In-order INT: Efficient Reordering of Out-of-order In-band Network Telemetry Packets via CPU-FPGA Co-design

Zongrui Sui[1,2], Lizhuang Tan[1,2], Huiling Shi[1,2], Wei Zhang[1,2], Peiying Zhang[1,3]
[1]*Key Laboratory of Computing Power Network and Information Security, Ministry of Education*
Shandong Computer Science Center (National Supercomputer Center in Jinan)
Qilu University of Technology (Shandong Academy of Sciences), Ji'nan, China
[2]*Shandong Provincial Key Laboratory of Computing Power Internet and Service Computing*
Shandong Fundamental Research Center for Computer Science, Jinan, China
[3]*Qingdao Institute of Software, College of Computer Science and Technology*
China University of Petroleum (East China), Qingdao, China
Email: zongruisui2002@gmail.com, lzhtan@qlu.edu.cn, shihl@sdas.org, wzhang@sdas.org, zhangpeiying@upc.edu.cn

***Abstract*—With the emergence of Software-Defined Networking (SDN) and data plane programmability, In-band Network Telemetry (INT) has become a novel network measurement approach, enabling end-to-end, hop-by-hop network visibility. However, due to the complexity of network topologies and limitations in telemetry orchestration schemes, telemetry servers often encounter severe out-of-order issues when processing incoming telemetry reports. This results in stale and unreliable data source for upper-layer telemetry applications. This paper proposes a CPU/FPGA co-designed out-of-order reordering system for INT, which combines online and offline sorting techniques to enhance the processing efficiency of telemetry servers. To the best of our knowledge, this is the first study addressing out-of-order reordering in INT. Experimental results show that compared to a CPU-only solution, our approach achieves a 12.5X speedup on the 10GE NIC.**

***Index Terms*—Network Measurement, In-band Network Telemetry, Heterogeneous Computing, Online Reordering, Offline Reordering, Reordering Acceleration**

I. Introduction

As an emerging network measurement technology, In-band Network Telemetry (INT)[1] embeds telemetry instructions and information into business traffic. It no longer uses a separate control/management plane or out-of-band channel, and can achieve end-to-end network status measurement. In recent years, INT-based network routing[2], congestion control[3], and fault detection[4] have significantly improved the efficiency of network control and management.

In-band Network Telemetry (INT) is a flow-level or path-level measurement method. However, when it is applied to network-wide measurement tasks, it faces a severe out-of-order problem. Based on the task orchestration[5][6][7], telemetry data is collected through multiple INT paths, which often partially overlap. Even telemetry packets originating from the same source flow may traverse different INT paths, resulting in inconsistencies between the order of generation and the order of arrival at the collector. If left unprocessed, such disorder can compromise the accuracy of temporal analysis for upper-layer applications. Therefore, reordering telemetry data at the collector side is essential to ensure data integrity and temporal correctness.

With the advancement of heterogeneous computing, CPU and Field Programmable Gate Array (FPGA) based heterogeneous architectures have been widely adopted for accelerating a variety of applications[8][9]. FPGA-based acceleration approaches leverage its highly parallel processing capability and pipelined architecture[10], while also offering lower power consumption[11]. From the perspective of accelerating the reordering process in In-band Network Telemetry (INT), the introduction of FPGA significantly enhances the reordering efficiency of the telemetry server, and releases more CPU computational resources for INT applications.

