FAST: A Fully-Concurrent Access Technique to All SRAM Rows for Enhanced Speed and Energy Efficiency in Data-Intensive Applications

Yiming Chen, Student Member, IEEE, Yushen Fu, Student Member, IEEE, Mingyen Lee, Student Member, IEEE, Sumitha George, Yongpan Liu, Senior Member, IEEE, Vijaykrishnan Narayanan, Fellow, IEEE, Huazhong Yang, Fellow, IEEE, and Xueqing Li, Senior Member, IEEE

Abstract—Compute-in-memory (CiM) is a promising approach to improving the computing speed and energy efficiency in data-intensive applications. Beyond existing CiM techniques of bitwise logic-in-memory operations and dot product operations, this paper extends the CiM paradigm with FAST, a new shift-based in-memory computation technique to handle high-concurrency operations on multiple rows in an SRAM. Such high-concurrency operations are widely seen in both conventional applications (e.g. the table update in a database), and emerging applications (e.g. the parallel weight update in neural network accelerators), in which low latency and low energy consumption are critical. The proposed shift-based CiM architecture is enabled by integrating the shifter function into each SRAM cell, and by creating a datapath that exploits the high-parallelism of shifting operations in multiple rows in the array. A 128-row 16-column shiftable SRAM in 65nm CMOS is designed to evaluate the proposed architecture. Postlayout SPICE simulations show average improvements of 4.4x energy efficiency and 96.0x speed over a conventional fully-digital memory-computing-separated scheme, when performing the 8-bit weight update task in a VGG-7 framework.

Index Terms—Compute-in-memory (CiM), on-chip learning, SRAM, high-concurrency access, weight update, database.

I. INTRODUCTION

High-concurrency access to a structured memory array could be widely seen in many data-intensive applications. Some examples include the table management in database and the weight matrix read and update in neural networks. Conventionally, the read and write access to multiple rows of an embedded memory array is carried out row by row, sequentially. This is due to the sharing of the bitlines and peripheral circuitry for many rows, so as to balance the density and latency well in most conventional computing tasks. However, in data-intensive high-concurrency memory access applications, it has caused significant latency and become a performance bottleneck, as illustrated in Fig. 1(a). Furthermore, the energy consumption per access is also high due to charging of long bitlines with large parasitic capacitance in read and write operations. Therefore, it is time to re-think the memory access pattern and supporting circuits for emerging high-concurrency applications.

Sumitha George is with The Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA (Email: sumitha.george@ndsu.edu).
Vijaykrishnan Narayanan is with The Department of Computer Science and Engineering, Penn State University, University Park, PA 16802, USA (Email: vijay@cse.psu.edu).
Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.
The initially concept of compute-in-memory (CiM), which was proposed to reduce the data transfer cost between memory and processing units, actually supports a variant of multi-row concurrent access. However, existing CiM techniques are limited in the computing functionalities. On the one hand, the memory data may still need to be delivered to the dedicated processing units for beyond-CiM computing, which needs high-concurrency raw-data read. On the other hand, high-currency memory update (e.g., the parallel neural network weight update), or simply, SRAM refreshing to support large networks, is still limited by the row-by-row access bottleneck.

To tackle these high-concurrency read and write problems, this paper proposes FAST, namely a fully concurrent access technique, supporting both CiM and non-CiM scenarios, as shown in Fig. 2. To demonstrate FAST, we take the static random-access memory (SRAM) in the CMOS technology for evaluations of functionality, latency, and energy efficiency. Itemized contributions include:

(i) A shift-based SRAM capable of reading and writing all rows with full row concurrency. The SRAM cell, array, and supporting circuitry are presented.

(ii) A fully concurrent in situ read-compute and update architecture supporting a category of CiM operations. This is achieved by adding a 1-bit ALU to each row.

