In-network Processing of the GPU-based Real-time DXT Compression*

Namgon Kim, Jae-Yong Yoo, Namgon Lucas Kim and JongWon Kim
Networked Computing Systems Lab., School of Information and Communications,
Gwangju Institute of Science and Technology (GIST)
Gwangju, 500-712, Korea
{ngkim, jyyoo, namgon, jongwon}@nm.gist.ac.kr

ABSTRACT

In this paper, we show our on-going work on realizing a PC-based network switch that provides low-latency in-network processing of GPU-based DXT compression.

1. INTRODUCTION

To assist the buildup of programmable network infrastructure capable of high-performance networking, several software routers have been introduced to provide fast packet-processing performance. PacketShader [1] introduces a software router platform that utilizes the high-throughput computing capability of GPU, and the same platform is applied to realize a transparent SSL (secure socket layer) proxy in SSLShader. As a first step to build a software router providing general in-network processing, in this paper, we show our on-going work on realizing a PC-based network switch that provides in-network processing of GPU-based DXT compression.

2. DXT COMPRESSION AS IN-NETWORK PROCESSING

In-network processing is the capability of a network switch applying additional processing on the packets before forwarding them to their destinations, as shown in Fig. 1. For a PC-based network switch, we implement a processing module by utilizing CPU or GPU. For light-complexity processing, CPU-based in-network processing can provide shorter processing delay while it could decrease the packet forwarding throughput as CPU also takes charge in packet forwarding. On the contrary, GPU-based in-network processing is better for tasks that can benefit from processing parallelism with many cores of GPU. However it could add more latency due to the expensive memory copies between host memory and GPU device memory.

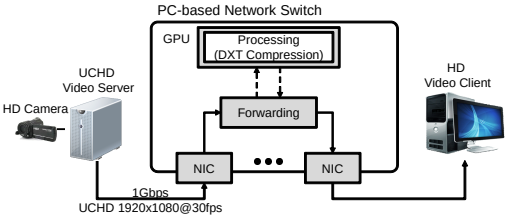


Figure 1: In-network processing in a PC-based network switch.

DXT (i.e., S3 texture compression) compression is a good example to apply GPU-based in-network processing. It is a light-weight compression scheme that compresses an image frame based on 4x4-pixel blocks. The size of each uncompressed pixel block is 32bytes with YUV422 format and is decreased to 8bytes with DXT compression. Compressing a single high-definition video frame with 1920x1080 pixels requires 129600 times of pixel-block compression operations which can be significantly reduced with hundreds core GPU [2]. We expect that in-network processing of GPU-based DXT compression can enable serving multiple uncompressed HD videos in real-time with several times less bandwidth.

3. LATENCY EVALUATION

For the evaluation, we setup a test environment as shown in Fig. 1. An uncompressed HD video server captures 1080i HD video from a HDMI camera and transmits the video to a HD video client. The PC-based network switch performs in-network DXT compression for the packets received from the video server and for-

*This research was supported in part by the KCC (Korea Communications Commission), Korea, under the Development of the core technology and virtualized programmable platform for Future Internet support program supervised by the KCA (Korea Communications Agency) (KCA-2011-09913-05006) and in part by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MEST) (No. 2011-0027558).

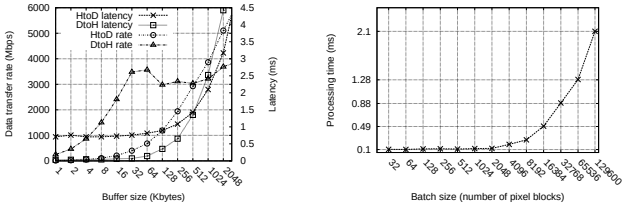
ACM CoNEXT Student Workshop, December 6, 2011, Tokyo, Japan.

wards the compressed packets to the video client. For sending a frame, it requires 2880 packets to transmit assuming MTU size as 1500bytes and a packet can contain 45 pixel blocks at most. The batch size is related to the number of pixel blocks in a packet. In this experiment, considering the performance limit of the network switch, we limit the frame rates of HD video to 20fps.

To provide real-time in-network processing, it is important to minimize the latency caused by the processing part. From Fig. 1, we can intuitively see the latency incurred by the memory copy 1) from host to device 2) from device to host, plus 3) that of DXT compression itself. The latency in memory copy can vary according to the buffer size. Also the time taken in DXT compression would be determined by the number of pixel blocks to compress. We perform following experiments to understand the relationship between control factors and latency.

3.1 Understanding the processing latency

We use following experiment configuration. We use an Intel Xeon single-core CPU 3.60GHz processor with 400MHz 2GB SDRAM memory. For GPU acceleration, we use a NVIDIA GTS450 card on a PCIe 1.0 x16 link, which has 192 stream processor cores and 1GB GDDR5 memory. For network cards, we use two NICs: an onboard Intel 82545GM 1GbE NIC and a Realtek 8169 1GbE NIC on a PCI slot. We use user-space Click to configure a PC-based network switch and apply in-network processing by implementing GPU-based DXT compression as a Click element.



(a) (b)

Figure 2: Experiment results: (a) Data transfer rate and latency in memory copy, (b) Processing time of GPU-based DXT compression.

We measure data transfer rate and latency between host and device memory over different buffer sizes and summarize the results in Fig. 2(a). For buffer sizes less than 128Kbytes, despite the small latency, we can only get data transfer rate around 1Gbps just close to that of NICs. When the buffer size is bigger than 1Mbytes, it takes more than 2ms to copy in one direction and the time increases linearly as the buffer size increases. Figure 2(b) shows the processing time of GPU-based DXT compression according to the number of pixel blocks. The processing time takes less than 0.2ms until 4096 pixel blocks. We can see that processing a single frame at a time, 129600 pixel blocks, takes 2.1ms and the processing time increases linearly for batch sizes bigger than 4096 pixel blocks. From the above two results, we can get minimum latency from a region between 2048 (=64Kbytes) and 8192 (=256Kbytes) pixel blocks.

3.2 Finding the minimum latency

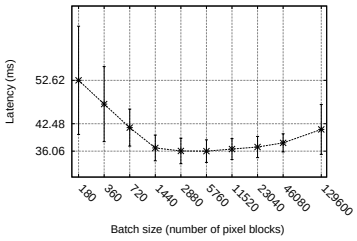


Figure 3: Latency of in-network DXT compression.

We measure the time for processing a frame in the network switch as shown in Fig. 3. For a frame, we compute the latency by the time difference between the arrival time of the first packet and the departure time of the last packet sent to the client. From fig. 3, we achieve the minimum latency, 36ms, with the batch size of 2880 (=90Kbytes) and 5760 (=180Kbytes) pixel blocks. It takes 41ms to complete the processing after all packets belonging to a frame arrives. With a batch size smaller than 1440 (=45Kbytes) pixel blocks, the time increases due to limited data transfer rate between host and device memories. As we increase the number of pixel blocks over 5760, it takes longer time. We also measure the time for forwarding the packets without processing them, which takes 42ms. This is because, through DXT compression, the number of packets to transmit is reduced while adding small latency for processing.

4. FUTURE WORK

In this work, we show preliminary results of GPU-based in-network DXT compression targeting low latency. In future, we plan to enhance the latency by applying other optimization techniques and increase the scalability of in-network processing with multiple 10GbE NICs.

5. REFERENCES

[1] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-accelerated software router. In *Proc. ACM SIGCOMM 2010*, Aug. 2010.

[2] N. L. Kim and J. Kim. Deploying GPU-based real-time DXT compression for networked visual sharing. In *Proc. APAN Network Research Workshop 2011*, Aug. 2011.