*electronics*

MDPI

*Article*

# Single-Instruction-Multiple-Data Instruction-Set-Based Heat Ranking Optimization for Massive Network Flow

**Lingling Tan [1,*], Yongyue Wang [2], Junkai Yi [1] and Fei Yang [1]**

[1] Institute of Automation, Beijing Information Science and Technology University, Beijing 100192, China; yijk@bistu.edu.cn (J.Y.); yangfei@bistu.edu.cn (F.Y.)
[2] Jiangsu Shuguang Optoelectric Co., Ltd., Yangzhou 225100, China; buaawyy@buaa.edu.cn
* Correspondence: tanlingling@bistu.edu.cn

**Abstract:** In order to cope with the massive scale of traffic and reduce the memory overhead of traffic statistics, the traffic statistics method based on the Sketch algorithm has become a research hotspot for traffic statistics. This paper studies the problem of the top-k flow statistics based on the Sketch algorithm and proposes a method to estimate the flow heat from massive network traffic using the Sketch algorithm and identify the *k*th flow with the highest heat by using a bitonic sort algorithm. In view of the performance difficulties of applying multiple hash functions in the implementation of the Sketch algorithm, the Single-Instruction-Multiple-Data (SIMD) instruction set is adopted to improve the performance of the Sketch algorithm so that SIMD instructions can process multiple fragments of data in a single step, implement multiple hash operations at the same time, compare and sort multiple flow tables at the same time. Thus, the throughput of the execution task is improved. Firstly, the elements of data flow are described and stored in the form of vectors, while the construction, analysis, and operation of data vectors are realized by SIMD instructions. Secondly, the multi-hash operation is simplified into a single vector operation, which reduces the CPU computing resource consumption of the Sketch algorithm. At the same time, the SIMD instruction set is used to optimize the parallel comparison operation of the flow table in a bitonic sort algorithm. Finally, the SIMD instruction set is used to optimize the functions in the Sketch algorithm and top-k sorting algorithm program, and the optimized code is tested and analyzed. The experimental results show that the time consumed by the advanced vector extensions (AVX)-instructions-optimized version has a significant reduction compared to the original version. When the length of KEY is 96 bytes, the instructions consumed by multiple hash functions account for less in the entire Sketch algorithm, and the time consumed by the optimized version of AVX is about 67.2% of that in the original version. As the length of KEY gradually increases to 256 bytes, the time consumed by the optimized version of AVX decreases to 53.8% of the original version. The simulation results show that the AVX optimization algorithm is effective in improving the measurement efficiency of network flow.

**Keywords:** SIMD instruction set; AVX instruction set; Sketch algorithm; top-k flow; flow heat ranking

check for updates

**Citation:** 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

Academic Editor: Myung-Sup Kim

Received: 14 November 2023
Revised: 10 December 2023
Accepted: 14 December 2023
Published: 16 December 2023

**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

Network measurement is an important basis for the development of a Software Defined Network (SDN) [1]. Network condition monitoring [2], network fault analysis [3], network security defense [4], and even network intelligence all depend on network measurement [5]. Network traffic detection is the most basic and important part of network measurement. Network traffic detection mainly collects network data flow continuously and analyzes network traffic. For example, once a malicious attacker takes advantage of some vulnerabilities in the system to obtain sensitive data in the network, traffic transmission is bound to occur. Traffic detection can extract the required target information from the malicious traffic so as to analyze the network behavior and take corresponding measures for network security defense [6]. Network traffic is characterized by few attributes, large

amounts of data, and large human influence factors, without typical behavior characteristics. Numerous studies on network traffic show that massive network traffic data are power-law distributed [7]. Because the flow distribution in the network is not balanced, most of the flow is very small, which is called cold flow. A small part of the flows is very large, which is called hot flow. However, the small amount of top-k heat flows with the biggest traffic has a greater impact on the network. For example, in data center networks, some applications require a large amount of data transmission, such as data mining and machine learning. Such hot flows may account for only 10% of the total flow, but they account for 90% of the total traffic [8]. Focusing full traffic analysis to top-k traffic analysis makes it possible to analyze massive network traffic in real time. For example, real-time monitoring and understanding of the top-k service providers in network traffic helps operators to understand the network status in real time and facilitate network management. J. Li et al. proposed a new Sketch algorithm, WavingSketch, which can be applied to four applications: finding top-k frequent items, finding top-k heavy changes, finding top-k persistent items, and finding top-k Super-Spreaders [9]. Alawadi A H et al. proposed a new stochastic performance evaluation model to detect and mark top-k heat flows, and then optimize these flows using dynamic load balancing, equal-cost multi-path routing (ECMP), and other traffic engineering technologies, which can improve throughput and flow completion time (FCT) [10]. L. Tang et al. presented the MV-Sketch algorithm, a fast and compact invertible Sketch algorithm that supports heavy flow detection with small and static memory allocation [11].

To rank the popularity of massive network traffic with the Sketch algorithm, it is often necessary to perform multiple hash operations on KEY with long length and sort large-scale flow tables [12], which requires a large amount of computation and takes a long time, because KEY needs to contain most of the flow identification at the link layer, network layer, and transport layer to adapt to various protocol types of flows and distinguish each specific flow. For a flow, according to its protocol type, multiple tuples in the packet are selected to construct the KEY of a hash function, and each tuple is filled into the corresponding position of KEY. KEY needs at least five tuples; the other tuples, such as the source MAC address and destination MAC address, can also be added to expand the length of KEY. The graphics processor (GPU), due to its powerful computing power and unique advantages in matrix operations, has undertaken most of the computing tasks of the entire computing process in many studies [13]. However, due to the very limited memory capacity of the GPU, it is usually more than a few levels worse than the CPU. As a result, the scale of network models that the GPU can support is largely limited. The CPU has, on the contrary, a generally much larger memory capacity than the GPU, but the computing power is far lower than the GPU [14], so it is neglected in large-scale long data processing, and its computing power is not fully excavated.

