 21 dies at 1.1V (bottom).

	CONV		FC		FIR		GRAPH	
Testbench	Cuda-Convnet 1st layer		AlexNet last layer		512 32-Tap Filter		Nearest Neighbor Traversal	
Input Size	24x24x3		24x1		32x10		192x192	
Parameter Size	5x5x3x64		1000x24		512x32		0	
Output Size	1x1x64		1000x1		512x10		192x192	
Bit Precision	8		8		4		1	
# Array used for compute	3		6		2		2	
# Wordline for computation	375		1000		512		192	
	cycle #	percentage	cycle #	percentage	cycle #	percentage	cycle #	percentage
Total Latency	39,628	100	33,434	100	251,290	100	1,572,628	100
Input loading	18,323	46.2	24	0.07	320	0.13	1,152	0.07
CRAM Compute	3,459	8.7	21,267	63.6	184,020	73.2	1,556,458	99.0
output readout	17,846	45.0	12,143	36.3	66,950	26.6	15,018	0.95
	cycle#	speedup	cycle #	speedup	cycle #	speedup	cycle #	speedup
Baseline	287,073	7.24x	1,174,032	35.1x	8,120,415	32.3x	164,456,448	114x

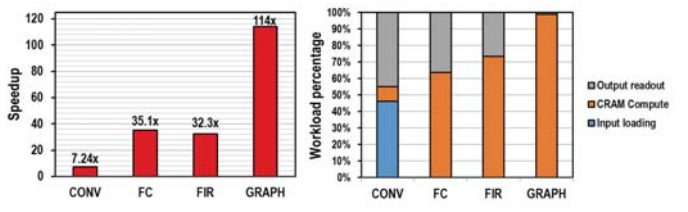
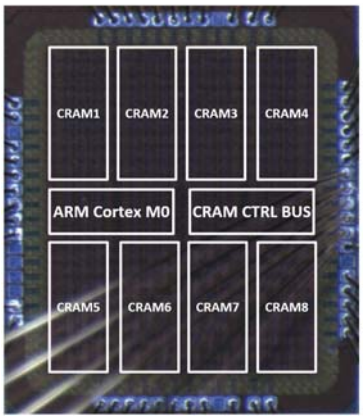


Figure 14.2.5: Performance comparison between CRAM and baseline scenario (top), workload breakdown (bottom).

	This Work	JSSC2017 [1]	ISSCC2018 [2]	ISSCC2018 [3]	ISSCC2018 [4]	VLSI2017 [5]
Technology	28nm	130nm	65nm	65nm	65nm	40nm
Supply Voltage	0.6~1.1V	1.2V	0.9~1.2V	0.65~1V	0.6~1V	0.65~0.9V
SRAM Macro size	16KB	2KB	2KB	16KB	0.5KB	8KB
SRAM bitcell	8T	6T	10T	6T	6T	10T
Method of Computing	Digital	Analog	Analog	Analog	Digital	Digital
Type of Supported Functions	Logic/Add/Sub/Mult/Div/FP	Add/Mult	Add/Mult	Add/Mult	Add/Mult	Logic
Bit precision	Arbitrary	5b (input) 1b (weight)	7b (input) 1b (weight)	16b (train) 8b (infer)	1b (input) 1b (weight)	Arbitrary
Die Area (mm2)	2.7	0.36	0.067	1.44	-	1.28
Max Freqency (MHz)	475	50	6.7	1000	435	90
Normalized Performance (GOPS)*	32.7	40.5	1.17	-	-	-
Performance per unit Area (GOPS/mm2)**	27.3	114	17.5	-	-	-
Normalized Energy Efficiency (TOPS/W) *	0.55 (mult) 5.27 (add)	0.16 (mult) 4.58 (add)	3.07	3.12	0.87	-

*All normalized to 8-by-8 bit Multiply and Add Operation
** Using SRAM area only

Figure 14.2.6: Comparison table.



Technology	28nm CMOS
Processor	ARM Cortex M0
Bitcell Size	0.405um x 1.93um (Logic-rule)
Chip Size	1.5 x 1.7 mm²
Supply Voltage	0.6 ~ 1.1V
Memory Capacity	128KB (8 x 16KB)
SRAM Macro Size	16KB
SRAM Sub Array	128 rows x 256 colums
Clock Frequency	475MHz @1.1V
Average power	105mW @1.1V

Figure 14.2.7: Die photo.

978-1-5386-8531-0/19/$31.00