(iii) Analysis and evaluations of the proposed techniques for both CiM and non-CiM scenarios. A showcase of a 128-row 16-column CMOS FAST SRAM is implemented, showing significant improvement of speed and energy efficiency over the conventional SRAM solutions.

In the rest of this paper, Section II presents the proposed FAST architecture. Section III evaluates the performance and costs. Section IV concludes this work.

II. PROPOSED FAST MEMORY ARCHITECTURE AND CIRCUITS

This section presents FAST, including the architecture, supporting circuits, and in situ computing capabilities. Simulation and evaluation results are provided in Section III.

A. Overall Architecture

The proposed system architecture is shown in Fig. 2. The bitline (BL) precharger and the row decoder are the same as those of a conventional SRAM array. The control decoder serves as an interface to the external processing units such as CPU or FPGA. The SRAM cell in the proposed architecture is designed to support in-cell shift function, so that each row could be cyclically shifted independently (to the right, for example). More circuit details will be introduced subsequently in Section II.B. Based on this, we add a 1-bit arithmetic logic unit (ALU) in each row, connecting the last cell and the first cell. This 1-bit ALU performs 1-bit logic computing, such as 1-bit add. By combining the shift operation and the 1-bit logic operation, multibit operation could be completed naturally in parallel between different rows, leading to a novel method of compute-in-memory (CiM).

According to the different functions of 1-bit ALU, this scheme could support applications that needs parallel updates to the stored data in several rows. One simple scenario is a high-concurrency access-intensive general cache, such as those of weight matrix during training and database table.

It is noted that the conventional SRAM-based CiM scheme focuses on generating outputs (e.g., dot product results) using the stored SRAM data as input operands and does not update the SRAM array [5]-[8]. In contrast, this proposed CiM architecture is capable of completing multibit in situ calculation with direct write-back support. In the following sections, we will take the weight update in neural network as an example to illustrate how to implement the multibit addition with cyclic right shift and 1-bit ALU.

B. SRAM In-Memory Parallel Shifter

As mentioned above, the shiftable SRAM design is a key enabler of the parallel in situ CiM architecture. In order to support in-row shift operation with less area and latency overhead, we propose a shiftable SRAM cell in Fig. 3, including the cell circuit structure in Fig. 3(a), the control flow chart in Fig. 3(b), and the step-by-step shift operations.

Each shiftable SRAM cell includes a conventional SRAM cell, a CMOS transmission gate controlled by $\varphi_1$ as the inter-cell switch, and two NMOS switches controlled by $\varphi_2$ and $\varphi_{2d}$ as intra-cell switches.
The shift operation between adjacent SRAM cells consists of three phases, shown as a shift-right function in Fig. 3. In phase 1, the intra-cell switches controlled by $\phi_1$ and $\phi_2$ are turned off and the inter-cell switch controlled by $\phi_1$ is turned on. The remnant charge at node X will drive the two inverters to generate a path from the left cell to the right. In phase 2 and phase 3, the intra-cell switch controlled by $\phi_2$ and $\phi_2d$ are turned on one by one with other switches remaining off, so that each SRAM cell forms a closed loop to stabilize its datum.

The timing of the control signals is shown in Fig. 3 (b). It could be conveniently generated by two-phase non-overlapping clock and a delay. The control signals $\phi_1$ and $\phi_2$ are non-overlapping to avoid the data loss caused by the simultaneous turning-on of the switches. The control signal $\phi_2d$ is set to $\phi_2$ with a slight delay to provide sufficient time for data restoration in phase 2. The delay circuit could be simply realized by two serially connected inverters.

C. One-Bit ALU for In-Row Computing Capabilities

Based on the above in-memory shifter, by adding a 1-bit ALU to the end of each row (between LSB and MSB), the high parallel memory computing operation could be realized, as showcased in Fig. 4(a) with a full-adder (FA) example. For a $q$-bit datum stored in the row, after $q$ right-shift cycles along with the 1-bit FA, the external add operand will be added to the row and the data in this row are restored. An example of $q=8$ is shown in Fig. 4(b). Note that in order to avoid the long connection from left-most cell to right-most cell, the cross-connection method shown in Figure 4(c) is adopted in layout.