The Single-Instruction-Multiple-Data (SIMD) is an extension of the CPU's basic instruction set, which can use one instruction to operate on multiple data, and simultaneously perform the same operation on each data point in a set of data vectors [15]. Thus, the technology of parallel processing in space is realized. Current microprocessors generally provide SIMD extension components, and the supported vector length is increasing. For example, Intel Corporation introduced a new vector-processing-oriented advanced vector extensions (AVX) series instruction set on the CPU, which adopts the SIMD mode. Jakobs et al. used Message Passing Interface (MPI) together with the AVX instruction set to obtain optimization of application programs [16]. Shafqat Khan et al. proposed an efficient sub-word parallelism (SWP)-enabled reduced instruction-set computer (RISC) architecture based on Streaming SIMD extensions (SSE) [17]. The SIMD instruction set of the processor includes SSE, AVX, AVX2, and AVX512, with different versions of the instruction set register varying in length and number. Al Hasib et al. investigated the effects on performance and energy using a data reuse methodology combined with parallelization and vectorization in multi- and many-core processors. In this reference, a full-search motion estimation kernel was evaluated on an Intel® (Santa Clara, CA, USA) Xeon Phi™

many-core processor with SSE and the AVX instruction set; the test results revealed that data-level parallel architectures was the design choice in many energy-efficient computing systems [18]. The instruction set can be used to process massive data hashing, comparison, and sorting operations so that the CPU as the effective computing power for flow heat ranking becomes possible. At the same time, because the memory capacity of the CPU is often much larger than that of the video memory of GPU, it is more suitable for Sketch algorithms and flow tables that need to occupy a lot of memory. Therefore, based on the AVX instruction set, it is necessary to optimize the hash calculation and flow table sorting of the Sketch algorithm and to develop the computational role and advantages of the CPU in large-scale data processing.

This paper takes network data packets as the object, counts network data flows based on the Sketch algorithm, and sorts top-k hot flows based on flow tables. This method can screen and sort the hot flow from massive amounts of traffic. In order to improve the performance of hash calculation and flow table sorting of the Sketch algorithm, the AVX instruction set is used to optimize the multi-data processing during the implementation of the Sketch algorithm and flow table, and a large number of serial processing is converted to the parallel processing of a single instruction stream and multi-data. Compared with the traditional single instruction stream single-data processing, the hash calculation and flow table sorting performance of the Sketch algorithm are greatly improved. The rest of this article is organized as follows. The second section introduces the mathematical principle of the Sketch algorithm and bitonic sort algorithm [19], and the basic idea of using the AVX instruction set to optimize the top-k flow heat ranking of network traffic. The third section realizes the construction and operation of a vector based on the AVX instruction set, studies the method of replacing traditional operation instructions with AVX instructions, and completes the optimization of the Sketch algorithm and bitonic sort algorithm. In the fourth section, experimental tests are conducted to verify the improvement of the AVX instructions on the performance optimization of the Sketch algorithm and bitonic sort algorithm.

## 2. Top-k Flow Heat Ranking Algorithm Based on AVX

### *2.1. Data Structure of the Sketch Algorithm*

There are a variety of packets in the network. If each packet is allocated a counter to store, although the measurement is accurate, the storage of all data flows and the need to search all the historical data flows will require a lot of memory space because of the huge amount of data in flow mining. Therefore, based on the hash value, the required storage space is determined according to the range of hash values [20]. Various packages are reclassified according to the hash value, and the data stream is compressed and stored in a smaller space, which can greatly reduce the storage space [21]. The hash function is defined as follows: given two data sets $X$ and $Y$, $R$ is a mapping relationship on the data sets $X$ and $Y$, and the functional relationship between them is shown below [22].

$$f_R : x \xrightarrow{R} y(\forall x \in X, y \in Y) \tag{1}$$

The method that uses hash to estimate the flow is called a Sketch-algorithm-based method. The Sketch algorithm has a hash based data structure, and the hash function is used to map the data flow to two-dimensional space to generate a summary information of the data flow to reduce the space overhead [23]. The commonly used string hash functions include BKDRHash, APHash, DJBHash, etc. Reference [24] presents a comparison of the performance of different hash functions. Increasing the range of hash functions or the number of hash functions can improve the accuracy of the Sketch algorithm, but also increases the complexity of the operations [25].

At present, with the increasing network traffic, the Sketch algorithm is widely used in data flow mining algorithms, and the demand for Sketch algorithm performance is also increasing. Among various Sketch algorithm structures, Count-min Sketch [26] proposed in 2004 has better performance. In recent years, most data flow mining algorithms based

on the Sketch algorithm have been improved based on Count-min Sketch. It randomly projects high-dimension traffic data to low-dimension space, which is a typical compression mapping. The Count-Min Sketch algorithm is composed of a two-dimensional array with a width of $w$ and a depth of $d$. Each element of the array corresponds to a counter, that is, counter [1,1], . . ., counter [$d$, $w$]. Each counter value of the array is initialized to 0. The Count-Min Sketch algorithm uses hash mapping rules to map elements in a data stream to the corresponding counter in each row for counting. The Count-min Sketch algorithm has the characteristics of fast update and query speed, high-speed arrival of data flow, and fast processing [27,28]. Figure 1 shows the summary method of the Sketch data stream processing method of the Count-min Sketch algorithm. The specific algorithm process is as follows.

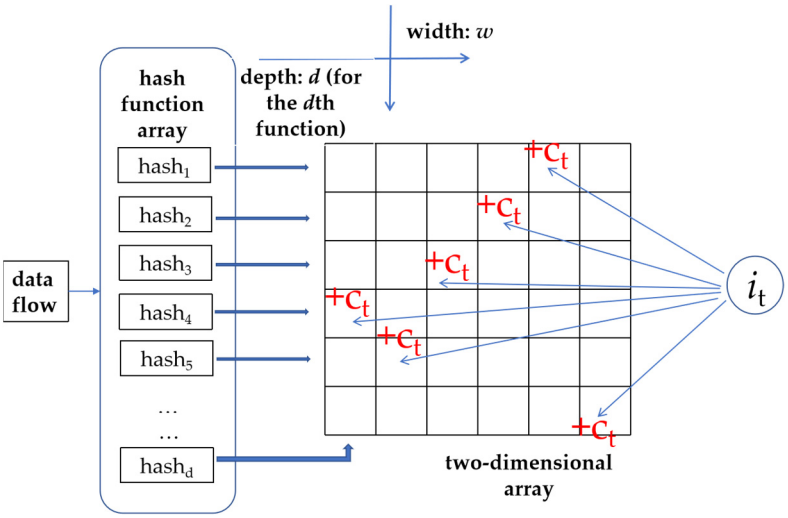


**Figure 1.** Summary method of the Sketch data stream.

The main actions of the Sketch algorithm are as follows:

(1) Hash mapping, which means taking $d$ pairwise independent hash functions $\text{hash}_1$, $\text{hash}_2$, . . ., $\text{hash}_d$ and constructing an array of $d$ rows of counters; each row uses a hash function to map the newly arrived element to a counter in that row to generate the Sketch's summary of that element.

(2) Update the two-dimension array. The input can be regarded as the data that arrives one after another. When the new elements ($j_t$, $c_t$) arrive at time $t$, the table entries are updated, and all the data streams are updated through this table to obtain the final two-dimensional array. The updating process is as follows. For each row in the two-dimensional array, $\text{hash}_i$ ($j$) is used to calculate the index of element $j$ in row $i$, and the counter value of the map is added to $c$. The formula is as follows.

$$counter(i, hash_i(j)) = counter(i, hash_i(j)) + c \quad (2)$$

(3) Query the result; the formula for the value corresponding to the query element $j$ is as follows:

$$Q(j) = \min(counter(i, hash_i(j))) \quad i = 1, 2, \cdots, depth \quad (3)$$

Different elements may conflict if they are mapped to the same counter through hash function calculation, so multiple hash functions are set up and the minimum hash value is taken to improve the accuracy of the measurement results. Multiple hash functions are set up and take the smallest hash value to improve the accuracy of the measurement results so as to avoid conflicts that may occur when different elements are mapped to the same counter after the hash function calculation. Therefore, the count of the row with the smallest query result is selected as the approximate result.

