GRAPHIC: Gather and Process Harmoniously in the Cache With High Parallelism and Flexibility

Yiming Chen, *Student Member, IEEE*, Mingyen Lee, *Student Member, IEEE*, Guohao Dai, *Member, IEEE*, Mufeng Zhou, *Student Member, IEEE*, Nagadastagiri Challapalle, Tianyi Wang, Yao Yu, Yongpan Liu, *Senior Member, IEEE*, Yu Wang, *Fellow, IEEE*, Huazhong Yang, *Fellow, IEEE*, Vijaykrishnan Narayanan, *Fellow, IEEE*, and Xueqing Li, *Senior Member, IEEE*

***Abstract*—In-memory computing (IMC) has been proposed to overcome the von Neumann bottleneck in data-intensive applications. However, existing IMC solutions could not achieve both high parallelism and high flexibility, which limits their application in more general scenarios: As a highly parallel IMC design, the functionality of a MAC crossbar is limited to the matrix-vector multiplication; Another IMC method of logic-in-memory (LiM) is more flexible in supporting different logic functions, but has low parallelism. To improve the LiM parallelism, we are inspired by investigating how the single-instruction, multiple-data (SIMD) instruction set in conventional CPU could potentially help to expand the number of LiM operands in one cycle. The biggest challenge is the inefficiency in handling non-continuous data in parallel due to the SIMD limitation of (i) continuous address, (ii) limited cache bandwidth, and (iii) large full-resolution parallel computing overheads. This article presents GRAPHIC, the first reported in-memory SIMD architecture that solves the parallelism and irregular data access challenges in applying SIMD to LiM. GRAPHIC exploits content-addressable memory (CAM) and row-wise-accessible SRAM. By providing the in-situ, full-parallelism, and low-overhead operations of address search, cache read-compute-and-update, GRAPHIC accomplishes high-efficiency gather and aggregation with high parallelism, high energy efficiency, low latency, and low area overheads. Experiments in both continuous data access and irregular data pattern applications show an average speedup of 5x over iso-area AVX-like LiM, and 3-5x over the emerging CAM-based accelerators of CAPE and GaaS-X in advanced techniques.**

***Index Terms*—Associative memory, single instruction multiple data, FAST SRAM, in-memory computing, logic in memory.**

I. Introduction

IN-MEMORY computing (IMC) attempts to improve the performance and energy efficiency by reducing the data movement in the conventional von Neumann architecture [1], [2]. While the IMC concept has been proposed decades ago, a majority of the IMC efforts today try to apply IMC in two categories of computing [3]: logic operations [4], [5], [6], [7], [8] and MAC operations [9], [10], [11], [12], [13]. Logic in memory (LiM) design aims at more general-purpose applications. Typically, two or three rows will be activated simultaneously on bitlines. The output results of the logic operation are obtained by peripheral circuits that support flexible algorithms. Differently, MAC-oriented design serves as a matrix-vector multiplication (MVM) accelerator. The outputs are usually obtained by comparators [14], ADCs [15], or adder-trees [16], [17], which can deal with dozens (or even far more) rows concurrently. Unfortunately, it is challenging to achieve both flexible logic operations and high parallelism. The MAC-oriented design is limited to bit-wise accumulation, making it impossible to perform more computation tasks. Meanwhile, it is difficult to apply LiM computing to more rows in an array at a time, which limits the inherent parallelism of LiM.

Therefore, there is a critical question: *how to enable both flexibility and parallelism in IMC?*

Notably, single instruction, multiple data (SIMD) [18], [19] as an instruction set architecture (ISA) could inspire the LiM design to improve parallelism by extending the access bandwidth. However, it faces several challenges, as shown in Fig. 1(a). First, the speedup from expanding bandwidth requires that the operands must be contiguous in the memory, which is not

Manuscript received 1 October 2022; revised 9 February 2023; accepted 22 June 2023. Date of publication 17 July 2023; date of current version 15 March 2024. This work was supported in part by the NSFC under Grants #U21B2030, #92264204, in part by Tsinghua University – Daimler Greater China Ltd. Joint Institute for Sustainable Mobility, and in part by the NSF under Grants #2008365, #2132918. *(Corresponding author: Xueqing Li.)*

Yiming Chen, Mingyen Lee, Mufeng Zhou, Yongpan Liu, Yu Wang, Huazhong Yang, and Xueqing Li are with the BNRist, Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: cym21@mails.tsinghua.edu.cn; lmy21@mails.tsinghua.edu.cn; zmf21@mails.tsinghua.edu.cn; ypliu@tsinghua.edu.cn; yu-wang@tsinghua.edu.cn; yanghz@tsinghua.edu.cn; xueqingli@tsinghua.edu.cn).

Guohao Dai is with Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai 200030, China (e-mail: daiguohao1992@gmail.com).

Nagadastagiri Challapalle is with Senior Deep Learning Hardware Architect, Nvidia, Santa Clara, CA 95050 USA (e-mail: nrc53@psu.edu).

Tianyi Wang and Yao Yu are with the Department of Research & Development Automated Driving System, Daimler Greater China Ltd., Beijing 100102, China (e-mail: tianyi.wang@mercedes-benz.com; yao.y.y@mercedes-benz.com).

Vijaykrishnan Narayanan is with the Department of Computer Science and Engineering, Penn State University, University Park, PA 16802 USA (e-mail: vijay@cse.psu.edu).

Digital Object Identifier 10.1109/TETC.2023.3290683

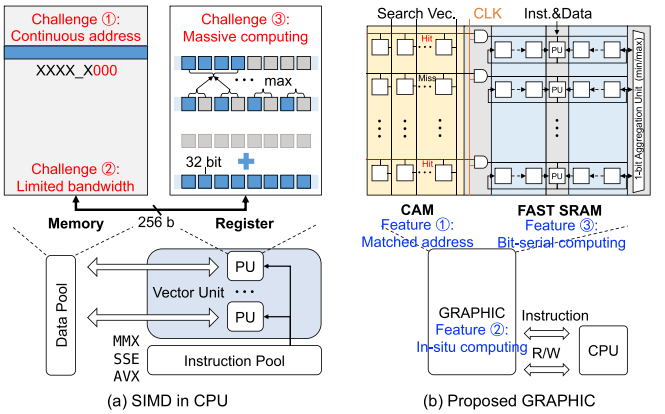

Fig. 1. Comparison between (a) conventional single-instruction multiple-data (SIMD) in CPU, and (b) SIMD in the proposed GRAPHIC cache.

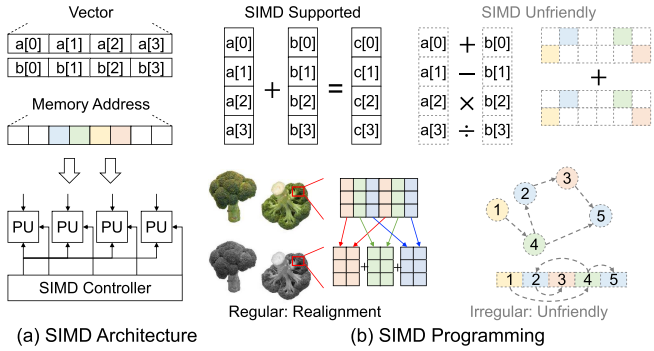

Fig. 2. Basics of SIMD (a) architecture, and (b) programming.

applicable to irregular parallel data, e.g., graph processing [20], [21], [22]. Second, the access bandwidth is limited by the peripheral circuits. Due to the on-chip memory access interface, the parallelism improvement from higher access bandwidth is limited. Finally, due to the increase in the number of operands, massive full-precision computing is limited by the power wall and area efficiency bottleneck.

In this paper, we propose GRAPHIC, an in-memory SIMD architecture, to overcome the challenges in flexible and parallel computing with irregular data. As illustrated in Fig. 1(b), GRAPHIC consists of content-addressable memory (CAM) [7], [23] and a recently proposed fully-concurrent row-wise-access SRAM, namely the FAST SRAM [24]. There are a few intriguing new features of GRAPHIC. First, the data matching mechanism of CAM provides an opportunity for parallel computation of discontinuous data in the memory. Second, the in situ computing method supported by FAST SRAM introduces an efficient way of parallel computing without data readout. Third, the parallel computing architecture provides high-area-efficiency processing in the memory. In addition to the regular vector operations in the SIMD instruction set, GRAPHIC also supports parallel aggregation in the whole memory, showing more advantages in certain applications, such as single-source shortest path (SSSP).

