Implementation of DXT Compression for UltraGrid

Ian Wesley-Smith, Miloš Liška, Petr Holub

CESNET Technical Report *x*/2008
2008–06–15

Abstract

This report describes an implementation of DXT compression in UltraGrid, allowing for low latency, high definition multi-party videoconferencing requiring only 250 Mbps of bandwidth. This represents a substantial decrease compared to the uncompressed video stream, which requires 1.5 Gbps, while having a minimal impact on latency. The DXT implementation includes a new client-side display which, when using DXT compressed data, takes full advantage of the graphics card, resulting in minimal CPU utilization. This report includes experimental evaluation of end-to-end latencies, CPU load, and traffic profiles for both uncompressed and DXT-compressed video.

Contents

1 Introduction

An uncompressed high-definition (HD) video transmission creates an unparalleled experince for participants of collaborative environments, providing both high image quality and low end to end latency of the transmission (considered in an end-to-end way, i.e., all the way from the camera to the display screen). Utilization of such an approach has been demonstrated by several teams around the world in recent years: The UltraGrid system provided by Pekins & Gharai [1, 2], a modified version of UltraGrid by CESNET team [3, 4], iHDTV systems [5], or the HDTV over IP project by NTT Laboratories [6]. Although, as noted earlier, uncompressed HD video provides the best image quality and lowest latency, its high bandwidth utilization, about 1.5 Gbps, severely hinders deployment of systems utilizing it.

In order to facilitate adoption in non 10GbE environments, we have integrated low-latency DXT compression [7–9] into UltraGrid. There is a series of DXT compression schemes named DXT1 through DXT5 [8], of which DXT1 provides the best compression ratio and thus has been actually used in UltraGrid. It is based on color indexing, where each 4×4 pixel block (so called *texel*) is converted into two 16 b colors[1] (c_1, c_2) and a lookup table comprising 16 color indices with 2 b per index. The index points to one of the following four colors then: $(c_1,\ ⅔c_1 + ⅓c_2,\ ⅓c_1 + ⅔c_2,\ c_2)$. Choice of the colors (c_1, c_2) is up to the implementation and is an important factor for quality of resulting images. Discussion of alpha channel handling has been omitted in the discussion of DXT as it is not used in UltraGrid. DXT compression was selected for the following reasons:

- 6:1 bitrate reduction (by converting 4×4 pixel block with 24 b/pixel into 4×16 b structure; coincidentally, the same 6:1 ratio is also achieved when comparing the original 10 b 4:2:2 HD-SDI stream at 1.5 Gbps to the resulting 8 b 4:4:4 stream at 250 Mbps).
- acceptable image quality on natural scenes (computer-generated images with long fine gradients may display more significant posterization).
- availability of an implementation in real-time [10] and for HD frame size and rate [14].
- availability of DXT texture decompression and rendering directly in graphics cards, even in low-cost models.

While the DXT compression has significant computational requirements, and thus requires a powerful enough sender computer, the receiver can be a fairly low cost computer equipped with a graphics card supporting DXT textures. The resulting data streams can be run over Gigabit Ethernet, which is now often available from LANs.

JPEG2000 Experiment Implementation of JPEG2000 was also attempted in order to achieve a higher-quality compressed video, ideally with GPU support for both encoding and decoding as JPEG2000 compression and decompression are fairly exhaustive processes when implemented on a CPU only. This, however, proved to be infeasible due to the high read-back latency from the GPU using Cg [11] (and CUDA [12] only became available at the time of development). An overview of the measured latency is shown in Figures 1 and 2, including both component-wise latency and the overall latency of the whole process. The measurements were performed using a NVidia 7600GS (with both 9000-series driver and the new

[1] Each color uses 5 b for red, 6 b for green, and 5 b for blue, i.e., 16-bit RGB 5:6:5.

100.14.11 driver) and a NVidia 8800GTX (with the 100.14.11 driver only). The results suggest that NVidia 8800-series cards are much better suited for general purpose computing compared to the older series, but still not usable for real-time compression using the Cg approach (this may significantly improve with CUDA approach).

2 Implementation

The implementation requires two parts: the *sender*, where compression of the incoming data is done, and the *receiver*, where decompression happens as a part of the rendering process. The basic assumption is that the data will be sent to more than one receiver (be it for 1:N distribution as in a virtual class, or full N:N collaboration where there are N – 1 receivers for each sender) and thus the playback part has to be as affordable as possible (both in terms of required CPU capacity and availability of required hardware). The sending part should also be optimized, but it is less critical compared to the receiving part. Another critical requirement is low end-to-end latency of the system, and as such the compression should introduce as little additional latency as possible. With the advent of general purpose GPU computing [13], we wanted to utilize the GPUs computational power to the maximum extent available at the time of development.