In this paper, the disordering of In-band Network Telemetry data is discussed, and we propose an INT (In-band Network Telemetry) processing scheme to accelerate the process of reordering, which enables both data and task parallelism. The scheme employs the optimized online reordering architecture, which reduces CPU computational resources, saves comparator resources compared with traditional architecture, and meets the requirements of real-time telemetry applications. On the basis of online reordering, we also propose an offline reordering architecture, which reduces the time required for offline reordering. Our main contributions are:

- We analyze the task orchestration of the In-band Network Telemetry during the telemetry phase and find that the out-of-order issue of INT is ignored. Therefore, a CPU–FPGA collaborative method for reordering In-band

This work was supported in part by the Shandong Provincial Natural Science Foundation under Grant No. ZR2022LZH015, ZR2023QF025, ZR2023LZH017 and ZR2024MF066, the National Research Foundation of Korea under Grant NRF-24109 FY2025, the National Natural Science Foundation of China under Grant 62402257 and 62471493, the China University Research Innovation Fund under Grant No.2023IT207, and the Talent Project of Qilu University of Technology (Shandong Academy of Sciences) under Grant No.2023RCKY141.

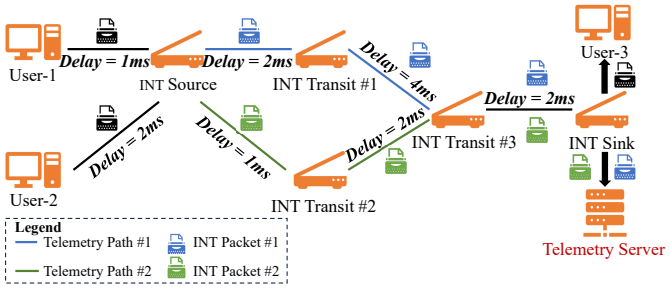

Fig. 1: In-band Network Telemetry

Network Telemetry packets is proposed.

- In this paper, we utilize the FPGA accelerator card to accelerate the reordering of INT packets, which improves the efficiency of reordering and reduces the utilization of CPU resources.
- The proposed method combines online and offline reordering for INT packets, and timely delivers ordered telemetry data to applications, thereby satisfying the real-time requirements of upper-layer telemetry applications.
- For multi-flow reordering, we propose an online reordering architecture based on time-multiplexing, which reduces the comparator resources by half.

The rest of this paper is organized as follows. Section 2 provides background. Section 3 describes the system design of in-order INT for CPU and FPGA architectures. Section 4 describes the details of the implementation of online reordering and offline reordering. The results of the experiment are discussed in Section 5. Finally, section 6 summarizes the total paper.

II. Background

A. In-band Network Telemetry

Different from traditional network measurement, In-band Network Telemetry combines packet forwarding with network measurement, allowing network devices (routers and switches) to insert device status information into packets during the transmission of service packets, which is is used to collect per-packet information in real time from end to end. Consequently, In-band Network Telemetry greatly improves the real-time visibility of the network and facilitates more efficient network diagnostics. In the INT architecture, the controller is responsible for orchestrating telemetry tasks, including selecting which network paths require monitoring, specifying the telemetry metadata to be collected (such as latency, congestion, and packet loss), and defining strategies such as sampling frequency. These tasks are then delivered to the corresponding INT Sources. The INT Source embeds telemetry instructions into data packets. INT Transit nodes insert telemetry data according to the embedded instructions, and the INT Sink receives the final packets carrying the collected telemetry information.

B. Out-of-order Issue of INT

In the following, we analysis the out-of-order issue of INT. Suppose that the In-band Network Telemetry server issues a telemetry command at 0 ms, and a set of telemetry path is formed according to the orchestration of task, that is, Telemetry Path 1 and Telemetry Path 2 are shown in Fig. 1. There is an overlap between Path 1 and Path 2 (INT Source, INT Transit #3, INT Sink). The 1st INT packet arrives at INT Source at 1 ms, and INT Source uploads its device status information with a timestamp of 1 ms. INT packet #2 arrives at INT Source at 2 ms, and INT Source does the same with it. When the two INT packets pass all the network devices on the telemetry path, the network devices do the above on them. Finally, INT packet #1 and INT packet #2 arrive at the INT Sink, which hands over the service packet-related information to User-3 and the In-band Network Telemetry data to the telemetry server. When the In-band Network Telemetry server processes the telemetry data, it will put the status information of different devices into the corresponding time series database based on the device ID number, and INT packet #2 will arrive at INT Sink earlier than INT packet #1 according to the link delay calculation. For the telemetry data of the INT Source of the device, the order of arrival to the telemetry server is [(2 ms, INT metadata 1), (1 ms, INT metadata 2)] (the time series database used is stored in ascending order of timestamps by default), so the INT out-of-order problem occurs, and a system needs to be designed to reorder the telemetry data.

