electronics

Article

A Scalable Sorting Network Based on Hybrid Algorithms for Accelerating Data Sorting

Xufeng Li [1,2], Li Zhou [1,*] and Yan Zhu [1]

[1] National Space Science Center, Chinese Academy of Sciences, Beijing 100190, China; lixufeng21@mails.ucas.ac.cn (X.L.)
[2] School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100190, China
* Correspondence: zhouli@nssc.ac.cn

Abstract: Sorting in sequential data mining is significantly improved through hardware acceleration, which becomes essential as data volume and complexity increase. This paper presents a scalable hybrid sorting network that maintains or improves performance while reducing computational load and hardware requirements. The network is composed of the pre-comparison odd–even sorting network (P-OESN) and the bidirectional insertion sorting network (BISN). A pre-comparison layer is introduced to the original OESN. This layer aims to place larger values in the first half of the input sequence and smaller values in the latter half. The number of iterations is reduced when the P-OESN transitions from fully parallel execution to iterative execution. A novel pipelined BISN architecture is proposed, which leads to enhanced operating frequency and throughput. The experimental results show that the pre-comparison layer reduces the number of iterations by 6% to 50%. Throughput is improved by more than four times, and operating frequency is increased by more than two times due to the pipelined BISN. The proposed hybrid sorting network reduces sorting time or resource usage, while enabling the sorting of large-scale data sets that other methods cannot support.

Keywords: field programmable gate array (FPGA); hybrid sorting network; scalable architecture; bidirectional processing; pre-comparison; pipeline technology

Academic Editor: Cecilio Angulo

Received: 19 December 2024
Revised: 21 January 2025
Accepted: 27 January 2025
Published: 1 February 2025

Citation: Li, X.; Zhou, L.; Zhu, Y. A Scalable Sorting Network Based on Hybrid Algorithms for Accelerating Data Sorting. *Electronics* **2025**, *14*, 579. https://doi.org/10.3390/electronics14030579

1. Introduction

Data mining, regarded as a vital decision support process, is used to apply machine learning [1], pattern recognition [2,3], statistics, databases [4], and visualization technologies [5]. This allows patterns to be extracted from large datasets, supporting informed decision-making. A significant branch of this field is sequential data mining, which focuses on detecting evolutionary patterns in sequential data. Such data include electromagnetic signals [6], radar navigation [7], communication streams [8], video sequences [9], robotic perception [10], and energy data [11]. This area of research is considered crucial for enhancing situational awareness in both military and civilian contexts.

Sorting data, a fundamental task in sequential data mining [12], can be substantially improved through hardware acceleration. As the volume and complexity of sequential data grow, rapid and resource-efficient hardware acceleration becomes essential for improving the efficiency of sequential data mining. In hardware-based sorting methods, including bitonic sorting networks (BSNs) [13–16], odd–even sorting networks (OESNs) [17,18], and insertion sorting networks (ISNs) [14,19–21], an exponential relationship is observed between the resources required and the size of the data [22]. Although high sorting speeds are achieved, the exponential increase in resource utilization lowers efficiency.

To address this challenge, researchers are developing methods that reduce computational load and hardware needs while maintaining or enhancing performance. Key contributions from recent research include the following:

1. In [23], a novel scalable hardware sorting architecture is introduced. This architecture is designed to address the challenges of high-performance sorting in resource-constrained systems. A dual-layer design is employed, consisting of the one-way linear insertion sorter (OLIS) and the cyclic bitonic merge tetwork (CBMN). The OLIS is based on insertion sort and is optimized for bandwidth. The properties of bitonic sequences are utilized by CBMN to efficiently merge the sequences sorted by the OLIS. It is demonstrated that hybrid sorting networks, compared to single networks, significantly reduce hardware resource usage while maintaining sorting efficiency. The scalability in [23] enables the sorting network to balance speed and resource consumption effectively. A scalable sorting network can convert parallel sorting units into iterative operations. As data size increases, the required number of sorting units also grows. By iteratively reusing certain sorting units rather than simply adding more, the network scale is reduced. Although a smaller scale may reduce throughput, a higher clock frequency can be achieved, allowing the overall speed to remain largely unaffected. Moreover, in resource-constrained environments, the scalability ensures that the architecture remains feasible.
2. In [24], a hybrid pipeline sorting architecture is introduced. This architecture consists of a bitonic sorter and cascaded insertion sorting units, forming an efficient hybrid sorting network. The bitonic sorter generates partially ordered sequences, while the cascaded units identify the maximum values and output fully sorted sequences in ascending order. This network achieves the same performance as [23], but with fewer resources. However, there is still room for optimization. The architecture incorporates a BSN that lacks scalable characteristics, meaning it does not support the scalability.
3. Inspired by previous works [24,25], Chen et al. introduced a sorting architecture that incorporates bidirectional processing into the cascade of insertion sorting units [26]. A novel bidirectional insertion sorting unit (BISU) is designed based on this concept. When a segment-sorted subsequence is received by the BISU, the maximum and minimum values are recorded. If these recorded values are updated, the original maximum and minimum values are inserted into the sorted subsequence simultaneously. This ensures that the order of the subsequence is preserved. Once all segment subsequences have passed through the cascading BISU, a fully sorted sequence is produced. The cascading BISU units form a bidirectional insertion sorting network (BISN), characterized by its scalability [27]. Resource consumption can be significantly reduced by decreasing the degree of BISU parallelism. Specifically, the number of BISUs is decreased, and a smaller number of BISUs is reused iteratively to complete the sorting task. Although some throughput is sacrificed, a substantial reduction in resource usage is achieved. Compared to conventional cascaded insertion sorting, the BISU offers higher efficiency and lower resource consumption, making it a more optimal choice.

In conclusion, the efficiency of hybrid networks compared to single networks, as well as the scalability in maintaining speed while reducing resource consumption, is demonstrated in [23]. In [24], a more efficient hybrid sorting network is proposed. Nonetheless, due to the use of a basic BSN, this approach lacks scalable characteristics. In [26], the BISN is introduced. This network could replace the unidirectional insertion network in [24], thereby improving sorting speed and reducing resource usage.

Based on previous research, a scalable hybrid sorting network has been proposed. This network is composed of a pre-comparison OESN (P-OESN) and BISN, both of which exhibit scalability. The scalability of the P-OESN and BISN imparts a scalable characteristic to the proposed network. The scalability of these networks is attributed to the identical nature of each comparison step, which allows for the iterative reuse of the same circuit components during the process. The sorting speed is determined by a combination of throughput and clock frequency. As data size increases, reusing specific circuit components may slightly reduce throughput. However, higher clock frequencies can still be achieved, meaning the sorting speed remains largely unaffected while resource usage is significantly reduced. In contrast to the network introduced in [28], the P-OESN is utilized in place of the BSN while maintaining the same time and resource complexity. To reduce the number of iterations required for scaled-down odd–even sorting, pre-comparisons are performed on the values in the first and second halves of the sequence. Additionally, the BISN is employed instead of a simple unidirectional insertion sorting network. This change further enhances the efficiency of the sorting network.

Specifically, the dataset to be sorted is first divided into several equal-length smaller datasets. These smaller datasets are then sequentially fed into the P-OESN for sorting. The BISN subsequently processes the partially ordered subsequences generated by the OESN, recording the maximum and minimum values of each subsequence. Once all subsequences have passed through the cascaded bidirectional insertion sorting units, a fully sorted dataset is obtained. This method simplifies the large-scale data sorting task into a series of smaller-scale sorting tasks.

