Appears in the Proceedings of the IEEE Int'l Symp. on Circuits and Systems (ISCAS), May 2020

Towards a Reconfigurable Bit-Serial/Bit-Parallel Vector Accelerator Using In-Situ Processing-In-SRAM

Khalid Al-Hawaj, Olalekan Afuye, Shady Agwa, Alyssa Apsel, Christopher Batten
School of Electrical and Computer Engineering, Cornell University, Ithaca, NY
{ka429, ota2, shady.agwa, aba25, cbatten}@cornell.edu

***Abstract*—Vector accelerators can efficiently execute regular data-parallel workloads, but they require expensive multi-ported register files to feed large vector ALUs. Recent work on in-situ processing-in-SRAM shows promise in enabling area-efficient vector acceleration. This work explores two different approaches to leveraging in-situ processing-in-SRAM: BS-VRAM, which uses bit-serial execution, and BP-VRAM, which uses bit-parallel execution. The two approaches have very different latency vs. throughput trade-offs. BS-VRAM requires more cycles per operation, but is able to execute thousands of operations in parallel, while BP-VRAM requires fewer cycles per operation, but can only execute hundreds of operations in parallel. This paper is the first work to perform a rigorous evaluation of bit-serial vs. bit-parallel in-situ processing-in-SRAM. Our results show that both approaches have similar area overheads. For 32-bit arithmetic operations, BS-VRAM improves throughput by 1.3–5.0× compared to BP-VRAM, while BP-VRAM improves latency by 3.0–23.0× compared to BS-VRAM.**

I. Introduction

Vector accelerators are seeing a resurgence in both general purpose and domain-specific processing [15, 17, 18]. These accelerators can achieve high performance on well-structured workloads by using a complex vector ALU and register file. To keep the vector ALU busy, vector register files are usually highly multi-ported which incurs significant area and energy overheads. Recent work on in-situ processing-in-SRAM attempts to reduce these overheads by fusing the vector ALU and register file. In-situ processing-in-SRAM uses bit-line computation to perform basic bit-wise logical operations in a single read of a traditional SRAM [8, 9]. Each SRAM column can be transformed into a bit-serial ALU by adding extra logic, multiplexing, and state elements in the peripheral circuitry. Alternatively, a set of SRAM columns can be grouped into a bit-parallel ALU by adding bit-parallel logic in the peripheral circuitry instead.

In this paper, we provide a detailed implementation for bit-serial vector RAM (BS-VRAM) and bit-parallel vector RAM (BP-VRAM) as two representative design points for processing-in-SRAM. These two flavors of VRAM support a variety of micro-operations for implementing macro-operations. Starting with an implementation of a traditional 28 nm 6T-SRAM in OpenRAM [6], we designed and laid out the additional peripheral circuitry required to implement BS-VRAM and BP-VRAM. Although surprisingly both designs have comparable area overhead, they have very different performance characteristics. For a 32-bit MAC operation, BS-VRAM has 5× higher throughput (6.4 GOPS) than BP-VRAM (1.3 GOPS), while BP-VRAM can achieve 6.5× lower latency (197.9 ns) when compared to BS-VRAM (1281 ns). BS-VRAM consumes lower energy per operation, but we discuss possible techniques to help close this gap. As the two designs require similar peripheral circuitry, this work can be seen as a step towards building a reconfigurable bit-serial/bit-parallel vector accelerator that is able to achieve either high-throughput or low-latency depending on the application requirements.

Our main contributions are: (1) a detailed circuit-level design of BS-/BP-VRAM in 28nm technology; (2) implementation of 17 macro-operations in BS-/BP-VRAM using micro-operations; (3) a detailed study of the trade-offs in area, cycle time, latency, throughput, and energy for BS-VRAM vs. BP-VRAM. To our knowledge, this is the first work to rigorously explore the trade-offs between a bit-serial vs. bit-parallel approach to in-situ processing-in-SRAM.