III. Architecture Design of In-order INT

Fig. 2 shows the block diagram of CPU/FPGA collaboration. The architecture of the proposed method is composed of host and FPGA. The host runs host program and telemetry application program. The host program schedules the reordering unit and manages data transfers between CPU and FPGA, and the application program utilizes telemetry data to perform relevant network functions that are depicted in Fig. 2.

The INT packet includes telemetry data from different network devices in multiple telemetry paths. The packet classifier in FPGA splits the coming INT and classifies the telemetry data into corresponding BRAM according to device-id. The multiple online reordering units in FPGA has been deployed, and all of them can run in parallel. Multiple online reordering units can correspond to reorder multiple telemetry data flows received in the In-band Network Telemetry. It should be noted that one online reordering unit is responsible for reordering by timestamp of telemetry data from only one network device.

The total BRAM is partitioned into read BRAM and write BRAM. The read BRAM functions as buffer for telemetry data awaiting reordering, and the online and offline reordering units access the read BRAM to load telemetry data to be sorted. Once reordering is complete, the telemetry data is written in the write BRAM, where it is temporarily buffered before being transferred to DDR4 and eventually returned to the host memory of host.

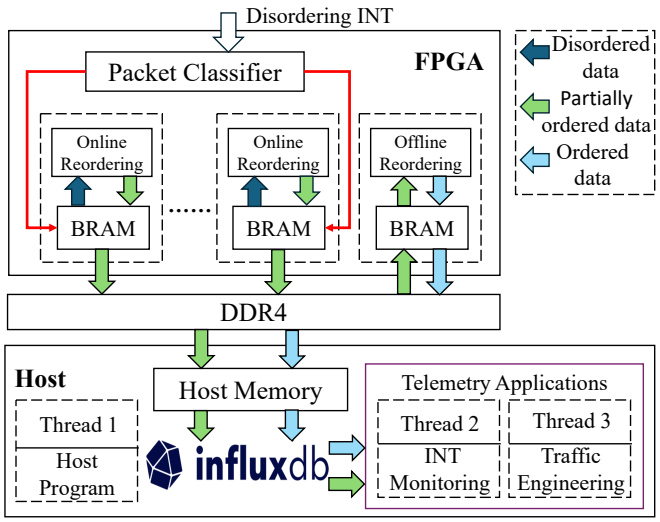

Fig. 2: The block diagram of collaboration

IV. Reordering Method and Implementation Details

A. Online and Offline Reordering

Our reordering work includes online reordering and offline reordering, online reordering refers to reordering during the process of receiving telemetry data packets, the reordering results are uploaded in real time, and offline reordering is reordering after all the data arrive. For upper-layer real-time telemetry applications, excessive latency cannot be tolerated, and real-time monitoring and diagnosis of network performance are required. Traffic engineering is a typical real-time telemetry application that continuously monitors network traffic, identifies bottlenecks and traffic imbalances, and redistributes traffic or adjusts paths accordingly, which demands the telemetry server provides real-time telemetry data for analysis. Thus, the telemetry server needs to perform online reordering of the telemetry data and promptly delivers the reordered data to the upper-layer real-time telemetry applications. And when all the telemetry data arrive, offline reordering is applied to reorder all the telemetry data.

