A Survey on Near Data Processing Accelerators for Graph Analytics

Makesh Tarun Chandran

Department of Electrical Engineering and Computer Science
Pennsylvania State University
Email: mzc88@psu.edu

***Abstract*—Graph algorithms are ubiquitous, from social science to machine learning analysis. They are also notorious for performing poorly in state-of-the-art computing systems, requiring more compute capability than they need. This is due to the memory bandwidth limitations of the current computing systems. In this regard, the Processing-In-Memory technology shows potential in tackling the bottleneck. A few graph analytics accelerators have been proposed based on the in-memory computing technology in recent years. This paper attempts to summarize the basic graph programming models, various in-memory technologies, and finally discuss the recently developed graph accelerator's endeavor at alleviating various performance bottlenecks.**

1. Introduction

Graph algorithms are an interesting class of application for which a single mac mini desktop equipped with SSD outperforms a medium sized cluster! [1] The primary reason for such a counter-intuitive behavior can be attributed to the irregular data-access pattern inherent in those algorithms. They also have a low computation to communication ratio which is exacerbated in the modern memory-bandwidth limited systems. However, graph algorithms are highly iterative and there is abundance of parallelism that can be exploited. These idiosyncrasies of graph algorithm render commercial HPC systems inefficient or unsuitable for accelerating such applications. In other words, there's a lot of scope for deriving performance improvements from a processor customised for graph processing. Extensive research on the topic in the past decade have revealed new insights on the algorithmic, programming model and hardware aspects of such systems. From the hardware perspective, emergence of in-memory computing technology has enabled us to overcome the limitation of the conventional Von-Neumann architecture. Logic units can now be integrated into the memory cells, evading the dreaded off-chip communication. Such technologies have given birth to Processing-In-Memory (PIM) based accelerators for graph analytics.

The rest of the paper is organised as follows. Two basic graph processing techniques are discussed in Section 2. Section 3 briefly introduces PIM techniques on different memory technologies. Section 4 discusses some of the state-of-the-art graph analytics accelerators building on the technologies described in Section 2 & 3.

2. Graph Processing

Current graph analytics frameworks are based on two fundamentally different computation models - Vertex-centric and Edge-centric processing. Among these two, Vertex-centric programming models are prominent as they are more intuitive than the other. Matrix Algebra-based systems which performs operations on the adjacency matrix of a graph is another intuitive model for graph processing.

2.1. Vertex-Centrix Processing

In Vertex-Centrix processing frameworks, the vertices are treated as first class objects and edges as association between two vertices. Vertex-Centric progrmming model provides an intuitive framework to construct a large scale graph program based on the computation on a single vertex. Two primary categories of the vertex-centric programming model are summarised below.

2.1.1. Pregel. Pregel [2] is a vertex-centric programming model following the Bulk Synchronous Parallel style of execution. The computation on vertices are represented as a sequence of supersteps with a global synchronisation among the vertices participating in a superstep. A superstep only contains the set of active nodes which was enabled by the vertices of previous superstep. The vertices in a superstep can be operated on in parallel and they generate messages to vertices of the next superstep. The program halts when there are no active vertices in the next superstep. Algorithm 1 is a pseudo code of a typical Pregel program. The Synchronicity of the model makes it easy to reason about and avoids any possibility deadlock or data races. One limitation with this model is the presence of the global barrier synchronisation, whose performance effects can be mitigated by having adequate parallelism slack. Some of the frameworks that are based on the Pregel programming are Giraph, Giraph++, Mizan, GPS, Pregelix and Pregel+.

2.1.2. Gather-Apply-Scatter. GAS [3] programming model splits graph processing into three steps - Gather, Apply

Algorithm 1 Pregel Abstraction

```
1: super_step ← 0
2: while active_vertices ≠ ∅ do
3:     for v ∈ active_vertices do
4:         v.value ← v.compute(msgs)
5:         v.send_message(neighbourhood(v))
6:         v.halt()
7: super_step ← super_step + 1
```

and Scatter. During the Gather phase a vertex accumulates information about its adjacent edges and processes it. The processed value is used to update itself and its neighbors in Apply and Scatter phase respectively. The updated neighbors are activated and will go through the GAS steps. The program halts when there are no updated vertices. The GAS model can be realised as both synchronous and asynchronous execution depending on the requirements. One fundamental difference from pregel is that vertices can directly access the neighborhood without having to be explicitly communicated to it. As a result, this requires shared memory abstraction for the GAS model to be realised, which increases the memory access latency in a distributed systems. Techniques such as data replication, prefetching, etc. can be employed to overcome the bottleneck. A basic sequence of steps involved in GAS model is presented in Algorithm 2. GraphLab, GraphChi and PowerGraph are some of the frameworks developed on GAS Abstraction.