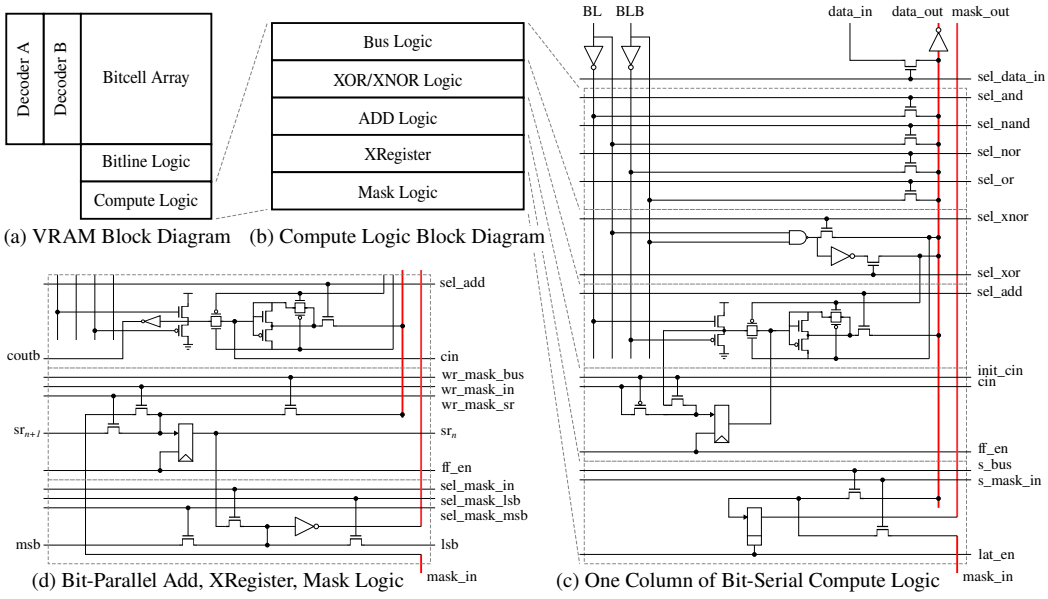
II. VRAM Circuits

BS-VRAM and BP-VRAM start with a basic 6T SRAM with support for bit-line computation (i.e., extra decoder and reconfigurable single-ended/differential sense amplifiers as in [8,9]). Bit-line computation simplifies the required peripheral logic to implement BS-VRAM and BP-VRAM. Figure 1 shows the additional peripheral circuitry required to enable BS-VRAM and BP-VRAM beyond what is necessary for bit-line computation.

A. Bit-Serial Compute Logic (BSCL)

The inputs to the BSCL are the output of each sense-amplifier and its complement. Bit-line computation provides bit-wise logical AND, NAND, OR, and NOR on these inputs.

The BSCL is composed of the following blocks. *Bus Logic:* BSCL uses a distributed bus with NMOS pass-transistors to choose between basic bit-wise logical operations (i.e., AND, NAND, OR, NOR). *XOR/XNOR Logic:* Computing XOR and NOR in BSCL requires an additional NAND gate and inverter. *ADD Logic:* BSCL uses a modified serial Manchester carry chain (MCC). As addition is computed bit serially, the carry out is stored for the subsequent cycle while previous carry is used as carry in. The carry in is XOR'ed with the bitwise logical XOR to compute the sum. *XRegister:* XRegister's input is multiplexed to choose either the carry out from the ADD logic or an input carry, which enables initializing the carry to zero for an addition or one for a subtraction. The output of the XRegister is the input to the ADD logic. *Mask Logic:* The mask logic in BSCL is a latch with a multiplexer to choose either the unbuffered bus or an input mask. The output of the latch is the mask used by the SRAM for writes. In a conventional write, the input mask will be chosen to be stored in the latch.



(a) VRAM Block Diagram (b) Compute Logic Block Diagram

(d) Bit-Parallel Add, XRegister, Mask Logic (c) One Column of Bit-Serial Compute Logic

Figure 1. VRAM Circuits – High-level block diagram of VRAM microarchitecture showing a column slice of circuit-level details of all the different logic blocks of both VRAM flavors. BP-VRAM shares many logic blocks implementation with BS-VRAM, shown in (c), and only differ in three blocks that are shown in (d). The boxed logic blocks are explained in section II from top to bottom. The blocks are: bus logic, XOR/XNOR logic, ADD logic, XRegister, and mask logic.