2.1 Sender

We have opted to use the FastDXT library [14, 15], which performs parallel compression of the data using CPUs. This library has been developed for compressing 4K video and can handle HD video in real time. It uses highly optimized code based on intrinsics, resulting in efficient code generation across various platforms.

The UltraGrid sender handles acquiring the video through one of its capture interfaces (DVS SDK for HDstation, Centaurus, Centaurus II and Centarus II LT cards on Linux, or very flexible QuickTime interface that allows for using various underlying capture cards). The acquired video data is extrapolated from 4:2:2 with 10 b per component to 4:4:4 sampling with 8 b per color component and then the DXT compression is performed using 3 threads running in parallel. It should be noted, however, that to decrease latency and computational requirements on the sender, the compressed image is still in YUV color space with conversion being done on the receiver. For sending with Jumbo frames enabled, the sender machine should have at least 4 cores (as 3 cores are completely busy with compression threads and one remaining core handles the other work). Four cores are also needed in order to send standard-sized 1500 B frames as the load increases only moderately, see Section 3.1.

2.2 Receiver

As noted above, we rely on DXT since its decompression through OpenGL has direct hardware support on vast majority of modern GPUs. Therefore, an OpenGL front end has been implemented for UltraGrid (the original implementation from [4] used only SDL front end for software-based rending). After decompressing the image, an OpenGL fragment shader is applied to convert from YUV to RGB color space. As all image processing occurs on the graphics card, the DXT OpenGL front end uses less resources than any other available front-end for UltraGrid. The OpenGL front end also supports scaling operation, so that the video can be played back using any screen resolution up to 1920×1200.

