BLOP: Batch-Level Order Preserving for GPU-Accelerated Packet Processing

Zhilong Zheng, Jun Bi, Heng Yu, Chen Sun, Jianping Wu*
Tsinghua University

CCS CONCEPTS

• **Networks → Middle boxes / network appliances**; *Network performance analysis*; Data path algorithms;

KEYWORDS

NFV, GPU, batch-level, order-preserving

ACM Reference format:
Zhilong Zheng, Jun Bi, Heng Yu, Chen Sun, Jianping Wu. 2017. BLOP: Batch-Level Order Preserving for GPU-Accelerated Packet Processing. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 2 pages.
https://doi.org/10.1145/3123878.3132013

1 INTRODUCTION

Recent advances in networks such as Network Function Virtualization (NFV) have driven the emergence of high-performance packet processing on commodity hardware. The nature of packet data naturally lends itself to *parallel processing*, and recent work has demonstrated that GPUs, which have thousands of cores, are well-suited to a number of packet-processing tasks [3].

A typical workflow of GPU-accelerated packet processing includes the following major steps [1]. (1) Multiple worker threads in multi-core CPUs keep fetching packets from NICs (based on DPDK) and storing them in host memory. (2) Anytime when a worker thread collects a *batch* of packets, it feeds them into the on-board memory on GPU and invokes a block of cores (threads) to process the packets in parallel. (3) After processing an entire *batch*, the core block in GPU informs the related worker thread in CPU to copy the *batch* back to host memory and send it out. GPU parallelism could deliver high throughput with relatively low cost [3].

However, the high performance brought by GPU parallelism comes with significant challenges. Especially, parallel processing could cause serious packet *order violations*, incur massive retransmission, and compromise end-to-end TCP throughput. The root cause is the ***double-edged sword of GPU parallelism***. Multiple core blocks in GPU processes their batch of packets individually and simultaneously. Due to different code complexity including branching, iteration, etc. in different core blocks, a latter packet batch could be processed and returned to the host memory *before* a former batch, causing ***batch-level out-of-order*** situations. Unfortunately, current researches did not focus on the challenge of order preserving to actually deliver the high performance of GPU. Despite that DPDK provides packet-level reordering interfaces, DPDK normally collects only a few packets for reordering, which cannot effectively alleviate batch-level out-of-order situations.

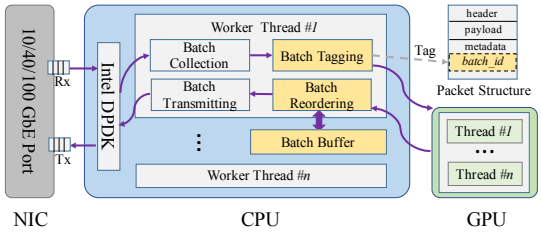

Figure 1: BLOP design overview

To address this challenge, we propose ***BLOP, a Batch-Level Order Preserving framework for GPU - accelerated packet processing***. We first measure the percentage of out-of-order packets (up to 26.1%) after GPU processing as the motivation for BLOP. Then we propose two *key observations* including (1) Packet order is strictly maintained inside a batch, and (2) The difference of processing time between core blocks is small, and packet reordering only needs to entangle very few batches. Finally, we carefully design the *BLOP framework* for order preserving. BLOP lightly *tags* batches in CPU before sending them to GPU, and efficiently *reorders* batches in CPU after GPU processing. Evaluations demonstrate that BLOP could achieve order preserving with little performance overhead.

2 BLOP DESIGN

2.1 Packet Out-of-Order Rate Measurement

To measure the rate of out-of-order packets after GPU processing, we build a platform according to a typical GPU-accelerated architecture [3]. The platform consists of two Dell R730 servers, each equipped with two Intel Xeon E5-2620 v3 CPUs (6 cores @2.40 GHz) and one dual-port Intel X520 10G NIC. The servers run Ubuntu with Linux kernel 3.19.0 with DPDK 16.07. One server carries a NVIDIA GTX 1070 GPU for packet processing, while the other server is used to generate TCP flows as the test traffic. Our DPDK-based packet generator could saturate the 10G NIC.

We have implemented two representative packet processing tasks including IPv4 forwarding and SHA-1 hashing on GPU. We send and collect TCP packets and identify out-of-order packets with unexpected *sequence numbers*. We calculate the average out-of-order packet percentage in all generated packets under different

*Zhilong Zheng, Jun Bi, Heng Yu, Chen Sun and Jianping Wu are with Institute for Network Sciences and Cyberspace, Tsinghua University, Department of Computer Science, Tsinghua University, and Tsinghua National Laboratory for Information Science and Technology (TNList). This work is supported by National Key R&D Program of China (2017YFB0801701) and the National Science Foundation of China (No.61472213). Jun Bi is the corresponding author.

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08.
https://doi.org/10.1145/3123878.3132013

packet rates (1 - 10Gbps), which follows the measurement and calculation methods introduced in [2]. Experimental results show that with the rise of packet rate from 1 to 10Gbps, the out-of-order packet percentage increases from 11.3% to 17.9% in IPv4 forwarding, and from 15.7% to 26.1% in SHA-1 hashing. This serious packet order violation could severely deteriorate end-to-end throughput, driving the need of an order preserving mechanism for GPU.

2.2 Key Observations

Now we present two key observations derived from the study of a typical GPU processing mechanism [1], and use the observations to instruct the design of BLOP.