### *2.2. Top-k Algorithm Based on Bitonic Sort*

The top-k flow statistics method plays an important role in network statistics. In the application of the Sketch algorithm, there has been much research on top-k flow statistics, such as the Elastic Sketch algorithm [29] and MV-Sketch algorithm [30]. Their basic idea is to maintain an additional data structure to count top-k flow. One of the problems involved is the replacement strategy. Different data structures and different replacement strategies will produce different computational overhead and storage overhead, and will also affect the statistical accuracy of top-k flows. A reasonable data structure is conducive to better implement the replacement strategy and facilitate the sorting of top-k flows. Therefore, this paper proposes a flow table based on a bitonic sequence to count the top-$k$ flow based on the Sketch algorithm.

Based on the Sketch algorithm, this paper adds a heat ranking flow table to track the top-k flows with the highest heat and the $L$ flows with the lowest heat by bitonic sort. Each flow in the flow table is arranged in a manner that the heat increases monotonically and then decreases monotonically. The specific implementation method is as follows. When the heat ranking function is turned on, after each message arrives, the heat of the current flow is first estimated by the Sketch algorithm, and then processed by the flow table after the heat reaches a certain threshold. The current flow is compared with the lowest heat flow in the flow table. If the heat of the current flow is higher, whether the current flow already exists in the flow table is checked. If so, the current flow count in the flow table is updated; otherwise, the lowest heat flow in the flow table is replaced with the current flow. In order to ensure the monotonicity of the bitonic sequence, when the count of a flow in the flow table is updated, the heat of this flow with the neighbor flow is compared. If the heat ranking changes, the position of this flow and the neighbor flow is exchanged. After replacing the lowest heat flow, it is necessary to insert the replaced new flow at the appropriate position to ensure that the flow table is arranged in a monotonically increasing and then monotonically decreasing manner. The lowest heat flow in the flow table is distributed on both sides of the flow table sequence, and the highest heat flow is concentrated in the middle of the flow table sequence.

Bitonic sorting is a sorting method based on the Batcher theorem [31]. Any bitonic sequence $A$ with a length of 2 $n$ is divided into $X$ and $Y$ of equal length. The elements in $X$ are compared with the elements in $Y$ one by one in the original order, that is, $a$ [$i$] is compared with $a$ [$i + n$] ($i < n$). The larger one is put into sequence $M$, and the smaller one is put into sequence $N$. Then, the obtained sequences $M$ and $N$ are still bitonic sequences, and any element in the sequence $M$ is not less than that in the sequence $N$. According to this principle, sequences $M$ and $N$ can be obtained by shuffling and comparing from a bitonic sequence with 2 $n$ elements, and then an ordered sequence can be obtained by processing two $n$-order bitonic mergers. Figure 2 shows the bitonic sorting diagram.

The heat ranking flow table is not a bitonic sequence at the beginning. It can be converted into a bitonic sequence through the bitonic merge [32] process. Two adjacent monotone sequences with opposite monotonicity are regarded as a bitonic sequence, and these two adjacent monotone sequences with opposite monotonicity are merged to generate a new bitonic sequence each time. For four adjacent sequences $A$, $B$, $C$, and $D$ with length $n$, sequence $A$ and $C$ are monotonically increasing and sequence $B$ and $D$ are monotonically decreasing. Through bitonic sorting once, sequence $A$ and $B$ can be merged into a monotonically increasing sequence with length 2 $n$ and sequence $C$ and $D$ can be merged into a monotonically decreasing sequence with length 2 $n$, and then the two adjacent sequences with length 2 $n$ are merged into a monotonic sequence with length 4 $n$. In this manner, an out-of-order sequence can be converted into a bitonic sequence. The schematic diagram of constructing a bitonic sequence is shown in Figure 3.