To maintain a largely unaffected sorting speed and reduce resource consumption, the scalability of the sorting network is used to control its size. Parallel operations involving multiple identical computing units are replaced by iterative processing using fewer computing units. In the P-OESN, each stage is composed of a fixed number of comparison and exchange units, with each unit responsible for comparing and swapping adjacent elements. In the most resource-constrained case, the entire sorting task can be completed with a single stage of comparison-exchange units through iterative processing, greatly reducing the hardware resource requirements. In the BISN, each stage consists of identical BISUs. By reusing the BISU in multiple iterations, the size of the network is effectively reduced. Although throughput may decrease with iterative processing, an optimal balance between resource usage and clock frequency can be achieved through careful trade-offs.

Furthermore, pipeline technology [29] was applied to optimize the architecture of the BISU, increasing its throughput. This optimization raised its operating frequency and improved throughput. In the proposed BISU architecture, operations are completed in five CLKs, unlike the original design, which completed all operations in two CLKs. The entire operation is divided into four pipeline stages, and the third stage is duplicated. The other stages are completed in one CLK, while the third stage requires two CLKs. Duplication of the third stage was implemented to meet the two-CLK requirement, allowing parallel processing. As a result, one set of outputs can be received by the third pipeline stage per CLK, eliminating delays caused by resource contention.

In summary, the main contributions of this paper are as follows:

1. A scalable hybrid sorting network, composed of a P-OESN and BISN, has been proposed to efficiently reduce resource utilization while maintaining performance.
2. Building on [24], the BSN is replaced with the P-OESN in the proposed network to achieve greater scalability, and pre-comparisons are incorporated to reduce iteration.
3. Based on [24], the unidirectional cascade insertion sorting network is replaced with the BISN. Meanwhile, pipeline technology is applied to the BISU architecture, where

throughput is increased by dividing operations into four stages and duplicating the third stage.

The remainder of this paper is organized as follows. Relevant studies are reviewed in Section 2. The proposed sorting methodology and hardware architecture are described in Section 3. In Section 4, the experimental results are presented and discussed, and comparisons are made between the performance of the proposed architecture and existing sorting designs. Finally, the conclusion of this paper is provided in Section 5.

2. Related Work

2.1. Odd–Even Sorting Network

An OESN is a type of parallel sorting network that efficiently sorts a sequence of values through a series of comparison and exchange operations, as shown in Figure 1. The architecture consists of a series of comparators that operate in a specific sequence. Each comparator compares two adjacent elements and swaps them if they are out of order.

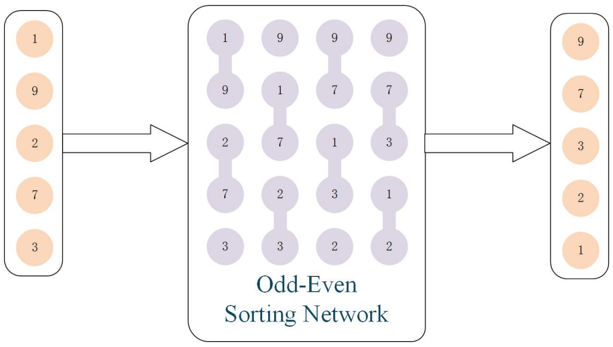

Figure 1. The architecture of the OESN from [30] for five input data elements.

The structure of an OESN is defined recursively. For n inputs, the network operates in two phases: the odd phase and the even phase. During the odd phase, comparisons are made between pairs of elements at odd and even indices (e.g., comparing elements at indices 1 and 2, 3, and 4, etc.). In the even phase, comparisons occur between pairs at even indices (e.g., indices 0 and 1, 2 and 3, etc.). This alternating pattern continues until the entire sequence is sorted. The overall number of comparators required for sorting n elements is approximately $n(n-1)/2$, and the time complexity for sorting is $O(nlogn)$. This network is particularly efficient for parallel architectures, where multiple comparisons can occur simultaneously.

The regularity of the OESN is a key feature that enables its scalability. This regularity arises from the repetitive structure of the network, where the same sequence of operations is applied uniformly across the elements being sorted. In practical terms, this means that each comparator operates under the same rules, and the design can be systematically extended to accommodate larger input sizes without fundamentally altering the underlying structure.

2.2. Bidirectional Insertion Sorting Network

The BISN operates in two phases, the recording phase and the insertion phase, as show in Figure 2. This design uses two storage arrays: the Smaller-Value Array (SVA) and the Larger-Value Array (LVA). Each array has a length of $N/2$ and a bit width matching the input data bit width K. In the recording phase, the system compares and swaps the minimum and maximum values of the segmented input subsequence with the corresponding values in the SVA and LVA arrays. After the recording phase is completed, the insertion phase begins. During this phase, if the minimum and maximum values from the input subsequence are swapped with the SVA and LVA values, the data are inserted into their correct position within the sorted subsequence to maintain subsequence order. The architecture is organized into $N/2$ stages. Each stage first performs the recording phase and then, the insertion phase. After all stages are processed, the SVA and LVA arrays store the data in ascending and descending order, respectively. The values in SVA are guaranteed to be smaller than those in LVA.

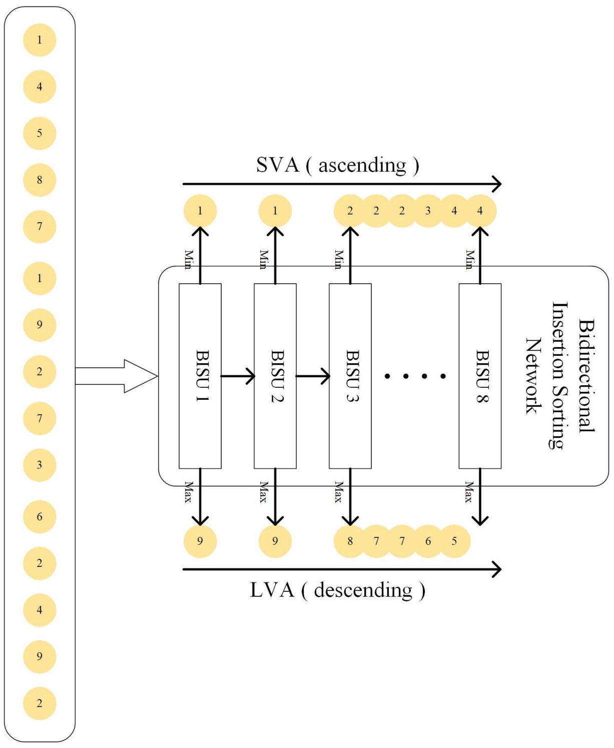

Figure 2. The architecture of the BISN from [26] for 15 input data elements.

The BISU serves as a fundamental component of the BISN. As illustrated in Figure 3, the BISU consists of two registers, Min and Max, and two components, OCM and BISL. The Min and Max registers belong to the SVA and LVA categories. They are utilized to record the maximum and minimum values of each ordered subsequence. The OCM functions as the controller for the BISU. It manages this unit through three signals: ENin, Vin, and INIT. The ENin signal acts as the enable signal. The sorting units remain inactive unless this signal is at a high level. Vin represents the Vout signal outputted from the preceding BISU, with Vout indicating the validity of the output data. The Min and Max registers are initialized to the maximum and minimum values that the registers can represent. During normal operation of the BISU, the value of Min will not exceed that of Max. Only during system initialization will the value of Min be greater than that of Max. This condition indicates that the values within both registers are invalid. When the INIT signal is high, it indicates the system initialization state. The OCM controls the BISU initialization based on the INIT signal. The initialization process involves writing the minimum and maximum values of the input subsequence into the Min and Max registers, respectively. The original values from the Min and Max registers are then transferred into the intermediate portion of the input subsequence.