The paper [14] does a performance evaluation of the various GAS and Pregel based large scale graph processing framework. GAS models enables data partitioning based on edges whereas pregel model is limited to vertex based partitioning. As many natural graphs follow power law like vertex-degree distribution, edge-based graph partitioning leads to better load balancing for such graphs.

Algorithm 2 GAS Abstraction

```
1: super_step ← 0
2: while active_vertices ≠ ∅ do
3:     for v ∈ active_vertices do
4:         accumulator ← sum(v.gather(neighbourhood(v))
5:         v.value ← v.apply(accumulator, val(v))
6:         v.scatter(neighbourhood(v))
7: super_step ← super_step + 1
```

2.2. Matrix Algebra-Based Systems

Before the advent of graph style processing frameworks such as Pregel, graph computations were modelled as matrix-vector multiplication. Every iteration executes the user defined function on the adjacency list(vector) of a vertex and update the adjacency matrix. Such matrix based computation enables the use of optimisations such as efficient data layouting, prefetching and use of standard linear algebra sub-routines. PEGASUS, GBASE and SystemML are three frameworks which incorporate matrix based graph computation in their framework. One drawback of matrix based graph computation is the amount of redundant work that is performed every step. Each iteration triggers a full matrix computation even though only a few of the node update their value. In contrast, in pregel like systems, computation is performed only on the active nodes which are bound to update their values. To overcome the limitation, some frameworks present a vertex-centrix programming framework and translate it to matrix computation underneath to exploit the best of both the worlds.

3. In-Memory Processing

The In-memory processing techniques focus on bridging the bandwidth gap between compute resources and memory elements by moving them close to each other. With new memory technologies, it is now possible to embed logical operations on the memory cell itself. These techniques have merits and demerits of their own which needs to be properly studied before picking one for an application.

3.1. Near-Data Processing

The Near-Data Processing (NDP) architectures [4], [10] focuses on bringing the data close to the memory by employing various techniques. They replace the conventional hardware-managed cache hierarchy with explicitly managed memory hierarchy. The management technique can be suited to application and system needs. Also these frameworks usually consider the memory hierarchy all the way till secondary storage, involving Disks and SSDs, to improve performance. However, it is possible to improve performance by extracting more bandwidth, when the processing is integrated within memory avoiding the off-chip communication.

3.2. Processing In-memory

Advances in 3D integration on memory devices gave rise opportunities to integrate logic into the memory cells. Some of the integration techniques with various memory technologies are discussed below.

3.2.1. SRAM-PIM. SRAM-PIM is about integrating basic bit-operations logic with the SRAM cells. The paper [21] presents a SRAM-PIM based pattern recognition accelerator, which exploits the relaxed precision and linearity requirements of pattern recognition applications to utilise the SRAM based compute memory. The work in [22] introduces a new content addressable memory (CAM) based on the conventional 6T SRAM cells with configurable memory and logic functions. The processing involved in a search operation in CAM is offloaded to the logic in the memory improving system performance and efficiency. The paper on Compute Caches [23] uses emerging bit-line SRAM circuit technology to re-purpose existing cache elements and transforms them into active very large vector computational

units. In-place Compute Cache reduces data movement overhead between a cache's sub-arrays and its controller while restricting the type of operations that can be supported and, the placement of operands. The work [24] is a machine learning accelerator based on SRAM PIM however with the overhead of large DACs. The paper [25] presents a novel 6T SRAM cell using the n-well as the write wordline to perform write operations and eliminate the write access transistors, achieving 15% area saving compared to conventional 8T SRAM.