B. Bit-Parallel Compute Logic (BPCL)

Bit-parallel compute logic reuses all but the last three blocks from BSCL. *ADD Logic:* Bit-parallel addition requires a carry propagation for which BPCL uses an MCC. The addition result is computed by XOR'ing the carry-in of every column with the bitwise logical XOR value. For better performance, lightweight buffering along the carry chain is inserted by using the inversion property of the adder. *XRegister:* The XRegister can act as shift register for the multiplier, controlling whether a µop is conditionally executed. The input of every flip-flop is multiplexed choosing either the input mask, the output of the compute logic on the bus, or the output of the XRegister of the column to the left (thus shifting right). *Mask Logic:* To generate the appropriate mask of every column, a multiplexer is added to select between the XRegister output of the current column, the XRegister output of the first column of the element (i.e., least-significant bit "LSB"), or the XRegister output of the last column of the element (i.e., most-significant bit "MSB"). For masking, multiplication uses the LSB while comparators use the MSB.

III. VRAM Micro-Programming

BS-VRAM and BP-VRAM implement single-cycle primitives µoperations (µops). This section describes the supported µops and illustrates how a sequence of µops can be used to implement a complex macro-operation.

A. Micro-Operations

Normal SRAM read and write in a VRAM use the `rd` and `wr` µops. The remaining µops are as follows:

Bit-line Compute (`blc`): This µop performs a read and uses the compute logic to generate AND, NAND, OR, NOR, and ADD. The sense-amps latch their outputs, meaning these logic value can be reused until the next `blc` or `rd`.

Writeback (`cond.wb.src`): After executing a `blc`, a writeback µop writes the selected source back to the SRAM. The writeback µop includes two parameters: a condition and a source selection. The condition (supported only by BP-VRAM) specifies whether the mask bit of a column would be set to: the corresponding `mask_in` bit, the element's LSB, or its MSB. BS-VRAM and BP-VRAM utilize the already existing write mask (native to SRAMs) to conditionally writeback.

Write to Mask (`wr_mask.src`): Instead of writing to the SRAM, BS-VRAM and BP-VRAM allow writing to the mask state element (which is the latch in BS-VRAM and the XRegister for BP-VRAM). An algorithm can generate the mask dynamically for subsequent conditional writes.

Shift Right Logical (`srl`): This µop (supported only by BP-VRAM) shifts the content of the XRegister to the right by one bit. Multiplication uses the LSB of the XRegister as a mask for conditional addition. By shifting, the algorithm can step through the bits of the multiplier from the LSB to the MSB.

Jump if not done (`j_n_done_{0,1}`): There are two counters for control flow, and each counter can be initialized to the desired bitwidth (e.g., 8, 32). This control µop decrements one of these counters (indicated with the suffix). If the counter is zero, the counter is reset and execution falls through to the next µop. If the counter is not zero, execution jumps to the label.

B. Macro-Operations

Both flavors of VRAM utilize arithmetic and control µops to implement multi-cycle macro-operations (see Table I). In BS-VRAM, single-loop control is used to implement simple macro-operations (e.g., bit-wise logic and addition) while nested-loop control is required for complex macro-operations (e.g., multiplication and division). In BP-VRAM, bit-parallel hardware is used for simple macro-operations while single-loop control is only required to implement complex macro-operations using a mixed bit-serial/bit-parallel approach. Figure 2 shows the implementation of addition and multiplication in both BS-/BP-VRAM. Addition in BS-VRAM is a loop

TABLE I. SUPPORTED MACRO-OPERATIONS

Macro-Operation	Cycle Count		# Temporary Rows	
	BS-VRAM	BP-VRAM	BS-VRAM	BP-VRAM
`add`	64	2	0	0
`sub`	128	4	0	0
`and,nand,or nor,xor,xnor`	64	2	0	0
`mul`	1185	133	0	1
`mac`	1153	132	0	1
`udiv`	1712	519	5	1
`rem`	1680	390	4	2
`slt,sle,sgt,sge`	162	6	1	0
`seq`	96	11	1	1

```
  loop:
1   blc          addr_a, addr_b
2   wb.add       addr_c
    ; j_n_done_0 loop
```