The key contributions of this paper include:

- GRAPHIC, a versatile SIMD accelerator that efficiently enables high parallel in-situ IMC with the support of discontinuous data access.
- A compiling process that converts the serial execution codes into a parallel operation stream in the proposed GRAPHIC.
- Demonstration of mapping of two different data access types in graph processing and image processing onto GRAPHIC.
- Full-stack application performance evaluations of GRAPHIC, in comparison with the SIMD ISA in CPU and recent CAM-based accelerators [25][26], in which GRAPHIC shows 6x and 4x performance over the CPU SIMD with irregular and continuous data, respectively, along with a 3x-5x average speedup over prior custom accelerator designs.

In the rest of this paper, Section II reviews the background of SIMD and CAM-based accelerators. Section III shows the details of the proposed GRAPHIC based on CAM and FAST SRAM. Section IV presents the method of deploying applications on GRAPHIC. Section V evaluates the proposed designs from the circuit level to the application level. Section VI discusses the overhead and future work. Section VII concludes this work.

II. Background

This section briefly introduces the basics of SIMD programming. The building blocks of GRAPHIC, including content-addressable memory (CAM) and fully-concurrent access SRAM (FAST SRAM) are also introduced.

A. SIMD Programming

The SIMD ISA, a compatible parallel computing technique, performs the same operation in multiple data simultaneously. Generally, SIMD in modern CPU is designed with respect to vector computation. As shown in Fig. 2(a), SIMD instructions still follow the von Neumann architecture. Taking SSE-128 instructions as an example, 2 sets of four 32-bit data as vectors stored continuously in the memory are loaded into registers of corresponding processing units (PUs). After executing the corresponding calculation in four 32-bit width arithmetic logic units, the results are written back to the data memory. Thus, it achieves up to 4x speedup compared to a single-core CPU.

Benefiting from vector computing in parallel, the modern CPU with SIMD aims to improve the performance of multimedia processing. However, there is not always such performance improvement by SIMD. It depends on how the data are organized in the memory and which algorithm is performed. As shown in Fig. 2(b), SIMD excels in processing homogeneous data, but it is not friendly to sparse, heterogeneous computing. As a typical application with regular data distribution, image processing can get a high-performance boost from SIMD, with data realignment. However, graph, as an efficient data structure to describe relationships in many emerging applications such

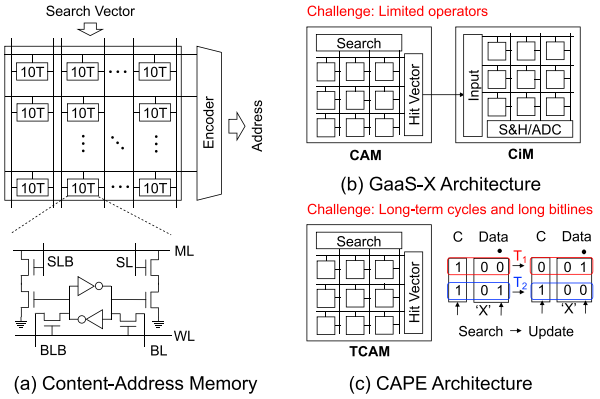

Fig. 3. Structure overview of (a) Content-Address Memory (CAM), (b) GaaS-X architecture [26], and (c) CAPE architecture [25].

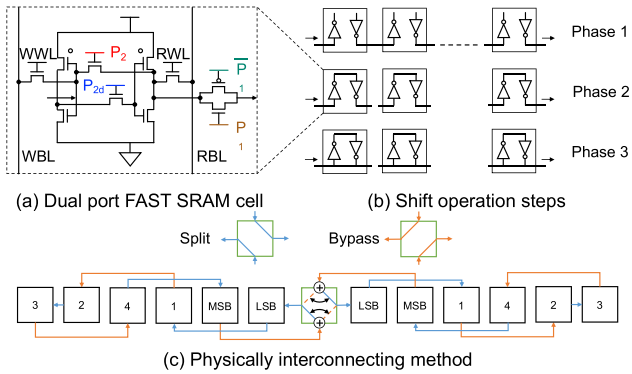

Fig. 4. FAST SRAM (a) 10T cell structure, (b) shift operation steps, and (c) interconnecting method.

as social, biological, and information systems, faces difficulty when applying SIMD due to the irregular data access.

B. Content-Addressable Memory (CAM) Based Accelerator

There is a challenge in simultaneous access to irregular data for high-concurrency computing. Recently, our work GaaS-X [26] reveals a chance to utilize CAM for irregular access. Single-cell and array structures of typical SRAM-based CAM are shown in Fig. 3(a). During a search in the CAM, the search vector and its inverted value are fed into SL and SLB, respectively. The match line (ML) will be discharged if there is one or more mismatch cell loaded on it. A priority encoder is used to output the address of the first row that matches the input.

Based on the functionality of matching, GaaS-X combines the CAM with MAC-oriented CiM to perform accumulation with certain selected rows in the CiM crossbar, as shown in Fig. 3(b). Various graph algorithms are implemented, such as PageRank, SSSP, BFS, and collaborative filtering. GaaS-X relieves the bottleneck of sparse-to-dense conversion to perform graph computing in CiM. However, the limitation of GaaS-X is also obvious. Only MAC calculations can be performed by the CiM. Although the special function unit (SFU) can perform some other operations, the overall area efficiency will be extremely low due to the redundant ADCs.

The content-addressable parallel processing paradigm (CAPP) [27] and associative computing (ASC) [28], [29] also provide a chance for in-situ computing in memory by CAM. A full-stack design based on the associative computing named CAPE [25] was proposed recently as a SIMD architecture, as shown in Fig. 3(c). The atomic operation of CAPE is to match and update the corresponding value in CAM columns. Based on this operator, bit-serial algorithms can be performed on CAPE. During the bit-serial calculation, only 1-bit of data will be computed in one cycle. In each step, CAPE searches for possible cases and updates the bits at the matched positions to the data obtained from the arithmetic unit. After a number of cycles, single full-precision computing could be completed for the entire array. Though CAPE could achieve high-performance improvement in densely formatted data computing, the difficulty of CAPE is long-term cycling. Using addition as an example, CAPE consumes 8x more time compared to the typical bit-serial algorithm. The performance deteriorates further with sparse data due to the parallelism decrease.

C. Fully-Concurrent Access SRAM (FAST SRAM)

Recently, a fully-concurrent access SRAM (FAST SRAM) [24] for the bit-serial algorithm was proposed and experimentally verified to perform high-parallel in-situ computing, as shown in Fig. 4.

FAST SRAM behaves as a memory that can cyclically shift independently in rows. The basic cell structure is shown in Fig. 4(a). Based on the conventional 6T SRAM, the FAST SRAM inserts two switches in the inverter ring controlled by P_2 and P_{2d}. Meanwhile, each two adjacent SRAMs are connected with a switch controlled by P_1. It is noted that the inverter ring can be broken during a write, which makes the single-ended read and write possible. This 10T FAST SRAM offers dual-port read and write functionality, i.e., simultaneous read and written in one cycle.

The fully concurrent shift operation steps of FAST SRAM are shown in Fig. 4(b). In phase 1, the inverter ring is broken and the inter-cell switches controlled by P_1 are closed to transfer the data between two adjacent SRAMs. In phase 2 and phase 3, the inter-cell switches are turned off and the switches controlled by P_2 and P_{2d} will be turned on in sequence to recover the inverter ring and maintain the data. This leads to a 1-bit right shift in a single cycle. Notably, P_2 and P_1 are generated by a 2-phase non-overlapping clock, and P_{2d} is set to P_2 with a slight delay to provide sufficient time to set up.

As shown in Fig. 4(c), the MSB and LSB are connected with a 1-bit processing unit (PU) to form a bit-serial computing unit. A folded interleaving layout connection is adopted to minimize the critical path delay of the interconnections between FAST SRAM cells. With this method, the longest interconnect line shrinks to at most two SRAM cells instead of all SRAM cells in the row. The arithmetic units of two adjacent rows of FAST SRAM can reuse the same PU. This allows the critical path of the binary operators to be minimized.