B. Online Reordering Architecture

To address packet reordering in networks, Hoang et al.[12] designed a sorting pipeline based on insertion reordering. However, the limitation of proposed multiple flows implement is that the resource utilization of FPGA is very high. In order to reduce the resource utilization on FPGA, we propose a parallel insertion sequencer architecture based on time-multiplexed comparator, which reduces the comparator by half compared with the traditional parallel insertion reordering architecture.The traditional architecture is N registers corresponding to N comparators. Because INT collects status information from multiple devices in the network, reordering INT involves the multi-flow approach, which allocates the telemetry data from multiple sources to the different sorting

TABLE I: Description of Parameters

Parameter	Description
T_{i}	Total number of telemetry packets in unit i
P_{i}	Insertion position in unit i
E_{ij}	Element in the j-th shift register of unit i
A_{i}	Element to be sorted in arriving packet i
R_{j}	Comparator result of comparing A_{i} with E_{ij}

unit, achieving higher throughput[13].It is important to note that when there are M flows in the INT, M reordering units are required, and the total number of comparators required C_{original} is

$$C_{\mathrm{original}} = N \times M. \tag{1}$$

Online reordering involves multi-flow reordering, and the method we propose allows two online reordering units to time-multiplex a shared set of comparators. In Fig. 3 the online reordering architecture for the reordering unit with a sorting window W = 16 is shown, which is also called the total number of available shift registers. The shown reordering units can run in parallel. To facilitate the explanation of the improved online reordering architecture we propose, we first explain the implement rules of single reordering unit. We take reordering unit 1 as an example in Fig. 3. The parameters necessary for the explanation of the method are listed in Tab. I. The insertion sorting process can be divided into the determination of the insertion position and the movement of the element followed by the insertion process. Determining the insertion position is a key step in the insertion reordering process. When the telemetry packet in the network arrives at the reordering unit's arriving packet-1, the timestamp in the telemetry packet will be compared simultaneously with the timestamps of all packets in the reordering unit 1. The rules for determining the insertion position P_1 are as follows.

1) If $A_1 > E_{1\mathrm{j}}$, the output of the comparator is 0, else the output is 1.
2) If $[R_{11},\ldots,R_{1\mathrm{k}}]$ = [0,…,0] and $[R_{1(\mathrm{k}+1)},\ldots,R_{1\,T_1}]$ = [1,…,1], the insertion position P_1 = k.
3) When $[R_{11},R_{12},\ldots,R_{1T_1}]$ = [0,0,…,0], the insertion position $P_1 = T_1 + 1$.
4) When $[R_{11},R_{12},\ldots,R_{1T_1}]$ = [1,1,…,1], the insertion position P_1 = 1.

According to the rules mentioned, the insertion position could be determined. And since the comparator works in parallel, the insertion position P_{i} is determined in one cycle. Then for $P_1 \le j \le T$, the E_{j} simultaneously completes the right shift followed by the A_1 to be inserted into P_1 in one cycle. As described in rule 4 above, when the insertion position is $T_1 + 1$, shifting is not required. Nevertheless, the insertion operation is still scheduled within a cycle. Therefore, a single-insertion reordering can be completed in only two cycles. In the following, we discuss how the two reordering units time-multiplex a set of comparators. We mention that when the reordering unit 1 is performing right shifts followed, the