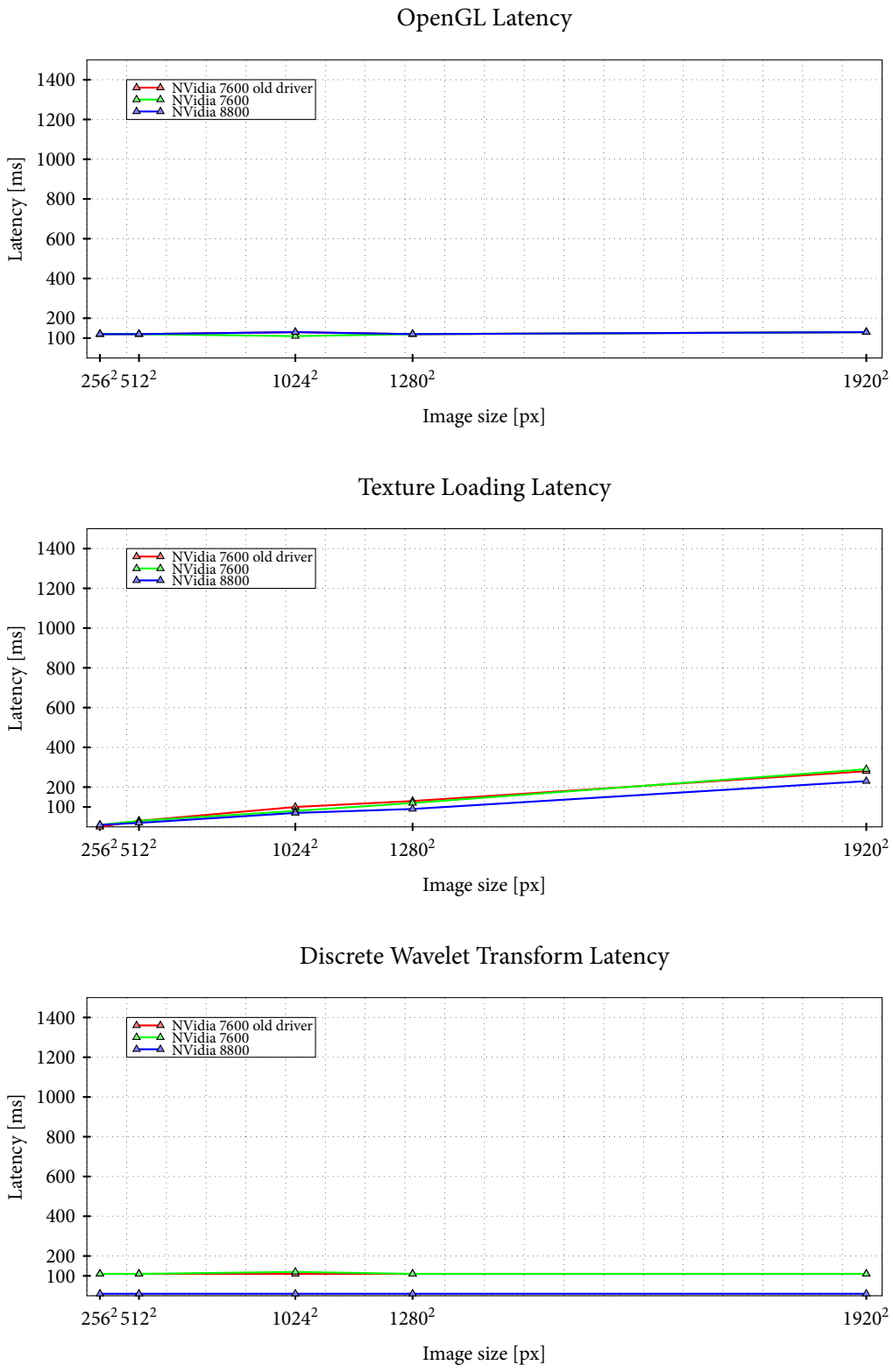

Figure 1: Latency measurements of discrete wavelet transform performed on a GPU.

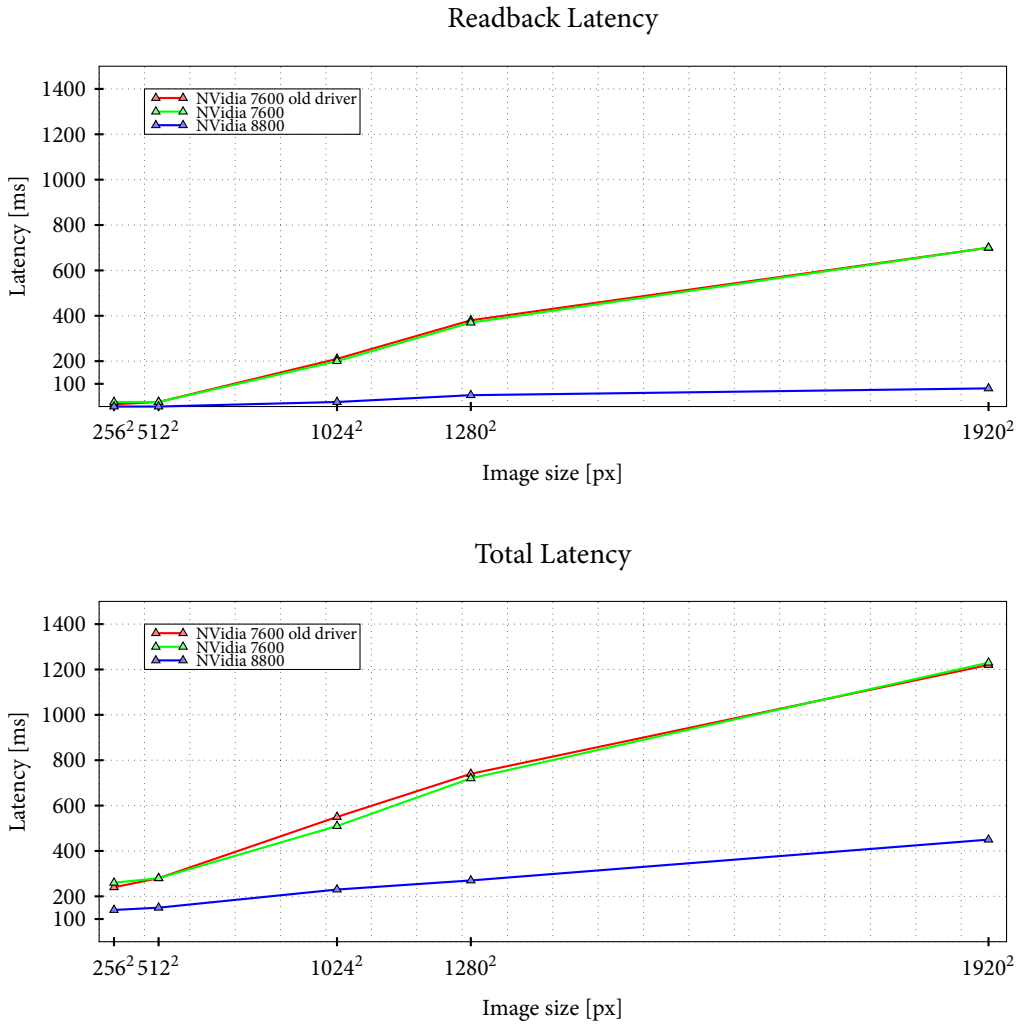

Figure 2: Latency measurements of discrete wavelet transform performed on a GPU.