OB #1: Packet order inside a batch is strictly maintained before and after GPU processing. After a batch is copied into GPU's on-board memory, each core in the core block processes one packet in the batch in an *in place* manner. Each core obtains a pointer to a packet, processes the packet, and stores the processed packet back to the pointer. Therefore, when copied back to the host memory, packets inside the batch possess strictly the same order as that before processing. This enlightens the *batch-level reordering* of BLOP. Since partial packets are already ordered, a sorting of all packets would be a waste of calculation time and resources.

OB #2: The difference of processing time between core blocks is small, and packet reordering only needs to entangle very few batches. First, in most cases, core blocks on the same GPU carry the same network function and process packets in the same way. Second, as a co-processor that only executes commands from CPU, GPU is cleanly isolated from the scheduling of the operating system. Third, one GPU core carries one thread and suffers no unpredictable latency penalty of context switching between threads. Finally, the difference of processing time could be effectively averaged for each packet by adopting large batch sizes [4]. Thus, we do not have to maintain a very large buffer to cache batches for reordering.

2.3 BLOP Framework Design

Instructed by the above observations, we propose the BLOP framework as shown in Figure 1. BLOP inserts two major functional modules inside the CPU, including a *Batch Tagging* module in the ingress pipeline from Network Interface Card (NIC) to GPU, and a *Batch Reordering* module in the egress pipeline from GPU to NIC. Next we describe the two modules in detail.

Batch Tagging: After a worker thread in CPU collects a full batch or timeout is reached, the Batch Tagging module would mark *every packet* inside the batch with the global unique *batch_id* in its *metadata*. The intuition here is that all output batches copied to the host memory after GPU processing *share a ring buffer*. By tagging every packet in a batch, we can easily gather all packets of a batch that could be reordered in the next step. Besides, the tagging can be conveniently implemented since DPDK already allocates a *metadata* field for every packet. In this way, batch tagging can be performed in a light-weighted manner, after which the batch is sent to the corresponding core block in GPU.

Batch Reordering: After GPU processing, the processed batch will be copied back to the shared ring buffer inside the host memory. The Batch Reordering module accumulates and monitors the *batch_id* of each new batch. If a new batch carries the expected

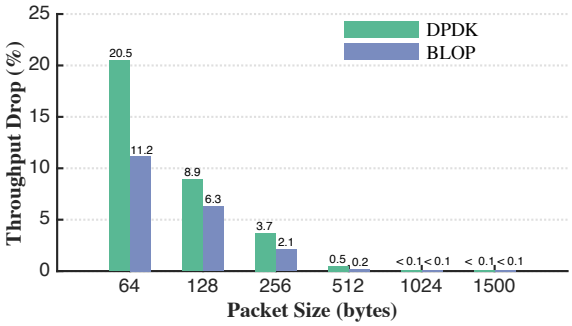

Figure 2: Throughput drop caused by order preserving

batch_id, this batch is directly transmitted to the NIC. Otherwise, the batch is moved to a *batch buffer* and waits for its turn. However, if any batch inside the batch buffer has waited for a pre-configured *timeout*, this batch is transmitted regardless of the id, which still could result in some extent of order violation. Nevertheless, this is a rare case according to OB #2.

3 IMPLEMENTATION AND EVALUATION

We have implemented BLOP on the platform introduced in §2.1, based on which we evaluate BLOP compared with the packet-level reordering in DPDK. We configure the batch size as 32 packets, and set the depth of the batch buffer as 4 batches. Similarly, we reserve a 128 packet buffer size for reordering in DPDK. We send TCP packets of different sizes under 10Gbps to the IPv4 forwarding task on the GPU, and measure the percentage of out-of-order packets with the two types of reordering mechanisms respectively, as well as the latency and throughput penalty for introducing the order preserving mechanisms. Evaluation results show that the two mechanisms could reduce the rate of out-of-order packets to an equally low level (around 4%), since they use the same amount of buffer. However, DPDK's packet reordering can cause 80 additional CPU cycles per packet, while BLOP only consumes 46 additional cycles for each packet. Furthermore, Figure 2 shows that DPDK incurs 2 × throughput drop of the GPU compared with BLOP. Above evaluations demonstrate the effect and efficiency of BLOP.

4 CONCLUSION AND FUTURE WORK

We have proposed BLOP, an effective and efficient order preserving framework for GPU-accelerated packet processing. In future, we will design advanced algorithms and buffer settings to achieve an optimal trade-off between *order preserving*, *performance*, and *resources*. We will also identify appropriate tasks to offload to GPU, and evaluate the rates of out-of-order packets of different tasks.

REFERENCES

[1] Younghwan Go, Muhammad Asim Jamshed, YoungGyoun Moon, Changho Hwang, and KyoungSoo Park. 2017. APUNet: Revitalizing GPU as Packet Processing Accelerator.. In *NSDI*. 83–96.

[2] S Govind, R Govindarajan, and Joy Kuri. 2007. Packet reordering in network processors. In *IPDPS*. IEEE.

[3] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. 2010. PacketShader: a GPU-accelerated software router. In *ACM SIGCOMM Computer Communication Review*, Vol. 40. ACM, 195–206.

[4] Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis. 2014. GASPP: A GPU-Accelerated Stateful Packet Processing Framework.. In *USENIX Annual Technical Conference*. 321–332.