The bidirectional insertion sorting logic (BISL) serves as the core functional unit of the BISU. It is responsible for recording the maximum and minimum values from the subsequence in the Max and Min registers, respectively. The BISL also manages the insertion of replaced data back into the subsequence while maintaining its ordered state. When determining whether to replace the data in the Min and Max registers, comparisons are made only with the first and last values of the subsequence. This approach is possible because the subsequence is inherently ordered. The data being replaced are inserted back into the subsequence according to the procedure outlined in Equation (1). In this equation, DI represents the input subsequence, DO denotes the output subsequence, RI_{MIN} corresponds to the data in the Min register, RI_{max} refers to the data in the Max register, and P signifies the length of the subsequence.

$$DO[i] = \begin{cases} RI_{\min} & DI[i] < RI_{\min}, DI[i+1] > RI_{\min} \\ DI[i-1] & DI[i] > RI_{\max}, DI[i-1] > RI_{\max} \\ DI[i] & RI_{\min} < DI[i] < RI_{\min} \\ DI[i+1] & DI[i] < RI_{\min}, DI[i+1] < RI_{\min} \\ RI_{\max} & DI[i] > RI_{\max}, DI[i-1] < RI_{\max} \end{cases} \quad i = 1,2,3,\cdots,P \tag{1}$$

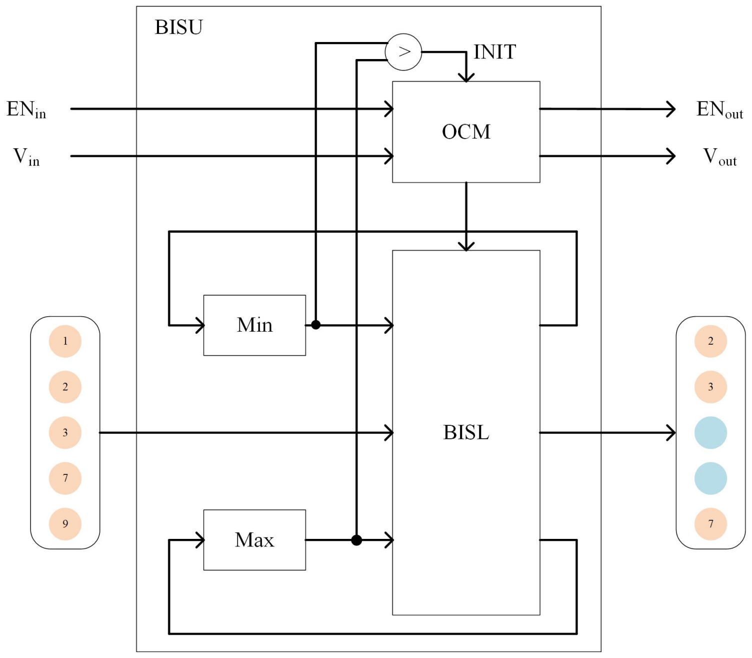

Figure 3. The architecture of the BISU.

3. Methodology and Architecture

In the proposed method, the sequence S of length m is divided into n subsequences SSi of length $l = m/n$, where i ranges from 1 to n, as illustrated in Figure 4. The parameter l is constrained to an odd number, as required by the original definition of the odd–even sorter. After division, all subsequences are processed by the P-OESN and BISN. The sorted subsequences $OSSi$ are output by the P-OESN. In the fully parallel configuration, the P-OESN includes l sorting layers along with one layer dedicated to pre-comparison. The BISN then receives $OSSi$, and the SVA and LVA arrays are produced. When m is even, $m/2$ valid data points are stored in each of the SVA and LVA arrays. When m is odd, $m/2$ valid data points are stored in the SVA and LVA, respectively. By reversing the LVA to obtain $\hat{\text{L}}$VA and concatenating it with the SVA, the sequence S is arranged in ascending order as $\bar{S}$ = SVA,$\hat{\text{L}}$VA. By reversing the SVA to obtain $\hat{\text{S}}$VA and concatenating it with the LVA, the sequence S is arranged in descending order as $\bar{S} =$ LVA, $\hat{\text{S}}$VA.

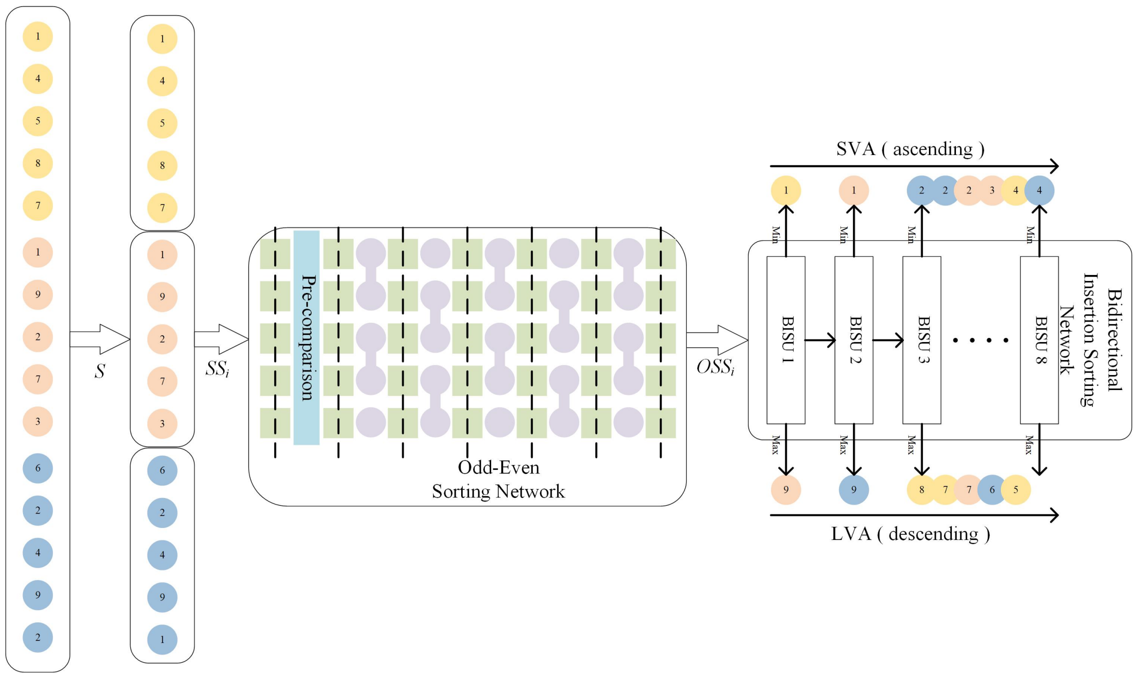

Figure 4. The architecture of the proposed sorting network in a fully parallel configuration.

3.1. Pre-Comparison Odd–Even Sorting Network

In the proposed method, a pre-comparison mechanism is introduced in the initial phase of OESN. In this mechanism, elements in the input subsequences $SS_{i,j}$, $i \in [1, n]$ and $j \in [1, l]$, are first compared and swapped. Specifically, $SS_{i,t}$ and $t \in [1, n/2]$ are compared and swapped with $SS_{i,l-t}$. This step ensures that, in a sequence intended to be sorted in descending order, the relatively larger values are placed in the upper half and the relatively smaller values are placed in the lower half of the sequence. In a sequence intended to be sorted in ascending order, the reverse is achieved. The worst-case scenario, where the input sequence is in the opposite order of the desired target order, is completely avoided by the pre-comparison mechanism. By rearranging the values within each subsequence before proceeding with further sorting operations, it helps avoid highly unbalanced distributions of data that would cause other sorting algorithms to degrade into their worst-case time complexities. As a result, the method ensures a more consistent performance, even in challenging input scenarios, effectively preventing the pitfalls of worst-case behavior and enhancing overall efficiency.