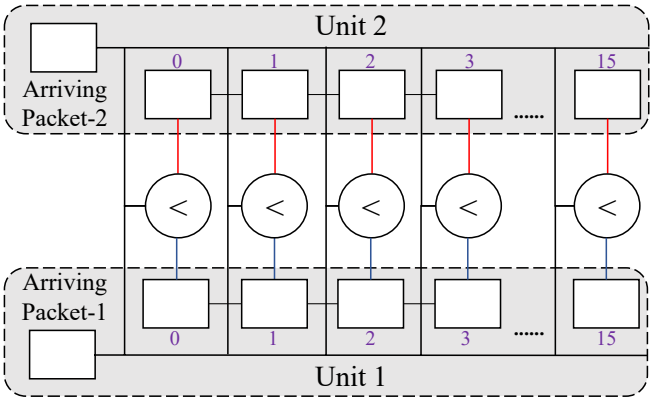

Fig. 3: The improved parallel insertion reordering framework

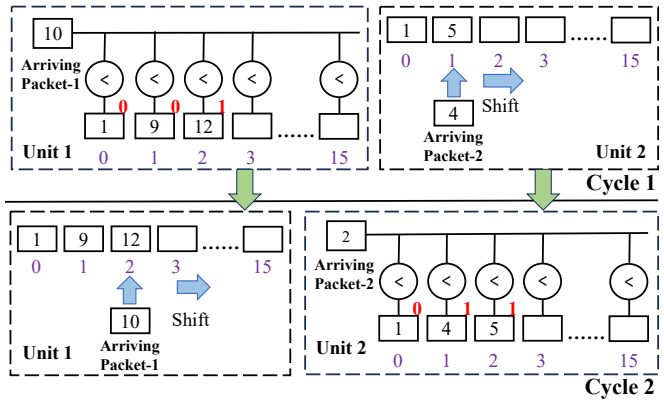

Fig. 4: The process of inserting reordering

comparators are in an idle cycle, which can be fully utilized by the reordering unit 2. The two reordering units in Fig. 3 respectively sort telemetry data from different network devices in the INT path, and the arriving packet unit stores earliest arriving telemetry data in the flow. Assume that the telemetry packet buffer queues of reordering unit 1 and unit 2 are [(10, data), (8, data)] and [(4, data), (6, data)], respectively. Therefore, A_1 and A_2 should respectively be $(10, \text{data})$ and $(6, \text{data})$. And in the previous cycle, unit 1 completed the element(9, data) insertion, while unit 2 completed the comparison. The following process is shown in Fig. 4. In cycle 1, the timestamp of the A_1 is compared with the timestamps of all the elements in the reordering unit 1, and [R_{11},R_{12},R_{13}] = [0,0,1], thus the insertion position P_1 is 3. At the same time, the right-shifting operation is completed followed by inserting element (4, data) in unit 2. In cycle 2, the shift is completed followed by the element (10, data) to be inserted. In cycle 2, the timestamp of A_2 is compared with the timestamps of all the elements in reordering unit 2, and [R_{21},R_{22},R_{23}] = [0,1,1], thus the insertion position is 2.

C. Offline Reordering Architecture

Offline Reordering is a solution to reorder large-scale telemetry data for telemetry applications. Pattern mining as a typical telemetry application, is a effective approach for monitoring network conditions[14]. Moreover, pattern mining on telemetry data enables the identification of recurrent congestion paths and anomalous device behaviors in the network. In this subsection, we discuss the architecture of offline reordering as well as the associated data transmission process.

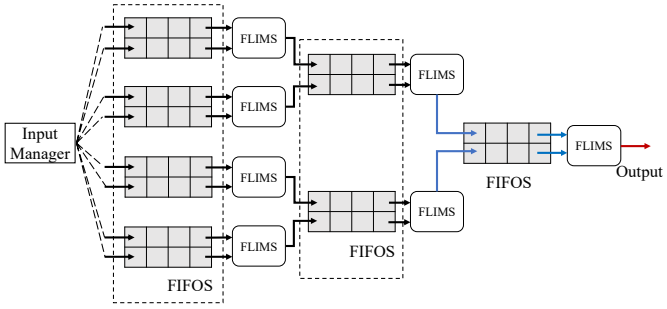

Fig. 5: The schematic diagram of offline reordering

After online reordering, the partially ordered telemetry data are ultimately stored in the time-series database. Merge sort is particularly effective when applied to partially ordered data, as it can efficiently merge sort sub-sequences into a globally ordered sequence.Recent research has proposed a variety of merge accelerators on FPGA[15]. In 2016, Wei Song et al. proposed the Parallel Merge Tree (PMT)[16] based on existing sorters, significantly improving performance compared to sequential sorters. Additionally, Philippos Papaphilippou et al. designed a Fast Lightweight Merge Sorter (FLiMS)[17] and introduced a new parallel architecture. By utilizing only half of the hardware resources, this architecture achieved superior FPGA performance, with lower resource utilization allowing for higher throughput. Thus the method we propose adopts FLiMS as the merge sorter in the parallel offline reordering architecture.