It is also noted that, during the multi-bit add, the FA carry bit needs to be stored temporarily. The circuit diagram of passing the carry bit to the next stage is shown in Fig. 5(a). In phase 1, FA calculates 1-bit addition and outputs the sum and carry-out bit. The switch $\phi_1$ is turned on while $\phi_2d$ is turned off to store the carry bit on the node T1. In phase 3, the carry will be transmitted through the switch $\phi_3$, which will be used as the input carry of the next stage. An example is shown in Fig. 5. (b) to showcase the workflow.

Fig. 7 shows the post-layout transient waveforms during the shift operations, in which the three phase control signals and the internal nodes of the four SRAM cells in a row are included. Fig. 8 shows the post-layout transient waveforms of the shift-
based add operations, in which the phase control signals and the internal nodes of the four SRAM cells in a row are provided.

III. BENCHMARKING AND DISCUSSION

This section evaluates the proposed FAST SRAM in terms of array-level power and latency, along with application testbenches. Simulation results of circuit transient waveforms, chip layout, etc. are included.

A. Chip Layout Design and Simulation Setting

To evaluate the performance of the proposed FAST SRAM, a chip has been designed in a 65nm CMOS technology, as shown in the die photograph in Fig. 6. Post-layout parasitic extraction has been carried out for more accurate SPICE simulations. The supply voltage is set to 1.0 V.

For comparison purposes, a fully-digital near-memory computing architecture is chosen as the baseline design (Fig. 9(a)). For general benchmarks, the proposed architecture and the baseline architecture are both based on the conventional 6T SRAM structure. While simulating the performance and costs during the weight update, we collect the energy consumption of each word update, and the latency of updating the whole array, i.e., the batch update latency.

B. Array Benchmarking

Energy Efficiency. Fig. 10 shows the energy consumption comparison. When the number of rows is greater than 2 times of the bit width, the proposed FAST SRAM scheme has higher energy efficiency. As mentioned above, this advantage comes from a shorter critical charging and discharging path of a memory access. When the bit width is much less than the number of rows (which is commonly adopted in general SRAM design to reduce the costs of the peripherals), the energy saving is significant. For example, the energy efficiency could be 4x higher than the baseline with 8-bit bit width and 512 rows (here the number of rows is 64x of the bit width).

Latency. Fig. 11 shows the latency comparison, in which the proposed FAST SRAM scheme shows up to hundreds of times speedup. The reason behind this advantage is straightforward: the latency of baseline depends on the number of rows in the array to carry out the operations row by row, while the proposed FAST SRAM support full-concurrency operations on all the rows and the latency depends on the bit width. When the

| Layer*  | Baseline (512 rows) | Our work (512 rows) |
|---------|---------------------|---------------------|
|         | Latency (μs) | Energy (fJ) | Latency (μs) | Energy (fJ) |
| conv1.1 | 1.04       | 3.14       | 0.01       | 0.71       |
| conv1.2 | 44.2       | 134        | 0.46       | 30.4       |
| conv2.1 | 88.5       | 268        | 0.92       | 60.8       |
| conv2.2 | 177        | 536        | 1.84       | 122        |
| conv3.1 | 354        | 1073       | 3.69       | 243        |
| conv3.2 | 708        | 2146       | 7.37       | 486        |
| fc1     | 2520       | 7633       | 26.2       | 1728       |
| fc2     | 3.07       | 9.31       | 0.03       | 2.11       |
| Total   | 3892       | 11806      | 40.54      | 2672       |

* Default model with 8-bit quantization of WAGE framework, VGG-7.
number of rows is larger compared to the bit width, the latency advantage of high parallelism becomes more significant.

