Sorting it out in Hardware: A State-of-the-Art Survey

Amir Hossein Jalilvand , Faeze S. Banitaba , Seyedeh Newsha Estiri , Sercan Aygun , *Member, IEEE*, and M. Hassan Najafi , *Senior Member, IEEE*

Abstract—Sorting is a fundamental operation in various applications and a traditional research topic in computer science. Improving the performance of sorting operations can have a significant impact on many application domains. For highperformance sorting, much attention has been paid to hardwarebased solutions. These are often realized with application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). Recently, in-memory sorting solutions have also been proposed to address the movement cost issue between memory and processing units, also known as Von Neumann bottleneck. Due to the complexity of the sorting algorithms, achieving an efficient hardware implementation for sorting data is challenging. A large body of prior solutions is built on compare-and-swap (CAS) units. These are categorized as comparison-based sorting. Some recent solutions offer comparison-free sorting. In this survey, we review the latest works in the area of hardwarebased sorting. We also discuss the recent hardware solutions for partial and stream sorting. Finally, we will discuss some important concerns that need to be considered in the future designs of sorting systems.

Index Terms—comparison-based sorting, comparison-free sorting, hardware-based sorting, in-memory sorting, partial sorting.

I. Introduction

Today, the data volume has increased significantly in many application domains. Processing data at the terabyte and petabyte levels has become routine. Processing large volumes of data is challenging and is expected to remain at an upward rate [1]. Sorting is one of the substantial operations in computer science performed for different purposes, from putting data in a specific order, such as *ascending* or *descending*, to find the minimum and maximum values, finding the median, and partial sorting to find the top-*m* greatest or smallest values. As Fig. 1 shows sorting is used in many application domains, from data merging to big data processing [2], [3], database operations [4] especially when the scale of files/data are very large, robotics [5]–[7], signal processing (*e.g.*, sorting radar signals) [8]–[10], and wireless networks [11].

Sorting the time series data according to their timestamps holds critical importance in numerous artificial intelligence (AI) applications, such as forecasting and anomaly detection [31], where the sequential occurrence of events is of paramount significance [32]. Wireless sensor network applications often incorporate genetic algorithms, with the 'Nondominated Sorting Genetic Algorithm (NSGA)' being a commonly employed and efficient approach requiring sorting [33].

The authors are with the School of Computing and Informatics, University of Louisiana, LA, 70504, USA. Email: {amir.jalilvand, faeze-sadat.banitaba1, seyedeh-newsha.estiri1, sercan.aygun, najafi}@louisiana.edu This work was supported in part by National Science Foundation (NSF) grant #2019511, the Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, and generous gifts from Cisco, Xilinx, and NVIDIA.

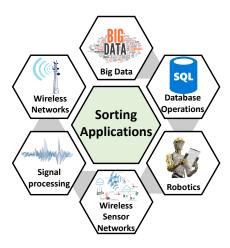


Fig. 1. Common applications of sorting: Big Data [12]–[14], Database operations [15]–[17], Robotics [18]–[21], Wireless Sensor Networks [22]–[26], Signal Processing [9], [10], [27]–[29], and Wireless Networks [30].

Additionally, wireless networks necessitate the implementation of sorting algorithms that are both energy-optimal and energy-balanced, such as enhanced sorting algorithms [24]. The concept of sorting also extends to the realm of robotic visual tasks. Much like traditional scalar sorting, the sorting of items based on attributes like color, shape, or other features within a robot's perceived environment constitutes a tangible engineering application of sorting [21]. In the field of robotics, object sorting is a significant task. Particularly in computer vision applications, sorting objects by robots based on their perceived environment is challenging [19]. Another intriguing application is to control greenhouse climatic factors through sorting networks [34]. For sorting large-scale datasets, some researchers adopt an external sorting methodology. External sorting serves as a solution for sorting vast datasets that cannot fit into the primary memory of a computing platform. Instead, it utilizes additional memory elements like hard disk drives, employing a sort-and-merge strategy [35]. Sorting also finds unconventional applications in signal processing. This extends from theoretical scalar sorting to sorting tasks in realworld signal processing. An illustrative example is radar signal sorting, a recent and intricate sorting challenge in the context of multi-function radar systems [28].

Improving the sorting speed can have a significant impact on all these applications. Many software- and hardware-based solutions have been proposed in the literature for high-performance sorting. Software-based solutions rely on powerful single/multi-core and graphics processing unit (GPU)-based processors for high performance [39]. Much attention has been paid to hardware sorting solutions, especially for

TABLE I

COMPARISON BETWEEN THE EXISTING SURVEYS FOR HARDWARE SOLUTIONS OF SORTING
(○: NOT COVERED, ▶: PARTIALLY COVERED, ♠: FULLY COVERED)

Article	Year	Sorting Networks	Comparison-based Sorting	Comparison-free Sorting	Partial Sorting	In-memory Sorting	
Jmaa et al. [36]	2019	О	•	О	О	О	
Skliarova [37]	2021	•	•	О	О	О	
Ali [38]	2022	Þ))	О	О	
This work		•	•	•	•	•	

applications that require very high-speed sorting [6], [40], [41]. These have been implemented using either application-specific integrated circuits (ASICs) or field-programmable gate arrays (FPGAs). Depending on the target applications, the hardware sorting units vary greatly in how they are configured and implemented. The number of inputs can be as low as nine for some image processing applications (*e.g.*, median filtering [42]) to tens of thousands [40], [41]. The data inputs have been binary values, integers, or floating-point numbers ranging from 4- to 256-bit precision.

Hardware cost and power consumption are the dominant concerns with hardware implementations. The total chip area is limited in many applications [43]. As fabrication technologies continue to scale, keeping chip temperatures low is an important goal since leakage current increases exponentially with temperature. Power consumption must be kept as low as possible. Developing low-cost, power-efficient hardware-based solutions to sorting is an important goal [41].

There is a large body of work on the design of customized sorting hardware. These works seek to utilize the hardware resources fully and to provide a custom, cost-effective hardware sorting engine. Developing hardware-efficient implementations for sorting algorithms is challenging, considering the complexity of these algorithms [40], [41], [44]. A significant amount of hardware resources is spent by comparators, memory elements including large global memories, complex pipelining, and complicated local and global control units [44].

Many of the prior hardware solutions are built on basic compare-and-swap (CAS) units that compare pairs of data and swap if needed. These solutions are categorized as comparison-based sorting. As shown in Fig. 2, each basic CAS unit is conventionally implemented with a binary comparator and two multiplexers (MUX) units [41]. Sorting networks of CAS units are frequently used for fast and parallel hardware sorting. Their inherent parallelism enables them to achieve sorting at a considerably faster rate than the fastest sequential software-based sorting algorithms. However, these CAS-based hardware solutions suffer from high hardware costs, especially when the *number* and *precision* of input data increase [45]. In the last few years, some comparison-free/quasi-comparisonfree sorting solutions have been proposed to address the challenges with comparison-based sorting designs. We will discuss these novel solutions in Section II-B.

Complete sorting sorts all items (N) of a list. Partial sorting has also been a popular sorting variant. Unlike complete sorting, partial sorting returns a list of the k smallest or largest elements in order where K < N [40], [75]. The cost of partial sorting is often substantially less than complete sorting when the number of sub-sorting attempts is small compared

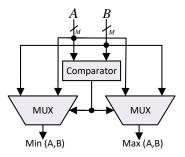


Fig. 2. Compare-and-Swap (CAS) operation in hardware.

to N. The other elements (above the k smallest ones) may also be sorted as in-place partial sorting or discarded, which is common in streaming partial sorts [76].

Despite many recent works in hardware-assisted sorting, no recent survey reviews the latest developments in this area. Studying the literature, we found three surveys discussing prior hardware-based sorting designs. These are compared in Table I. Jmaa et al. [36] compare the performance of the hardware implementations of popular sorting algorithms (i.e., Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Heap Sort, Shell Sort, Merge Sort, and Tim Sort) in terms of execution time, standard deviation, and resource utilization. They synthesized the designs on a Zynq-7000 FPGA platform. Skliarova [37] reviewed different implementation approaches for network-based hardware accelerators for *sorting*, *searching*, and *counting* tasks. Ali [38] looked closely at comparison-based and comparisonfree hardware solutions for sorting. As in-memory and partial sorting are relatively emerging topics, these previous surveys do not cover them. Motivated by this, this work reviews the latest hardware solutions for complete, partial and in-memory sorting, covering both comparison-based and comparison-free approaches. Table II summarizes and classifies the important works we study in this article.

The remainder of this paper is organized as follows. Section II reviews complete sorting solutions. Section III reviews hardware solutions for partial sorting. Section IV discusses recent works on emerging in-memory sorting. Section V discusses open challenges and future works. Finally, Section VI concludes the paper.

II. COMPLETE SORTING METHODS

We begin by reviewing recent works on complete sorting, which processes all the data to sort them in an ascending or descending order. We divide our discussion into two categories of *comparison-based* and *comparison-free* sorting.