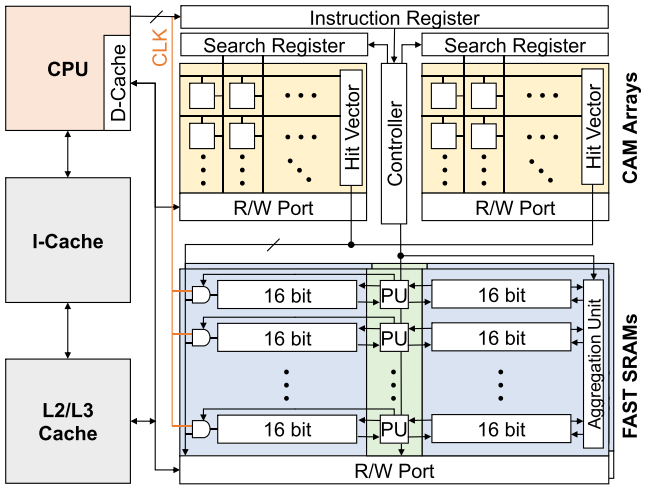

Fig. 5. Proposed GRAPHIC architecture overview.

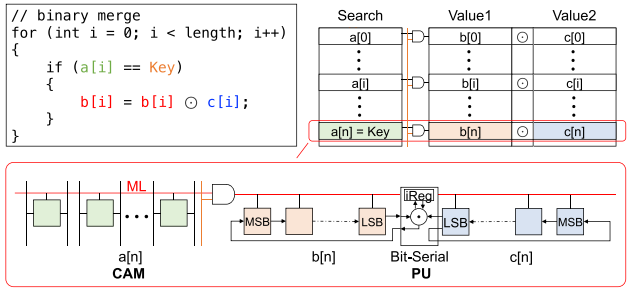

Fig. 6. GRAPHIC binary operator instructions with multiple data.

III. GRAPHIC Architecture

This section presents the proposed in-memory SIMD accelerator, GRAPHIC. To understand the support of full-array-parallel data processing, this section will introduce the overall GRAPHIC architecture, the merge operation, and the aggregation operation. The architecture and operations exploit CAM and FAST SRAM to leverage the high-parallelism computation of data matching and in-situ bit-serial computing to support various algorithms and applications.

A. Overall Architecture

Fig. 5 shows the overall architecture of GRAPHIC. To obtain compatibility with existing platforms, GRAPHIC serves as both the L1 data cache and SIMD units of the CPU. In the cache mode, the data transferred between the CPU and GRAPHIC is conducted through the read/write port. The details related to the group association are not shown in the illustration. In the computing mode, the GRAPHIC architecture works with three major modules: (i) the GRAPHIC controller, (ii) the CAM crossbars, and (iii) the FAST SRAM banks. The controller accepts instructions from the CPU and extracts the control vector into the operation registers. The CAM is responsible for parallel searching and generating the hit vector to drive the clock of the corresponding rows in the FAST SRAM to select data for computing. The FAST SRAM has the capability of performing parallel bit-serial computing in all selected rows independently. During each cycle, the PUs and aggregation modules perform 1-bit arithmetic operations such as full-add, Boolean logic, data multiplexing, etc. With the multi-cycle support of the controller, full-precision operations such as subtraction, multiplication, and minimization are supported. Thanks to the in-situ computing in the FAST SRAM, the context is well preserved when switching between computing and cache modes. As the FAST SRAM has been verified by tape-out [24] to be able to perform all-row-parallel data processing, the throughput is improved significantly compared with the area-costly low-density register files in the conventional SIMD extension in CPU.

B. Merge Operation Mapping

As a SIMD architecture, GRAPHIC can compile codes that serially process lists and hash tables into parallel instructions. These operations are collectively known as "merge" operations. With the support of FAST SRAM, GRAPHIC can support both inter-list merging and immediate instructions with different bit widths.

Considering the binary operator instructions of key-value pairs in GRAPHIC, as shown in Fig. 6, the key for each value is stored in the CAM crossbars and the values are stored in the FAST SRAM. First, key matching is performed in the CAM to search for rows that store the operands. The search result will drive the clock generator in FAST SRAM to cyclically shift the bits in corresponding rows. For a PU with binary operators, there are two 1-bit external inputs, namely target operand b[i] and intermediate operand c[i], from the LSB side (see Fig. 6) of the FAST SRAM rows for each cycle of bit-serial computing. Besides, internal registers (I Reg) are also deployed in PU for data computing (e.g., carry bits in full-add) and state indications (e.g., data processing, processing finished, idle). The supported arithmetic operations performed in the PU include 1-bit full addition, Boolean logic, and assignment. In each cycle, the PU calculation result is written back to the MSB cell of the target operand while the input from the intermediate operand is written to its own MSB cell. After N cycles (N = value bit widths), full-precision addition, logical computation, and list copying operators are completed. Note that shift operations are also supported by simply configuring the cycles of the driving clock. Furthermore, by combining these operators, more complex operations such as multiplication can be performed.

The immediate instructions with multiple data in GRAPHIC can be used when the selected element of a list or hash table performs the same calculation with an identical value *Imm*, as shown in Fig. 7. Compared to the binary operator instructions, one input of PU also comes from the LSB cell of target operand, but the other input of *Imm* is sent from the external registers instead of the SRAM cells. At this point, the left and right sides of a FAST SRAM row can be reconfigured as two values with halved bit width, or, one single value with full bit width. The PU corresponding to each value will receive a shift input from the FAST SRAM row, and a global input from the immediate value. The PU output will be updated directly to this row.

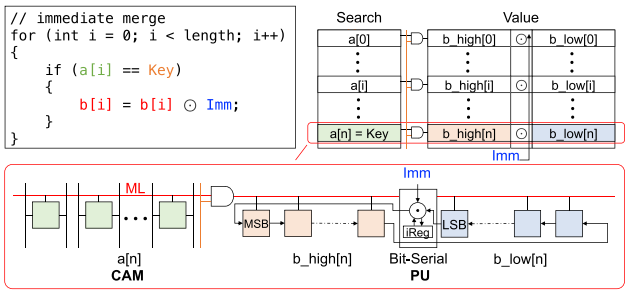

Fig. 7. GRAPHIC immediate instructions with multiple data.

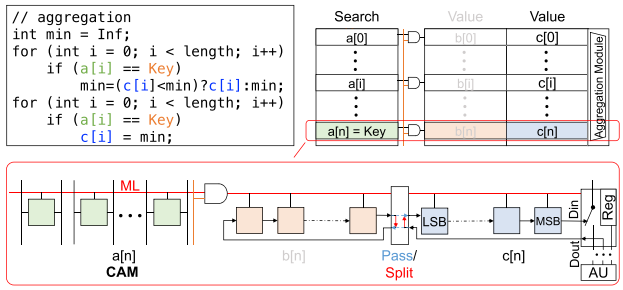

Fig. 8. GRAPHIC aggregation instructions with multiple data.

Algorithm 1: Minimize the List by Bit-Serial.

Input: A list C[1 …m] of n-bit data width
Output: A list of stored minimum values
Initialization: Conf[1 …m]Deny[1 …m] to 0
  Conf[i]: 1-bit, whether C[i] is confirmed as minimum
  Deny[i]: 1-bit, whether C[i] is not minimum
  Min1: 1-bit, the output before minimum is located
  Min2: 1-bit, the output after minimum is located
  End: 1-bit, whether the minimum is located
  One: 1-bit, used to decide the exists of minimum
for i **in** 1:n // executed in n cycle
  for j **in** 1:m // parallel in FAST SRAM
    Din[j] ← **ith significant bit in** C[j]
    D[j] ← !(Din[j] | Deny[j])
    K[j] ← Din[j] **&** Conf[j]
  Min1 ← **!**D[1] **&** **!**D[2] **&** **…** **&** **!**D[m]
  Min2 ← K [1] | K [2] | **…** | K[m]
  End ← **!(!**Conf[1] **&** **!**Conf[2] **&** **…** **&** **!**Conf[m])
  One ← **Sum(!**D[1 …m]) **is** 1
  If End
    Min ← Min2
  Else
    Min ← Min1
  for j **in** 1:m // parallel, iReg update in FAST SRAM
    Conf[j] ← Conf[j] | (**!**Din[j] **&** One)
    Deny[j] ← Deny[j] | (Din[j] **&** Min1)
  Dout ← Min

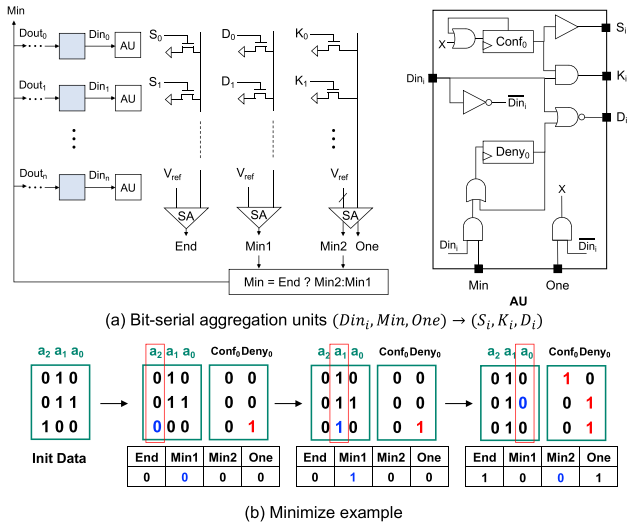

Fig. 9. (a) Circuits and (b) an example of aggregation operation: minimize.