(a) `add` in BS-VRAM

```
1   blc          addr_a, addr_b
2   wb.add       addr_c
```

(b) `add` in BP-VRAM

```
1   set_cin      1
2   wb_mask      <(1)
  init:
3   wr           addr_C <(0)
    ; j_n_done_0 init
  iter:
4   rd           addr_B
5   wr_mask.and
  iter_add:
6   blc          addr_C, addr_A
7   wb.add       addr_C
    ; j_n_done_1 iter_add
8   j_n_done_0   iter
```

(c) `mul` in BS-VRAM

```
1   wr           addr_c <(0)
2   rd           addr_a
3   wb.and t0
4   rd           addr_b
5   wr_mask.and
  iter:
6   blc          addr_c, t0
7   msb.wr.add   addr_c
    ; srl
8   rd           t0
9   wb.add       t0
    ; j_n_done_0 iter
```

(d) `mul` in BP-VRAM

Figure 2. `add` and `mul` Macro-Operations – Each macro-op is used as `op c, a, b`, where `a` and `b` are inputs and the resulting value would be stored in `c`. `addr_X` is the row address of variable `X`. Labels are at beginning of line with colon. Control μops use labels as destination to redirect execution flow. `<(X)`: indicates setting the data_in port of the VRAM to the specified value `X`. Semicolon (`;`) indicates a mini-op composed of two μops.

where each iteration uses a bit-line compute μop (`blc`) followed by writing back the result of the ADD logic (`wb.add`). Whereas, addition in BP-VRAM only requires one bit-line compute μop (`blc`) and one write back (`wb.add`).

IV. EVALUATION

This section discusses the evaluation methodology as well as the area, performance, and energy results of BS-/BP-VRAM.

A. Methodology

OpenRAM [6] was adapted to produce a layout for a traditional 6T-SRAM targeting a 28nm technology node. OpenRAM was also extended to produce a layout for bit-line compute-capable SRAM (BC-SRAM) by adding an extra decoder and reconfigurable single-ended/differential sense-amplifiers. Finally, OpenRAM was used to produce the layout for BS-VRAM and BP-VRAM based on BC-SRAM by adding the layout for the compute logic.

Figure 3 shows the layout of BS-VRAM. This layout is used to create a detailed extracted netlist for cycle time and

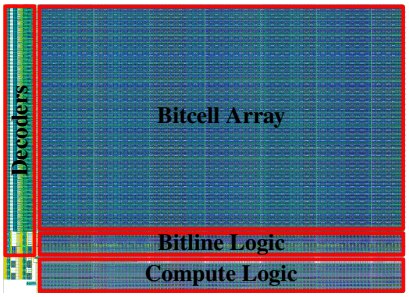


Figure 3. BS-VRAM Layout – BP-VRAM have identical floorplan and similar sizing.

TABLE II. SUB-ARRAY AREA COMPARSION

Design	Area (μm^2)
SRAM	22,868 (1.00×)
BL-SRAM	23,722 (1.04×)
BS-VRAM	24,940 (1.09×)
BP-VRAM	24,786 (1.08×)

SRAM = Traditional 6T SRAM, BL-SRAM = bit-line compute capable SRAM, BS-VRAM = bit-serial vector RAM, BP-VRAM = bit-parallel vector RAM.

energy analysis. The energy of each μop is estimated by averaging the energy consumed for 10 random inputs. Macro-operation energies are estimated by accumulating the energy of all executed μops. The cycle time is estimated by simulating the worst-case inputs on the extracted netlist.

B. Results

Since OpenRAM's SRAM implementation does not use pushed design rules [7, 10], bitcells are roughly 80% larger compared to an equivalent SRAM generated by a commercial memory compiler. Due to larger bitcells, OpenRAM's SRAM consumes around 1.5× write energy and 3× read energy. The read energy difference is attributed to less optimized sense-amplifiers that require a larger voltage drop on the bitlines. OpenRAM's SRAM achieves an operating frequency of 1.1 GHz compared to commercial SRAM compiler, which can achieve up to 2 GHz. While using OpenRAM without pushed design rules obviously incurs significant overhead compared to a commercial memory compiler, OpenRAM also enables detailed layout design of BS-/BP-VRAM and rigorous comparative analysis. Our techniques will also apply to pushed design rule SRAMs.