TABLE II
PRIOR ART FOR HARDWARE SOLUTIONS OF SORTING

Category	Reference	Year	Idea	Design
Comparison Based	Farmahini et al. [40]	2013	Sorting using hierarchical smaller blocks	•
	Lin et al. [46]	2017	Pointer-like iterative architecture	•
	Najafi et al. [41]	2018	Bit-stream-based and time-encoded unary design	•
	Norollah et al. [47]	2019	Consecutive normal and reverse sorting	*
	Jelodari et al. [48]	2020	Vertex indexing in graph representation of inputs	*
	Papaphilippou et al [49], [50]	2020	Recursive parallel merge tree	*
	Preethi et al. [51]	2021	Clock gating techniques to improve power consumption	•
	Prince et al. [52]	2023	Sorting weighted stochastic bit-streams	•
	Abdel-Hafeez et al. [44]	2017	One-hot weight representation	•
	Bhargav et al. [53]	2019	FSM module for sorting	*
Comparison	Chen et al. [54]	2021	Bidirectional architecture improving sorting cycle [53]	♠ / ↔
Free	Sri <i>et al.</i> [55]	2022	Improving the boundary-finding module of [54]	+
	Ray et al. [45]	2022	k-Degree Parallel Comparison-free Hardware Sorter	*
	Jalilvand et al. [56]	2022	Comparison-free sorting architecture based on unary computing	•
	Yu et al. [57]	2011	Spike sorting hardware	+
	Campobello et al. [58]	2012	Maximum - minimum finder	*
	Subramaniam et al. [59]	2017	Median finder	+
Partial	Korat <i>et al.</i> [60]	2017	Odd-even merge sorting	+
Sorting	Valencia et al. [61]	2019	Minimum distance calculation for spike sorting	+
	Zhang et al. [62]	2021	Min-max sorting architecture	*
	Yan et al. [63]	2021	Determining a certain k largest or smallest numbers	+
In- Memory	Wu et al. [64]	2015	Data sorting in flash memory (NAND flash-based)	*
	Samardzic et al. [65]	2020	Bonsai Sorting on CPU-FPGA focused on DRAM-scale sorting	÷/*
	Li et al. [66]	2020	Parallel sorting via hybrid memory cubes	A
	Prasad et al. [67]	2021	Memristor-based data ranking and min/max computation	+
	Chu et al. [68]	2021	Detecting partially ordered for cost reduction	٥
	Riahi Alam et al. [69]	2022	High-performance and energy-efficient data sorting	*
	Yu et al. [70]	2022	Column-skipping algorithm for memristive memory	*
	Zokaee et al. [71]	2022	Sorting large datasets based on sample sort	+
	Lenjani <i>et al.</i> [72], [73]	2022	Optimizing external sorting for NVM-DRAM hybrid storage	+
	Liu <i>et al.</i> [74]	2023	Minimizing NVM write operations	*

♣ → ASIC, ❖ → FPGA, ☀ → DRAM, ◆ → In-Memory, ♣ → Memristor-based, ★ → Adaptive Memristor, ☆ → In-Logic-Layer Based, ♣ → In-Logic-Layer Based, ♣ → In-Memory Friendly

A. Comparison-based

Farmahini *et al.* [40] proposed a comparison-based design that employs efficient techniques for constructing high-throughput, low-latency sorting units using smaller building blocks in a hierarchical manner. Their design includes *N*-to-*M sorting* and *max-set-selection* units. They extensively discuss the structure, performance, and resource requirements of these units. Despite its primary focus on integer numbers, their design efficiently accommodates two's complement and floating-point numbers, as the comparators utilized in their compare-and-exchange (CAE) blocks can be substituted accordingly.

Some sorting applications do not need to sort all input data. Instead, the application may only require the identification of the M largest or M smallest values from a set of N inputs. These algorithms are called *partial sorters* and will be discussed later in this survey. In an N-to-M max-set-selection unit used in the sorting designs of [40], only the M largest inputs are required in no specific order.

Lin et al. [46] proposed a hardware acceleration architecture for real-time sorting of M out of N inputs. Their design benefits from moving indexes instead of data and is called a pointer-like design. They reduce power consumption by reducing switching activities and signal transitions while maintaining high throughput. Their sorting approach has a complexity of $O(\log_2^2 M)$. The primary contributor to power consumption is the switching activities of registers. To effectively reduce power, they recommend modification to the

register transfer level (RTL) design. Notably, signal transitions increase when the input dataset is larger or when the bit width of the input sample is significant. They propose to incorporate additional registers to represent the position of each input sample. So, only the indexes need to be migrated from register to register. When N inputs are present, the complete index can be represented using only $\log_2 N$ bits, irrespective of whether the bit-widths are 8-bit, 16-bit, or more. While modifications may increase the total cell area, they achieve a substantial reduction in dynamic power dissipation.

Executing the sorting process using a single module is impractical for large input datasets, as it requires high I/O bandwidth and large cell area. To mitigate this issue, Lin *et al.* [46] proposed to reuse smaller sorting units as the core module and combine these small units with other control units to implement an iterative architecture. Fig. 3 shows their proposed architecture. Users have the flexibility to select different sorting units as the core module, enabling them to trade off throughput for resource constraints.

Najafi *et al.* [41], [77] developed an area- and power-efficient hardware design for complete sorting based on *unary* computing (UC). They convert the data from binary to unary bit-streams to sort them in the unary domain. Their approach replaces the conventional complex design of the *CAS* unit implemented based on binary radix with a simple unary-based design made of simple standard AND and OR gates. Fig. 4 demonstrates how a *CAS* block is implemented in the unary domain. When two unary bit-streams of equal length

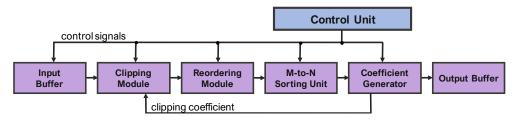


Fig. 3. Lin et al. [46] iterative sorting system. An iterative architecture is designed by repeatedly employing a smaller sorting unit to process streaming input data. Within this iterative max-set-selection or partial sorting system, the input remains constant, contingent on the type of sorting unit in use. Users have the flexibility to select different sorting units as the Core Module, allowing them to strike a balance between throughput and resource constraints. Importantly, this iterative architecture imposes no limitations on the volume of input samples it can handle. As the input size scales, resource consumption remains constant, effectively mitigating resource overhead challenges. In addition to the application of the low-power sorting module, the design also incorporates an adaptive clipping mechanism and a reordering module. These elements are instrumental in further reducing the switching activities of registers. The adaptive clipping coefficient increases in pair with the temporal results during the sorting process, serving to block a substantial number of samples.

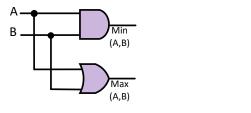
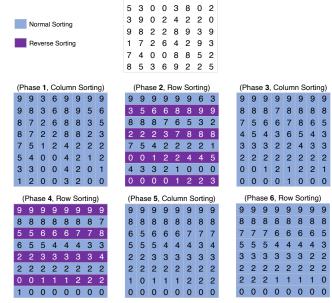


Fig. 4. The hardware implementation of a Unary CAS block [41].

are connected to the inputs, an AND gate yields the minimum value, whereas an OR gate produces the maximum value. An overhead of this unary design is the cost of converting data from binary to unary representation. However, compared to the cost savings in the computation circuits, this conversion overhead is insignificant. They report an area and power saving of more than 90% for implementing a 256-input complete sorting network. The unary design of [41] consists of simple logic gates independent of data size. The computation accuracy is controlled by the length of bit-streams. The longer the bit-stream, the higher the accuracy. But processing long unary bit-streams can result in long latency with the sorting design of [41]. This causes runtime overhead compared to the conventional binary process. While the latency may be tolerated in many applications, they introduce a time-based unary design to mitigate the latency issue. They encode the input data to pulse-width modulated signals. The data value is determined by the duty cycle in this approach. At the cost of slight accuracy loss, the time-based approach significantly reduces the latency.

Prince et al. [52] combined the bit-stream capabilities of stochastic computing (SC) with binary weighting, reducing latency of bit-stream-based sorting. The approach offers good scalability and cost-efficiency compared to SC and traditional binary methods, making it an efficient solution for sorting tasks. They use a weighted bit-stream converter to generate weighted bit-streams for an adaptable sorting network. Unlike conventional SC bit-streams, each bit in the weighted bit-streams retains its weight as a standard binary value. This conversion reduces the number of bits in SC from 2^N to N for N-bit precision, resulting in a substantial reduction in latency and energy consumption by shifting from exponential to linear representation. They propose a new lock-and-swap (LAS) unit to sort weighted bit-streams. Their LAS-based sorting network can determine the result of comparing different input values



64 Input Records
9 7 1 0 8 2 1 6
8 2 3 6 4 9 0 1

Fig. 5. The MDSA with 64 input records, forming an 8×8 matrix [47].

early and then map the inputs to the corresponding outputs based on shorter weighted bit-streams.

Norollah *et al.* [47] presented a novel multidimensional sorting algorithm (MDSA) and its corresponding architecture, a real-time hardware sorter (RTHS), to efficiently sort large sequences of records. MDSA reduces the required resources, enhances memory efficiency, and has a minimal negative impact on execution time, even when the number of input records increases. To sort large sequences of records, MDSA divides a sequence into smaller segments, which are then sorted separately. As shown in Fig. 5, the MDSA algorithm consists of six consecutive phases and two modes: normal and reverse sorting. The sorting network organizes the records in descending and ascending order for normal and reverse modes, respectively. In each phase, the sorting networks are fed by a group of input records to sort independently.

The authors in [47] claim that their sorting method is more beneficial for resource conservation (memory efficiency) while providing high performance. Fig. 6 shows the complete archi-

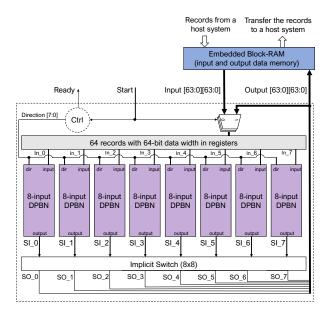


Fig. 6. Real-time hardware sorter (RTHS) architecture for 8×8 matrix records [47].

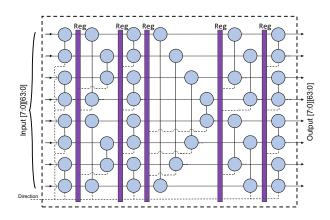


Fig. 7. Dual-mode pipeline bitonic network (DPBN) unit for 8 inputs [47]. The direction signal indicates the mode for sorting: normal or reverse.

tecture of RTHS. In this design, pipelining is used to reduce the critical path in dual-mode pipeline bitonic networks (DPBNs). Fig. 7 shows a DPBN unit for 8 inputs. The number of pipeline stages in a DPBN is directly proportional to its number of steps, which can be computed by $(1/2\log_2(N)(\log_2(N)+1))$, where N is the number of inputs. The implicit switch is done by fixed wiring and so is completely static. This hardwired switching does not require additional routing resources and has minimal overhead.

Jelodari *et al.* [48] proposed a low-complexity sorting network design, which maps the unsorted input data to a graph. In this graph, the vertices represent inputs and are fully connected through directed edges as shown in Fig. 8. This structure allows comparing all inputs with each other through the directed edges connecting their corresponding vertices. At each end of any graph edge, the corresponding vertex is tagged by 0 or 1. The tags of the vertices connected by an edge are always complementary. The outgoing tag "1" means the source vertex is greater than or equal to the sink vertex. The sum of

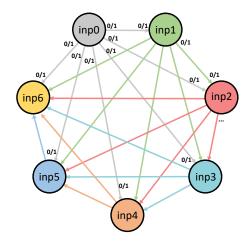


Fig. 8. The graph representation of [48]. Each input, represented by a vertex, is linked to all other vertices through directed edges, indicating a directed, fully connected graph.

the tags assigned to each vertex indicates the position of the corresponding input data in the sorted output.