Fig. 12 provides more simulation results to highlight the trend of the latency and energy consumption of our scheme under different bit width and number of rows.

C. Application Benchmarking

WAGE [9] is an integer training framework for edge devices, which could limit the weight, activation, and gradient of each layer to a lower bit width integer in training and reasoning. At the end of the gradient calculation, millions of integers may need to be updated at the same time.

Table 1 compares the latency and energy performance of weight update during training for different layers under a configuration of 512 rows in the WAGE benchmark. Compared with the fully-digital near-memory computing architecture, our design shows 4 times of energy saving and 100 times of speedup. Apparently, the proposed FAST SRAM CiM architecture benefits from the concurrent operations.

D. Overheads, More Discussions, and Future Works.

In the experimental chip design, the proposed FAST SRAM architecture adopts ten transistors per cell, including six original SRAM cell transistors and four switch transistors. This extra transistor count brings about 70% area overhead in our design. The area overhead of shift control signal generation is only about 10% in a 16-column scenario. Although the area overhead could be up to 80% of the core area, the improvement of speed and latency is much more significant.

It is also noted that, the proposed SRAM subarray could be used as a general cache, especially for data-intensive applications such as multimedia processing and encryption. In addition, it could realize more complex functions by replacing the 1-bit full adder into other 1-bit operation units. This architecture could serve as a weight update accelerator with high energy efficiency for inference acceleration, database index search and other applications with high-concurrency row-by-row operations. Future work may also consider a reconfigurable design to deal with more versatile calculations such as float-point adder or integer multiplier.

IV. Conclusion

This paper has proposed a novel architecture, namely FAST SRAM, which is capable of dealing with high-concurrency row-wise memory operations. A chip design in 65nm CMOS technology has been implemented to demonstrate the efficiency in parallel memory access and weight update of neural network accelerators. The overhead of the proposed design is mainly the area overhead. Future work that further harnesses the parallelism, flexibility, reconfigurability could be meaningful in data-intensive applications where high-currency memory access may be the performance bottleneck.

REFERENCES

[1] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose and R. W. Linderman, “Memristor crossbar-based neuromorphic computing system: a case study,” in IEEE Transactions on Neural Networks and Learning Systems, vol. 25, no. 10, pp. 1864-1878, Oct. 2014.
[2] S. Jain, A. Ranjan, K. Roy and A. Raghunathan, “Computing in memory with spin-transfer torque magnetic RAM,” in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 26, no. 3, pp. 470-483, March 2018.
[3] D. Reis, M. Niemier, and X. S. Hu, “Computing in memory with FeFETs,” in Proceedings of the International Symposium on Low Power Electronics and Design, Seattle WA USA, Jul. 2018, pp. 1–6.
[4] J. Zhang, Z. Wang, and N. Verma, “A machine-learning classifier implemented in a standard 6T SRAM array,” in 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits), Honolulu, HI, USA, Jun. 2016, pp. 1–2.
[5] X. Si, J. Chen and Y. Tu et al., “A twin-8T SRAM computation-in-memory macro for multiple bit CNN based machine learning.” in International Solid-State Circuits Conference (ISSCC), Feb. 2019, pp. 397-399.
[6] A. Biswas and A. P. Chandrakasan, “CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks,” in IEEE Journal Solid-State Circuits, vol. 54, no. 1, pp. 217–230, Jan. 2019.
[7] J. Yang, Y. Kong and Z. Wang et al., “Sandwich-RAM: An energy-efficient in-memory BWN architecture with pulse-width modulation,” in International Solid-State Circuits Conference, pp. 394-396, Feb. 2019.
[8] A. Biswas et al., “Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications,” in International Solid-State Circuits Conference (ISSCC), Feb. 2018, pp. 488-489.
[9] S. Wu, G. Li, F. Chen, and L. Shi. “Training and inference with integers in deep neural networks” in arXiv preprint, arXiv:1802.04680, 2018.