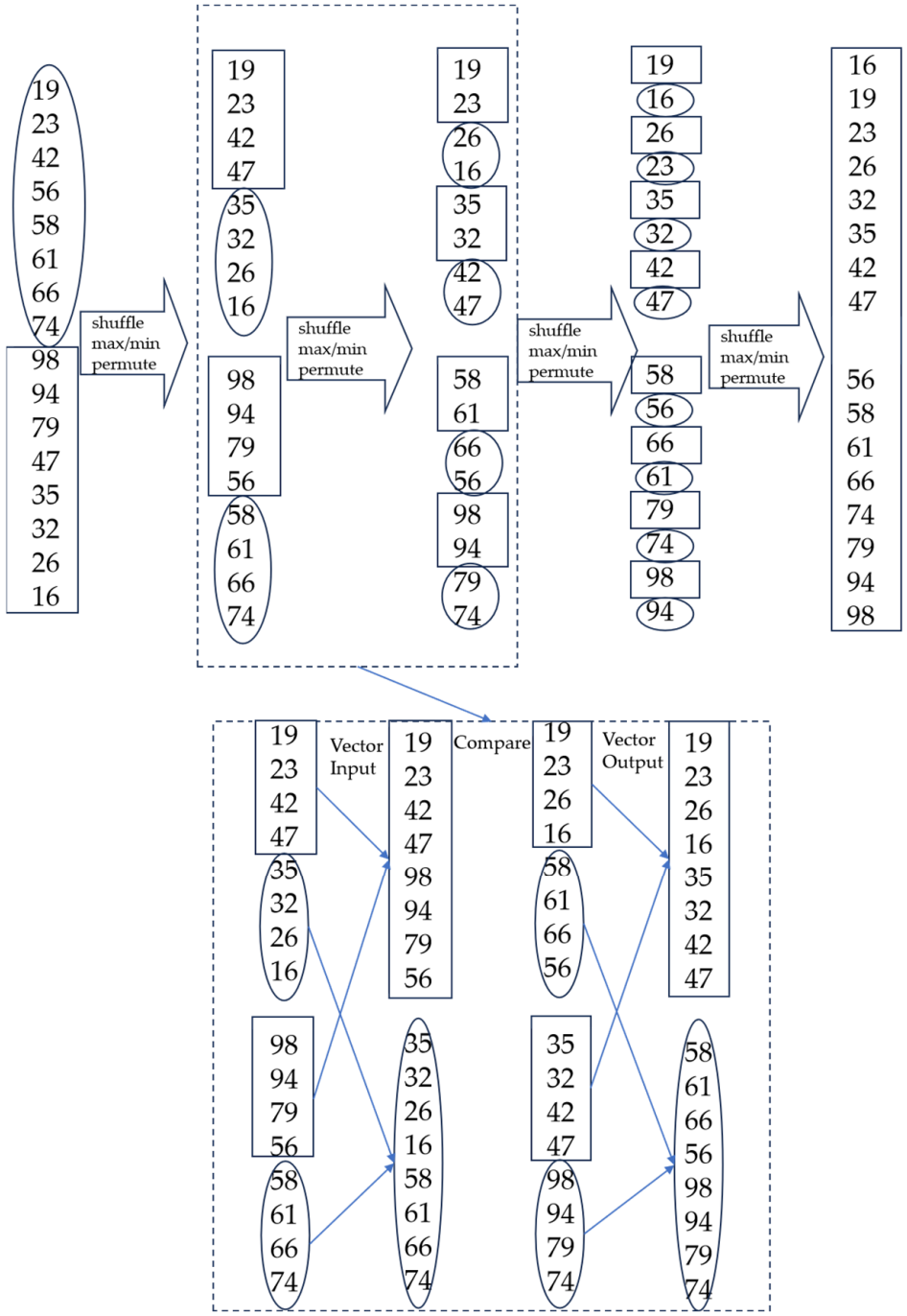


**Figure 2.** Bitonic sorting diagram.

Bitonic sort is a sorting algorithm that can easily implement parallel computing. Through SIMD instructions and parallel operations of multiple cores, the operation performance can be greatly accelerated. However, traditional serial sorting algorithms, such as fast sorting [33], face difficulty in using SIMD instructions to achieve parallel operations. In addition, the bitonic sort algorithm is more suitable for arrays with a length of $2^n$, so the flow table specification and the $k$ value of top-k generally take the power of 2.

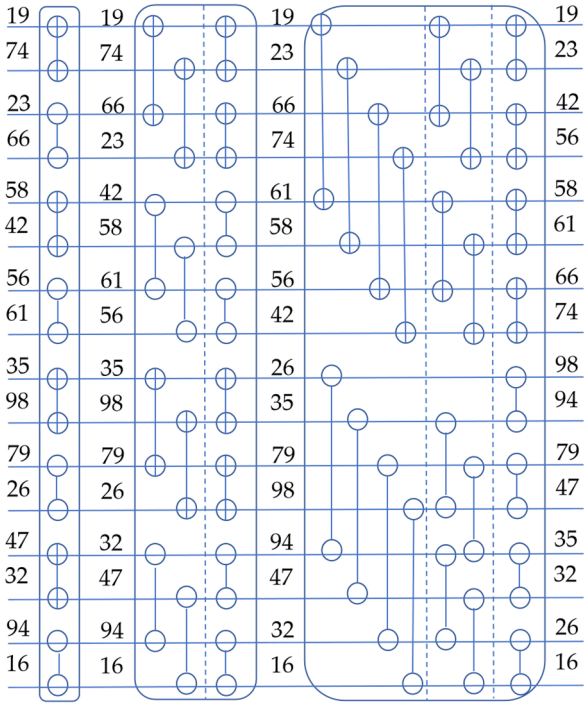


**Figure 3.** Schematic diagram of constructing a bitonic sequence.

### *2.3. Overview of AVX Technology*

The idea of classification is used in the Count-Min Sketch algorithm. Group network flows with the same hash value into one class, and use the same counter to count, which means describing the characteristics of all data with a small amount of data. Although accuracy is sacrificed, the cost of data storage is reduced. However, multiple hash functions are used in the implementation of the Count-min Sketch algorithm, which requires $d \times w$ counters to ensure high query accuracy, resulting in high requirements on memory resources. Compared with the traditional x64 instruction set, SIMD has a significant advantage in the processing speed of data-intensive operations. In order to solve the problem of memory consumption during the implementation of the Sketch algorithm, AVX technology is used to optimize the performance of the Sketch algorithm. The AVX instruction set is designed and implemented based on the idea of SIMD technology, which processes multiple channels of data in memory at the same time in one CPU instruction execution cycle, and processes multiple fragments of data in a single step, which can improve data processing efficiency. The AVX instruction supports 256-bit vector operations and can process eight 32-bit data at a time [34], making the hash operation nearly 4–8 times faster. Compared with traditional instructions, AVX instructions save access time and improve data processing speed, thus improving the throughput of executing tasks.

## 3. Optimization of Flow Heat Ranking Algorithm Based on AVX

### *3.1. Algorithm of Flow Heat Ranking*

The construction and operation of vectors are realized based on the AVX instruction set, the method of replacing traditional operation instructions with AVX instructions is studied, and the optimization of the Sketch algorithm is completed. This method can realize the simultaneous calculation of multiple hash functions and reduce the instruction operation consumption of the Sketch algorithm. It has the characteristics of high reliability, simple implementation, and saving CPU instruction consumption, thus improving the efficiency of the Sketch algorithm and meeting the requirements of top-k hot flow identification and classification in a high-speed network environment. Figure 4 shows the flow chart of Sketch algorithm optimization based on the AVX instruction set.

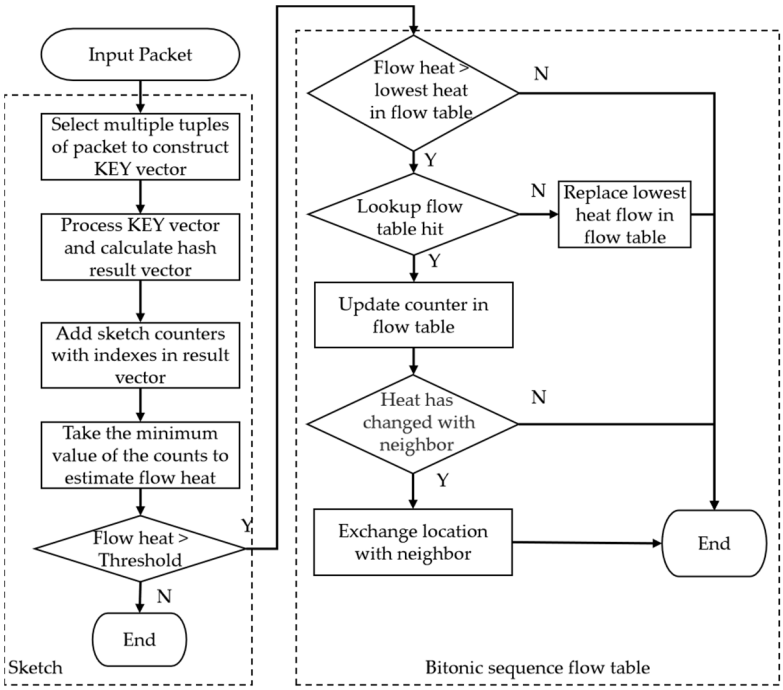


**Figure 4.** Optimization flow chart of the Sketch algorithm based on the AVX instruction set.

The algorithm for flow heat ranking is shown in Algorithm 1.