The architecture of offline reordering is shown based on FPGA in Fig. 5. To facilitate the explanation of the operational mechanism of the Input Manager, we treat every 32 telemetry data as a data block. After online sorting, the data within a block is ordered. However, the telemetry data across multiple blocks may be unordered. The Input Manager streams the telemetry data from a single data block into one FIFO, with the telemetry data from eight data blocks being streamed into eight separate FIFO.The process of data movement will be explained in the fourth paragraph of this subsection. FLiMS, as the merge sorter in this parallel reordering architecture, merges the telemetry data from two sorted lists into a single sorted queue. The implementation rules of FLiMS are as followed.

1) All FLiMS in Fig. 5 can run in parallel, provided they follow rules 2 and 3.
2) FLiMS stops functioning when any FIFO in two corresponding input FIFOs is empty.
3) FLiMS suspends if its output FIFO is full.

In the following, we describe the process of data transfer.Initially, the host transfers the data to be sorted into the

FPGA's DDR4 through Direct Memory Access (DMA) process in Fig. 2. Subsequently, the Input Manager transfers the data from DDR4 to the on-chip buffers (FIFOS) in bursts[18].

V. EVALUATION

In this section, we begin by presenting the details of our experimental setup. Then, We discuss the impact of different reordering window size in online sorting on the improvement of disorder and the effect on delivery latency. Finally, we discuss the performance of the overall reordering and its speedup ratio compared to the performance of reordering by using CPU.

A. Experimental Setup

Our system uses the Xilinx VU37P FPGA, equipped with 32 GB of DDR4 memory (split into 2 channels) and 10 Gbps Ethernet ports. The CPU used in the experiment is Intel Xeon Gold 5318Y in host. The INT is encapsulated in UDP, and the telemetry packet size is 32 B.

B. Latency and improvement of disorder of a single online reordering unit

To facilitate the description of the disorder level of a sequence, we define the disorder level D. Inversion count C and inversion distance sum S can collectively measure the disorder level of a sequence, with different weights assigned to their contributions. For a given sequence $a = [a_1, a_2, \ldots, a_N]$, where each inversion pair (a_i, a_j) satisfies $i < j$ and $a_i > a_j$, we can define the inversion distance sum S as the total sum of the distances of all inversion pairs is

$$S = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \mathbf{1}(a_i > a_j) \cdot (j - i), \tag{2}$$

where $\mathbf{1}(a_i > a_j)$ is an indicator function, which takes the value of 1 when $a_i > a_j$, indicating the existence of a inversion pair for that element. The $j - i$ is the distance between the inversion pair. And $C_{\max}$ and $S_{\max}$ represent the maximum Inversion Count and maximum Inversion Distance Sum of a sequence with size N, respectively. We can get:

$$S_{max} = \frac{1}{6}N(N-1)(N+1), C_{max} = \frac{N(N-1)}{2}. \tag{3}$$

The disorder level D is

$$D = \alpha \frac{C}{C_{\max}} + \beta \frac{S}{S_{max}}, 0 \leq D \leq 1, \tag{4}$$

In equation (4), α and β respectively represent the weights of the Inversion Count and Inversion Distance Sum. N represents the size of the sequence. The condition that α and β must satisfy is shown:

$$\alpha + \beta = 1, \tag{5}$$

where α and β represent the weights of the Inversion Count and Inversion Distance Sum in their influence on D.