Papaphilippou et al. [49], [50] introduced a merge sorter tailored for small lists, with the capability to merge sublists recursively. This feature sets their solution apart from most large-scale sorters, often reliant on pre-sorted sublists or established hardware sorter modules. Their design bridges the gap between high-throughput and many-leaf sorters by merge sorters, allowing customization of bandwidth, data, and payload width. They assess the applicability of their solution in their specialized in-house context, specifically for database analytics. This involved calculating the count of distinct values per key (group) from a dataset comprising key-value pairs. They integrated a fully-pipelined high-throughput stream processor seamlessly with the sorter's output, enabling real-time result generation. Their streamlined process eliminates the need for temporary data storage, exemplifying task-pipelining for efficient data processing. Fig. 9 shows their setup. They incorporate a fast lightweight merge sorter (FLiMS) as a key component within their parallel merge/sort tree. The FLiMS unit combines two separate -already sorted- lists. The design is characterized by w linear sorters, where w signifies the degree of parallelism being employed. Each individual linear sorter has a length of k/w, with k representing the total merge capacity or the size of the sorted chunk. This architectural arrangement sorts an input dataset comprising "k" elements while adeptly merging already sorted lists of varying lengths.

Preethi *et al.* [51] investigated the use of clock gating technique to design low-power sorters. The bubble sort, bitonic sort, and odd-even sorting algorithms are redesigned to make them low-power using the clock gating technique. The implementation results showed that clock gating can reduce the dynamic power consumption of sorters by 47.5% with no significant impact on the performance.

B. Comparison-free

Comparison-free sorting designs do not involve direct element comparisons. Instead, they employ alternative approaches

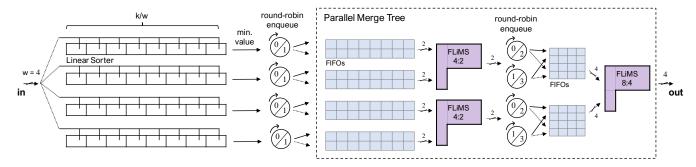


Fig. 9. The high-throughput sorting system of [49] sorts data quickly while merging them efficiently at a rate of 4. The system uses a specific design where wire widths are chosen based on multiples of the width of the data values. This structure possesses the capability to perform both the sorting of an input containing "k" elements and the merging of "k" sorted lists of variable lengths. In sorting mode, a 2-bit counter value is appended to the most significant bits of all outputs from the linear sorters. This 2-bit counter is incremented whenever a new sorted chunk is flushed to the "Parallel Merge Tree." This plays a vital role in the FLiMS (fast lightweight merge sorter) system [50], ensuring correct sorting prioritization for independently sorted chunks.

to accomplish efficient sorting. In recent years, there has been a notable surge in research and development of this type of sorting. In this section, we will provide an overview of these advancements.

Abdel-Hafeez and Gordon [44] proposed a comparison-free sorting algorithm for large datasets. The method operates on the elements' one-hot weight representation, a unique count weight associated with each of the N elements. The input elements are inserted into a binary matrix of size $N \times 1$, where each element is k bits. Concurrently, the input elements are converted to a one-hot weight representation and stored in a one-hot matrix of size $N \times H$. In this matrix, each stored element is of size H-bit and H = N gives a one-hot matrix of size N-bit \times N-bit. The one-hot matrix is transposed to a transpose matrix of size $N \times N$, which is multiplied by the binary matrix to produce a sorted matrix. An example of this method is illustrated in Fig. 10. The total number of sorting cycles is linearly proportional to the number of input data elements N. The architecture of [44] is a high-performance and low-area design for hardware implementation.

Bhargav and Prabhu [53] later proposed an algorithm for comparison-free sorting using finite-state machines (FSMs). Their FSM consists of six states that describe the functionality of a comparison-free sorting algorithm dealing with N inputs. Their proposed design shows 53% and 68% savings in area and power consumption compared to the design of [44].

Chen at al. [54] improve the number of sorting cycles, which range from [2N to 2N+2K-1] to [1.5N to $2N+(\frac{2^k}{2})-2]$. Their proposed architecture improves the performance of the unidirectional architecture in [44] by reducing the total number of sorting cycles via bidirectional sorting along with two auxiliary modules. One of the auxiliary modules is *boundary finding*, which is designed to record the maximum and minimum values of the input data for the high-index part (max H and min H) and the low-index part (max L and min L). As shown in Fig. 11, the boundary values are stored in four K-bit registers where K is the bit-width of input data. In the initial state of the circuit, the values of max H, min H, max L, and min L are set to 2K/2, 2K-1, 0, and (2K/2)-1, respectively. A *binary finding* module shortens the range for index searching by finding the boundaries of

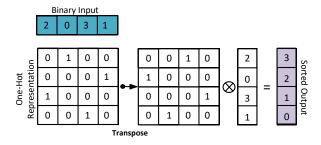


Fig. 10. Example of sorting four input data with the method of [44].

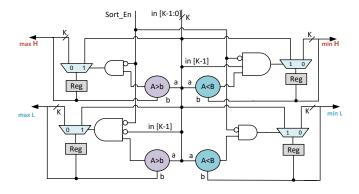


Fig. 11. Architecture of the boundary founding module used in [54].

the range. Bidirectional sorting allows the sorting tasks to be conducted concurrently in the high- and low-index parts of the architecture. Sri *et al.* [55] reduce the area, delay, and power consumption of the design of [54] by improving the boundary finding module. The improvements are achieved by removing the AND gates and MUX components.

Ray and Ghosh [45] developed an architecture for parallel comparison-free sorting based on a model presented earlier in [78]. This work sorts N data elements completely by utilizing N iterations with speed-up of $\frac{n}{\lceil \frac{n}{k} \rceil + k}$ compared to non-parallel architectures.

Jalilvand *et al.* [56] proposed a fast and low-cost comparison-free sorting architecture based on UC. Similar to [79], [80], their method iteratively finds the index of the

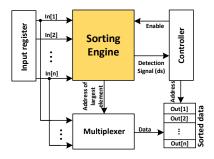


Fig. 12. High-level architecture of the comparison-free unary sorter in [56].

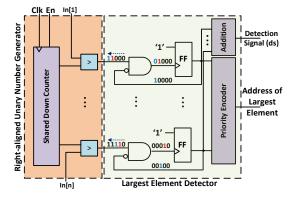


Fig. 13. The Unary Sorting Engine proposed in [56].

maximum value by converting data to left-aligned unary bitstreams and finding the first "1" in the generated bit-streams. Fig. 12 shows the high-level architecture. The architecture includes a sorting engine, a controller, and a multiplexer. The design reads unsorted data from the input registers and performs sorting by finding the address of the maximum number at each step. Fig. 13 shows the architecture of the sorting engine. The sorting engine contains simple logic and converts data to right-alighted unary bit-streams. It returns the index of the bit-stream corresponding to the maximum value. This is done by finding the bit-stream that produces the first "1". The design also employs a controller that gets a duplication sign signal from the sorting engine and puts the next value to the output sorted register.

Finally, Yoon [81] proposed a sorting engine based on the radix-2 sorting algorithm. Their sorting engine avoids comparison by creating and distributing data into buckets according to the radix-2 sorting.

III. PARTIAL SORTING

Partial sorting is primarily used to sort the top-k largest or smallest values out of N elements, where k < N. Partial sortings have been used for determining the minimum and maximum values, finding more than one relative maximum and minimum (max-set min-set selection), merging of partially sorted data, and approximate partial sorting [40], [75], [82], [83]. Finding the minimum and maximum values among a set of data has been particularly an important target of partial sorting. FPGA has been a popular platform for implementing this type of sorting in hardware.

Yan et al. [63] proposed an architecture for determining the k largest or smallest numbers on FPGA. Their work allows selecting two min/max subsets with a real-time hardware partial sorter (RTHPS) structure consisting of even-odd swap blocks, a bitonic sorting network, and parallel swap blocks. Korat et al. [60] proposed a sorting algorithm that partially sorts the odd and even parts in a vector structure. Their method guarantees a linear time complexity with O(n). The hardware unit includes two multiplexers and a comparator, which is responsible for ordering input pairs. Their FPGA-based hardware design implemented on a Xilinx VIRTEX-7 VC707 FPGA consumes 136 LUTs and 181 registers with a working frequency of 370 MHz when sorting eight inputs.

Median sorting is another practice of partial sorting with wide application in image processing, particularly for image enhancement. Various hardware designs for median filtering have been proposed in the literature. Subramaniam et al. [59] proposed a hardware design for finding the median value of a set of data. They employ selective comparators as a means to locate the median, allowing for partial sorting with fewer elements compared to the conventional designs that necessitate a fully sorted list. CAS operations are obtained using a comparator and two 2-to-1 MUXs. They implement the design on an FPGA (Xilinx FPGA Virtex 4 XC4VSX25) and evaluate it using an image processing case study [59]. Using a pipelined architecture, Cadenas et al. [86], [87] proposed a median filtering architecture using accumulative parallel counters. Najafi et al. [41] further implemented a low-cost median filtering design based on UC by converting data to unary bit-streams and processing them in the unary domain using simple standard AND and OR gates. Finally, Riahi Alam et al. [69] proposed a binary and a unary architecture for energy-efficient median filtering completely in memory.

Finding the maximum and minimum values is one of the current topics in in-memory computing applications. Zhang *et al.* [62] proposed an in-memory min-max sorting architecture in DRAM technology for fast and big data applications (see Fig. 14). Sorting and graph processing applications are provided with an architecture that produces results 50 times faster than a GPU. This architecture includes two-row decoders, a one-column decoder, a modified logic sense amplifier with a typical sense amplifier (TSA), one latch per bit-line, a pseudo-OR gate, and one priority encoder (for the resultant index of minimum and maximum locations).

Campobello *et al.* [58] discuss sorting networks' complexity and propose a multi-input maximum finder circuit. Their design finds the maximum value by using an XNOR comparator, a zero catcher (via Q-port feedbacked D flip-flip), a buffer with enable for each input, an OR gate, and a D-flip-flop.