3.2.2. DRAM-PIM. DRAM-PIM technology builds on the practically demonstrated 3D stacking of silicon dies. This technology not only reduces the communication latency across banks but also offers provisions to embed simple logic layers in the memory die. The Hybrid Memory Cube (HMC) [15], [16] supports upto 8 layers of DRAM dies and an additional logic layer integrated by Through-Silicon Vias (TSV). HMC provides more than 10x increase in bandwidth with less than 1/3rd the energy consumed by a commercial DDR4 module. The embedded logic layer can be equipped with basic operations to pre-process the data before providing it to the host processor for further processing. This reduces the amount of data movement between the processor and the memory leading to efficient usage of limited memory bandwidth and lowered energy consumption.

The performance of HMC is limited by the bandwidth between the memory banks and logic layer. Another technique to improve the DRAM performance is by integrating logic with the memory cells and sense amplifier. The paper [18] models the bitwise AND and OR operations on data across two DRAM rows, within the memory banks. This techniques achieves over 9X throughput and 50x reduced energy consumption for bulk bitwise operations. However, the read operations on DRAM cells are destructive and needs to be backed up before invoking any operations on them. The DRISA paper [19] presents modification at the cell (1-Transistor and 1-Capacitor) and sense amplifier level to optimise the above overheads. The architecture presented in the In-Memory Intelligence(IMI) paper [20] uses simple bit-serial computing elements attached to the DRAM array's sense amplifier. The IMI architecture is a standard DRAM in form and function with the ability for massive SIMD parallelism and a standard and familiar programming model.

3.2.3. NVM-PIM. The charge-based memory technology doesn't scale as much as the CMOS technology and has lead to a gap in performance between the memory and logic. The recently developed resistive memory technology [27] shows promising capabilities in bridging the gap between the storage and processor. Resistive memories including Phase change memories and Spin torque memories change the resistance of the storage elements and can scale better than the charge-based memories. However, as the resistive memories need to change the phase of the element, it incurs high write energy and long write latency. In spite of the above limitation, it offers non-volatility, high density and low overall power-consumption compared to the CMOS technology [26].

One class of PIM technique [28], [29] use the memresistive devices connected in a crossbar structure with peripheral circuitry to realise various logic functions. The other class of PIM technique [30], [31] exploit analog properties of memresistive devices to realise logic on the memory cell itself. The work Memresistor-Aided Logic (MAGIC) [30] realises the NOR and other basic operation on a memory cell while retaining the input data. The idea in MAGIC is further extended in [32] to support bitwise addition and subtraction using memresistive crossbar. This marks of the state of the art in NVM-based Processing-In-Memory technology.

4. Accelerators

An Ideal graph processing system is where the performance increases proportional to the size of the graphs that can be stored in the system. Unfortunately, in conventional systems, memory bandwidth remains almost constant irrespective of the memory capacity due to pin count limitation per chip. Traditional caching mechanisms fail to sustain the memory throughput requirement due to the application's poor locality. Following is the summary of accelerators with unconventional memory design, developed to mitigate the performance bottleneck in graph processing systems.

4.1. Graphicionado

Graphicionado [4] is a custom hardware accelerator based on the Gather-Apply-Scatter model. The execution in each of the phases are broken down into pipeline stages and custom hardware units are deployed to perform the operations. There are about 10 stages in the Processing phase and 6 stages in the Update phase. A vertex can only be in one of the stage making the system inherently atomic. The memory subsystem is also redesigned to suit graph processing. Scratchpad memory is used to buffer the temporary updates to the vertices, and an edge Id table which stores the first edge of each vertex, eliminating most of the random accesses to the memory. Other optimizations such as streamed data access, and prefetching are also supported.