Selecting data by CAM without data alignment avoids the bottleneck of traditional SIMD, as SIMD requires data alignment before performing parallel computation. Also, the in situ computing feature of GRAPHIC supported by the FAST SRAM reduces the energy consumption of redundant memory access. Thus, performing merging operations in GRAPHIC is highlighted with both high energy efficiency and high area efficiency benefits.

C. Aggregation Operation Mapping

To extend the vector processing capability of the conventional SIMD, GRAPHIC supports aggregation operations in multimedia and graph applications. As shown in Fig. 8, row-wise parallel bit-serial computing is performed in the aggregation units (AUs).

Unlike the additive aggregation used in MAC-oriented CiM design supported with ADCs or adder-tree, GRAPHIC shows a lower-cost way to perform the minimum aggregation of massive data. In GaaS-X [26] and the SIMD extension in CPU, minimization is performed using full-precision digital logic, which introduces a large area overhead and power consumption.

Some modifications based on the 10T FAST SRAM are made, so as to support minimization and maximization. Previous FAST SRAM performed only the right shift to support arithmetic operations due to only one switch between adjacent cells. The modification is to add an extra right-to-left pathway controlled by one more switch to achieve a bi-directional shift. That is about 20% cell area overhead and less than 5% overall area overhead. FAST SRAM with the support of a bi-directional shift allows the bit to be fed to the PUs in order from MSB to LSB for comparison operators, such as “set less than (SLT)”. The internal register (iReg) is used to store the comparison result of the current bit and will be fixed after being set to ‘1’.

Further, based on the bi-direction FAST SRAM, we can improve this one-by-one comparison into the minimization operation in Fig. 8 on the array. The aggregation unit used for minimization is shown in Fig. 9(a). This operation updates all

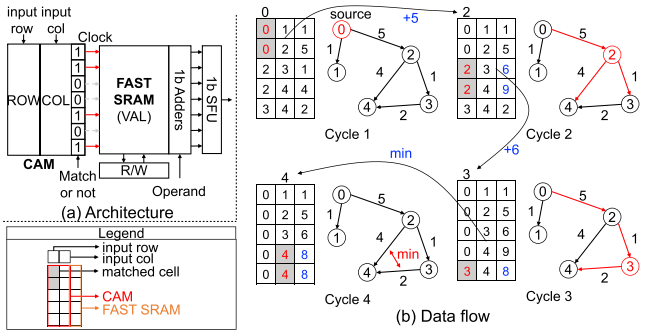

Fig. 10. (a) The GRAPHIC Architecture and (b) Operation Example of Single Source Shortest Path (SSSP).

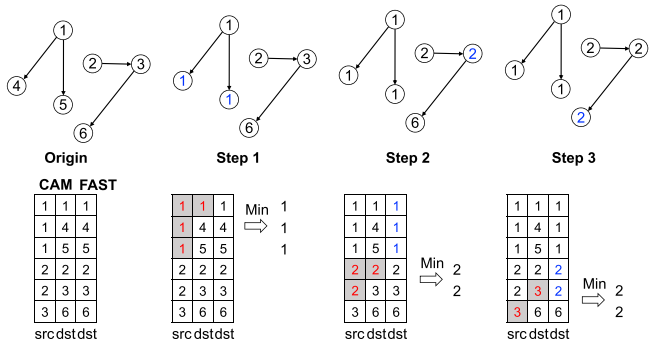

Fig. 11. An operation example of connected component (CC).

the selected rows to the smallest one. An example is illustrated in Fig. 9(b). In principle, the minimization operator traverses all bit locations of selected rows from MSB to LSB. When the minimum value has not been located, and a '0' exists among AD inputs, '0' is written back to the MSB cells of all selected rows since values with '0' on that bit location are smaller ones. Otherwise, '1' will be written back. Such a process will be stopped if the sense amplifier detects one and only one '0' in the inputs K_i, $i = 0, \ldots, n$. This row is the location for the minimum value. Meanwhile, *END* and the register *"Conf"* of the corresponding row are set to 1. The following bits written back to all rows are the bits corresponding to that minimum value. This method utilizes bitline discharging and sense amplifiers in the aggregation unit to achieve extremely low-cost of latency and energy consumption in the aggregation operation without direct SRAM access.

The details are shown in Algorithm 1. This bit-serial aggregation can convert minimization and maximization in an array to a constant-cycle operation, which can achieve a large speedup in the corresponding applications.

IV. Application Mapping

In this section, a method to map applications to the SIMD instructions provided by GRAPHIC is provided. Also in detail, different levels of applications are presented as examples, including graph traversal operation SSSP, graph relationship analysis connected component (CC), basic operators for image process reverse, and practical image processing algorithms (histogram equilibrium and haze removal).

A. Graph Processing Application

Single Source Shortest Path (SSSP): The SSSP algorithm is an important operator in the graph data structure, which is to designate a start vertex and compute the distances from all other vertices to the start vertex. The distance is calculated by summing the weights of the edges of connected vertices. Due to the highly random graph data accessing, SSSP is a bottleneck for many graph-related algorithms, such as pathfinding algorithms and computer-aided design.

Fig. 10 illustrates how to deploy SSSP on the GRAPHIC architecture based on Dijkstra algorithm. Consider an implementation sample shown in Fig. 10(a), each row of CAMs and FAST SRAMs stores the graph information in a coordinate (COO) format (source, destination, and weights of an edge). The source and destination pairs for each edge are loaded onto CAM crossbars, while the weights are loaded onto FAST SRAM. In the SSSP algorithms, the GRAPHIC basic operators include matched value addition and matched minimization. The control flow is based on breadth-first search (BFS).

Fig. 10(b) shows the data flow for the SSSP algorithm under the sample graph. Starting from the source vertex (vertex 0), the neighbors (vertex 1 and vertex 2) are selected in an ascending order of the distance to the source vertex to perform BFS. The operator to find the minimum distance from the source can be modified by the aggregation algorithm in Section III.C. In this process, GRAPHIC does not update the original data, but directly selects and reads the row activated by the register Conf. Since vertex 1 is terminated with no CAM row matched, vertex 2 is sent to CAM to perform the matched value addition operation to update the distance of the vertex 2 neighbors (vertex 3 and vertex 4). Meanwhile, the edge start point will also be updated as the source vertex. The process above is applied on each vertex started from the source vertex to update its distance to the source until a duplicate destination appears in a matched addition operation, which means multiple paths from the current vertex to the source have been found. At this point, the matched minimization operator through the matched destination will update the path to be the shortest one. Those operations will be repeated until all stored sources of all the edges are the starting vertex.

Connected component (CC): The CC algorithm is another operator in the graph, which is to mark all the connected vertex with the same group ID (always the smallest vertex ID in the subgraph). In general, this algorithm requires traversing all edges to pass the minimum ID, which will suffer from frequent and irregular memory access. However, the proposed GRAPHIC-based algorithms only require the traversal of vertices, bringing significant throughput improvement compared with conventional CPU processing since the number of edges will be orders of magnitude larger than the number of vertices.