In a fully parallel OESN, the pre-comparison mechanism does not reduce resource requirements. Instead, an additional layer is required to implement this mechanism. The pre-comparison layer becomes a burden under fully parallel conditions. However, in scalable configurations, converting the fully parallel OESN to an iterative OESN allows the pre-comparison mechanism to effectively reduce the number of iterations. This reduction is achieved with a relatively low resource cost. As shown in Figure 5, the OESN is a scalable network. Its scalability allows the network to be reduced and to perform sorting iteratively.

For large-scale sorting tasks, significant data transfer bandwidth is often required. Bandwidths can reach 128 units of 32-bit data or even more. In a fully parallel OESN, the number of layers must match the number of data items sorted in one pass. Such a large-scale sorting network would consume extensive resources. This demand makes implementation impractical. In these cases, the network scalability is utilized to convert it from a fully parallel to an iterative operation.

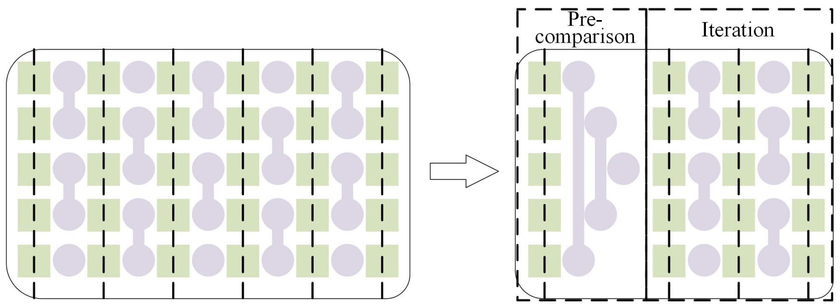

Figure 5. A five-input P-OESN transitions from a fully parallel configuration to an iterative configuration.

In the OESN, the pre-comparison layer is retained. The subsequent layers are then reduced to even-numbered layers. This reduction is based on OESN sorting operations. Sorting operations involve comparison and swapping at odd positions and at even positions. Although both parts use identical operations in equal quantities, they differ in position. To minimize control logic and improve the operating frequency, operations at odd positions are not interchanged with those at even positions. Thus, the reduced sorting network retains an equal number of odd-position and even-position comparison and swapping layers. These two parts alternate during operation.

For sorting 129 of 32 bits data, a fully parallel OESN would require 129 layers. Each layer would contain 64 compare-and-swap units, totaling 1,677,032 bits registers and 8256 compare-and-swap units. By using an iterative approach, sorting can be achieved with only a pre-sorting layer and two additional OESN layers. This approach requires only 516,32 bits registers and 130 compare-and-swap units. As a result, resource usage is significantly reduced, making the design feasible.

3.2. Pipeline-Based Bidirectional Insertion Sorting Unit Architecture

The BISN is structured by connecting multiple BISUs in series. The architecture of a BISU is presented in Figure 6. When the selection signal is 1, the 2-to-1 multiplexer selects the signal from the upper path. When the selection signal is 0, the signal from the lower path is enabled. A four-stage pipeline is employed. In the first pipeline stage, Min and Max values are compared and updated with $OSS_{i,1}$ and $OSS_{i,l}$. Due to the ascending order of the OSS_i sequence, this process establishes the minimum and maximum values. In the second pipeline stage, a 1-bit counter (Counter) is initialized to 1b0. When Counter is 1b0, data from the second stage are directed to the upper bidirectional insertion unit (BIU) in the third pipeline stage. When Counter is 1b1, data flow to the lower BIU unit in the third stage. Control signals are generated by comparing Counter with 1b0 and 1b1. These signals enable the write functions of the inter-stage registers, thereby regulating data flow. Since each BIU requires two clock cycles for processing, duplication of the BIU unit is implemented. This duplication prevents pipeline blocking and resolves structural hazards. Additional pipeline stages are included. The second and fourth stages are responsible for data distribution and merging, respectively. The third pipeline stage contains two BIU units, each of which receives data from the second stage during each clock cycle. In the fourth pipeline stage, a 1-bit counter (Counter), initialized to 1b1, controls data forwarding. When Counter is 1b1, the output from the upper BIU in the third stage is directed to the output register. When Counter is 1b0, the output from the lower BIU is sent to the output register.

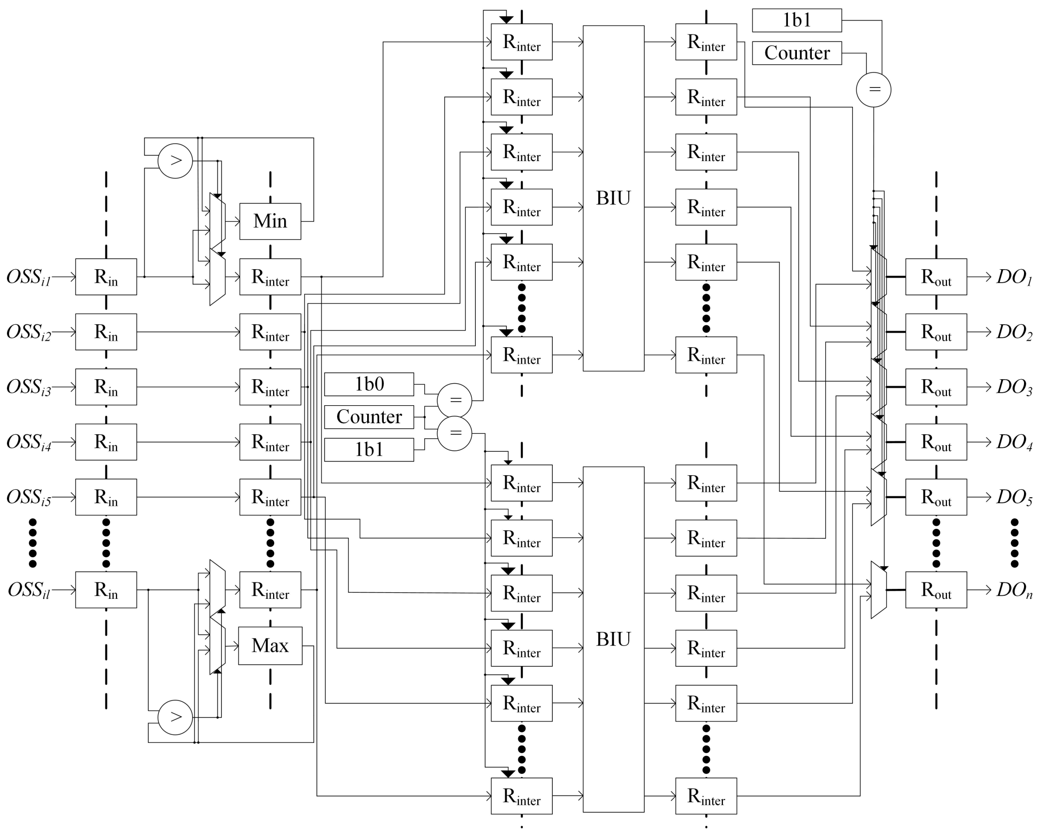

Figure 6. The architecture of the proposed BISU implemented using pipelining technology.