To scale the processing unit, the pipeline stages are split into source-oriented and destination-oriented stages and the multiple instance of each them are inter-connected through a crossbar. The parallelization of source and destination streams eliminates memory access conflicts as each of the streams access regions exclusively assigned to it. To achieve this, graph slicing techniques are used to create partitions of graphs which have minimum interaction with each other. This trivializes the scaling of the scratchpad memory system to simply creating m instances of scratchpad units for m source-and-destination streams. The accelerator achieves upto 6x speedup while consuming less than 2% of energy of compared to the state-of-the-art software graphs analytics framework running on a 16-core Haswell Xeon server. However, for large graphs it is not feasible to hold all the relevant metadata in scratchpad and data movement

TABLE 1: Comparison of various features of Graph Analytics Accelerators

	Graphicianado	**GraphH**	**Tesseract**	**GraphR**	**GraphP**
Programing Model	Gather-Apply-Scatter	Gather-Apply-Broadcast	Pregel-like	Pregel-like	Pregel like + Two phase Update
Execution Model	Pipelined	Message passing and shared memory	Message passing and iterative	Sparse matrix computation	Message passing and iterative
Computing Mode	Sync	Sync	Sync & Async	Sync	Sync & Async
Memory Organisation	Explicitly managed in Scratchpad	Memory-disk hybrid	DRAM-PIM (HMC)	ReRAM-PIM	DRAM-PIM (HMC)
Core	Custom pipeline processor	General purpose processor	In-order core with prefetcher	ReRam Crossbar & ALUs	In-order core with prefetcher
Generality	Any Vertex program	Any Vertex or Edge based program	Any Vertex program	Vertex program in SpMV	Any Vertex program

across the hierarchy degrades the system performance for such large inputs.

4.2. GraphH

GraphH [11] is a high performance distributed big graph analytics framework optimized for a small number of clusters. It is built on a spark-based graph processing engine, a distributed file-system and MPI based graph processing engine. The underlying principle is to implement a memory-disk hybrid approach which reduces the data movement overhead by maximising the reuse of in-memory data. In this regard, three techniques are employed:

1) The graph is evenly split into tiles and distributed to servers for performing computation while the whole graph is replicated in all the systems
2) Gather-Apply-Broadcast(GAB) is used to represent the distribution of tiles across the server.
3) Edge Cache system -lines are evicted only when cache is full - is employed to utilise the idle memory to reduce disk IO overhead.

The difference between GAB and GAS is that GAB maintains replicas of data in every server, which have to be updated by a 'Broadcast' after every 'Apply' step. Various graph partitioning mechanisms such as Hash-based Edge-cut, Intelligent Vertex-cut and Stream partitioning are also analysed for communication and computation overheads on small clusters. Performance evaluation shows that GraphH can be 7.8X faster than the in-memory systems such as Pregel+ and Powergraph while processing generic graphs, and 100X faster than the out-of-core systems such as GraphD and Chaos.

4.3. Tesseract

The Tesseract architecture [5] exploits the internal bandwidth of the HMC-RAM, which is an order of magnitude higher than the off-chip bandwidth, by integrating a logic layer within the memory die. The HMC in Tesseract is organised as vaults which are vertical slices of memory in a cube. Each vault is composed of a 16-bank DRAM partition and a dedicated memory controller. The logic layer of a vault is equipped with a single issue in-order core. Each cube can host upto 32 vaults and 8 high-speed serial links as off-chip interface. The Tesseract cores (cubes) are integrated with the host as a memory-mapped co-processor device.

The host distributes jobs between the cores based on the location of data relevant to the job. During execution data access to the remote cubes are facilitated through a low cost message passing mechanism. The communication mechanism is inherently atomic and its latency can be hidden through asynchronous communication. The remote access latency can also be overcome by employing a data prefetcher which derives heuristics from the program and execution trace. The programming interface consists of a set of APIs capturing basic functions in the underlying hardware, on which any commercial graph analytics framework can be built. The proposed system was evaluated on five state-of-the-art graph processing applications and achieve a speedup of over 14x compared to the standard DDR3 based systems. Tesseract also achieves memory-capacity proportional performance, which is the key to handling increasing amounts of data in a cost-effective manner. However, when scaling the system with multiple Tesseract cores, the off-chip communication latency incurred while accessing data in a remote core becomes the performance bottleneck.