Fig. 11 illustrates the procedure of how CC works in GRAPHIC. The source and destination pairs for each edge

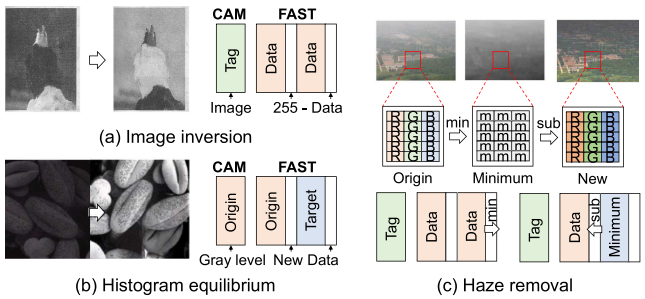

Fig. 12. Operation examples of image processing applications: (a) image inversion, (b) histogram equilibrium, and (c) haze removal.

are loaded onto the CAM crossbars, while the FAST SRAM is used to store the group ID, initialed with the destination vertex id. Starting from the smallest vertex ID, the matching is performed in both source and destination columns in CAM. The minimization operation is performed on the selected rows in the FAST SRAM, which will update the group ID with the smallest one. This operation will be repeated until all vertices are visited.

B. Image Processing Application

Unlike irregular graph applications, image processing is a class of applications where data are distributed continuously and computation could be performed sequentially, which is friendly to SIMD. The important feature of GRAPHIC is the capability of performing the in situ computation of images, which outperforms SIMD or other near-memory computing architectures, leading to reduced data movement and improved energy efficiency.

Fig. 12(a) shows a basic operator for image processing in a single channel: inversion. If the image pixel is in an unsigned 8-bit format, then the inversion is calculated as 255 minus the value of the original image. The image information is stored in the FAST SRAM, while the CAM is used to identify the attributes of the FAST SRAM row. The attributes will be matched as a tag to select all the pixels of the area to be operated in the inversion. At this point, immediate instruction with multiple data is performed in the FAST SRAM.

Fig. 12(b) demonstrates an advanced algorithm called the histogram equilibrium. In some scenarios due to lighting, texture, etc., the photograph may be difficult to read. Histogram equilibrium may enhance the image visibility, by mapping the image grayscale level to a new level with statistical methods. The grayscale information will be stored in both CAM and corresponding rows in the FAST SRAM. For an image loaded on the FAST SRAM, iterate through all its gray levels to be matched in CAMs and update them to new values accordingly in the FAST SRAM. This match-update operation can be implemented not only for histogram equilibrium but also in various image enhancement operations with improved parallelism.

Further, a specific application of haze removal is presented in Fig. 12(c). Objects covered in fog or haze are difficult to be distinguished. A hazing removal algorithm based on statistical analysis was proposed [30]. Although neural network-based algorithms are well performed in various scenarios, such a lightweight and effective algorithm is vibrant on small devices at the edge. When the haze removal algorithm is executed, an image is firstly divided into multiple small blocks Ω, and in each block, the smallest RGB values are found, i.e.,

$$m = J^{dark}(x) = \min_{y \in \Omega(x)} \left(\min_{c \in \{r,g,b\}} J^C(y) \right) \quad (1)$$

For an image without haze or fog, the value m is close to 0. For an image covered with haze, the value m represents the intensity of the haze in that block. Next, simply subtract the corresponding value m from each block to complete the haze removal. When it comes to the GRAPHIC architecture, the FAST SRAM stores the image information (the left side and the right side save the same pixel), while CAM is used to select a certain block in the whole image. Each block performs two cycles of operation. In the first cycle, the minimization on the selected block in the FAST SRAM on the right side. In the second cycle, the minimum value is subtracted from the FAST SRAM on the left side to complete the haze removal algorithm for that block.

Furthermore, combined with the image inversion algorithm, the haze removal could be modified as a night vision algorithm [31]. To accomplish this task, a dark image is reversed first to a white image. Then the haze removal is performed on it. Finally, reverse the image back to the dark version. The visibility of the original dark image may be improved.

V. Evaluation

In this section, experiments from the circuit level to the application level with the GRAPHIC architecture constructed by specific parameters are carried out to evaluate the performance of GRAPHIC. The proposed GRAPHIC is compared with the baseline architectures and the improvement in energy efficiency and speedup is presented.

A. Circuit-Level Evaluation

The extended FAST SRAM subarray of 16 columns by 256 rows is simulated in Cadence Virtuoso on TSMC 65nm PDK. The design of the processing unit (PU) and aggregation unit (AU) is compiled by Synopsys DC. The overall control logic is also evaluated by the DC compiler. A 6T cell content-address memory based on split-wordline [7] is implemented to be reconfigured as SRAM, BCAM, or TCAM according to different applications. The latency and power consumption of the on-chip SRAM arrays are evaluated by the 65nm CACTI model [32]. As an L1 cache, the external circuitry of the FAST SRAM and its cache controller are the same as the dual port SRAM modeled by CACTI.

Timing of FAST SRAM: Multiple cycles to perform a single operation is the latency bottleneck of the bit-serial style computing method. FAST SRAM uses local access and computing, which can greatly increase the frequency. The delay of FAST SRAM obtained from the simulation is 120ps. Considering the clock jitter, skew, and other factors, the FAST SRAM is finally driven by a 5GHz clock. In addition, benefiting from the

TABLE I
PARAMETERS OF THE GRAPHIC DESIGN

	Component	Configuration	Area
FAST SRAM	Array	32×256×8	0.478 mm^2
	1-bit PU	2×256×8	0.057 mm^2
	1-bit AU	256×8	0.053 mm^2
	Signal generators		0.005 mm^2
CAM	Crossbar	32×256×8	0.578 mm^2
Input Buffer	Instruction registers	64×8	0.004 mm^2
	Search registers	32×8	0.002 mm^2
Controller	Finite state machine		0.003 mm^2
Total	-	128 Kb	1.18 mm^2

TABLE II
METRICS OF A SUBSET OF VECTOR INSTRUCTIONS SUPPORTED BY GRAPHIC COMPARED TO CAPE

Instruction (n bits)		CAPE		GRAPHIC	
		Total cycle	Operation energy (pJ)	Total cycle	Operation energy (pJ)
Arithmetic	**vadd**	$8n + 2$	8.4	n	5.4
	vsub	$8n + 2$	8.4	n	5.4
	vmul	$4n^2 - 4n$	99.9	$(3n^2 - n)/2$	256.5
Logic	**vand**	3	0.4	2	0.3
	vor	3	0.4	2	0.3
Comparison	**vmslt**	$3n + 6$	3.2	n	5.4
Data	**vmov**	$2n$	12.8	n	5.4
Aggregation	**min**	Unsupported		n	5.4
	max			$2n$	10.8

simplicity of 1-bit arithmetic logic, the PU is synthesized as an additional unit in the FAST SRAM shifting loop at 5GHz. Since the bitwidth of a FAST SRAM [24] array is fixed, a larger FAST SRAM could be built by dynamic bitwidth reconfiguration. This makes FAST SRAM scalable for various cache designs.

Energy of FAST SRAM: FAST SRAM exhibits high energy efficiency by eliminating the impact of the large parasitic capacitance loads on the bitline in SRAM access. It is also noted that, the energy consumption could be further improved by taking care of the redundant shifting operations in activated rows. There are two techniques applied in GRAPHIC for this purpose. First, a routing unit inserted in a word can terminate the shifting loop earlier to reduce the flips. This can be applied in image processing since in most cases the data bit width is only 8 bits. Second, the addressing function of CAM is used to clock-gate the FAST SRAM rows. This is widely used in graph processing because the data access and computing will be sparse. In this case, pruning useless shifting is the key to improving energy efficiency.

B. Macro-Level Evaluation

Considering a reasonable L1 cache size of 128Kb, a specific design of GRAPHIC with 128Kb memory capacity used in the evaluation is summarized in Table I.

The GRAPHIC unit consists of a 256-row 32-bit CAM and a FAST SRAM of the same size. Each row of the FAST SRAM is divided by 1-bit PU into two 16-bit left and right parts. Each GRAPHIC unit accepts inputs from the 64-bit instruction registers and the 32-bit search registers to perform computing and matching, respectively.

The macro consists of 8 basic GRAPHIC units, managed by the controller. The controller is responsible for communicating with the CPU and distributing data and control vectors to the registers of GRAPHIC. A finite state machine is used to control the FAST SRAM shift behavior under one vector/aggregation instruction.

The designed access memory bitwidth is 512 bits (64-bit x 8), the same as the baseline AVX-512 (a SIMD instruction set in CPU).