The hardware structure of the BIU is depicted in Figure 7. In this configuration, the l inter-stage pipeline registers, PR_i where $i \in 1 - l$, from the preceding stage are used as inputs. After two clock cycles, outputs are directed to the next stage l inter-stage pipeline registers, AR_i where $i \in 1 - l$. The BIU is constructed from $l - 4$ BILs and four sets of fixed logic. Following the first pipeline stage, the sequence stored in the current inter-stage registers no longer retains order. Comparisons occur between the first register and the Min register, as well as between the last register and the Max register, resulting in potential data exchanges. The BIU then reorders the sequence of length l by inserting the values at both ends into the central section, which remains ordered with a length of $l - 2$. This insertion is effectively a selection process, where values are chosen from PR_1, PR_{i-1}, PR_i, PR_{i+1}, and PR_l, ensuring that only values smaller than the two highest and larger than the two lowest are retained. For boundary registers PR_1, PR_2, PR_{l-1}, and PR_l, the complete set of five values is unavailable. Comparisons and exchanges are, therefore, performed as follows. PR_1 is compared with PR_2, yielding AR_1 upon exchange. PR_2 is compared with PR_1, PR_3, and PR_l, yielding AR_2 upon exchange. PR_{l-1} is compared with PR_1, PR_{l-2}, and PR_l, yielding AR_{l-1} upon exchange. PR_l is compared with PR_{l-1}, yielding AR_l upon exchange. Insertion operations for intermediate positions are managed by the BIL units. Each BIL selects an appropriate value from PR_1, PR_{i-1}, PR_i, PR_{i+1}, and PR_l for output as AR_i.

The BIL functions to select a value LO from five inputs, ensuring that LO is greater than two of these inputs and less than the remaining two. It is assumed that inputs LI_2, LI_3, and LI_4 are arranged in sequential order, with LI_1 less than LI_5. The conditions are structured as follows: For LI_1, it is required that LI_1 be greater than LI_3 and less than LI_4. For LI_2, the condition is met when LI_2 is greater than LI_5. For LI_3, the requirements are fulfilled if LI_2 is greater than LI_1 and LI_4 is less than LI_5. For LI_4, compliance is achieved if LI_4 is less than LI_1. For LI_5, the condition is satisfied when LI_5 is greater than LI_2 and less than LI_3. In accordance with this logic, the circuit displayed in Figure 8 is designed, incorporating only comparators and two-way selectors.

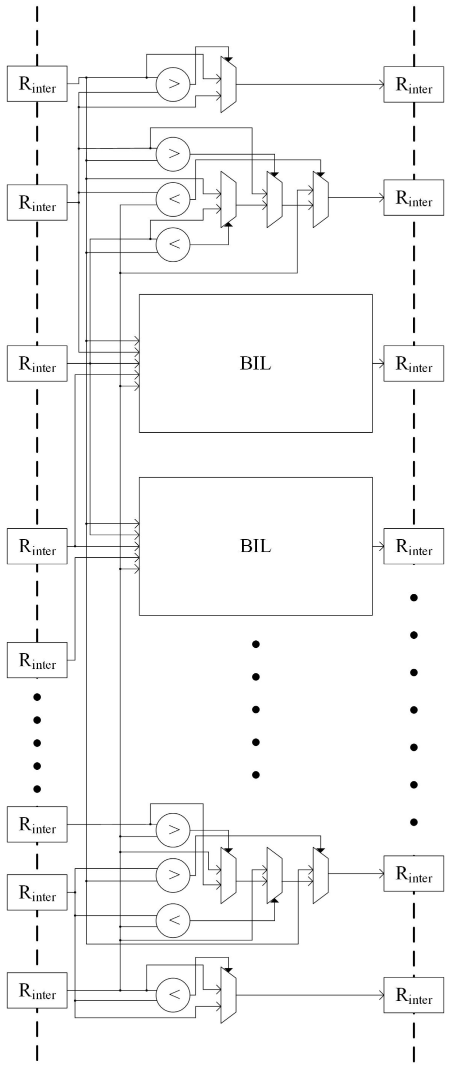

Figure 7. The architecture of the BIU.

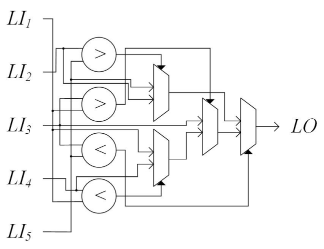

Figure 8. The architecture of the BIL.

As illustrated in Figure 9, the fully parallel BISN is converted to an iterative BISN. This conversion effectively reduces the number of BISUs required, ensuring the feasibility for large-scale data applications. For a sequence of length m, a fully parallel configuration requires $\lceil m/2 \rceil$ BISU. However, in a scalable configuration, the same task can be completed using between 1 and $\lceil m/2 \rceil$ BISU. When g BISU are used to sort a sequence of length m, resource usage is reduced to g/m of the original requirement. Throughput is also decreased to g/m of the original. This scalable approach, thus, reduces resource consumption at the expense of throughput.

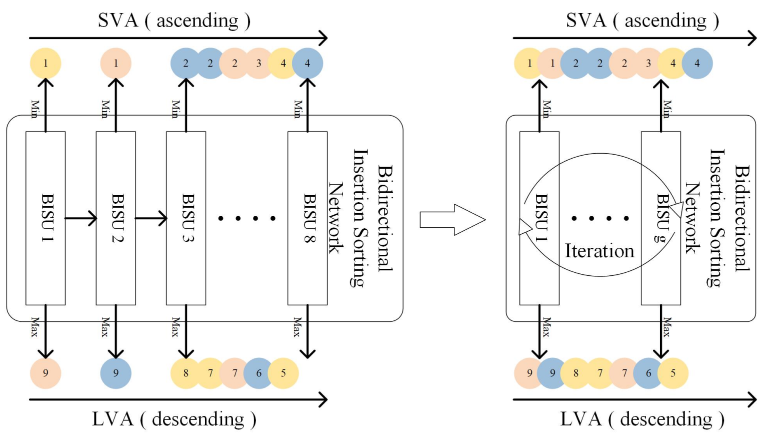

Figure 9. A BISN transitions from a fully parallel configuration to an iterative configuration.

4. Experiment and Discussion

The experiments are divided into three main components: the proposed P-OESN, the pipeline-based BISN architecture, and the scalable hybrid sorting network. In the P-OESN section, experiments will verify if the pre-comparison functionality significantly reduces the required iterations in odd–even sorting. In the pipeline-based BISN architecture section, experiments will validate whether the pipelined BISU achieves higher operating frequencies and throughput compared to the original BISU. In the scalable hybrid sorting network section, this network will be applied in various typical scenarios, with data provided on resource utilization, throughput, and operating frequency. Two settings, input parallelism and the length of the data to be sorted, are considered for the typical scenarios. Common input parallelism configurations include processing one, two, four, eight, or 16 data items at a time. The lengths of the data to be sorted include smaller datasets of 1K items and larger datasets of 1M items.

The experimental data are generated as random values using the random module in a Python 3.6 environment. Data types in electromagnetic signals, radar navigation, communication streams, and video sequences are typically represented as floating-point numbers. Therefore, floating-point numbers were selected for the experiment. To assess the influence of data types on system performance, metrics such as operating frequency, resource utilization, and throughput are measured for 64-bit double, 32-bit single, and 16-bit half floating-point numbers [31].

The sorting network was implemented on the Xilinx ZC706 evaluation board [32]. The FPGA used is the Zynq-7000 XC7Z045-2FFG900C SoC, featuring two ARM Cortex-A9 MPCore processors, 218,600 LUTs, 437,200 FFs, and 19.16 Mb of BRAM.