4.4. GraphR

GraphR [6] leverages on the fact that graph computations are inherently tolerant to imprecision and resilient to errors, to perform approximate computations on the graph, using ReRAM crossbar discussed in section 2.2. The GraphR architecture contains two key components : memory ReRAM and graph engine. The graph is stored in memresistor-based RAM (called ReRAM) in compressed sparse representation of its adjacency matrix. The graph engine is equipped with ReRAM crossbars, Analog to Digital converters, a simple ALU and IO registers to cache IO values. The schemes for mapping two prominent operations - parallel-MAC and parallel-ADD are described in the paper. To facilitate the

data movement between GE and memory, streaming-apply mechanism is employed. Graphs are stored in coloumn major order, and vertices in a coloumn are streamed onto GEs to be processed together. GraphR can be plugged in as an accelerator to a host processor which assigns a subgraph of a large graph for processing. The experiment results show that GraphR achieves 16X speedup and a 33X energy saving on geometric mean compared to a CPU baseline system. Compared to GPU, GraphR achieves 1.69X to 2.19X speedup and consumes 4.7X to 8.9X less energy. GraphR gains a speedup of upto 4X, and is 10X more energy efficiency compared to Tesseract [5]. However, the whole idea rests on the assumption that the proposed memresistive device with its energy characteristics is practically feasible, which has not been demonstrated yet. Also, the SpMV based computation model limits the number of graph applications that can be efficiently supported by the proposed architecture.

4.5. GraphP

The motivation behind GraphP [7] is to reduce the excessive cross-cube communication overhead through SerDes links whose bandwidth is much less than the bandwidth available within a cube. The design considers data organisation as the first-order design criteria in mitigating the bottleneck. The techniques developed for the same are -

1) 'Source-Cut' partitioning - Split the graph such that all the incoming edges of a vertex are contained in the same cube.
2) 'Two-phase vertex' programming model - *GenUpdate* to generate the vertex value update on all incoming edges & *ApplyUpdate* to apply the update to each vertex. The two phases can be overlapped, with each other and the execution, to further hide the latency.

With these optimisations, the proposed architecture is able to to achieve a speedup of up to 1.7x and energy efficiency of 89% compared to Tesseract. This is at the expense of programmability of the hardware, due to the introduction of Two-phase vertex programming. Also Source-Cut partitioning may not be feasible for power law like vertex degree distribution where some vertices have large number of edges which may not fit inside a cube.

5. Conclusion and Future Work

The primary challenge in graph analytics systems has been handling the disparity between communication and computation requirements. Further performance improvements have to come from redesigning the memory subsystems as the processing capability has hit a wall. The advent of new memory technology has opened up a new design space that can advance the existing computing systems. The architectures discussed above were proposed in the last three years, after in-memory processing became feasible.

Other survey in Graph Analytics accelerators are [12], [13], containing hardware or software stack focused perspective of the topic.

References

[1] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. 2012. GraphChi: large-scale graph computation on just a PC. In Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12). USENIX Association, Berkeley, CA, USA, 31-46.

[2] Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 135-146.

[3] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: distributed graph-parallel computation on natural graphs. In Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12). USENIX Association, Berkeley, CA, USA, 17-30.

[4] T. J. Ham, L. Wu, N. Sundaram, N. Satish and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, 2016, pp. 1-13.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA), Portland, OR, 2015, pp. 105-117.

[6] L. Song, Y. Zhuo, X. Qian, H. Li and Y. Chen, "GraphR: Accelerating Graph Processing Using ReRAM," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, 2018, pp. 531-543.

[7] M. Zhang et al., "GraphP: Reducing Communication for PIM-Based Graph Processing with Efficient Data Partition," 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Vienna, 2018, pp. 544-557.

[8] Muhammad Imran, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. 2015. Processing Social Media Messages in Mass Emergency: A Survey. ACM Comput. Surv. 47, 4, Article 67 (June 2015), 38 pages.

[9] C. Unsalan and B. Sirmacek, "Road Network Detection Using Probabilistic and Graph Theoretical Methods," in IEEE Transactions on Geoscience and Remote Sensing, vol. 50, no. 11, pp. 4441-4453, Nov. 2012.

[10] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. 2016. Biscuit: a framework for near-data processing of big data workloads. SIGARCH Comput. Archit. News 44, 3 (June 2016), 153-165.