C. System Evaluation

To support general-purpose applications, we abstract the GRAPHIC operations into several SIMD instructions. Table II shows the metrics of a subset of instructions supported by GRAPHIC compared with the CAPE architecture. Both GRAPHIC and CAPE adopt the bit-serial computing method. Given the bit width n, the number of execution cycles is related to n because of the bit-serial execution method. The energy consumption of GRAPHIC is obtained based on simulations. The energy consumption of CAPE is from [25].

Arithmetic instructions perform an arithmetic calculation between two vectors and write the result back to the target vector. Table II shows the number of cycles and the power consumption of GRAPHIC and the baseline CAPE, using vector-to-vector addition, subtraction, and multiplication as examples. It is noted that CAPE needs to traverse all its possibilities for each bit. For arithmetic calculations in CAPE, each bit takes 8 cycles. In comparison, GRAPHIC needs only 1 cycle to deal with each bit. In terms of power consumption, GRAPHIC consumes 35.7% less energy compared with CAPE in 32-bit addition and subtraction. However, due to the high flip rate mentioned earlier, the power consumption is greater than CAPE when performing multiplication with a high number of cycles.

Logic instructions perform Boolean functions on Boolean vectors, which are widely used in control flow. For FAST SRAM, the minimum cycle shift period is 8 (by route units). However, the logic operation is only performed to the LSB, which will incur redundant shift operations in a uni-directional FAST SRAM. At this point, the bi-directional FAST SRAM could provide a redundance-shift-free solution. When executing a logical operation on GRAPHIC, PU selects the corresponding logical operation path. First, it performs a right shift to send the LSB to the PU and the result to the MSB. Then the PU selects the direct pass (without calculation) and the FAST SRAM performs a left shift to write the calculation result in the MSB back to the LSB. In this way, GRAPHIC achieves 2-cycle logic operation, which is more efficient than the 3-cycle logic operation of CAPE.

TABLE III
SPECIFICATION SUMMARY

Specifications	AVX-like LiM	GaaS-X	CAPE	GRAPHIC
Technique node	65nm CMOS	32nm RRAM	7nm CMOS	65nm CMOS
Storage size (Kb)	128	98304	18432	128
Area (mm^2)	1.23	2.69	11.3	1.18
Throughput (Tbps)	4.19	3.35	524.52	18.62
Computing method	Full prec.	ADC	Bit-serial	Bit-serial
VADD/VSUB	Supported	Supported	Supported	Supported
VMUL	Supported	Supported	Supported	Supported
VSHIFT	Supported	Not supported	Not supported	Supported
VLOGIC	Supported	Not supported	Supported	Supported
VMOV	Supported	Not supported	Supported	Supported
MAX/MIN	Supported	Supported	Not supported	Supported
Irregular pattern	Not supported	Supported	Not supported	Supported
Area efficiency[1] (Tbps/mm^2)	3.39	1.24	46.4	15.8
Energy efficiency[1] (16bit)	19.38 pJ/OP	1.03 pJ/OP (6 bit)	4.20 pJ/OP	1.26 pJ/OP
Energy efficiency[1] (32bit)	38.75 pJ/OP		8.40 pJ/OP	5.40 pJ/OP

*1: 32-bit arithmetic logic between two values

TABLE IV
UTILIZATION AND SPEEDUP SUMMARY

Tasks		CAPE		GRAPHIC-512		GRAPHIC-2k	
Specification		utilization	speedup	utilization	speedup	utilization	speedup
graph computing	sssp	0.08%	0.4	19.8%	4.4	5.0%	4.4
	cc	0.3%	1.4	39.6%	8.8	9.9%	8.8
image processing	hist equal	15.8%	4.1	65.9%	4.4	32.5%	8.6
	haze removal	0.9%	1.0	99.6%	3.9	98.6%	15.4
	night vision	10.8%	1.3	77.0%	4.2	54.5%	10.9

Comparison instructions compare the corresponding values of two vectors in parallel. Those operations are similar to an arithmetic operation, requiring only one cycle per bit, which shows a 3x speedup compared with CAPE.

Data movement is used to copy the data from one vector to another, which will be used when a temporary copy is needed.

Aggregation instructions are used to count the data in a vector and give an eigenvalue. Taking the example of finding the minimum value in a list, SIMD in CPU does the comparison in parallel in the form of a binary tree. Therefore, a total logarithmic level of time duration is required. The CAPE architecture cannot support native aggregation operations. The proposed GRAPHIC architecture performs the min and max calculation by the aggregation unit. The minimization simply iterates over the operands stored in the FAST SRAM from MSB to LSB to replace all values in the list with the minimum value. At the same time, in order to reuse the aggregation units in the maximization operator, n more cycles of operations are needed to invert the operands in a batch.

The above operands are integers in the two's complement format. Since the FAST SRAM is naturally shifting independently between rows in the array, it is capable to store floating point numbers in rows and align them before calculation, future work is promising.

D. Metrics Comparison

The baseline is AVX-like LiM with the same functionality as GRAPHIC. It is used to evaluate the efficiency of CAM-based GaaS-X, CAPE, and the proposed GRAPHIC architecture, as shown in Table III.

AVX-like LiM consists of a 128Kb cache (the same as the GRAPHIC design) and 9 PU. Each PU supports 16 parallel 32-bit operands. A custom design is implemented by Synopsys DC. The specifications of the SRAM cache design are from CACTI [33]. As a macro based on the SIMD instruction set, the baseline can perform a variety of operations, including logic, arithmetic, and aggregation.

GaaS-X [26] includes 64Mb RRAM-based CiM crossbars and 32Mb RRAM-based CAM arrays. Benefiting from the extremely high density of the RRAM memory, GaaS-X obtains a capacity beyond the other efforts in Table II. However, the flexibility issues and the area overhead associated with massive ADCs make the computational throughput the lowest. Furthermore, it does not support different operators like VSHIFT, VLOGIC, and VMOV in Table II.

CAPE [25] performs as an associative computing engine in the CPU. For each 32x32 SRAM CAM array as a PE, a total of 131K vector lanes could be performed simultaneously. With an advanced 7nm process, CAPE achieves the highest area efficiency. However, the frequent bitline charging in CAPE incurs lower energy efficiency compared with GaaS-X and the proposed GRAPHIC.

GRAPHIC is proposed to obtain the highest energy efficiency under the same bit width while supporting irregular data access. In addition, it has 5x area efficiency improvement compared to the baseline, and 3x area efficiency improvement compared to CAPE in a normalized process.

E. Task Evaluation

We designed a custom cycle-granularity simulator for speedup evaluation of GRAPHIC. The simulator splits the computing tasks into instructions that can be executed on GRAPHIC and the baseline architecture. The overall task execution time is obtained based on the metrics in Table II.

The applications of the two classic data access patterns in Section IV are used to evaluate the overall speedup. In Fig. 13, the graph computing contains many irregular data flows, and the image processing is featured with continuous regular data mapping. Fig. 13(a) compares the speedup of GRAPHIC against GaaS-X and CAPE. Fig. 13(b) shows the comparison of energy consumption between GRAPHIC and CAPE.

In applications with irregular data access (SSSP and CC), GaaS-X and GRAPHIC architectures with matched access structures achieve 4x-8x speedup ratios with respect to the AVX-like method. GRAPHIC is about 50% faster in graph computing compared to GaaS-X, due to the unique aggregation mechanism in GRAPHIC and higher area efficiency than ADCs used in GaaS-X. On contrary, CAPE architectures that require dense computing to take advantage of acceleration are difficult to obtain speedup in sparse computation. As shown in Table IV, although CAPE has a high total throughput, the low utilization

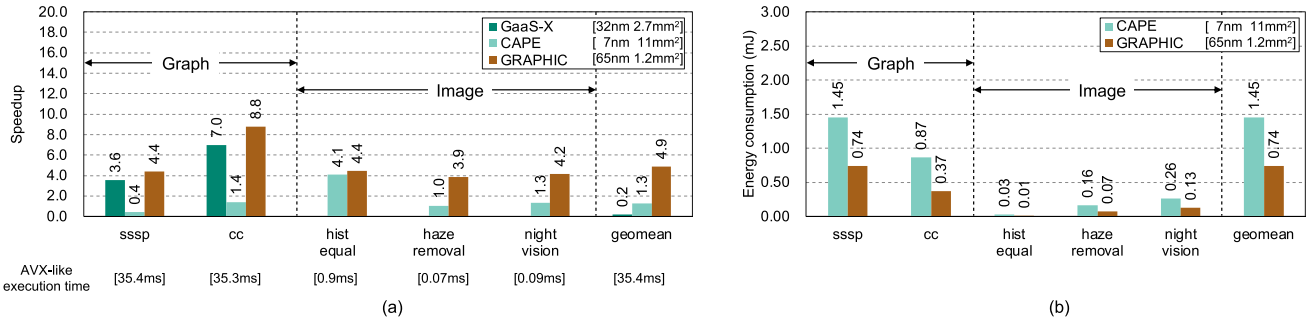

Fig. 13. (a) Speedup ratio with respect to AVX-like and (b) energy consumption of GRAPHIC compared to GaaS-X [26], and CAPE [25].

in graph computation hinders CAPE from taking advantage of it.