Partial sorting can also be used as an intermediary tool to help understand data, *e.g.*, to find outliers [88]. This includes the complex task of *spike sorting* in brain-inspired computing. Spike sorting encompasses algorithms designed to identify individual spikes from extracellular neural recordings and classify them based on their shapes, attributing these detected spikes to their respective originating neurons. This sorting process differs from conventional sorting as it involves machine learning-related steps such as detection, feature extraction, and

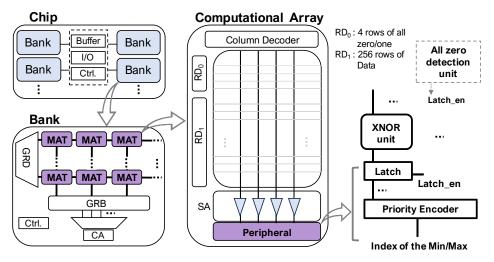


Fig. 14. The MIN-MAX PIM architecture proposed by Zhang *et al.* [62] preserves the original memory hierarchy with each DRAM chip divided into multiple banks. These partitions share the Input-Output and buffers. Each bank comprises multiple memory matrices (MATs), which are essentially DRAM subarrays. The design supports Ambit [84], [85] logic and instructions with enhanced support for the Dual Row Activation (DRA) mechanism, thus providing compatibility with XNOR operations. The Computational Array includes (i) two-row decoders, (ii) one column decoder, (iii) modified logic sense-amplifier, (iv) one latch per bit-line, (v) pseudo-OR gate, and (vi) one priority encoder. The circuit for zero detection uses a pseudo OR gate. This gate is employed to govern the update of the matching vector latch. The priority encoder returns the index of the minimum or maximum value for partial sorting purposes.

classification. Instead of straightforward scalar sorting, spike sorting resembles the segmentation of patterns within brain signal pulses [89]. Spike sorting involves partial sorting for tasks such as early learning termination, outlier analysis, and spike activity thresholding. In the literature, spike sorting for a unit activity may encompass partial sorting to separate multiunit activity into distinct groups of single-unit activity [90]. The segmentation of spike data plays a significant role in distinguishing specific activities within the overall spike data. Within the realm of spike processing, some studies underscore partial sorting for outlier analysis of the spikes [88], while others commend it for thresholding operations [91]. Valencia and Alimohammad [61], [92] implement a hardware module for spike sorting architecture. Their design incorporates a template matching unit to compute the minimum distance between spikes during the spike sorting process. Fig. 15 depicts the spike sorting design, which relies on template checks and minimum distance calculations. Such advancement in hardware-powered sorting is expected to open new research avenues in emerging machine-learning models, particularly brain-inspired computing.

IV. IN-MEMORY SORTING

In traditional processors, data are retrieved from disk storage and loaded into memory for processing. In this conventional approach, a significant portion of the total processing time and energy consumption is wasted for transferring data between the memory and processing unit. Most prior sorting designs are implemented based on this Von-Neumann architecture with separate memory and processing units [93]. In-memory computation (IMC) –aka processing-in-memory (PIM)– is a promising solution to address this data movement bottleneck. In this processing approach, the chip memory is used for both storage and computation [94]. To address the data movement issue and improve sorting speed, *in-memory sorting* [67],

[69], [70], [95] has been proposed. In particular, the special properties of non-volatile memories (NVMs) make them a promising candidate for efficient sorting in memory.

Chu et al. [68] proposed an NVM-friendly sorting algorithm called "NVMSorting". NVMSorting is a modification of the MONTRES algorithm [96], a sorting algorithm resembling merge sort, designed for flash memory. MONTRES aims to enhance performance by minimizing I/O operations and reducing the generation of temporary data during sorting. It includes a run generation phase and a run merge phase, employing optimized block selection, continuous run expansion, and onthe-fly merging for efficient data organization. NVMSorting has the ability to detect partially ordered runs by using a new concept, called *natural run*, to reduce the sorting cost. A natural run consists of multiple blocks. The items within each block are not required to be sorted, but the items between any two consecutive blocks are ordered. In the first step, the algorithm searches for the partially ordered runs (i.e., natural runs) in the input data. The next step is the run generation, which is based on a merge-on-the-fly mechanism and a run expansion mechanism. DRAM is divided into two sections: I) workspace for the natural runs, and II) workspace for the other input data. Chu et al. take advantage of the NVM's byteaddressable capability to merge the runs. Their evaluations show that NVMSorting is more efficient than the traditional merge sorting algorithms in terms of execution time (t) and number of NVM writes (w). However, if the dataset is entirely random, NVMSorting can achieve similar performance to MONTRES, hybrid sort, and external sort [97].

Li *et al.* [66] proposed a PIM architecture called IMC-Sort to perform parallel sort operations using a hybrid memory cube (HMC). As shown in Fig. 16, IMC-Sort is comprised of sorting units that are specifically designed to operate within each HMC vault's logic layer. The control unit of the HMC vault is enhanced with some logic to carry out the sorting process.

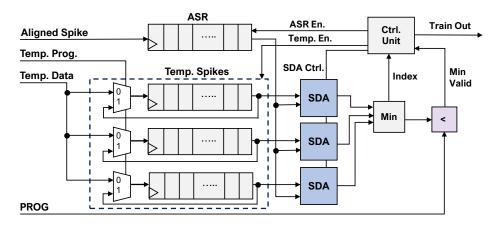


Fig. 15. Template matching-based architecture for spike sorting. The aligned spike is directed into an ASR (Aligned Shift Register) module, which has been set up for parallel input and serial output. The values stored in the templates and the ASR module are then transferred into some SDA (Squared Difference Accumulator) units. These SDA units are used to calculate and accumulate the squared differences between the spike waveform preserved in the ASR and templates. The MIN unit identifies and conveys the minimum value to the comparator, along with the index of the minimum value, which is then passed on to the Control Unit. The substantial reduction of raw data to sorted spikes is achieved by transmitting only those sorted spikes (in a partial sorting manner) that match a small set of frequently encountered waveforms [61], [92].

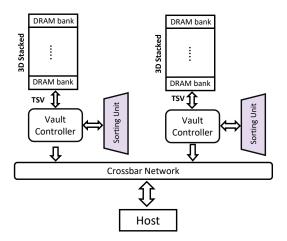


Fig. 16. Overall architecture of the IMC-Sort. A single stack HMC vault is composed of several DRAM banks that are linked to the logic layer via through-silicon vias (TSV) [66].

The sorting units in IMC-Sort are capable of parallel access and utilize the HMC crossbar network to communicate with one another. In an "Intra-vault merging" step, they utilize a chunking technique to accommodate a range of input sequence lengths using a fixed number of CAS units and a fixed input permutation unit. They divide the sequence into chunks of a specific size determined by the number of CAS units. Then, they sort the chunks. Finally, the sorted values are merged into a single sorted sequence. On the other hand, an "Intervault merging" step combines the sorted values or sequences from all vaults to produce a globally sorted sequence. IMC-Sort delivers $16.8 \times$ and $1.1 \times$ speedup and $375.5 \times$ and $13.6 \times$ reduction in energy consumption compared to the widely used CPU implementation and a state-of-the-art near memory custom sort accelerator, respectively [98], [99], [65], [100], [101].

Riahi Alam et al. [69] proposed the first in-array (in-memory) architectures for high-performance and energy-

efficient data sorting completely in memory using memristive devices. They introduce two different architectures. The first architecture, "Binary Sorting," is based on the conventional weighted binary representation, while the second architecture, "Unary Sorting," is based on the non-weighted unary representation. Both of these sorting designs achieve a significant reduction in the processing time compared to prior off-memory binary and unary sorting designs. The memristor technology they used is based on the stateful logic in which the input and output are both presented as the state of input and output memristors. In stateful logic, values are stored and maintained within memristive switches through their resistance states. These switches not only store logic values but also perform logical operations, exhibiting both memory and computational capabilities [102]–[104]. They implement the boolean operations with memristor-aided logic (MAGIC) [103] in a crossbar implementation. Each MAGIC logic gate utilizes memristors as inputs, which contain previously stored data, and additional memristor functions as the output. Parallel architectures such as CAS-based sorting networks can be executed efficiently within the memory using these IMC logic operations [69].

In the first design, the memory is split into multiple partitions to enable parallel execution of different CAS operations of each bitonic CAS stage. The number of partitions indicates the number of CAS units that can run in parallel. The first two inputs of each partition are sorted using a basic sorting operation. Then the maximum value of each basic sort operation is copied to another partition determined by the sorting network. The second design is a complete unary sort system that follows the same approach as the binary implementation but represents and processes the data in the *unary* domain with uniform unary bit-streams [105]. The comparison operations are implemented in this design based on a basic unary sorting unit. Their performance evaluation results show a significant latency and energy consumption reduction compared to the conventional off-memory designs. On average, their inmemory binary sorting resulted in a 14x reduction in latency and a 37× reduction in energy consumption. On the other hand, the average latency and energy reductions for the inmemory unary sorting design were much greater, at $1200\times$ and $138\times$, respectively. Further, they implemented two in-memory binary and unary designs for Median filtering based on their developed in-memory basic sorting units. Their results showed an energy reduction of $14\times$ (binary) and $5.6\times$ (unary) for a 3×3 -based image processing system, and $3.1\times$ and $12\times$ energy reduction for binary and unary median filtering, respectively, for a 5×5 -based image processing system compared to their corresponding off-memory designs.

Today's systems often face memory bandwidth constraints that can limit their performance. The efficiency of the sorting algorithms can be significantly impacted by the available memory bandwidth. To overcome the bandwidth problem in large-scale sorting applications, Prasad et al. [67] proposed an iterative in-memory min/max computation technique. They applied a novel mechanism called "RIME", which enhances bandwidth efficiency by enabling extensive in-situ bit-wise comparisons. RIME eliminates unnecessary data movement on the memory interface, resulting in improved performance. They provide an API library with significant control over essential in-situ operations like ranking, sorting, and merging. With RIME, users can efficiently manage and manipulate data. To perform bit-serial min/max operation, they execute an iterative search for bit value (1 or 0) within individual columns of a data array using a 1T1R memristive memory. In each iteration of the search, a match vector is generated to identify which rows in the array should be eliminated from the dataset. The memory array must be capable of performing two additional operations, namely bitwise column search and selective row exclusion.

The algorithm starts by examining the binary values of all bit positions, beginning from the most significant bit position in a set of numbers. This process is carried out using a kstep algorithm, during which some of the non-minimum or non-maximum values may be removed from the set at each step. At each step, a selection of matching numbers is formed by searching for "1" at the current bit position. The selected numbers are removed from the set only if the set and selection are unequal. This results in all the final remaining numbers in the set having the minimum value. By eliminating the unnecessary data movement for finding min/max of given data, their sorting operation obtains a bandwidth complexity of O(N). With the suggested in-memory min/max locator, the costs of accessing bandwidth when searching for the k^{th} value in a range of data decrease to k operations, which shows a bandwidth complexity of O(k). Their simulation results on a group of advanced parallel sorting algorithms demonstrate a significant increase in throughput ranging from $12.4\times$ to $50.7 \times$ when using RIME.