**Algorithm 1:** Algorithm for flow heat ranking

Input: packet
Output: flow hot rank

```
1.    key = GetKey(packet);
2.    index = hash_v(key);
3.    insert_sketch(index)
4.    flowCnt = query_sketch(index)
5.    if (flowCnt > threshold)
6.      if (flowCnt > flow_table.min)
7.        if (flow_table.find(key) == 1)
8.          flow_table.delete_min;
9.          flow_table.insert_flow(key, flowCnt);
10.       else
11.         flow_table.update_flow(key, flowCnt);
12.         while (flowCnt > flow_table.neighbor_cnt)
13.           flow_table.switch(key);
```

Corresponding with Step 1 to Step 13 of the algorithm for flow heat ranking, we provide a detailed explanation of each step of the procedure below.

1. Obtain multiple tuples from packets to form vector KEY.
2. Index vector of the Sketch algorithm is calculated by the AVX instruction set according to vector KEY.
3. According to the index vector, the indexes of multiple counters are obtained, and the multiple counters corresponding to these indexes are increased.
4. The flow heat is estimated by taking the minimum value of multiple counters according to the Sketch algorithm.
5. If the heat of the flow is bigger than the threshold.

6. If the heat of the flow is bigger than the flow with the smallest counter in the flow table.
7. If the flow is already in the flow table.
8. Then the flow with the smallest counter is deleted in the flow table.
9. The flow table is inserted into the current flow, and inserted into the appropriate position according to the heat of the flow, maintaining bitonic peculiarity.
10. Update the heat of the current flow in the flow table.
11. If the current flow heat is higher than that of the neighbor.
12. In the flow table, the current flow switches positions with the neighbor, maintaining bitonic peculiarity.

### *3.2. Optimize Sketch Algorithm with AVX*

AVX instructions support 256-bit-wide vector operations, and hash function seeds and results can be represented by 64-bit-wide numbers, so a single AVX instruction can implement four hash function operations at the same time. The specific implementation flow of the Sketch algorithm is as follows:

(1) Initialize the hash function and select a set of four hash functions constructed using a random number modding method, which means supposing that the hash function $\text{hash}_i(j)$ of row $i$ is as follows:

$$hash_i(j) = \{[(k1_i \times j) + k2_i] \,\%\text{mod}\, \} \,\%width \quad (i = 1, 2, 3, 4) \tag{4}$$

where $j$ is the data element and $k1$ and $k2$ are randomly generated integers, which are hash seeds and initial values, respectively. The effective bit width of $j$, $k1$, and $k2$ is 32. The multiplication result is represented by 64 bits, and the effective lower 32 bits are taken by the modulus operation, that is, mod is the base of a modulus operation, mod = $2^{32}$; *width* is the width of the two-dimension array. Equation (4) maps each incoming data element $j$ to an integer by $\text{hash}_i(j)$ calculation. Different hash functions are constructed by different initial values $k2_i$ and hash seeds $k1_i$. Multiplication and addition of different hash functions for the same element are accelerated by AVX vector operations.

(2) Construct the vector: according to the initial value of each hash function, use the instruction _mm256_set_epi32 to construct the hash result vector. According to the seeds of each hash function, a hash seed vector is constructed. For the data stream to be processed, use instruction _mm256_set1_epi32 to make four copies and construct the data element vector. Because the effective bit width of each seed, hash result, and data element is 32-bit, the actual bit width of these values in the vector is 64-bit to prevent multiplication transgression. The composition structure of each vector is shown in Figure 5.

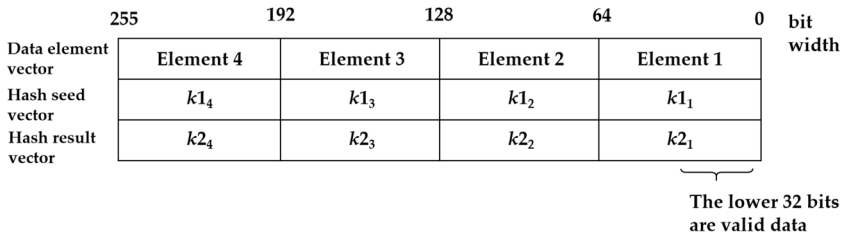


**Figure 5.** Composition structure of the vectors.

(3) Hash operation means using _mm256_mul_epi32 and _mm256_add_epi32 instructions to achieve vector multiplication and addition, using AVX instruction to complete the operation of multiple hash functions, and completing the calculation of four

hash results after traversing all packets, thus improving the performance of the Sketch algorithm.

$$\text{hash\_v} = \_\text{mm256\_mul\_epi32(hash\_v, seed\_v)} \quad (5)$$

$$\text{hash\_v} = \_\text{mm256\_add\_epi32(hash\_v, data\_v)} \quad (6)$$

(4) Sketch algorithm counting means extracting the final results of four hash functions from the hash result vector to obtain four counters by taking the minimum value of the traffic count results of the Count-min Sketch algorithm.

### *3.3. Bitonic Sort Optimizing with AVX*

The AVX instructions support 256-bit-wide vector operations, and the heat of flows in the flow table can be represented by 32-bit-wide numbers, so an AVX instruction can achieve the heat ordering of eight streams at the same time. The specific implementation process of the sorting algorithm is as follows:

(1) Initialize the bitonic sequence. The number of streams supported in the flow table is limited, and the number is usually set to a power of 2, while the number of flows in the real network is massive. At the beginning, the flow table is empty. When a packet arrives, the Sketch algorithm is used to estimate the heat of the flow. When the heat exceeds the threshold, the flow is inserted into the flow table. This is because the heat value returned by the Sketch algorithm will only be estimated to be larger, not smaller. A flow whose heat estimation value by Sketch is greater than the threshold may not be a heat flow, but a flow whose estimation value is less than the threshold is definitely not a heat flow. When the flow table is full or needs to be top-k sorted, the bitonic merge process is converted into a bitonic sequence, ensuring that the flow table is arranged in a monotonically increasing and then monotonically decreasing manner.

(2) Update the bitonic sequence. After the flow table is filled, when a new flow arrives, it needs to be compared with the flow with the lowest heat in the flow table. If the heat of the new flow is lower, there is no need to insert the flow table. If the heat of the flow is higher than the lowest heat of the flow table, whether the new flow has been inserted into the flow table is checked. If the heat value of the flow is higher, it only needs to update the heat value of the flow in the flow table. If not, the flow is replaced with a new flow. After the heat updating or new flow replacement, the heat of the new flow are compared with their neighbor, the appropriate location is found through binary search [35], and the new flow is moved to the location, so as to ensure the monotony of the bitonic sequence.