4.1. Pre-Compare Function

To verify the effectiveness of the pre-comparison function in reducing the number of iterations required in odd–even sorting, a comparative experiment was conducted. The experiment compared the number of iterations needed for odd–even sorting without the pre-comparison function against the iterations needed when the pre-comparison function was applied. In this experiment, data were sorted using only a single odd–even sorting unit. This sorting unit is the smallest unit capable of sorting once in odd-numbered positions and once in even-numbered positions. Since the P-OESN is designed to sort subsequences that are typically equal to or a multiple of the bandwidth, sequence lengths from 8 to 4096 were selected. The sequence length began at 8 and increased in increments of 8 until reaching 4096. For each sequence length, experiments were repeated 100 times. The averages were then calculated to eliminate the influence of random effects.

The results of the experiment are presented in Figure 10. In Figure 10a, the x-axis represents the sequence length, while the y-axis indicates the number of iterations required to sort the sequence with a single odd–even sorting unit. The blue line represents the case without the pre-comparison function, and the red line represents the case with the pre-comparison function. As shown, fewer iterations were required when the pre-comparison function was applied.

Figure 10b shows the number of iterations saved when the pre-comparison function was applied, across varying sequence lengths. The x-axis represents the sequence length, while the y-axis shows the difference in iterations needed between cases without and with the pre-comparison function. The results show that, for a sequence length of 3920, the most significant reduction of 104 iterations was achieved. Overall, as the sequence length increased, the number of saved iterations also increased, despite some fluctuations in the data.

Figure 10c presents the acceleration achieved by applying the pre-comparison function across different sequence lengths. Here, the x-axis represents the sequence length, and the y-axis shows the ratio of iterations between cases without and with the pre-comparison function. For a sequence length of 8, a maximum speedup ratio of 1.4996 was observed. As the sequence length increased, the acceleration ratio gradually decreased, eventually stabilizing around 1.06.

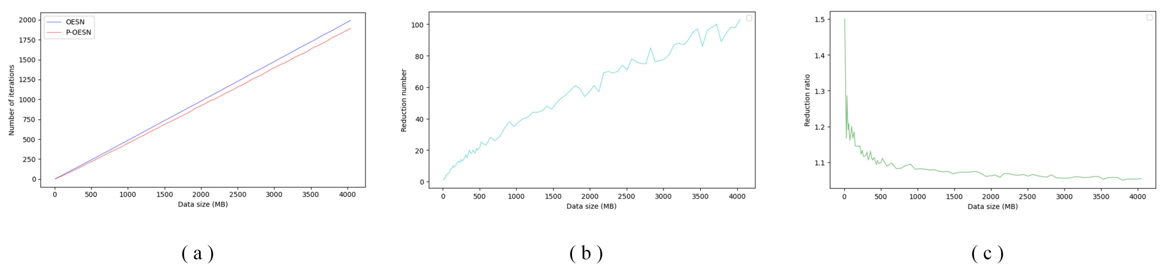

Figure 10. Comparison of the number of iterations before and after the introduction of the pre-comparison layer. (**a**) The number of iterations. (**b**) The saved number of iterations. (**c**) The ratio of the number of iterations between the two cases.

4.2. Pipeline-Based Bidirectional Insertion Sorting Unit Architecture

The BISU architecture in [26] is not a pipelined design, whereas the BISU architecture we propose is a fully pipelined design. The acceleration effect of pipeline technology on the BISU was evaluated by comparing the architecture in [26] with the proposed architecture. It is often found that pipeline technology increases the operating frequency of the design, thereby improving throughput. However, the drawback of this technology is also evident, as more resources are required. The comparison was conducted under different data widths, types, and input sequence lengths. Common sequence lengths of 128, 256, and 512 were included. The two architectures were compared in terms of clock frequency, resource utilization, and throughput. For the comparison of the two architectures, standard data formats were used, specifically, half, single, and double-precision floating-point numbers. Different data formats result in varying resource utilization, which, in turn, affects the operating frequency and throughput. Since the BISU performs consistently across different data contents, only the data formats were considered, without taking the actual data content into account. The results for half, single, and double-precision floating-point numbers are presented in Tables 1, 2, and 3, respectively. In the resource utilization section, columns from left to right indicate the usage of slice LUTs, Slice Registers, and F7 Muxes. Other resources, such as block RAM and DSPs, are excluded from the tables as they were not utilized. Since FPGA resource utilization for the architecture proposed in [26] was not provided, the architecture was implemented on the platform used in this study to ensure consistency in reporting.

Table 1. Overall performance analysis of the pipelined BISU and the naive BISU under half-precision floating-point conditions.

	Ours					[26]				
Length	Frequency	Resource Utilization			Throughput	Frequency	Resource Utilization			Throughput
	(MHz)	LUTs	Reg	F7	(GB/s)	(MHz)	LUTs	Reg	F7	(GB/s)
64	470	10,149	9547	1482	117	221	4595	1640	655	27.625
128	468	20,656	19,506	2528	234	220	8003	3380	1264	55
256	453	41,360	30,795	6912	453	219	15,152	6775	3568	109.5

Table 2. Overall performance analysis of the pipelined BISU and the naive BISU under single-precision floating-point conditions.

	Ours					[26]				
Length	Frequency	Resource Utilization			Throughput	Frequency	Resource Utilization			Throughput
	(MHz)	LUTs	Reg	F7	(GB/s)	(MHz)	LUTs	Reg	F7	(GB/s)
64	416	18,298	18,972	3136	208	202	7306	3257	1760	50.5
128	409	36,886	38,099	5376	409	203	15,081	6563	2848	101.5
256	405	77,804	75,737	13,184	810	200	29,600	13,041	6752	200

Table 3. Overall performance analysis of the pipelined BISU and the naive BISU under double-precision floating-point conditions.

	Ours					[26]				
Length	Frequency	Resource Utilization			Throughput	Frequency	Resource Utilization			Throughput
	(MHz)	LUTs	Reg	F7	(GB/s)	(MHz)	LUTs	Reg	F7	(GB/s)
64	470	10,149	9547	1482	117	221	4595	1640	655	27.625
128	468	20,656	19,506	2528	234	220	8003	3380	1264	55
256	453	41,360	30,795	6912	453	219	15,152	6775	3568	109.5

Our proposed architecture achieves a throughput that is 4.117 times higher than the architecture presented in [26], while using 3.396 times more resources. Specifically, the utilization of slice LUTs, Slice Registers, and F7 Muxes is 5.563 times, 5.633 times, and 1.9921 times that of [26], respectively. The experimental results show that this increase in resource usage leads to a throughput gain of 4.117 times. This yields a resource-to-throughput ratio of 1.212, which demonstrates the efficiency of our proposed architecture in enhancing BISU speed. In our design, BISU is divided into five clock cycles. Operations in the BIU that cannot be completed in a single clock cycle are handled using module replication, where two modules alternately execute during each clock cycle. This module replication significantly increases resource overhead but results in a substantial improvement in operating frequency. The architecture proposed in [26] does not employ such a multi-cycle pipelined structure. Instead, it achieves high operating frequencies primarily through hardware advantage TSMC 90nm. The pipeline architecture in our design enables a clock frequency that is more than twice that of the basic implementation. Furthermore, continuous data intake is allowed by the pipeline structure, with new data received at each clock cycle without any waiting period. In contrast, the basic implementation can only receive new data once the current set is fully processed, allowing data intake only every two clock cycles. This increase in the clock frequency and data reception rate allows the proposed architecture to deliver over four times the throughput of [26]. Although resource usage is increased, the acceleration achieved significantly exceeds the increase in resource requirements.