Table II shows that BS-/BP-VRAM incur marginal area overhead compared to a traditional 6T-SRAM ($\leq$10%). BS-VRAM operates at slightly lower clock frequency (900 MHz) compared to the SRAM (1.1 GHz) mainly due to the switch from a differential sense-amp to a reconfigurable sense-amp. The MCC is on the critical path in BP-VRAM resulting in an operating frequency of 645 MHz. Table III shows BP-VRAM, despite its lower frequency, can achieve 23× lower latency compared to BS-VRAM. However, BS-VRAM is able to achieve 5× higher throughput. Even with lower latency, BP-VRAM struggles to compensate for fewer ALUs compared to BS-VRAM. Per sub-array, BS-VRAM has 32× more ALUs, while BP-VRAM reduces latency by only 23×.

Table III shows the energy comparison between BS-VRAM and BP-VRAM for `add` and `mul` operating on 8-bit and 32-bit data. BS-VRAM can reduce the number of executed μops for 8-bit data (and thus the energy), while BP-VRAM essentially always operates on 32-bit data. For example, BS-VRAM requires only 1.2 pJ for an 8-bit addition, while BP-VRAM requires 4.8 pJ. BP-VRAM's energy efficiency could be improved by using transmission gates to segment the MCC. For 32-bit data, BP-VRAM's `add` energy is similar to BS-VRAM, but bit-parallel `mul` consumes more en-

TABLE III. DETAILED COMPARSION TABLE BETWEEN BS-VRAM AND BP-VRAM

		8-bit		32-bit	
		BS	BP	BS	BP
Latency (ns)	`add`	17.8	**3.1** (5.8×)	71.1	**3.1** (23.0×)
	`mul`	116.7	**57.4** (2.0×)	1316.7	**206.2** (6.4×)
Throughput (GOPS)	`add`	**14.4** (5.5×)	2.6	**3.6** (1.4×)	2.6
	`mul`	**2.2** (15.7×)	0.14	**0.2** (5.0×)	0.04
Energy (pJ/Op)	`add`	1.2	4.8	4.7	4.8
	`mul`	9.0	58.1	112.5	221.3

ergy due the multiplicand shifts required to generate partial-products (i.e., more reads and writes).

V. COMPARISON TO PRIOR WORK

Processing-in-memory has shown promise in increasing performance and energy efficiency by moving the computation closer to the memory [2,3,11–14,16,20,22]. Specifically, recent work on processing-in-SRAM uses bit-line compute to push logic into the SRAM with minimal area-overhead. Prior work demonstrates the potential for bit-line compute by transforming the cache subsystem of a chip multi-processor into different engines: a bit-parallel bit-wise logic engine [1]; a fixed-function accelerator for neural networks [4]; and a SIMT accelerator [5].

Jeloka et al. were the earliest to introduce the concept of bit-line compute, where computations are performed digitally inside the SRAM [8, 9]. Wang et al. propose CRAM which extends bit-line compute to an 8T SRAM and include support for integer arithmetic [19]. Instead of performing the computation vertically, the 8T bitcell allows the computations to be performed horizontally in the compute bitlines. Additional functionality is implemented using bit-serial logic in the periphery with appropriate multiplexing. Sub-array banking helps mitigate area overhead by sharing column decoders and compute logic with neighboring sub-arrays.

Table IV shows a comparison of BS-VRAM and BP-VRAM against prior work. Despite CRAM's single-cycle read-write, BS-VRAM is has higher throughput because: all sub-arrays in BS-VRAM can be active at the same time while CRAM can only use half of the subarrays due to banking; per sub-array, BS-VRAM has twice computing bit-lines resulting in twice the ALUs compared to CRAM; and BS-VRAM operates at a higher frequency. The use of a 6T bitcells help BS-VRAM and BP-VRAM in occupying only half the area of CRAM. Energy efficiency of CRAM is roughly 3× higher than BS-VRAM because CRAM uses lower wordline voltage while performing a bit-line computation as well as utilizing a more optimized sense-amp that reduces read and bit-line computation energy.