To evaluate the impact of different reordering window sizes W on disorder improvement and delivery latency during the online reordering phase, the described FPGA was used to receive 1024 INT packets containing telemetry information from a specific device. The experimental results are shown in the Fig. 6. In this experiment, the values of α and β are set to 0.5 and 0.5. The experimental results indicate that when the reordering window size W varies within a range of small values, increasing the window size does not significantly improve the disorder improvement, while delivery latency almost increases exponentially. When a larger sorting window is chosen, the improvement in disorder is very significant, especially when the window size is set to 128, where the disorder level is nearly eliminated, although the delivery latency is also very high, reaching 1590 ns. Therefore, the choice of reordering window size W should be selected based on the real-time application's requirements for disorder improvement and delivery latency.

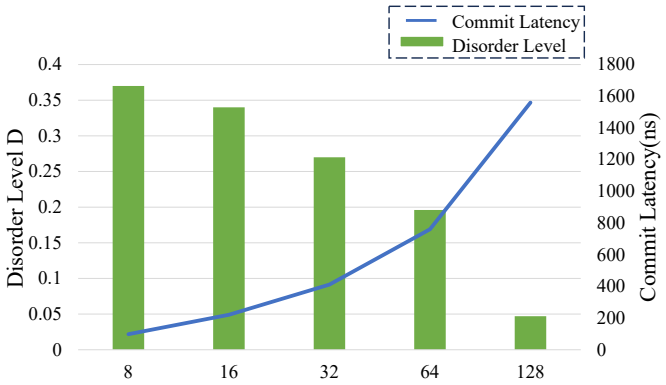

Fig. 6: The impact of different sorting window sizes on disorder improvement and delivery latency

C. Reordering performance

In this experiment, we evaluate the overall reordering performance. We monitor the network performance of a network device in the telemetry path and reorder its telemetry data of 2 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB, 256 MB and 512 MB. The total reordering time is the sum of the online reordering time and the offline reordering time. Similarly, we perform both online reordering and offline reordering on the CPU for the above data. The online reordering uses the insertion sort algorithm, while the offline reordering uses std::sort(). The GCC C++ Standard Library's std::sort() is considered a widely available and excellent baseline for evaluating high-performance sorting in software. Finally, we calculate the speedup corresponding to different data sizes.

The experimental results are shown in Fig. 7.The results show that when the size of the data gradually increases, our acceleration performance gradually improves. When the data size is 32MB, the acceleration ratio is 7.4, whereas when the data size reaches 512MB, the acceleration ratio can reach 12.5. In conclution, the data size is relatively large, the acceleration effect of our proposed CPU/FPGA-based reordering method becomes more efficient.

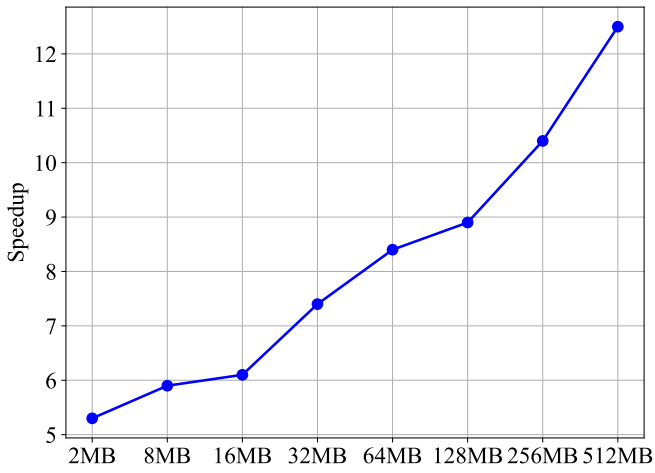

Fig. 7: The speedup of FPGA over CPU for different data sizes

VI. Conclusion

In this paper, we propose a method based on a CPU-FPGA heterogeneous platform to accelerate reordering for INT. This method combines online reordering with offline reordering, meeting the real-time requirements of telemetry applications. Our experimental results prove that the proposed method can accelerate the entire reordering process of telemetry data. In the future, more efficient reordering algorithms and more adequate hardware resource utilization are important research directions for improving the performance of In-band Network Telemetry out-of-order reordering.