However, with intensive data access, the SIMD-based baseline and CAPE architecture will perform well in image-enhancing tasks (such as hist equal). In these intensive tasks, the GRAPHIC architecture can achieve performance close to that of the CAPE architecture of a larger area. Considering area normalization, GRAPHIC has a 5x-10x speedup compared to CAPE. And in tasks that require aggregation, such as the haze removal and night vision algorithm, GRAPHIC has a 3.4x improvement over CAPE and a 31x improvement considering the area normalization. GaaS-X is limited by the accumulation provided by the MAC-oriented CiM module and cannot provide acceleration for image processing computations. As shown in Table IV, the utilization rate in CAPE is significantly higher in intensive tasks such as image processing.

Overall, the GRAPHIC architecture achieves an average speedup of 3.8x-4.8x speedup compared to the AVX-like, GaaS-X, and CAPE architecture.

VI. Discussion

GRAPHIC breaks the dilemma of flexibility and parallelism faced in existing in-memory computing frameworks. With the help of the FAST SRAM and CAM, GRAPHIC supports high parallelism and irregular data selection for parallel computing. While this work only implements a partial SIMD instruction set, more operators could be supported in the future with more implemented SIMD instructions.

Sensitivity: The number of processing elements determines the maximum throughput. However, more circuits do not necessarily perform better on specific tasks. Taking graph computing and image processing as examples, increasing the array size of GRAPHIC has different impacts. For graph computing, as shown in Table IV, the utilization rate in GRAPHIC is still low. Therefore, it may not obtain better performance by directly increasing the array size. Instead, applying subgraph segmentation may be more helpful. For image processing, the bottleneck of the 128Kb GRAPHIC design is the array size. Adding computing resources here will improve the task performance.

Overhead: While achieving high-efficiency in-memory SIMD, GRAPHIC may introduce some overhead. The first that comes to mind could be the area occupation compared to the conventional SRAM. Although the bi-direction FAST SRAM at the cell level has 36% area overhead over an 8T dual-port SRAM, the overall area overhead is about only 2% when considering the peripheral circuits in the cache. Another overhead could be the additional power consumption when performing regular access (not computing) to the FAST SRAM, considering the increased parasitic capacitance inside the cell. Actually, the overall access power overhead is only about 1% because the main sources of the access energy are still the bitline charging activities and the SA operation.

Limitation: It is noted that there are some functions that are currently difficult to support with GRAPHIC, GaaS-X, and CAPE. The first function is a non-one-to-one data selection, such as using a convolutional kernel in an image for expansion or sharpening algorithms. An alternative way is to realign and copy the image data by a number of times equal to the kernel size. However, frequent data movement would incur serious energy overhead, which will eliminate the energy efficiency of IMC for this function. Another difficult function is the operations between adjacent data. Because a compute operation in GRAPHIC is restricted to two rows of the same index in two FAST SRAMs, parallel computation located in rows of a different row index, e.g., two adjacent rows, will not be supported.

Future Work: Floating-point support is an important SIMD feature. One potential way to implement floating-point calculations in GRAPHIC is to store a set of floating-point numbers in two arrays. One array is used to store the exponent and the other is used to store the mantissa. The floating-point calculation starts by aligning the exponents. The number of bits to be shifted in the mantissa needs to be obtained using subtraction in parallel. The array storing the mantissa is shifted separately according to the result of the calculation. After the exponent is aligned, the floating-point calculation would be performed according to the arithmetic logic. Preliminary investigation shows an area overhead of roughly 25% for the single-precision floating-point format because of the intermediate value during exponent alignment. Future work in complete end-to-end implementation and optimization would be meaningful. Finer granularity of bitwidth reconfigurability is also a promising direction, as it achieves better performance on some tasks at the cost of extra area and

control overheads, such as reconfiguring four 8-bit FAST SRAM subarrays instead of using the original two 16-bit subarrays.

VII. Conclusion

In this paper, a high-flexibility and high-parallelism in-memory computing architecture, namely GRAPHIC, is proposed to deal with the dilemma of existing MAC-oriented and logic-in-memory approaches. GRAPHIC utilizes the matching functionality of content-address memory (CAM) and the independently parallel computing of the FAST SRAM. On the one hand, GRAPHIC is capable of supporting SIMD instruction subsets with the in-memory computing method; on the other hand, GRAPHIC supports irregular data processing in parallel to optimize the graph processing algorithms. Therefore, GRAPHIC has established a new paradigm of energy-efficient, flexible, and high-parallel in-memory computing for various tasks including dense mapping and irregular access.

References

[1] "Beyond von Neumann," *Nat. Nanotechnol.*, vol. 15, no. 7, pp. 507–507, Jul. 2020, doi: 10.1038/s41565-020-0738-x.

[2] J. Backus, "Can programming be liberated from the von Neumann style?: A functional style and its algebra of programs," *Commun. ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978.

[3] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and trends of SRAM-based computing-in-memory for AI Edge devices," *IEEE Trans. Circuits Syst. I*, vol. 68, no. 5, pp. 1773–1786, May 2021, doi: 10.1109/TCSI.2021.3064189.

[4] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-SRAM: Enabling in-memory Boolean computations in CMOS static random access memories," *IEEE Trans. Circuits Syst. I*, vol. 65, no. 12, pp. 4219–4232, Dec. 2018, doi: 10.1109/TCSI.2018.2848999.

[5] M. Lee et al., "FeFET-based low-power bitwise logic-in-memory with direct write-back and data-adaptive dynamic sensing interface," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Des.*, Boston MA, 2020, pp. 127–132, doi: 10.1145/3370748.3406572.

[6] J. Chen, W. Zhao, Y. Wang, Y. Shu, W. Jiang, and Y. Ha, "A reliable 8T SRAM for high-speed searching and logic-in-memory operations," *IEEE Trans. VLSI Syst.*, vol. 30, no. 6, pp. 769–780, Jun. 2022, doi: 10.1109/TVLSI.2022.3164756.

[7] "A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T Bit cell enabling logic-in-memory," *IEEE J. Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016.

[8] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 5, pp. 1123–1136, May 2020.

[9] S. Angizi, Z. He, F. Parveen, and D. Fan, "IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proc. IEEE 23rd Asia South Pacific Des. Automat. Conf.*, 2018, pp. 111–116.

[10] X. Si et al., "A local computing cell and 6T SRAM-based computing-in-memory macro with 8-b MAC operation for edge AI chips," *IEEE J. Solid-State Circuits*, vol. 56, no. 9, pp. 2817–2831, Sep. 2021, doi: 10.1109/JSSC.2021.3073254.

[11] J.-W. Su et al., "15.2 A 28nm 64Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *Proc. IEEE Int. Solid- State Circuits Conf.*, San Francisco, CA, USA, 2020, pp. 240–242.

[12] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, and J. P. Kulkarni, "16.2 eDRAM-CIM: Compute-in-memory design with reconfigurable embedded-dynamic-memory array realizing adaptive data converters and charge-domain computing," in *Proc. IEEE Int. Solid- State Circuits Conf.*, San Francisco, CA, USA, 2021, pp. 248–250.

[13] J. Yue et al., "15.2 A 2.75-to-75.9 TOPS/W computing-in-memory NN processor supporting set-associate block-wise zero skipping and ping-pong CIM with simultaneous computation and weight updating," in *Proc. IEEE Int. Solid- State Circuits Conf.*, San Francisco, CA, USA, 2021, pp. 238–240, doi: 10.1109/ISSCC42613.2021.9365958.