4.3. Proposed Architecture

The results reported in [24] were compared in detail with the architecture we propose. The same experimental setup as in [24] was used for the comparison. A 32-bit unsigned integer was chosen as the experimental data type. The test data, with lengths of 1K and 1M, were randomly generated using the random module in a Python environment. To ensure the objectivity of the experiments, the same dataset was used to test each architecture. The same implementation platform, Xilinx Vivado 2017.4 on a Xilinx Virtex-7 FPGA XC7VX485T FFG1157-2 [33], was utilized. In the experiment, P represents data parallelism, which refers to the number of data items input to the sorting unit at one time. The input sequence lengths, L, were set to 1K and 1M. Here, 1K refers to 1024 data elements, while 1M refers to $1024 * 1024$ data elements. Total latency is the time required for the architecture to complete the sorting process. For the proposed architecture, multiple configurations were tested. In these configurations, C represents the number of pre-comparison layers, O represents the number of odd–even sorting layers, and B represents the number of BISUs. The corresponding experimental results are presented in Table 4.

Table 4. Comparison of the implementation results of various sorting networks.

Architecture	P	L	LUT	Register	Frequency (MHz)	Total Latency
[34]	1	1K	50,176	67,582	369	5.54 µs
[35]	1	1K	59,160	32,768	0.5	4.09 ms
[36]	1	1K	50,175	65,536	364	5.69 µs
[24]	2	1K	99,376	101,442	383	4.01 µs
[24]	4	1K	197,921	166,788	358	3.58 µs
[24]	8	1K	-	-	-	-
[24]	16	1K	-	-	-	-
Proposed ($C = 0$ $O = 3$ $B = 69$)	3	1K 1M	73,283	51,867	490	5.28 µs 5.47 s
Proposed ($C = 0$ $O = 3$ $B = 207$)	3	1M	219,287	154,953	410	2.17 s
Proposed ($C = 0$ $O = 5$ $B = 41$)	5	1K 1M	57,642	50,321	488	5.25 µs 5.44 s
Proposed ($C = 0$ $O = 5$ $B = 159$)	5	1M	219,814	190,809	406	1.68 s
Proposed ($C = 0$ $O = 9$ $B = 23$)	9	1K 1M	64,733	51,632	488	5.24 µs 5.44 s
Proposed ($C = 0$ $O = 9$ $B = 82$)	9	1M	221,799	175,308	407	1.66 s
Proposed ($C = 0$ $O = 17$ $B = 13$)	17	1K 1M	81,407	62,959	488	5.25 µs 5.07 s
Proposed ($C = 0$ $O = 17$ $B = 39$)	17	1M	217,283	164,125	411	1.55 s
Proposed ($C = 1$ $O = 6$ $B = 20$)	33	1M	216,704	160,698	403	1.50 s
Proposed ($C = 1$ $O = 12$ $B = 11$)	65	1M	243,149	198,895	392	1.48 s

The experimental results show that the total latency of the proposed architecture is 5.24 µs under optimal condition and 5.28 µs under the worst condition when L = 1K. Compared to [34–36], the total latency of our proposed architecture increased by 0.26 µs, 4088 µs, and 0.41 µs, respectively. In contrast, compared to [24], the total latency of our proposed architecture decreased by 1.27 µs and 1.7 µs, respectively. In terms of resource usage, at P = 2, the LUT and Register usage of our proposed architecture is 73.74% and 51.12%, respectively, of that in [24]. At P = 4, the LUT and Register usage is 29.12% and 30.17%, respectively, of that in [24]. As P increases, the resource utilization of our architecture becomes increasingly lower compared to [24]. These results demonstrate that the scalable hybrid sorting network, composed of the P-OESN and BISN, can efficiently reduce resource utilization while largely maintaining performance. Although the total latency of our proposed architecture is slightly higher, it can support a variety of input data lengths. For instance, when L = 8 and L = 16, the resources used in [24] exceed the available limits and cannot be implemented. However, our proposed architecture remains feasible at

$P = 3, 5, 9, 17$, and 33. Compared to other sorting networks, not only can we sort 1K data items, but we can also handle 1M data items with a relatively low processing time.

5. Conclusions

The proposed scalable hybrid sorting network, composed of the P-OESN and BISN, demonstrates significant improvements in both efficiency and scalability. The experimental results show that resource utilization is substantially reduced while maintaining throughput. The architecture's ability to handle large input sizes, such as 1K and 1M, highlights its adaptability to a wide range of applications. The integration of pre-comparison mechanisms and iterative processing enables substantial reductions in resource consumption without sacrificing performance. Through detailed experimentation, the proposed BISU architecture achieves up to 4.117 times the throughput of existing designs, while utilizing only 3.396 times the resources. The proposed design outperforms several existing methods in terms of throughput and supports larger datasets, making it suitable for high-speed, large-scale sorting applications. Its ability to sustain high clock frequencies and efficiently reuse circuit components underscores the practical advantages of the hybrid sorting network. The scalability of this architecture ensures its feasibility for large-scale data applications, where traditional designs may fall short. This hybrid sorting network not only enhances performance but also addresses real-world constraints such as resource limitations and varying input data lengths. Its successful implementation on FPGA platforms demonstrates its practical feasibility, and the performance improvements make it a strong candidate for high-speed data processing applications, including radar navigation, communication systems, and video sequences. The scalable hybrid sorting network developed in this study represents an advancement in hardware-based sorting technologies. By efficiently balancing resource usage and throughput, it lays the groundwork for further optimization and customization in sequential data mining tasks. Future work could focus on extending this architecture to handle even larger datasets and more complex sorting requirements.

Author Contributions: Methodology, X.L.; software, X.L.; validation, X.L. and L.Z.; formal analysis, X.L.; investigation, X.L. and L.Z.; resources, X.L. and Y.Z.; data curation, X.L.; writing—original draft preparation, X.L.; writing—review and editing, L.Z.; supervision, Y.Z.; project administration, X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Belcastro, L.; Marozzo, F.; Talia, D.; Trunfio, P. ParSoDA: High-level parallel programming for social data mining. *Soc. Netw. Anal. Min.* **2019**, *9*, 4.
2. Ma, L.C.; Hsu, P.P. Prediction of web browsing behavior based on sequential data mining. *Int. J. Electron. Commer. Stud.* **2022**, *13*, 1–20.
3. Fournier-Viger, P.; Lin, J.C.W.; Kiran, R.U.; Koh, Y.S.; Thomas, R. A survey of sequential pattern mining. *Data Sci. Pattern Recognit.* **2017**, *1*, 54–77.
4. Korytkowski, M.; Nowak, J.; Nowicki, R.; Milkowska, K.; Scherer, M.; Goetzen, P. Sequential Data Mining of Network Traffic in URL Logs. In Proceedings of the Artificial Intelligence and Soft Computing: 18th International Conference, ICAISC 2019, Zakopane, Poland, 16–20 June 2019; Proceedings, Part I 18; Springer: Berlin/Heidelberg, Germany, 2019; pp. 125–130.
5. Jian, L.; Wang, C.; Liu, Y.; Liang, S.; Yi, W.; Shi, Y. Parallel data mining techniques on graphics processing unit with compute unified device architecture (CUDA). *J. Supercomput.* **2013**, *64*, 942–967.