Yu et al. [70] improve the speed and performance of Parasad et al.'s design by proposing a column-skipping algorithm that keeps track of the column read conditions and skips those that are leading 0's or have been processed previously (see Fig. 17). A bank manager enables column-skipping for datasets stored in different banks of the memristive memory. For detecting and skipping redundant column reads the algorithm records

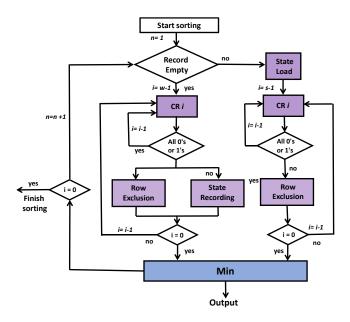


Fig. 17. Iterative min search with proposed column-skipping algorithm [70].

the k most recent row exclusion states and their corresponding column indexes, which can be reloaded to avoid repeating these states.

To tackle the sorting challenges of large-scale datasets, Zokaee et al. [71] proposed Sky-Sorter, a cutting-edge sorting accelerator powered by Skyrmion Racetrack Memory (SRM). Sky-Sorter leverages the unique capabilities of SRM, enabling the storage of 128 bits of data within a single racetrack. Sky-Sorter adopts the sample sort algorithm, which encompasses a sequence of four essential steps: sampling, splitting marker sorting, partitioning, and bucket sorting. First, it employs a random sampling technique to estimate the distribution of the dataset. This sampled subset is then sorted, and specific records are selected as splitting markers. The markers are crucial for defining the boundaries of nonoverlapping buckets. The next step involves partitioning, where all records, excluding the splitting markers, are allocated to appropriate buckets based on their relationship to the markers. Lastly, each bucket is sorted individually, and the results are concatenated to produce the final sorted sequence. Bucket sorting, known for its high parallelizability, is the key to this algorithm's efficiency, with the distribution of bucket sizes playing a crucial role in maintaining balance. To achieve balanced distribution and prevent load imbalances during bucket sorting, it is essential to distribute records evenly across all buckets. Larger random sampling sizes contribute to more accurate estimates of the data distribution and less variability in bucket sizes. The algorithm ensures that the probability of any bucket exceeding an upper size limit is nearly zero. In rare cases where a bucket size surpasses this threshold, the algorithm triggers the resampling of splitting markers to maintain uniformity in bucket sizes. The fundamental cell structure of SRM is composed of four integral parts. These components encompass two injectors devoted to the creation of skyrmions, a detector designed for the precise detection of skyrmions, a nanotrack to facilitate the controlled motion of these skyrmions and peripheral circuits that support and coordinate the functionality of the entire cell. The authors claim that Sky-Sorter improves the throughput per Watt \sim 4× over prior FPGA-, Processing Near Memory (PNM)-, and PIM-based accelerators when sorting with a high bandwidth memory DRAM, a DDR4 DRAM, and an SSD [65]–[67].

To address the challenges of sorting vast datasets with limited memory resources, significant efforts have been dedicated to enhancing external sorting algorithms. While efforts have been made to enhance external sorting algorithms, few have considered the I/O requests and byte-addressable characteristics of NVM. Liu et al. [74] proposed LazySort, an external sorting algorithm tailored to the NVM-DRAM hybrid storage architecture. LazySort leverages NVM's byte-addressable feature and locally ordered data to minimize write operations to NVM. It comprises two stages: run generation and merge. To enhance efficiency, they introduce an optimization strategy known as RunMerge for the merge stage. RunMerge intelligently merges non-intersecting data blocks based on the range of an index table records, reducing the total number of runs and memory usage. To validate the performance, they established a real NVM-DRAM experimental platform and conducted comprehensive experiments. The results showed LazySort's superior time performance and significantly reduced NVM write operations. Compared to traditional external sorting algorithms, LazySort reduced sorting time by 93.08% and minimized NVM write operations by 49.50%. This design then addresses an important need for efficient external sorting methods for NVM-DRAM hybrid storage.

Lenjani et al. [72] proposed Pulley, an algorithm/hardware co-optimization technique for in-memory sorting. Pulley uses 3D-stacked memories. They employ Fulcrum [73] for the baseline PIM architecture. Fulcrum inputs data into a singleword arithmetic logic unit (ALU) in a sequential manner and enables operations that involve data dependencies as well as operations based on a predicate. In Fulcrum, every pair of subarrays has three row-wide buffers called Walkers. In the radix sorting proposed in Fulcrum, all buckets have the same length, and a bucket in each pass can always fit in one subarray. For efficient sorting of large data using Fulcrum, Lenjani et al. modified the design by calculating the exact length of each bucket and the position of each key within that bucket. In the first step, the keys of each processing unit are sorted locally. In this step, the keys are dichotomized into two buckets (Bucket0 and Bucket1). The subarray-level processing unit (SPU) starts Bucket0 from the bottom of the space and fills it upward, and starts Bucket1 from the end of the space and fills it downward. In the next step, each SPU generates the histogram values of the first 256 buckets iteratively, and all SPUs reduce the histogram values of each of the 256 buckets in the lowest subarray. In Pulley, each vault's core in the logic layer performs a prefix-sum on all the shared sub-arrays in the vaults. Then, the cores in the vaults aggregate their prefix-sum arrays. They evaluate Pulley in 1-device and 6-device settings, where each device has four stacks of 8-GB memories. Compared to IMC-Sort, Pulley has a lower working frequency.

Wu and Huang [64] introduced a novel sorting technique specifically tailored to NAND flash-based storage systems,

aiming to optimize performance and efficiency. They propose a record rearrangement and replacement method for unclustered sorting, which involves scanning sorted tags to efficiently rearrange records and minimize unnecessary page reads during the process. They introduce a strategic decision rule to harness the advantages of both clustered and unclustered sorting approaches. This rule categorizes records based on their length and then selects the most appropriate sorting method (clustered or unclustered) for each category, followed by merging the sorted results. They reuse data to reduce page writes by detecting content similarities in the output buffer and marking logical addresses in the address translation table for potential reuse. They provide a comprehensive I/O analysis, comparing the performance of clustered sorting, unclustered sorting, MinSort, and FAST in terms of page reads and writes. Finally, they implement and test the proposed methods on real hardware, including an Intel SSD and a Hitachi HDD, demonstrating significant performance improvements compared to traditional external sorting methods.

Samardzic et al. [65] introduced "Bonsai," an adaptive sorting solution that leverages merge tree architecture to optimize sorting performance across a wide range of data sizes, from megabytes to terabytes, on a single CPU-FPGA server node. Bonsai's adaptability is achieved by considering various factors, including computational resources, memory sizes, memory bandwidths, and record width. It employs analytical performance and resource models to configure the merge tree architecture to match the available hardware and problem sizes. Their approach can enhance sorting efficiency on a single FPGA while also being used as a foundation for potential use in larger distributed sorting systems. Bonsai's primary objective is to minimize sorting time by selecting the optimal adaptive merge tree configuration based on the hardware, merger architecture, and input size. They demonstrate the feasibility of implementing merge trees on FPGAs, highlighting their superior performance across various problem sizes, particularly for DRAM-scale sorting. Bonsai achieves significant speedup over CPU, FPGA, and GPU-based sorting implementations, along with impressive bandwidth efficiency improvements, making it an appealing solution for adaptive sorting.

V. OPEN CHALLENGES

Although significant strides have been made in the field of hardware sorting, numerous challenges persist, warranting further research and innovation. In this section, we explore the ongoing challenges within the research on hardware-assisted sorting. Addressing these challenges can result in sorting solutions that are more efficient in different aspects, from performance to footprint area, power, and energy consumption. These challenges are elaborated on in the following sections.

A. Algorithmic Considerations

With recent research opportunities and emerging sorting solutions such as in-memory and partial sorting, future research needs to explore potential avenues for radically novel sorting

TABLE III VARIOUS SORTING NETWORK CONFIGURATIONS AND THEIR PRINCIPAL CHARACTERISTICS BY ZULUAGA $\it et~al.~[106]$

Architecture	Logic	Storage	Storage Type	Streaming Width	Fully Streaming
Bitonic and odd-even sorting network (Batcher, 1968, [107])	$O(n \log^2 n)$	$O(n \log^2 n)$	Flip-flop	n	Yes
Folded bitonic sorting network (Stone, 1971, [108])	O(n)	O(n)	Flip-flop	n	No
Odd-even transposition sorting network (Knuth, 1968, [109])	$O(n^2)$	$O(n^2)$	Flip-flop	n	Yes
Folded odd-even transposition sorting network (Knuth, 1968, [109])	O(n)	O(n)	Flip-flop	n	No
AKS sorting network (Ajtai et al., 1983, [110])	$O(n \log n)$	$O(n \log n)$	Flip-flop	n	Yes
Linear sorter (Lee and Tsai, 1995, [111] & Perez-Andrade et al., 2009, [112])	O(n)	O(n)	Flip-flop	1	Yes
Interleaved linear sorter (ILS) (Ortiz and Andrews, 2010, [113])	O(wn)	O(wn)	Flip-flop	$1 \le w \le n$	Yes
Shear-sort (2D mesh) (Scherson and Sen, 1989, [114])	O(n)	O(n)	Flip-flop	n	No
Streaming sorting network (Zuluaga et al., 2016, [106])	$O(w \log^2 n)$	$O(n \log^2 n)$	RAM	$2 \le w < n$	Yes
Folded streaming sorting network (Zuluaga et al., 2016, [106])	O(w)	O(n)	RAM	$2 \le w < n$	No

Chen and Prasanna [115] also assess the performance of sorting architectures, considering their O(n) complexities. Their evaluation encompasses considerations of latency, logic usage, memory usage, throughput, and memory throughput. They study merge sort, in-place sort, bitonic sort, and sort-merge join.

architectures, from algorithmic considerations to hardwarelevel enhancements. For instance, when developing new sorting algorithms, it is crucial to commence with an initial argument considering a time complexity of O(n). Table III enumerates various sorting network architectures and highlights key features emphasized by Zuluaga et al. [106]. Assessing the evolution of sorting architectures, an emerging trend involves using RAM devices for a new sorting approach known as stream sorting [106], [116]. Stream sorting takes ndata words as input and produces w words per clock cycle across n/w clock cycles. The sorter achieves a throughput of w if it operates in a fully streaming manner, implying no waiting time between consecutive input sets. Without a fully stream network, the throughput will be less than w words per cycle. We anticipate that one of the pivotal challenges lies in devising algorithms tailored specifically for hardware design, addressing pipeline and parallel processing concerns. Solutions such as stream sorting represent cutting-edge approaches for achieving a more efficient design right from the initial stages, optimizing both memory utilization and time complexity.