References

[1] L. Tan, W. Su, W. Zhang, *et al.*, "In-band Network Telemetry: A Survey," *Computer Networks*, vol. 186, p. 107 763, 2021.

[2] W. Gao, J. Huang, N. Jiang, *et al.*, "HPLB: High precision load balancing based on in-band network telemetry in data center networks," *Peer-to-Peer Networking and Applications*, vol. 15, no. 6, pp. 2503–2515, 2022.

[3] Y. Li, R. Miao, H. H. Liu, *et al.*, "HPCC: High precision congestion control," in *SIGCOMM'19*, ACM, 2019, pp. 44–58.

[4] K. Zhang, W. Su, H. Shi, K. Zhang, and W. Zhang, "Grayint–detection and localization of gray failures via hybrid in-band network telemetry," in *APNOMS'23*, IEEE, 2023, pp. 405–408.

[5] R. Hohemberger, A. G. Castro, F. G. Vogt, *et al.*, "Orchestrating In-Band Data Plane Telemetry With Machine Learning," *IEEE Communications Letters*, vol. 23, no. 12, pp. 2247–2251, 2019.

[6] Z. Zhang, W. Su, and L. Tan, "In-band network telemetry task orchestration based on multi-objective optimization," in *APNOMS'21*, IEEE, 2021, pp. 354–357.

[7] T. Ouyang, H. Yao, W. He, T. Mai, F. Wang, and F. R. Yu, "Self-adaptive dynamic in-band network telemetry orchestration for balancing accuracy and stability," *IEEE Transactions on Network and Service Management*, 2025.

[8] Y.-K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "In-depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, pp. 1–20, 2019.

[9] M. Huang, A. Shen, K. Li, *et al.*, "EdgeLLM: A Highly Efficient CPU-FPGA Heterogeneous Edge Accelerator for Large Language Models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2025.

[10] M. Kalemati, A. D. Nayeri, and S. Koohi, "Pip-SW: Pipeline Architectures for Accelerating Smith-Waterman Algorithm on FPGA Platforms," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–12, 2024.

[11] M. S. Alam, U. Khan, M. Hasan, and O. Farooq, "Energy Efficient FPGA Implementation of an Epileptic Seizure Detection System using a QDA Classifier," *Expert Systems with Applications*, vol. 249, p. 123 755, 2024.

[12] V. Q. Hoang and Y. Chen, "Cost-effective Network Reordering using FPGA," *Sensors*, vol. 23, no. 2, p. 819, 2023.

[13] Y. Thomas, G. Xylomenos, C. Tsilopoulos, and G. C. Polyzos, "Multi-flow Congestion Control with Network Assistance," in *NETWORKING'16*, IEEE, 2016, pp. 440–448.

[14] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, "A survey of Sequential Pattern Mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.

[15] P. Papaphilippou, C. Brooks, and W. Luk, "An Adaptable High-throughput FPGA Merge Sorter for Accelerating Database Analytics," in *FPL'20*, IEEE, 2020, pp. 65–72.

[16] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel Hardware Merge Sorter," in *FCCM'16*, 2016, pp. 95–102.

[17] P. Papaphilippou, C. Brooks, and W. Luk, "FLiMS: Fast Lightweight Merge Sorter," in *FPT'18*, IEEE, 2018, pp. 78–85.

[18] C. Ferry, T. Yuki, S. Derrien, and S. Rajopadhye, "Increasing FPGA Accelerators Memory Bandwidth With a Burst-Friendly Memory Layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 5, pp. 1546–1559, 2023.