(3) Order the bitonic sequence. The bitonic sequence with length of 2 $n$ is divided into $X$ and $Y$ with equal length, and a set of data are taken from $X$ and $Y$ for construction vectors. When the length of $X$ and $Y$ is greater than the length of the vector, the _mm256_loadu_ps instruction is used to take continuous data for construction vectors. When the length of $X$ and $Y$ is equal to or less than the length of the vector, the instruction _mm256_shuffle_ps is used for construction vectors by shuffling. The instructions _mm256_max_ps and _mm256_min_ps are used to compare the two vectors; the larger vector is put into the sequence $M$ and the smaller vector is put into the sequence $N$, and the resulting sequences $M$ and $N$ are still bitonic sequences. According to this principle, a bitonic sequence with 2 $n$ elements can firstly be obtained by shuffling and comparing operations to sequences $M$ and $N$. And then the ordered sequence can be obtained by recursively performing bitonic sorting of subsequences.

## 4. Analysis of Experiments and Results

The specific experimental environment of this paper is as following. The Windows 11 Professional Edition operating system, 11th Gen Intel® Core™ i9-11950H @ 2.60GHz processor, 32 GB of RAM, and Visual Studio 2022 development platform are adopted. The objective of the experimental test is to verify the function of the flow heat ranking algorithm

and verify the improvement of the AVX instruction set on the performance optimization of the Sketch algorithm and bitonic sorting algorithm. For this purpose, two versions of the flow heat ranking algorithm are implemented: the original version and the optimized version. The original version is not optimized using the AVX instruction set. The AVX-optimized version means that the AVX instructions are used to complete the calculation of multiple hash functions at one time, and the multiple flow table sorting is completed at one time.

This paper uses several datasets obtained from the MAWI Working Group Traffic Archive [36]. The three data sets were collected from different time periods to collect traffic packets from the carrier backbone network. For each data set, the five tuples of the packet are extracted, including the source IP address, destination IP address, source port number, destination port number, and transport protocol. In addition, the MAC address and Differentiated Services Code Point (DSCP) of the IPv4 packet are extracted to construct a hash KEY of different length.

For all data sets, the two versions of the flow heat ranking algorithm achieve exactly the same function, and the Sketch's counters used for the final ranking result in the two versions being the same. Here is the simulation to compare the performance gap between the two versions.

In the experiment of this paper, in order to verify the optimization effect of the AVX instructions over time, a hash operation is used for the counting and query actions of the Sketch algorithm. For a flow, when the flow counter returned by its Sketch algorithm is greater than the threshold of 10, it will be selected into the flow table processing process; otherwise, it will be regarded as a small flow and no flow table processing is required. The experimental data set information is as following in Table 1.

**Table 1.** Experimental data set information.

| Data Set | Message Number | Flow Number | Flow Number in Flow Table |
|---|---|---|---|
| data set 1 | 4,774,122 | 1,196,845 | 9339 |
| data set 2 | 7,664,905 | 1,691,924 | 14,791 |
| data set 3 | 10,163,203 | 1,996,525 | 18,541 |

For all data sets, the two versions of the flow heat ranking algorithm should achieve exactly the same function; the counter estimated by the Sketch algorithm and the final ranking result should be the same. Both versions are developed based on the CPU, making full use of the CPU's large memory characteristics, so the memory space consumption of the two versions is consistent. The following is a simulation to compare the performance gap between the two versions in CPU instruction execution time.

The original version of flow heat ranking algorithm and the version optimized by the AVX instruction set are used for simulation, and the performance of lookup, flow table construction, and sorting processing in the Sketch algorithm are compared; the simulation results are shown in Table 2. Table 2 shows a running time comparison between the original version and the AVX-optimized version under conditions of different data sets and different KEY lengths.

The following conclusions can be drawn from Table 2. Taking the test results of data set 3 as an example, in terms of hash operation, when the length of KEY is 128 bytes, the running time of sketch decreases from 3,917,605 us in the original version to 2,576,302 us in the optimized version, and the time consumed by the AVX-optimized version is 65.8% of the original version. When the length of KEY is short, the instructions consumed by multiple hash functions account for less in the whole process. For example, when the length of KEY is 96 bytes, the running time is reduced from 3,109,784 us to 2,089,397 us, and the time consumed by the AVX-optimized version is about 67.2% of the original version. As the length of KEY gradually increases, the proportion of instructions consumed by multiple hash functions in the whole Sketch also gradually increases. For example, when the length of KEY is 256 bytes, the running time is reduced from 6,321,628 us to

3,399,399 us, and the time consumed by the AVX-optimized version is reduced to 53.8% of the original version. Figure 6 gives the performance comparison of the original Sketch algorithm with the AVX-optimized Sketch algorithm. As can be seen from the comparison diagram in Figure 6, the longer the length of KEY, the more operation time the original version needs to perform a hash operation in Sketch, and the better the optimization effect of the AVX-optimized version.

**Table 2.** Simulation results.

| Test Set | Key Length (B) | Processing Time (μs) | | Optimize Ratio |
|---|---|---|---|---|
| | | Original | AVX Optimization | |
| Test set 1 | 64 | 1,173,567 | 935,560 | 79.7% |
| | 96 | 1,647,833 | 1,039,376 | 63.1% |
| | 128 | 1,857,836 | 1,156,706 | 62.3% |
| | 256 | 3,257,260 | 1,699,635 | 52.2% |
| Test set 2 | 64 | 1,606,498 | 1,261,235 | 78.5% |
| | 96 | 2,361,363 | 1,733,468 | 73.4% |
| | 128 | 2,664,560 | 2,162,273 | 81.1% |
| | 256 | 4,709,894 | 2,429,931 | 51.6% |
| Test set 3 | 64 | 2,293,531 | 1,837,161 | 80.1% |
| | 96 | 3,109,784 | 2,089,397 | 67.2% |
| | 128 | 3,917,605 | 2,576,302 | 65.8% |
| | 256 | 6,321,628 | 3,399,399 | 53.8% |

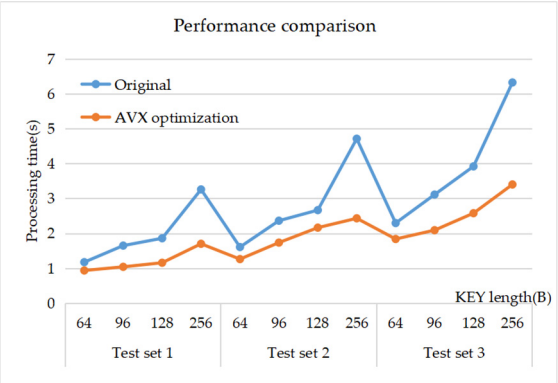


**Figure 6.** Performance comparison of the original Sketch algorithm with the AVX-optimized Sketch algorithm.

In one test, the same data set is read and both versions of the algorithm are applied for heat ranking of a massive network flow. The CPU instruction consumption of major operations in the two versions is calculated in one test, as shown in Table 3. In the heat ranking of a massive network flow algorithm, the basic processing of the data set consumes 18% of the CPU instructions. The original version consumes 51.6% of CPU instructions; the AVX-optimized version consumes 30.4% of the CPU instructions. The management operations of flow tables in the two versions are basically the same, and both of them consume about 7.8% of CPU instructions. The AVX-optimized version mainly reduces the count and access costs of Sketch, which are reduced from 43.8% in the original version to 22.7% in the AVX-optimized version. Figure 7 shows the pie chart of CPU instruction consumption of each module in the original version and AVX-optimized version of the algorithm for heat ranking of massive network flows, which is an intuitive presentation of Table 3.