Although CRAM achieves 2–3× higher energy efficiency in 8-bit and 32-bit `mac`, the gap between BS-VRAM and CRAM can easily be closed by scaling the BS-VRAM supply voltage. Considering BS-VRAM achieves around 16–18× higher throughput, by scaling the voltage to 0.6 V, BS-VRAM can achieve similar energy efficiency to CRAM while maintaining higher throughput.

TABLE IV. COMPARSION TO PRIOR WORK

Paper	VRAM		ISSCC'19	JSSC'16	VLSI'17
	BS	BP	[19]	[9]	[21]
Technology	28nm	28nm	28nm	28nm	40nm
Voltage	0.9V	0.9V	0.9V	0.9V	0.9V
SRAM Capacity	128kB	128kB	128kB	128kB	128kB
SRAM Macro	4kB	4kB	16kB	0.5kB	8kB
SRAM Bitcell	6T	6T	8T	6T	10T
Precision	Arb.	32b	Arb.	Arb.	Arb.
Freq (MHz)	900	645	225	594	90
Area (mm^2)*	1.1	1.1	2.7	0.7	1.28
Logic Ops	✓	✓	✓	✓(a)	✓(b)
Basic Int Ops	✓	✓	✓		
Cmplx Int Ops	✓	✓	✓(c)		
Cmp Ops	✓	✓	✓(d)		
Search			✓	✓	
FX Ops	✓	✓			
FP Ops			✓		
8b MAC GOPS	76.0	4.5	4.2	n/a	n/a
8b MAC GOPS/W	115.5	17.2	245.5	n/a	n/a
32b MAC GOPS	6.4	1.2	0.4	n/a	n/a
32b MAC GOPS/W	9.0	4.5	22.5	n/a	n/a

* Area is extrapolated to a full chip with 128kB total capcaity considering 80% density; BS-VRAM and BP-VRAM consider 10% overhead for a controller. Cmplx = Complex, Cmps = comparators, FX = fixed-point, FP = floating-point. Logic ops: `and`, `nand`, `or`, `nor`, `xor`, `xnor`. Basic int ops: `add`, `sub`. Cmplex int ops: `mul`, `udiv`, `rem`. Cmps ops: `slt`, `sle`, `sgt`, `sge`, `seq`. FX Ops: `addfx`, `subfx`, `mulfx`, `udivfx`. FP Ops: `addfp`, `subfp`, `mulfp`, `udivfp`. (a) limited to `and`, `nor`. (b) limited to `and`, `nor`, `xor`. (c) limited to `mul`, `udiv` (d) limited to `slt`, `sgt`, `seq`. n/a = the corresponding work does not support this functionality.

VI. CONCLUSION

Leveraging in-situ processing-in-SRAM opens a rich design space for vector accelerators by reducing area and energy costs. Considering BS-VRAM and BP-VRAM as representative design points for bit-serial and bit-parallel approaches, our exploration shows that bit-serial achieves higher throughput compared to bit-parallel, while bit-parallel has lower latency. Both approaches incur comparable area overhead. Although the bit-serial has lower energy, we believe adding better μop support for some macro-operation can bridge the gap. Finally, both designs share a significant amount of circuitry. A reconfigurable design is possible by adding multiplexing to break the addition chain into individual adders, thus transforming bit-parallel into bit-serial and vice-versa.

ACKNOWLEDGMENT

This work was supported in part by NSF E2CDA Award #1740136, the Semiconductor Research Corporation (SRC) as nCORE task 2758.002 and 2758.004, and the Center for Applications Driving Architectures (ADA), one of six centers of JUMP, a SRC program co-sponsored by DARPA, as well as equipment, tool, and/or physical IP donations from Intel, Synopsys, Cadence, and ARM. The authors acknowledge and thank Al Molnar for his advice on SRAM design, and José Martínez and Helena Caminal for useful discussion on various processing-in-memory architectures. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of any funding agency.

REFERENCES

[1] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das. Compute Caches. *Int'l Symp. on High-Performance Computer Architecture*, Feb 2017.

[2] J. Ahn, S. Yoo, O. Mutlu, and K. Choi. PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture. *Int'l Symp. on Computer Architecture*, Jun 2015.