[14] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "CMP-PIM: An energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. 55th Annu. Des. Automat. Conf.*, San Francisco CA, 2018, pp. 1–6, doi: 10.1145/3195970.3196009.

[15] A. Biswas and A. P. Chandrakasan, "CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 54, no. 1, pp. 217–230, Jan. 2019, doi: 10.1109/JSSC.2018.2880918.

[16] R. Guo et al., "A 5.1pJ/Neuron 127.3us/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65nm CMOS," in *Proc. IEEE Symp. VLSI Circuits*, Kyoto, Japan, 2019, pp. C120–C121, doi: 10.23919/VLSIC.2019.8778028.

[17] F. Tu et al., "A 28nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 reconfigurable digital CIM processor with unified FP/INT pipeline and bitwise in-memory booth multiplication for cloud deep learning acceleration," in *Proc. IEEE Int. Solid- State Circuits Conf.*, San Francisco, CA, USA, 2022, pp. 1–3, doi: 10.1109/ISSCC42614.2022.9731762.

[18] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Trans. Comput.*, vol. C–21, no. 9, pp. 948–960, Sep. 1972, doi: 10.1109/TC.1972.5009071.

[19] G. Conte, S. Tommesani, and F. Zanichelli, "The long and winding road to high-performance image processing with MMX/SSE," in *Proc. IEEE 5th Int. Workshop Comput. Architectures Mach. Percep.*, Padova, Italy, 2000, pp. 302–310, doi: 10.1109/CAMP.2000.875989.

[20] S. Zhou, C. Chelmis, and V. K. Prasanna, "Accelerating large-scale single-source shortest path on FPGA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshop*, Hyderabad, India, 2015, pp. 129–136, doi: 10.1109/IPDPSW.2015.130.

[21] L. Arge, G. S. Brodal, and L. Toma, "On external-memory MST, SSSP, and multi-way planar graph separation," in *Algorithm Theory - SWAT 2000*, vol. 1851, Berlin, Germany: Springer, 2000, pp. 433–447, doi: 10.1007/3-540-44985-X_37.

[22] T. Geng et al., "AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing," in *Proc. IEEE/ACM 53rd Annu. Int. Symp. Microarchitecture*, Athens, Greece, 2020, pp. 922–936, doi: 10.1109/MICRO50266.2020.00079.

[23] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006, doi: 10.1109/JSSC.2005.864128.

[24] Y. Chen et al., "FAST: A fully-concurrent access SRAM topology for high row-wise parallelism applications based on dynamic shift operations," *IEEE Trans. Circuits Syst. II*, vol. 70, no. 4, pp. 1605–1609, Apr. 2023, doi: 10.1109/TCSII.2022.3231589.

[25] H. Caminal et al., "CAPE: A content-addressable processing engine," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture*, Seoul, Korea (South), 2021, pp. 557–569, doi: 10.1109/HPCA51647.2021.00054.

[26] N. Challapalle et al., "GaaS-X: Graph analytics accelerator supporting sparse data representation using crossbar architectures," in *Proc. IEEE/ACM 47th Annu. Int. Symp. Comput. Architecture*, Valencia, Spain, 2020, pp. 433–445, doi: 10.1109/ISCA45697.2020.00044.

[27] C. C. Foster, *Content Addressable Parallel Processors*. Hoboken, NJ, USA: Wiley, 1976.

[28] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, and C. Asthagiri, "ASC: An associative-computing paradigm," *Computer*, vol. 27, no. 11, pp. 19–25, Nov. 1994, doi: 10.1109/2.330039.

[29] G. E. Sayre, "Staran: An associative approach to multiprocessor architecture," in *Computer Architecture*, vol. 4, W. Händler and R. K. Bell, Eds. Berlin, Germany: Springer, 1976, pp. 199–221, doi: 10.1007/978-3-642-66400-7_9.

[30] K. He, J. Sun, and X. Tang, "Single image haze removal using dark channel prior," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 12, pp. 2341–2353, Dec. 2011, doi: 10.1109/TPAMI.2010.168.

[31] X. Dong et al., "Fast efficient algorithm for enhancement of low lighting video," in *Proc. IEEE Int. Conf. Multimedia Expo*, Barcelona, Spain, 2011, pp. 1–6, doi: 10.1109/ICME.2011.6012107.

[32] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, pp. 1–25, Jul. 2017, doi: 10.1145/3085572.

[33] N. P. Jouppi, A. B. Kahng, N. Muralimanohar, and V. Srinivas, "CACTI-IO: CACTI with OFF-chip power-area-timing models," *IEEE Trans. VLSI Syst.*, vol. 23, no. 7, pp. 1254–1267, Jul. 2015, doi: 10.1109/TVLSI.2014.2334635.

Yiming Chen (Student Member, IEEE) received the BS degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2021. He is currently working toward the PhD degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His current research interests include computing-in-memory architecture and co-optimization on artificial intelligence.

Mingyen Lee (Student Member, IEEE) received the BS degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2021. He is currently working toward the master's degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research interests mainly include energy-area-efficient computing-in-memory designs.

Guohao Dai (Member, IEEE) received the BS and PhD (with honor) degrees from Tsinghua University, Beijing, in 2014 and 2019. He is joining Shanghai Jiao Tong University, Shanghai, China, as an associate professor. His research interests include mainly focuses on large-scale sparse graph computing, heterogeneous hardware computing, emerging hardware architecture, and etc.

Mufeng Zhou (Student Member, IEEE) received the BS degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2021. He is currently working toward the master's degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China. His research interests mainly include computing-in-memory circuit and system.

Nagadastagiri Challapalle received the PhD degree in computer science and engineering from the Pennsylvania State University, State College, PA, and his thesis proposed processing-in-memory architectures for deep learning and graph analytics applications. He is a hardware architect with Nvidia in Santa Clara, CA, USA. His research interests include the areas of performance and power modeling for parallel computer architectures, and efficient scale-out architectures for deep learning training applications.

Tianyi Wang received the BS degree from Shandong University, and the MS degree from Ulm University. After joining Mercedes-Benz in 2012, he worked in different roles in crowd data analysis and advanced driver assistance systems. He is currently in charge of ADAS development and validation.

Yao Yu received the BS degree from the Beijing Institute of Technology, and the MS degree from the Karlsruhe Institute of Technology. She joined the Department of Research & Development Automated Driving System since 2018. She is currently developing an automated driving system-data recorder.

Yongpan Liu (Senior Member, IEEE) received the BS, MS, and PhD degrees from Tsinghua University, in 1999, 2002, and 2007, respectively. He is currently a full professor with the Department of Electronic Engineering, Tsinghua University, China. He is a Program Committee Member for ISSCC, A-SSCC and DAC.

Yu Wang (Fellow, IEEE) received the BS and PhD (with honor) degrees from Tsinghua University, Beijing, in 2002 and 2007. He is currently a tenured professor with the Department of Electronic Engineering, Tsinghua University. His research interests include brain inspired computing, application specific hardware computing, parallel circuit analysis, and power/reliability aware system design methodology.

Huazhong Yang (Fellow, IEEE) received the BS degree in microelectronics and the MS and PhD degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, where he has been a full professor since 1998.

Vijaykrishnan Narayanan (Fellow, IEEE) received the BS degree in computer science and engineering from the University of Madras, Chennai, India, in 1993, and the PhD degree in computer science and engineering from the University of South Florida, Tampa, FL, USA, in 1998. He is currently the Robert Noll chair professor of Computer Science and Engineering and Electrical Engineering with Pennsylvania State University, University Park, PA. He is also the co-director of the Microsystems Design Laboratory.

Xueqing Li (Senior Member, IEEE) received the BS and PhD degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2007 and 2013, respectively. He is currently an associate professor with the Department of Electronic Engineering, Tsinghua University. From 2013 to 2017, he was a postdoctoral researcher with the Department of Computer Science and Engineering, Penn State University, State College, PA. He joined the Department of Electronic Engineering, Tsinghua University, as an assistant professor, in 2018. He has more than 100 publications. He holds more than 20 patents. His research interests include emerging memory, memory-oriented computing, and high-performance data converters. He is currently the TPC member of DAC, ICCAD, ASP-DAC, GLSVLSI, ISVLSI, etc.