B. Power and Energy Efficiency

The issue of power usage holds significant importance in current and future hardware designs. Given that sorting designs are being incorporated into a range of embedded and power-limited systems, the reduction of power consumption takes on a vital role. Future works must delve into innovative strategies for ultra-low-power hardware. These could encompass advanced clock gating, dynamic voltage scaling, and enhanced management of data transfer to curtail the energy consumption tied to the implementation of sorting designs. Additionally, by loosening accuracy demands and taking advantage of approximate computing techniques, hardware has the capacity to execute computations with fewer resources.

C. Resource Limitations

Hardware designs must operate within the boundaries defined by accessible resources such as registers, memory, and processing units. Striving to optimize the utilization of these resources while upholding performance is challenging, especially when dealing with intricate sorting algorithms that

exhibit diverse computational demands. Lin *et al.* [46] provide a trade-off between throughput and resources. UC-based solutions (*e.g.*, [41], [56], [69]) have successfully achieved hardware sorting designs with extremely simple digital logic. However, they achieved this at the cost of an exponential increase in latency. Developing future sorting systems based on such emerging computing systems that operate on simple data representations [41], [117], [118] is a promising path forward.

D. Latency vs. Throughput Trade-off

Designing hardware sorting systems necessitates finding the right compromise between latency (the duration of a single sorting operation) and throughput (the number of sorting operations completed within a specific period). Designers must achieve an optimum point based on the application expectations and hardware constraints.

E. Parallelism

Sorting algorithms encompass repetitive and regular processes that hold the potential for improvement with parallelization and pipelining. Nonetheless, implementing efficient parallel/pipelined hardware architectures (e.g., [50]) and the oversight of data inter-dependencies can intricate these endeavors. Striking a harmonious equilibrium amidst diverse processing units while upholding synchronization and communication can pose a considerable challenge. PIM solutions hold significant promise for the highly parallel execution of future sorting architectures.

F. Adaptation

Numerous practical applications demand data sorting in dynamic and ever-evolving streams. Crafting hardware-based sorting designs capable of adeptly managing these dynamic inputs in real-time presents a multifaceted difficulty. It is imperative for researchers to delve into adaptive algorithms capable of flexibly adapting to shifting input patterns. This adaptability should ensure sustained, efficient sorting performance while minimizing any notable additional workload.

G. Customization

Hardware sorting designs may need to be customized for specific applications or environments. This requires flexibility in the design process (e.g., [46], [50]) to accommodate different requirements. From different data types to various data precisions (i.e., bit-widths), size of the dataset, and hardware constraints (e.g., area and power budget), achieving the best performance may require customized hardware. However, the higher design time and cost of implementing customized hardware must also be considered.

H. Data Movement and Memory Access

Optimal memory access is pivotal for sorting algorithms, and hardware architectures must strive to curtail data transfer and cache-related inefficiencies. Sorting entails frequent data comparisons and exchanges, introducing the potential for irregular memory access patterns. Effectively handling these access patterns is imperative to avert potential performance bottlenecks. The problem aggregates in big data applications where the sorting engine is expected to sort a large set of data.

I. Technology Scaling

Hardware designs might necessitate adjustments to accommodate technological shifts, such as advancements in the semiconductor manufacturing process. Designers must meticulously evaluate the potentials and consequences of technological scaling on factors such as performance, area, power and energy usage, and various design parameters.

VI. CONCLUSION

Sorting is one of the crucial operations in computer science, widely used in many application domains, from data merging to big data processing, database operations, robotics, wireless sensor networks, signal processing, and wireless networks. A substantial body of work is dedicated to designing hardware-based sorting. In this survey, we reviewed the latest developments in hardware-based sorting, encompassing both comparison-based and comparison-free solutions. Comparison-based solutions tend to incur high hardware costs, particularly as the volume and precision of data increase. Comparison-free solutions have recently been proposed to overcome the challenges associated with compare-and-swapbased sorting designs. We reviewed recent hardware solutions for partial sorting and stream sorting, which are used to sort the top-k largest or smallest values of the dataset. We also studied the latest emerging in-memory solutions for sorting operations. Finally, we outlined the challenges in developing future hardware sorting, aiming to provide readers with insights into the next generation of sorting systems.

REFERENCES

- [1] W. Fan and A. Bifet, "Mining big data: Current status, and forecast to the future," SIGKDD Explor. Newsl., vol. 14, p. 1–5, apr 2013.
- [2] D. J. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, et al., "Faster sorting algorithms discovered using deep reinforcement learning," *Nature*, vol. 618, no. 7964, pp. 257–263, 2023.

- [3] S. Pang, J. Wang, and X. Yi, "Application of loan lost-linking customer path correlated index model and network sorting search algorithm based on big data environment," *Neural Comp. and Apps.*, pp. 1–28, 2022.
- [4] T. Do, G. Graefe, and J. Naughton, "Efficient sorting, duplicate removal, grouping, and aggregation," ACM Trans. Database Syst., vol. 47, jan 2023.
- [5] M. L. Dezaki, S. Hatami, A. Zolfagharian, and M. Bodaghi, "A pneumatic conveyor robot for color detection and sorting," *Cognitive Robotics*, vol. 2, pp. 60–72, 2022.
- [6] G. Montesdeoca, V. Asanza, K. Chica, and D. H. Peluffo-Ordóñez, "Analysis of sorting algorithms using a wsn and environmental pollution data based on fpga," in 2022 International Conference on Applied Electronics (AE), pp. 1–4, 2022.
- [7] S. Shirvani Moghaddam and K. Shirvani Moghaddam, "A threshold-based sorting algorithm for dense wireless sensor systems and communication networks," *IET Wireless Sensor Systems*, vol. 13, no. 2, pp. 37–47, 2023.
- [8] W. Guo and S. Li, "Fast binary counters and compressors generated by sorting network," *IEEE Transactions on Very Large Scale Integration* (VLSI) Systems, vol. 29, no. 6, pp. 1220–1230, 2021.
- [9] C. Kotropoulos, M. Pappas, and I. Pitas, "Sorting networks using l/sub p/ mean comparators for signal processing applications," *IEEE Trans. on Signal Processing*, vol. 50, no. 11, pp. 2716–2729, 2002.
- [10] C.-Y. Lu and C.-M. Wu, "A hardware design approach of sorting for flexray-based clock synchronization," in 2011 IEEE/SICE International Symposium on System Integration (SII), pp. 1400–1405, 2011.
- [11] A. Ivanov, D. Yarotsky, M. Stoliarenko, and A. Frolov, "Smart sorting in massive mimo detection," in 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 1–6, 2018.
- [12] J. Munro and M. Paterson, "Selection and sorting with limited storage," Theoretical Computer Science, vol. 12, no. 3, pp. 315–323, 1980.
- [13] H. Chen, S. Madaminov, M. Ferdman, and P. Milder, "Fpga-accelerated samplesort for large data sets," in ACM/SIGDA Int. Symp. on Field-Prog. Gate Arrays, p. 222–232, 2020.
- [14] M. OrHai and C. Teuscher, "Spatial sorting algorithms for parallel computing in networks," in 2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops, pp. 73–78, 2011.
- [15] W. Teich and H. C. Zeidler, "Data handling and dedicated hardware for the sort problem," in *Database Machines* (H.-O. Leilich and M. Missikoff, eds.), (Berlin, Heidelberg), pp. 205–226, Springer Berlin Heidelberg, 1983.
- [16] G. Graefe, "Implementing sorting in database systems," ACM Comput. Surv., vol. 38, no. 3, p. 10–es, 2006.
- [17] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson, "High-performance sorting on networks of workstations," SIGMOD Rec., vol. 26, p. 243–254, jun 1997.
- [18] B. Shang, R. Crowder, and K.-P. Zauner, "Swarm behavioral sorting based on robotic hardware variation," in 2014 4th International Conference On Simulation And Modeling Methodologies, Technologies And Applications (SIMULTECH), pp. 631–636, 2014.
- [19] H. Bui, H. Nguyen, H. M. La, and S. Li, "A deep learning-based autonomous robotmanipulator for sorting application," *CoRR*, vol. abs/2009.03565, 2020.
- [20] D. Guo, H. Liu, and F. Sun, "Audio–visual language instruction understanding for robotic sorting," *Robotics and Autonomous Systems*, vol. 159, p. 104271, 2023.
- [21] A. S. Shaikat, S. Akter, and U. Salma, "Computer vision based industrial robotic arm for sorting objects by color and height," *Journal* of Engineering Advancements, vol. 1, p. 116–122, Dec. 2020.
- [22] Y. Tang, D. Huang, R. Li, and Z. Huang, "A non-dominated sorting genetic algorithm based on voronoi diagram for deployment of wireless sensor networks on 3-d terrains," *Electronics*, vol. 11, no. 19, 2022.
- [23] J. L. Bordim, K. Nakano, and H. Shen, "Sorting on single-channel wireless sensor networks," in *Proceedings 2002 International Symposium* on *Parallel Architectures, Algorithms, and Networks*, (Los Alamitos, CA, USA), p. 0153, IEEE Computer Society, may 2002.
- [24] M. Singh and V. K. Prasanna, "Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network," in *Proceedings of* the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)., 2003.
- [25] S. Shirvani Moghaddam and K. Shirvani Moghaddam, "A threshold-based sorting algorithm for dense wireless sensor systems and communication networks," *IET Wireless Sensor Systems*, vol. 13, no. 2, pp. 37–47, 2023.