6. Capraro, G.; Farina, A.; Griffiths, H.; Wicks, M. Knowledge-based radar signal and data processing: A tutorial review. *IEEE Signal Process. Mag.* **2006**, *23*, 18–29. https://doi.org/10.1109/MSP.2006.1593334.
7. Shaban, A. Fundamentals of Satellite Remote Sensing. In *Satellite Monitoring of Water Resources in the Middle East*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 1–14.
8. Rashidi, P.; Krishnan, N.; Cook, D.J. Chapter 19 - Discovering and Tracking Patterns of Interest in Security Sensor Streams. In *Handbook on Securing Cyber-Physical Critical Infrastructure*; Das, S.K., Kant, K., Zhang, N., Eds.; Morgan Kaufmann: Boston, MA, USA, 2012; pp. 481–504. https://doi.org/10.1016/B978-0-12-415815-3.00019-4.
9. Al-Kaff, A.; Martin, D.; Garcia, F.; de la Escalera, A.; Armingol, J.M. Survey of computer vision algorithms and applications for unmanned aerial vehicles. *Expert Syst. Appl.* **2018**, *92*, 447–463.
10. Dogan, A.; Birant, D. Machine learning and data mining in manufacturing. *Expert Syst. Appl.* **2021**, *166*, 114060. https://doi.org/10.1016/j.eswa.2020.114060.
11. Ramos, S.; Soares, J.; Forouzandeh, Z.; Vale, Z. Data Mining Techniques Applied to Power Systems. In *Intelligent Data Mining and Analysis in Power and Energy Systems: Models and Applications for Smarter Efficient Power Systems*; IEEE: New York, NY, USA, 2023; pp. 69–103. https://doi.org/10.1002/9781119834052.ch4.
12. chung Fu, T. A review on time series data mining. *Eng. Appl. Artif. Intell.* **2011**, *24*, 164–181. https://doi.org/10.1016/j.engappai.2010.09.007.
13. Papaphilippou, P.; Luk, W.; Brooks, C. FLiMS: A Fast Lightweight 2-way Merger for Sorting. *IEEE Trans. Comput.* **2022**, *71*, 3215–3226. https://doi.org/10.1109/TC.2022.3146509.
14. Norollah, A.; Derafshi, D.; Beitollahi, H.; Fazeli, M. RTHS: A Low-Cost High-Performance Real-Time Hardware Sorter, Using a Multidimensional Sorting Algorithm. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1601–1613. https://doi.org/10.1109/TVLSI.2019.2912554.
15. Tan, L.; Wang, Y.; Yi, J.; Yang, F. Single-Instruction-Multiple-Data Instruction-Set-Based Heat Ranking Optimization for Massive Network Flow. *Electronics* **2023**, *12*, 5026. https://doi.org/10.3390/electronics12245026.
16. Chen, R.; Siriyal, S.; Prasanna, V. Energy and memory efficient mapping of bitonic sorting on FPGA. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; pp. 240–249.
17. Skliarova, I. A Survey of Network-Based Hardware Accelerators. *Electronics* **2022**, *11*. https://doi.org/10.3390/electronics11071029.
18. Hematian, A.; Chuprat, S.; Manaf, A.A.; Parsazadeh, N. Zero-delay FPGA-based odd-even sorting network. In Proceedings of the 2013 IEEE Symposium on Computers & Informatics (ISCI), Langkawi, Malaysia, 7–9 April 2013; pp. 128–131.
19. Ortiz, J.; Andrews, D. A configurable high-throughput linear sorter system. In Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), Atlanta, GA, USA, 19–23 April 2010; pp. 1–8. https://doi.org/10.1109/IPDPSW.2010.5470730.
20. Perez-Andrade, R.; Cumplido, R.; Del Campo, F.M.; Feregrino-Uribe, C. A Versatile Linear Insertion Sorter Based on a FIFO Scheme. In Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI, Montpellier, France, 7–9 April 2008; pp. 357–362. https://doi.org/10.1109/ISVLSI.2008.14.
21. Marszałek, Z. Parallelization of Modified Merge Sort Algorithm. *Symmetry* **2017**, *9*, 176. https://doi.org/10.3390/sym9090176.
22. Kent, R.B.; Pattichis, M.S. Design of high-speed multiway merge sorting networks using fast single-stage N-sorters and N-filters. *IEEE Access* **2022**, *10*, 77980–77992.
23. Oh, H.W.; Park, J.; Lee, S.E. DL-Sort: A Hybrid Approach to Scalable Hardware-Accelerated Fully-Streaming Sorting. *IEEE Trans. Circuits Syst. II: Express Briefs* **2024**, *71*, 2549–2553. https://doi.org/10.1109/TCSII.2024.3377255.
24. Chen, W.; Li, W.; Yu, F. A Hybrid Pipelined Architecture for High Performance Top-K Sorting on FPGA. *IEEE Trans. Circuits Syst. II: Express Briefs* **2020**, *67*, 1449–1453. https://doi.org/10.1109/TCSII.2019.2938892.
25. Chen, W.T.; Chen, R.D.; Chen, P.Y.; Hsiao, Y.C. A High-Performance Bidirectional Architecture for the Quasi-Comparison-Free Sorting Algorithm. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **2021**, *68*, 1493–1506. https://doi.org/10.1109/TCSI.2020.3048955.
26. Chen, Y.R.; Ho, C.C.; Chen, W.T.; Chen, P.Y. A Low-Cost Pipelined Architecture Based on a Hybrid Sorting Algorithm. *IEEE Trans. Circuits Syst. I: Regul. Pap.* **2024**, *71*, 717–730. https://doi.org/10.1109/TCSI.2023.3342929.
27. Papaphilippou, P.; Brooks, C.; Luk, W. An Adaptable High-Throughput FPGA Merge Sorter for Accelerating Database Analytics. In Proceedings of the 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, 31 August–4 September 2020; pp. 65–72. https://doi.org/10.1109/FPL50879.2020.00021.
28. Batcher, K.E. Sorting networks and their applications. In Proceedings of the Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April–2 May 1968; AFIPS ’68; Spring: Berlin/Heidelberg, Germany, 1968; pp. 307–314. https://doi.org/10.1145/1468075.1468121.

29. Reddy, B.N.K.; Sarangam, K.; Dandeliya, S.; Naidu, S.P.S.; Kumar, N. Accelerating Sorting Performance on FPGA: Combining Quick Sort and Heap Sort through Hybrid Pipelining. In Proceedings of the 2023 IEEE International Symposium on Smart Electronic Systems (iSES), Ahmedabad, India, 18–20 December 2023; pp. 405–408. https://doi.org/10.1109/iSES58672.2023.00090.
30. Yang, Q.; Du, Z.; Zhang, S. Optimized GPU Sorting Algorithms on Special Input Distributions. In Proceedings of the 2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science, Guilin, China, 19–22 October 2012; pp. 57–61. https://doi.org/10.1109/DCABES.2012.57.
31. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*; IEEE Standard for Floating-Point Arithmetic. IEEE: New York, NY, USA, 2019; pp. 1–84. https://doi.org/10.1109/IEEESTD.2019.8766229.
32. xilinx. ZC706 Evaluation Board for the Zynq-7000 XC7Z045 SoC. 2019. Available online: https://docs.amd.com/v/u/en-US/ug954-zc706-eval-board-xc7z045-ap-soc (accessed on 9 August 2019).
33. xilinx. DS180 7 Series FPGAs Data Sheet: Overview. 2019. Available online: https://docs.amd.com/v/u/en-US/ds180_7Series_Overview (accessed on 8 September 2020).
34. Chen, T.; Li, W.; Yu, F.; Xing, Q. Modular Serial Pipelined Sorting Architecture for Continuous Variable-Length Sequences with a Very Simple Control Strategy. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* **2017**, *100-A*, 1074–1078.
35. Dong, S.; Wang, X.; Wang, X. A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA. In Proceedings of the 2009 2nd International Congress on Image and Signal Processing, Tianjin, China, 17–19 October 2009; pp. 1–4. https://doi.org/10.1109/CISP.2009.5302455.
36. Lin, C.S.; Liu, B.D. Design of a pipelined and expandable sorting architecture with simple control scheme. In Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS), Scottsdale, AZ, USA, 26–29 May 2002; Volume 4, pp. IV–IV. https://doi.org/10.1109/ISCAS.2002.1010428.