**Table 3.** CPU instructions consumption for major operations in both versions.

| Process | Operation | Percent of Instruction Consumption | Total |
|---|---|---|---|
| Original | Sketch counts<br>Sketch query<br>Flow table | 19.9%<br>23.9%<br>7.8% | 51.6% |
| AVX-optimized | Sketch counts<br>Sketch query<br>Flow table | 10.4%<br>12.3%<br>7.7% | 30.4% |
| Basic process | Read data set<br>Others | 14.8%<br>3.2% | 18.0% |

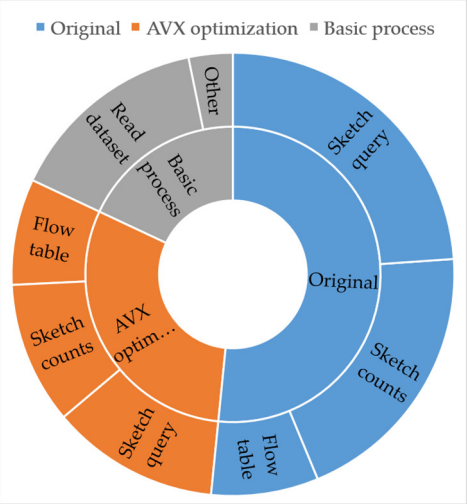


**Figure 7.** Pie chart of CPU instruction consumption of each module in the original version and AVX-optimized version.

## 5. Conclusions and Future Work

In this paper, we propose a method to implement and optimize the Sketch algorithm and bitonic sort algorithm using the AVX instruction set, which is applied to heat ranking for massive network flows. This method takes network data packets as the object, counts network flows based on the Sketch algorithm, and sorts flows based on a flow table to select the top-k hot flows from massive network traffic. In order to improve real-time performance of the algorithm for top-k hot flow ranking, it is necessary to solve the problem that the existing Sketch algorithm needs to perform multiple hash operations on KEY with long length, which consumes a lot of CPU instruction computing resources. This paper uses the AVX instruction set to optimize multi-data processing during the implementation of Sketch and flow table, and converts a large amount of serial processing into parallel processing of single-instruction and multi-data. Using one vector operation to realize multiple hash operations that need to be executed successively and using one vector sorting to realize comparison and exchange operations that need to be executed successively, it has theoretical reliability, reduces the consumption of CPU instructions, greatly improves the computing efficiency, and thus improves the efficiency of network flow measurement. This method is an effective way to improve the overall computing power of the system without increasing the hardware cost.

Based on the real traffic in the data set, this paper verifies the improvement of optimization performance of the Sketch algorithm and bitonic sort algorithm by the AVX instruction set through experimental tests. The experimental results show that compared with the traditional single-instruction single-data processing, the optimization method proposed in this paper can greatly improve the performance of hash calculation and flow table sorting in Sketch. In terms of hash function operation, when the length of KEY is

short, the instructions consumed by multiple hash functions account for less in the whole Sketch algorithm, and the optimization effect of the AVX-optimized version decreases. As the length of KEY gradually increases, the proportion of instructions consumed by multiple hash functions in the whole Sketch also gradually increases, and the optimization effect of the AVX-optimized version will also be more obvious.

In the future research, with increasing deployment of the IPv6 protocol, we plan to further expand KEY's length of hash operation for IPv6 traffic, supporting longer IP addresses, simultaneously supporting IPv4 and other protocol traffic detection. On the other hand, using machine learning to optimize the flow heat ranking algorithm is considered, which means using the network traffic characteristics extracted from a small number of traffic data packets to quickly predict and identify heat flow, so as to improve the speed and accuracy of flow heat ranking. We also intend to consider using the powerful parallel processing capability of the GPU to further improve the parallelism and real-time performance of the heat ranking algorithm for massive network traffic.

**Author Contributions:** Conceptualization, software, and writing—original draft L.T.; validation, Y.W.; methodology, J.Y.; project administration, F.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Data is contained within the article.

**Conflicts of Interest:** Author Yongyue Wang was employed by the company Jiangsu Shuguang Optoelectric. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as potential conflict of interest.

## References

1. Akhunzada, A.; Ahmed, E.; Gani, A.; Khan, M.K.; Imran, M.; Guizani, S. Securing software defined networks: Taxonomy, requirements, and open issues. *IEEE Commun. Mag.* **2015**, *53*, 36–44. [CrossRef]
2. Hosseini, S.; Zade, B.M.H. New hybrid method for attack detection using combination of evolutionary algorithms, SVM, and ANN. *Comput. Netw.* **2020**, *173*, 107168. [CrossRef]
3. Wu, Z.; Lu, K.; Wang, X.; Chi, W. Topology-aware network fault influence domain analysis. *Comput. Electr. Eng.* **2017**, *57*, 266–280. [CrossRef]
4. Kong, D.; Shen, Y.; Chen, X.; Cheng, Q.; Liu, H.; Zhang, D.; Liu, X.; Chen, S.; Wu, C. Combination Attacks and Defenses on SDN Topology Discovery. *IEEE/ACM Trans. Netw.* **2023**, *31*, 904–919. [CrossRef]
5. Wei, W.; Chen, Y.; Lin, Q.; Ji, J.; Wong, K.-C.; Li, J. Multi-objective evolving long—Short term memory networks with attention for network intrusion detection. *Appl. Soft Comput.* **2023**, *139*, 110216. [CrossRef]
6. Qing, W.; Hongju, C. Computer Network Security and Defense Technology Research. In Proceedings of the 2016 Eighth International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Macau, China, 11–12 March 2016; pp. 155–157. [CrossRef]
7. Zhang, H.; Shen, Y.; Thai, M.T. Robustness of power-law networks: Its assessment and optimization. *J. Comb. Optim.* **2016**, *32*, 696–720. [CrossRef]
8. Mogul, J.C.; Tourrilhes, J.; Yalagandula, P.; Sharma, P.; Curtis, A.R.; Banerjee, S. DevoFlow: Cost-effective flow management for high performance enterprise networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Monterey, CA, USA, 20–21 October 2010; pp. 1–6.
9. Li, J.; Li, Z.; Xu, Y.; Jiang, S.; Yang, T.; Cui, B.; Dai, Y.; Zhang, G. WavingSketch: An unbiased and generic sketch for finding top-k items in data streams. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, 6–10 July 2020; pp. 1574–1584.
10. Alawadi, A.H.; Zaher, M.; Molnár, S. Methods for Predicting Behavior of Elephant Flows in Data Center Networks. *Infocommun. J.* **2019**, *6*, 34–41. [CrossRef]
11. Tang, L.; Huang, Q.; Lee, P.P.C. A Fast and Compact Invertible Sketch for Network-Wide Heavy Flow Detection. *IEEE/ACM Trans. Netw.* **2020**, *28*, 2350–2363. [CrossRef]
12. Huang, J.; Zhang, W.; Li, Y.; Li, L.; Li, Z.; Ye, J.; Wang, J. ChainSketch: An efffcient and accurate sketch for heavy flow detection. *IEEE/ACM Trans. Netw.* **2023**, *31*, 738–753. [CrossRef]
13. Pan, Z.; Zhang, F.; Li, H.; Zhang, C.; Du, X.; Deng, D. G-SLIDE: A GPU-Based Sub-Linear Deep Learning Engine via LSH Sparsification. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 3015–3027. [CrossRef]
14. Liu, J.; Li, X.; Hu, F.Q. Performance comparison on parallel CPU and GPU algorithms for two dimensional unified gas-kinetic scheme. *Adv. Appl. Math. Mech.* **2020**, *12*, 1247–1260.