- [26] M. Moshref, R. Al-Sayyed, and S. Al-Sharaeh, "An enhanced multiobjective non-dominated sorting genetic routing algorithm for improving the qos in wireless sensor networks," *IEEE Access*, vol. 9, pp. 149176–149195, 2021.
- [27] S.-H. Shiau and C.-B. Yang, "A fast sorting algorithm and its generalization on broadcast communications," in *Computing and Combinatorics* (D.-Z. Du, P. Eades, V. Estivill-Castro, X. Lin, and A. Sharma, eds.), pp. 252–261, 2000.
- [28] K. Chi, J. Shen, Y. Li, Y. Li, and S. Wang, "Multi-function radar signal sorting based on complex network," *IEEE Signal Processing Letters*, vol. 28, pp. 91–95, 2021.
- [29] J. Wang, C. Hou, and F. Qu, "Multi-threshold fuzzy clustering sorting algorithm," in 2017 Progress In Electromagnetics Research Symposium - Spring (PIERS), pp. 889–892, 2017.
- [30] S.-H. Shiau and C.-B. Yang, "Generalization of sorting in single hop wireless networks," *IEICE Trans. Inf. Syst.*, vol. 89-D, pp. 1432–1439, 2006
- [31] B. Abbasi, J. Calder, and A. M. Oberman, "Anomaly detection and classification for streaming data using pdes," SIAM Journal on Applied Mathematics, vol. 78, p. 921–941, Jan 2018.
- [32] X. Zhang, H. Zhang, S. Song, X. Huang, C. Wang, and J. Wang, "Backward-sort for time series in apache iotdb," in 2023 IEEE 39th Intern. Conf. on Data Engineering (ICDE), pp. 3196–3208, 2023.
- [33] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions* on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, 2002.
- [34] B. C. L. Ramírez, G. Guzmán, W. Alhalabi, N. Cruz-Cortés, M. Torres-Ruiz, and M. Moreno-Ibarra, "On the usage of sorting networks to control greenhouse climatic factors," *International Journal of Distributed Sensor Networks*, vol. 14, no. 2, p. 1550147718756871, 2018.
- [35] W. Chen, Y. Liu, Z. Chen, F. Liu, and N. Xiao, "External sorting algorithm: State-of-the-art and future directions," *IOP Conf. Series: Materials Science and Engineering*, vol. 806, p. 012040, apr 2020.
- [36] Y. Ben Jmaa, R. Ben Atitallah, D. Duvivier, and M. Ben Jemaa, "A comparative study of sorting algorithms with fpga acceleration by high level synthesis," *Comp. y Sistemas*, vol. 23, pp. 213–230, 2019.
- [37] I. Skliarova, "A survey of network-based hardware accelerators," *Electronics*, vol. 11, no. 7, 2022.
- [38] R. Ali, "Hardware solution to sorting algorithms: A review," Turkish Journal of Computer and Mathematics Education (TURCOMAT), vol. 13, no. 2, pp. 254–272, 2022.
- [39] D. P. Singh, I. Joshi, and J. Choudhary, "Survey of gpu based sorting algorithms," Int. J. Parallel Program., vol. 46, p. 1017–1034, dec 2018.
- [40] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1389–1402, 2013.
- [41] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471– 1480, 2018.
- [42] H. H. Draz, N. E. Elashker, and M. M. A. Mahmoud, "Optimized algorithms and hardware implementation of median filter for image processing," *Circuits, Systems, and Signal Processing*, vol. 42, pp. 5545–5558, Apr 2023.
- [43] J. Chhugani, A. D. Nguyen, V. W. Lee, W. Macy, M. Hagog, Y.-K. Chen, A. Baransi, S. Kumar, and P. Dubey, "Efficient implementation of sorting on multi-core simd cpu architecture," *Proc. VLDB Endow.*, vol. 1, p. 1313–1324, aug 2008.
- [44] S. Abdel-Hafeez and A. Gordon-Ross, "An efficient o(n) comparison-free sorting algorithm," *IEEE Transactions on Very Large Scale Integration Sys.*, vol. 25, no. 6, pp. 1930–1942, 2017.
- [45] S. S. Ray and S. Ghosh, "k-degree parallel comparison-free hardware sorter for complete sorting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.
- [46] S.-H. Lin, P.-Y. Chen, and Y.-N. Lin, "Hardware design of low-power high-throughput sorting unit," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1383–1395, 2017.
- [47] A. Norollah, D. Derafshi, H. Beitollahi, and M. Fazeli, "Rths: A low-cost high-performance real-time hardware sorter, using a multidimensional sorting algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 7, pp. 1601–1613, 2019.
- [48] P. T. Jelodari, M. P. Kordasiabi, S. Sheikhaei, and B. Forouzandeh, "An o(1) time complexity sorting network for small number of inputs with hardware implementation," *Microprocessors and Microsystems*, vol. 77, p. 103203, 2020.

- [49] P. Papaphilippou, C. Brooks, and W. Luk, "An adaptable high-throughput fpga merge sorter for accelerating database analytics," in 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 65–72, 2020.
- [50] P. Papaphilippou, C. Brooks, and W. Luk, "Flims: Fast lightweight merge sorter," in 2018 International Conference on Field-Programmable Technology (FPT), pp. 78–85, IEEE, 2018.
- [51] P. Preethi, K. Mohan, K. Sudeendra Kumar, and K. K. Mahapatra, "Low power sorters using clock gating," in 2021 IEEE International Symposium on Smart Electronic Systems (iSES), pp. 6–11, 2021.
- [52] B. Prince, M. H. Najafi, and B. Li, "Scalable low-cost sorting network with weighted bit-streams," in 2023 24th International Symposium on Quality Electronic Design (ISQED), pp. 1–6, IEEE, 2023.
- [53] T. Bhargav and E. Prabhu, "Power and area efficient fsm with comparison-free sorting algorithm for write-evaluate phase and readsort phase," in Advances in Signal Processing and Intelligent Recognition Systems: 4th International Symposium SIRS 2018, Bangalore, India, pp. 433–442, Springer, 2019.
- [54] W.-T. Chen, R.-D. Chen, P.-Y. Chen, and Y.-C. Hsiao, "A high-performance bidirectional architecture for the quasi-comparison-free sorting algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 4, pp. 1493–1506, 2021.
- [55] G. C. Sri, S. A. K. V. Hanuma, N. Harshita, and S. Agrawal, "An efficient quasi comparison-free bidirectional architecture for sorting algorithm," in 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), pp. 1–8, 2022.
- [56] A. H. Jalilvand, S. N. Estiri, S. Naderi, M. H. Najafi, and M. Imani, "A fast and low-cost comparison-free sorting engine with unary computing: Late breaking results," in the 59th ACM/IEEE Design Automation Conference, DAC '22, p. 1390–1391, 2022.
- [57] B. Yu, T. Mak, X. Li, F. Xia, A. Yakovlev, Y. Sun, and C.-S. Poon, "Real-time fpga-based multichannel spike sorting using hebbian eigenfilters," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 4, pp. 502–515, 2011.
- [58] G. Campobello, G. Patanè, and M. Russo, "On the complexity of min-max sorting networks," *Info. Sci.*, vol. 190, pp. 178–191, 2012.
- [59] J. Subramaniam, J. Raju, and D. Ebenezer, "Fast median-finding word comparator array," *Elect. Lett.*, vol. 53, no. 21, pp. 1402–1404, 2017.
- [60] U. A. Korat, P. Yadav, and H. Shah, "An efficient hardware implementation of vector-based odd-even merge sorting," in *IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pp. 654–657, 2017.
- [61] D. Valencia and A. Alimohammad, "An efficient hardware architecture for template matching-based spike sorting," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 481–492, 2019.
- [62] F. Zhang, S. Angizi, and D. Fan, "Max-pim: Fast and efficient max/min searching in dram," in 58th Design Automation Conference DAC, pp. 211–216, 2021.
- [63] D. Yan, W.-X. Wang, L. Zuo, and X.-W. Zhang, "A novel scheme for real-time max/min-set-selection sorters on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 7, pp. 2665–2669, 2021.
- [64] C.-H. Wu and K.-Y. Huang, "Data sorting in flash memory," ACM Transactions on Storage (TOS), vol. 11, no. 2, pp. 1–25, 2015.
- [65] N. Samardzic, W. Qiao, V. Aggarwal, M.-C. F. Chang, and J. Cong, "Bonsai: High-performance adaptive merge tree sorting," in 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 282–294, IEEE, 2020.
- [66] Z. Li, N. Challapalle, A. K. Ramanathan, and V. Narayanan, "Imc-sort: In-memory parallel sorting architecture using hybrid memory cube," in the 2020 on Great Lakes Symposium on VLSI, pp. 45–50, 2020.
- [67] A. K. Prasad, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, "Memristive data ranking," in 2021 IEEE International Symposium on High Performance Computer Architecture (HPCA), (Seoul, South Korea), pp. 440–452, 2021.
- [68] Z. Chu, Y. Luo, P. Jin, and S. Wan, "Nymsorting: Efficient sorting on non-volatile memory," in *The 33rd International Conference on Software Engineering & Knowledge Engineering (SEKE 2021)*, 2021.
- [69] M. R. Alam, M. H. Najafi, and N. Taherinejad, "Sorting in memristive memory," ACM Journal on Emerging Technologies in Computing Systems (JETC), vol. 18, no. 4, pp. 1–21, 2022.
- [70] L. Yu, Z. Jing, Y. Yang, and Y. Tao, "Fast and scalable memristive in-memory sorting with column-skipping algorithm," in 2022 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 590– 594, IEEE, 2022.