[3] J. B. Brockman, S. Thoziyoor, S. K. Kuntz, and P. M. Kogge. A Low Cost, Multithreaded Processing-in-Memory System. *Proc. of Workshop on Memory Performance Issues*, Jun 2004.

[4] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer†, D. Sylvester, D. Blaauw, and R. Das. Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks. *Int'l Symp. on Computer Architecture*, Jul 2018.

[5] D. Fujiki, S. Mahlke, and R. Das. Duality Cache for Data Parallel Acceleration. *Int'l Symp. on Computer Architecture*, Jun 2019.

[6] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar. OpenRAM: An Open-Source Memory Compiler. *Int'l Conf. on Computer-Aided Design (ICCAD)*, Jan 2016.

[7] M. Ishida, T. Kawakami, A. Tsuji, N. Kawamoto, M. Motoyoshi, and N. Ouchi. A Novel 6T-SRAM Cell Technology Designed with Rectangular Patterns Scalable Beyond 0.18/spl mu/m Generation and Desirable for Ultra High Speed Operation. *International Electron Devices Meeting*, Dec 1998.

[8] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. A Configurable TCAM/BCAM/SRAM Using 28nm Push-Rule 6T Bit Cell. *Symp. on Very Large-Scale Integration Circuits (VLSIC)*, Jun 2015.

[9] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T BitCell Enabling Logic-in-Memory. *IEEE Journal of Solid-State Circuits*, Apr 2016.

[10] T. Jhaveri, A. Strojwas, L. Pileggi, and V. Rovner. Enabling Technology Scaling with "In Production" Lithography Processes. *Optical Microlithography XXI*, Feb 2008.

[11] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. *Int'l Symp. on Computer Architecture*, Aug 2016.

[12] M. Oskin, F. T. Chong, and T. Sherwood. Active Pages: A Computation Model for Intelligent Memory. *Int'l Symp. on Computer Architecture*, Jun 1998.

[13] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. *IEEE Micro*, Mar 1997.

[14] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li. NDC: Analyzing the Impact of 3D-Stacked Memory+Logic Devices on MapReduce Workloads. *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Jun 2014.

[15] RISC-V Foundation. RISC-V "V" Vector Extension. *https://github.com/riscv/riscv-v-spec/releases/download/0.7.1/riscv-v-spec-0.7.1.pdf*, Jun 2019.

[16] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry. RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization. *Int'l Symp. on Microarchitecture*, Dec 2013.

[17] N. Stephens. ARMv8-A Next-Generation Vector Architecture for HPC. *Symp. on High Performance Chips (Hot Chips)*, Aug 2016.

[18] N. Stephens, S. Biles, M. Boettcher, J. Eapen, M. Eyole, G. Gabrielli, M. Horsnell, G. Magklis, A. Martinez, N. Premillieu, A. Reid, A. Rico, and P. Walker. The ARM Scalable Vector Extension. *IEEE Micro*, Mar 2017.

[19] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester. A Compute SRAM with Bit-Serial Integer/Floating-Point Operations for Programmable In-Memory Vector Acceleration. *Int'l Solid-State Circuits Conf.*, Feb 2019.

[20] D. P. Zhang, N. Jayasena, A. Lyashevsky, J. L. Greathouse, L. Xu, and M. Ignatowski. TOP-PIM: Throughput-Oriented Programmable Processing in Memory. *Proc. of Int'l Symp. on High-Performance Parallel and Distributed Computing*, Jun 2014.

[21] Y. Zhang, L. Xu, K. Yang, Q. Dong, S. Jeloka, D. Blaauw, and D. Sylveste. Recryptor: A Reconfigurable In-Memory Cryptographic Cortex-M0 Processor for IoT. *Symp. on Very Large-Scale Integration Circuits (VLSIC)*, Jun 2017.

[22] Q. Zhu, B. Akin, H. E. Sumbul, F. Sadi, J. C. Hoe, L. Pileggi, and F. Franchetti. A 3D-Stacked Logic-in-Memory Accelerator for Application-Specific Data Intensive Computing. *Int'l 3D Systems Integration Conf.*, Jan 2014.