15. Geng, T.; Waeijen, L.; Peemen, M.; Corporaal, H.; He, Y. MacSim: A MAC-Enabled High-Performance Low-Power SIMD Architecture. In Proceedings of the 2016 Euromicro Conference on Digital System Design (DSD), Limassol, Cyprus, 31 August–2 September 2016; pp. 160–167. [CrossRef]
16. Jakobs, T.; Kratzsch, S.; Rünger, G. Analyzing Data Reordering of a combined MPI and AVX execution of a Jacobi Method. In Proceedings of the 2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Naples, Italy, 1–3 March 2023; pp. 159–163. [CrossRef]
17. Khan, S.; Rashid, M.; Javaid, F. A high performance processor architecture for multimedia applications. *Comput. Electr. Eng.* **2018**, *66*, 14–29. [CrossRef]
18. Al Hasib, A.; Natvig, L.; Kjeldsberg, P.G.; Cebrián, J.M. Energy Efficiency Effects of Vectorization in Data Reuse Transformations for Many-Core Processors—A Case Study. *J. Low Power Electron. Appl.* **2017**, *7*, 5. [CrossRef]
19. Mu, Q.; Cui, L.; Song, Y. The implementation and optimization of Bitonic sort algorithm based on CUDA. *Comput. Sci.* **2015**, *40*, 553–556.
20. Zhu, H.; Zhang, Y.; Zhang, L.; He, G.; Liu, L.; Liu, N. SA Sketch: A self-adaption sketch framework for high-speed network: NA. *Concurr. Comput. Pract. Exp.* **2020**, *1*, e5891. [CrossRef]
21. Li, D.; Du, R.; Liu, Z.; Yang, T.; Cui, B. Multi-copy Cuckoo Hashing. In Proceedings of the IEEE 35th International Conference on Data Engineering, Macao, China, 8–11 April 2019; pp. 1226–1237.
22. Yoshioka, M.; Hiraguri, T.; Yoshino, H. Performance evaluation of sketch schemes on traffic anomaly detection accuracy. *IEICE Commun. Express* **2017**, *6*, 399–404. [CrossRef]
23. Yang, T.; Zhang, H.; Wang, H.; Shahzad, M.; Liu, X.; Xin, Q.; Li, X. FID-sketch: An accurate sketch to store frequencies in data streams. *World Wide Web* **2019**, *22*, 2675–2696. [CrossRef]
24. Deng, F.; Yu, Z.; Song, H.; Zhao, R.; Zheng, Q.; Li, Z.; He, H.; Zhang, Y.; Guo, F. An efficient policy evaluation engine with locomotive algorithm. *Clust. Comput.* **2021**, *24*, 1505–1524. [CrossRef]
25. Li, S.; Luo, L.; Guo, D.; Zhang, Q.; Fu, P. A survey of sketches in traffic measurement: Design, optimization, application and implementation. *arXiv* **2020**, arXiv:2012.07214.
26. Cormode, G.; Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. In Proceedings of the 2004 Latin American Symposium on Theoretical Informatics, Buenos Aires, Argentina, 5–8 April 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 29–38.
27. Sisovic, S.; Bakaric, M.B.; Matetic, M. Reducing data stream complexity by applying Count-Min algorithm and discretization procedure. In Proceedings of the IEEE Fourth International Conference on Big Data Computing Service & Applications, Bamberg, Germany, 26–29 March 2018.
28. Rottenstreich, O.; Reviriego, P.; Porat, E.; Muthukrishnan, S. Avoiding Flow Size Overestimation in the Count-Min Sketch with Bloom Filter Constructions. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 3662–3676. [CrossRef]
29. Yang, T.; Jiang, J.; Liu, P.; Huang, Q.; Gong, J.; Zhou, Y.; Miao, R.; Li, X.; Uhlig, S. Adaptive Measurements Using One Elastic Sketch. *IEEE/ACM Trans. Netw.* **2019**, *27*, 2236–2251. [CrossRef]
30. Tang, L.; Huang, Q.; Lee, P.P.C. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 2026–2034. [CrossRef]
31. Zhang, J.; Zhang, W.; Yuan, J.; Wang, H. Implementing bitonic sorting on optical network-on-chip with bus topology. *Photonic Netw. Commun.* **2020**, *39*, 129–134. [CrossRef]
32. Ranković, V.; Kos, A.; Milutinović, V. Bitonic Merge Sort Implementation on the Maxeler Dataflow Supercomputing System. *IPSI BgD Trans. Internet Res.* **2013**, *9*, 5–10.
33. Marszałek, Z. Parallelization of fast sort algorithm. In Proceedings of the Information and Software Technologies: 23rd International Conference, ICIST 2017, Druskininkai, Lithuania, 12–14 October 2017.
34. Amiri, H.; Shahbahrami, A. SIMD programming using Intel vector extensions. *J. Parallel Distrib. Comput.* **2020**, *135*, 83–100. [CrossRef]
35. Nowak, R. Generalized binary search. In Proceedings of the 2008 46th Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, USA, 23–26 September 2008; pp. 568–574. [CrossRef]
36. Cho, K.; Mitsuya, K.; Kato, A. Traffic data repository at the wide project. ser. USENIX 2000 FREENIX Track. USENIX. In Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, CA, USA, 18–23 June 2000.


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.