- [71] F. Zokaee, F. Chen, G. Sun, and L. Jiang, "Sky-sorter: A processing-in-memory architecture for large-scale sorting," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 480–493, 2022.
- [72] M. Lenjani, A. Ahmed, and K. Skadron, "Pulley: An algorithm/hardware co-optimization for in-memory sorting," *IEEE Computer Architecture Letters*, vol. 21, no. 2, pp. 109–112, 2022.
- [73] M. Lenjani, P. Gonzalez, E. Sadredini, S. Li, Y. Xie, A. Akel, S. Eilert, M. R. Stan, and K. Skadron, "Fulcrum: A simplified control and access mechanism toward flexible and practical in-situ accelerators," in 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 556–569, IEEE, 2020.
- [74] Y. Liu, Y. Ou, W. Chen, Z. Chen, and N. Xiao, "Lazysort: A customized sorting algorithm for non-volatile memory," *Info. Sci.*, vol. 641, p. 119137, 2023.
- [75] A. Rjabov, "Hardware-based systems for partial sorting of streaming data," in 2016 15th Biennial Baltic Electronics Conference (BEC), pp. 59–62, 2016.
- [76] J. M. Chambers, "Algorithm 410: Partial sorting," Commun. ACM, vol. 14, p. 357–358, may 1971.
- [77] M. H. Najafi, D. J. Lilja, M. Riedel, and K. Bazargan, "Power and area efficient sorting networks using unary processing," in 2017 IEEE Intern. Conf. on Computer Design (ICCD), pp. 125–128, 2017.
- [78] S. Ghosh, S. Dasgupta, and S. S. Ray, "A comparison-free hardware sorting engine," in 2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), pp. 586–591, IEEE, 2019.
- [79] A. H. Jalilvand, M. H. Najafi, and M. Fazeli, "Fuzzy-logic using unary bit-stream processing," in 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5, IEEE, 2020.
- [80] M. Najafi, A. H. Jalilvand, and M. Fazeli, "Method and architecture for fuzzy-logic using unary processing," Dec. 9 2021. US Patent App. 17/340,834.
- [81] M. Yoon, "A novel architecture of asynchronous sorting engine module for asic design," JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, vol. 22, no. 4, pp. 224–233, 2022.
- [82] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, Jan. 2008.
- [83] B. Yuce, H. F. Ugurdag, S. Gören, and G. Dundar, "A fast circuit topology for finding the maximum of n k-bit numbers," in 2013 IEEE 21st Symposium on Computer Arithmetic, pp. 59–66, 2013.
- [84] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in 50th IEEE/ACM MICRO, pp. 273–287, 2017.
- [85] S. Angizi and D. Fan, "Accelerating bulk bit-wise x(n)or operation in processing-in-dram platform," 2019. arXiv, https://arxiv.org/abs/1904.05782.
- [86] J. O. Cadenas, G. M. Megson, and R. S. Sherratt, "Median filter architecture by accumulative parallel counters," *IEEE Trans. on Circuits* and Systems II: Express Briefs, vol. 62, no. 7, pp. 661–665, 2015.
- [87] J. O. Cadenas, G. M. Megson, R. S. Sherratt, and P. Huerta, "Fast median calculation method," *Electronics Letters*, vol. 48, pp. 558– 560(2), May 2012.
- [88] P. Mitra and H. Bokil, Observed brain dynamics. New York, NY: Oxford University Press, Jan. 2008.
- [89] T. Zhang, C. Lammie, M. R. Azghadi, A. Amirsoleimani, M. Ahmadi, and R. Genov, "Toward a formalized approach for spike sorting algorithms and hardware evaluation," 2022 IEEE MWSCAS, Aug 2022.
- [90] S. Gibson, J. W. Judy, and D. Marković, "Spike sorting: The first step in decoding the brain: The first step in decoding the brain," *IEEE Signal Processing Magazine*, vol. 29, no. 1, pp. 124–143, 2012.
- [91] B. P. Christie, D. M. Tat, Z. T. Irwin, V. Gilja, P. Nuyujukian, J. D. Foster, S. I. Ryu, K. V. Shenoy, D. E. Thompson, and C. A. Chestek, "Comparison of spike sorting and thresholding of voltage waveforms for intracortical brain-machine interface performance," *J Neural Eng*, vol. 12, p. 016009, Dec. 2014.
- [92] D. Valencia and A. Alimohammad, "An efficient hardware architecture for template matching-based spike sorting," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 481–492, 2019.
- [93] M. Godfrey and D. Hendry, "The computer as von neumann planned it," *IEEE Annals of the History of Computing*, vol. 15, pp. 11–21, 1993.
- [94] P. Trancoso, "Moving to memoryland: in-memory computation for existing applications," in *Proceedings of the 12th ACM International* Conference on Computing Frontiers, pp. 1–6, 2015.

- [95] K. A. Ali, A. Baghdadi, E. Dupraz, M. Leonardon, M. Rizk, and J. Diguet, "Mol-based in-memory computing of binary neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI)* Systems, vol. 30, pp. 869–880, jul 2022.
- [96] A. Laga, J. Boukhobza, F. Singhoff, and M. Koskas, "Montres: Merge on-the-run external sorting algorithm for large data volumes on ssd based storage systems," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1689–1702, 2017.
- [97] S. D. Viglas, "Write-limited sorts and joins for persistent memory," Proceedings of the VLDB Endowment, vol. 7, no. 5, pp. 413–424, 2014.
- [98] A. Farmahini-Farahani, H. J. Duwe III, M. J. Schulte, and K. Compton, "Modular design of high-throughput, low-latency sorting units," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1389–1402, 2012.
- [99] S. H. Pugsley, A. Deb, R. Balasubramonian, and F. Li, "Fixed-function hardware sorting accelerators for near data mapreduce execution," in 2015 33rd IEEE International Conference on Computer Design (ICCD), pp. 439–442, IEEE, 2015.
- [100] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on fpga," in 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 103–110, IEEE, 2016.
- [101] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A hybrid design for high performance large-scale sorting on fpga," in *IEEE 2015 ReConFig*, pp. 1–6, 2015.
- [102] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, no. 7290, 2010.
- [103] S. Kvatinsky, D. Belousov, S. Liman, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Magic—memristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, no. 11, pp. 895–899, 2014.
- [104] S. Gupta, M. Imani, and T. Rosing, "Felix: Fast and energy-efficient logic in memory," in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–7, IEEE, 2018.
- [105] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471– 1480, 2018.
- [106] M. Zuluaga, P. Milder, and M. Püschel, "Streaming sorting networks," ACM Transactions on Design Automation of Electronic Systems, vol. 21, no. 4, 2016.
- [107] K. E. Batcher, "Sorting networks and their applications," p. 307–314, 1968
- [108] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers*, vol. C-20, no. 2, pp. 153–161, 1971.
- [109] D. E. Knuth, *The art of computer programming*. Addison-Wesley Pub. Co., 1968.
- [110] M. Ajtai, J. Komlós, and E. Szemerédi, "An 0(n log n) sorting network," in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, p. 1–9, 1983.
- [111] C. Y. Lee and J. M. Tsai, "A shift register architecture for high-speed data sorting," *Journal of VLSI signal processing systems for signal*, image and video technology, vol. 11, pp. 273–280, Dec 1995.
- [112] R. Perez-Andrade, R. Cumplido, C. Feregrino-Uribe, and F. Martin Del Campo, "A versatile linear insertion sorter based on an fifo scheme," *Microelectronics Journal*, vol. 40, no. 12, pp. 1705–1713, 2009.
- [113] J. Ortiz and D. Andrews, "A configurable high-throughput linear sorter system," in 2010 IEEE International Symp. on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1–8, 2010.
- [114] I. Scherson and S. Sen, "Parallel sorting in two-dimensional vlsi models of computation," *IEEE Transactions on Computers*, vol. 38, no. 2, pp. 238–249, 1989.
- [115] R. Chen and V. K. Prasanna, "Computer generation of high throughput and memory efficient sorting designs on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3100–3113, 2017.
- [116] M. Petrović and V. Milovanović, "A chisel generator of parameterizable and runtime reconfigurable linear insertion streaming sorters," in 2021 IEEE 32nd International Conference on Microelectronics (MIEL), pp. 251–254, 2021.
- [117] M. H. Najafi, D. Jenson, D. J. Lilja, and M. D. Riedel, "Performing stochastic computation deterministically," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 12, pp. 2925–2938, 2019.
- [118] S. Aygun, M. S. Moghadam, M. H. Najafi, and M. Imani, "Learning from hypervectors: A survey on hypervector encoding," arXiv preprint arXiv:2308.00685, 2023.



Amir Hossein Jalilvand received a B.Sc. degree in Computer Engineering from Bu-Ali Sina University, Hamadan, Iran, and an M.Sc. degree in Computer Engineering - Computer Architecture from Iran University of Science and Technology, Tehran, Iran Currently, he is a Ph.D. candidate in Computer Engineering. His research interests include Stochastic and Unary Computing, Computer Architecture, Fuzzy Logic, and machine learning.



Faeze S. Banitaba (S'23) received a B.Sc. degree in Computer Engineering from Shahid Beheshti University, Tehran, Iran. She completed her M.Sc. degree in Computer Architecture from the University of Tehran in 2017. Currently, she is a Ph.D. student in Computer Engineering. Her research interests rely on Stochastic Computing, Hardware Security for Neural Networks, and Hyperdimensional Computing. She was recognized as a DAC Young Fellow 2023.



Seyedeh Newsha Estiri received a B.Sc. degree in Computer Engineering - Hardware System Design from Iran University of Science and Technology, Tehran, Iran, in 2020. Currently, she is a second-year Ph.D. student in computer engineering at the University of Louisiana at Lafayette. Her research interests include Stochastic computing, Machine Learning Applications and Hardware, and Computer Architecture. Newsha was selected as a DAC Young Fellow in DAC 2022.



Sercan Aygun (S'09-M'22) received a B.Sc. degree in Electrical & Electronics Engineering and a double major in Computer Engineering from Eskisehir Osmangazi University, Turkey, in 2013. He completed his M.Sc. degree in Electronics Engineering from Istanbul Technical University in 2015 and a second M.Sc. degree in Computer Engineering from Anadolu University in 2016. Dr. Aygun received his Ph.D. in Electronics Engineering from Istanbul Technical University in 2022. Dr. Aygun's Ph.D. work has appeared in several Ph.D. Forums of top-

tier conferences, such as DAC, DATE, and ESWEEK. He received the Best Scientific Research Award of the ACM SIGBED Student Research Competition (SRC) ESWEEK 2022 and the Best Paper Award at GLSVLSI'23. Dr. Aygun's Ph.D. work was recognized with the Best Scientific Application Ph.D. Award by the Turkish Electronic Manufacturers Association. He is currently a postdoctoral researcher at the University of Louisiana at Lafayette, USA. He works on emerging computing technologies, including stochastic computing in computer vision and machine learning.



M. Hassan Najafi (S'15-M'18-SM'23) received the B.Sc. degree in Computer Engineering from the University of Isfahan, Iran, the M.Sc. degree in Computer Architecture from the University of Tehran, Iran, and the Ph.D. degree in Electrical Engineering from the University of Minnesota, Twin Cities, USA, in 2011, 2014, and 2018, respectively. He is currently an Assistant Professor with the School of Computing and Informatics, University of Louisiana, LA, USA. His research interests include stochastic and approximate computing, unary processing, in-

memory computing, and hyperdimensional computing. He has authored/co-authored more than 60 peer-reviewed papers and has been granted 5 U.S. patents with more pending. In recognition of his research, he received the 2018 EDAA Outstanding Dissertation Award, the Doctoral Dissertation Fellowship from the University of Minnesota, and the Best Paper Award at the ICCD'17 and GLSVLSI'23. Dr. Najafi has been an editor for the IEEE Journal on Emerging and Selected Topics in Circuits and Systems.