[11] G. Dai et al., "GraphH: A Processing-in-Memory Architecture for Large-scale Graph Processing," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

[12] Systems for Big Graph Analytics by Yan, Da, Tian, Yuanyuan, Cheng, James

[13] Gupta, S. (2018). Processing in Memory using Emerging Memory Technologies. UC San Diego. ProQuest ID: Merritt ID: ark:/13030/m5bg7kwn. Retrieved from https://escholarship.org/uc/item/95z3z84c

[14] Omar Batarfi, Radwa El Shawi, Ayman G. Fayoumi, Reza Nouri, Seyed-Mehdi-Reza Beheshti, Ahmed Barnawi, and Sherif Sakr. 2015. Large scale graph processing systems: survey and an experimental evaluation. Cluster Computing 18, 3

[15] J. T. Pawlowski, "Hybrid memory cube (HMC)," 2011 IEEE Hot Chips 23 Symposium (HCS), Stanford, CA, USA, 2011, pp. 1-24.

[16] R. Nair et al., "Active Memory Cube: A processing-in-memory architecture for exascale systems," in IBM Journal of Research and Development, vol. 59, no. 2/3, pp. 17:1-17:14, March-May 2015.

[17] Q. Dong et al., "A 4 + 2T SRAM for Searching and In-Memory Computing With 0.3-V V_{DDmin}," in IEEE Journal of Solid-State Circuits, vol. 53, no. 4, pp. 1006-1015, April 2018.

[18] V. Seshadri et al., "Fast Bulk Bitwise AND and OR in DRAM," in IEEE Computer Architecture Letters, vol. 14, no. 2, pp. 127-131, 1 July-Dec. 2015.

[19] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: a DRAM-based Reconfigurable In-Situ Accelerator. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17). ACM, New York, NY, USA, 288-301

[20] T. Finkbeiner, G. Hush, T. Larsen, P. Lea, J. Leidel and T. Manning, "In-Memory Intelligence," in IEEE Micro, vol. 37, no. 4, pp. 30-38, 2017. doi: 10.1109/MM.2017.3211117

[21] M. Kang, M. Keel, N. R. Shanbhag, S. Eilert and K. Curewitz, "An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM," 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Florence, 2014, pp. 8326-8330.

[22] S. Jeloka, N. B. Akesh, D. Sylvester and D. Blaauw, "A 28 nm Configurable Memory (TCAM/BCAM/SRAM) Using Push-Rule 6T Bit Cell Enabling Logic-in-Memory," in IEEE Journal of Solid-State Circuits, vol. 51, no. 4, pp. 1009-1021, April 2016.

[23] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw and R. Das, "Compute Caches," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 481-492.

[24] J. Zhang, Z. Wang and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," in IEEE Journal of Solid-State Circuits, vol. 52, no. 4, pp. 915-924, April 2017.

[25] Q. Dong et al., "A 4 + 2T SRAM for Searching and In-Memory Computing With 0.3-V V_{DDmin}," in IEEE Journal of Solid-State Circuits, vol. 53, no. 4, pp. 1006-1015, April 2018.

[26] Qing Guo, Xiaochen Guo, Yuxin Bai, and Engin pek. 2011. A resistive TCAM accelerator for data-intensive computing. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44). ACM, New York, NY, USA, 339-350.

[27] Dmitri B Strukov, Gregory S Snider, et al. The missing memristor found. Nature, 453(7191):8083, 2008.

[28] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen and H. Yang, "Memristor-based approximated computation," International Symposium on Low Power Electronics and Design (ISLPED), Beijing, 2013, pp. 242-247.

[29] Yongtae Kim, Yong Zhang, and Peng Li. 2015. A Reconfigurable Digital Neuromorphic Processor with Memristive Synaptic Crossbar for Cognitive Computing. J. Emerg. Technol. Comput. Syst. 11, 4, Article 38 (April 2015), 25 pages.

[30] S. Kvatinsky et al., "MAGICMemristor-Aided Logic," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 61, no. 11, pp. 895-899, Nov. 2014.

[31] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny and U. C. Weiser, "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 10, pp. 2054-2066, Oct. 2014.

[32] N. Talati, S. Gupta, P. Mane and S. Kvatinsky, "Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC)," in IEEE Transactions on Nanotechnology, vol. 15, no. 4, pp. 635-650, July 2016.