The implementation has been tested on various NVidia cards (7000 and 8000 series) and it performed as expected. The problems were however when we tried to run the DXT receiver on Mac Mini with built-in Intel GMA 950 graphics card as it seems there is no support for OpenGL 2.0 on this GPU (OpenGL 1.4 is only supported). The compilation of shader fragment for color space conversion fails on this card probably because GLSL extensions are not fully supported—this problem will be worked upon in the future.

3 Performance Evaluation

In order to analyze behavior of the system, we have performed measurements of end-to-end latencies. This measurement methodology uses the following setup: a generator computer is used to display a time-changing pattern to analyze the latency on an attached LCD screen. The screen of the generator computer is captured by the SONY HVR-Z1E camera attached to the sender computer using an analog to HD-SDI converter (AJA HD10A). The data is received by the receiver computer and displayed on an attached LCD screen. For all the measurements, sender and receiver were connected directly back to back (without a switch) using 2 meters of a single-mode fiber. Both generator screen and receiver screen were captured by a still digital camera and the latency difference has been read out. The same principle has been used to carry out end-to-end latency measurements in [3]. This technical report also updates results for uncompressed video shown is [3] as the implementation of both UltraGrid and DVS SDK has improved since then.

The Linux machines (both sender and receiver) for measuring latencies with Centaurus and Centaurus II cards both with and without DXT compression were set up as follows:

- ▷ 2× processor AMD Opteron Dual Core 2.6 GHz
- ▷ 2 GB RAM
- ▷ 10GbE network interface card Myricom Myri-10GE (PCIe, i.e., PCI Express)
- ▷ Centaurus (PCI-X) or Centaurus II (PCIe) capture card

The MacOS X machine which has been used again as both sender and receiver has been configured as follows:

- ▷ Intel-based Mac Pro
- ▷ 2× Intel Xeon Quad Core 3 GHz
- ▷ 4 GB RAM
- ▷ 10GbE network interface card Myricom Myri-10GE (PCIe)
- ▷ Blackmagic Decklink HD Pro capture card (PCIe)

3.1 Results

Latency. Measurement results are summarized in Table 1. Error of the measurement is given by refresh rate of the 60 Hz LCD screen which is 16 ms.

Sender load. Results indicate two positive results: (1) DXT compression has only a very small impact on the latency and (2) latency has improved since publication of results in [3]. Also it is obvious that, in terms of latency, the newer and cheaper Centaurus II card performs better than its predecessor. This is likely due to the fact that it produces a lower computational load on the system, and thus it performs significantly better when DXT compression is occurring (this may be attributed at least in part to its PCIe interface).

Configuration	*Latency [ms]*
Linux, Centaurus, no compression, Jumbo frames	90±8
Linux, Centaurus II, no compression, Jumbo frames	85±8
Linux, Centaurus, DXT compression, Jumbo frames	130±8
Linux, Centaurus II, DXT compression, Jumbo frames	95±8
MacOS X, DeckLink Pro HD, no compression, Jumbo frames	148±8
MacOS X, DeckLink Pro HD, DXT compression, Jumbo frames	178±8

Table 1: Latency evaluation results summary.

Configuration	*CPU load*
Linux, no compression, Jumbo frames	26 %
Linux, DXT compression, Jumbo frames	332 %
Linux, DXT compression, 1500B frames	350 %
MacOS X, no compression, Jumbo frames	46 %
MacOS X, DXT compression, Jumbo frames	324 %
MacOS X, DXT compression, 1500B frames	350 %

Table 2: Sender CPU load evaluation results summary.

CPU load was measured for both Linux and Mac Pro machine senders sending uncompressed and DXT compressed HD video. Results are summarized in Table 2.

The CPU load is measured in total for all CPU cores. 100 % CPU load in this case means either one fully loaded core or equivalent load distributed among number of cores. In case of sending DXT compressed streams the DXT compression was fully loading three cores on both Linux and Mac Pro sender. Such a requirement was easily matched with quad-core (Linux sender) and octo-core (Mac Pro sender) setups.

Receiver load. CPU loads on the receiving machines and different HD video streams are given in Table 3. An expected observation is a low CPU load imposed by receiving and displaying DXT compressed streams. For the Linux receiver, the CPU load was only 17 % of an Opteron 245 core when using DXT compressed stream with Jumbo frames enabled.

Configuration	*CPU load*
Linux, no compression, Jumbo frames	123 %
Linux, DXT compression, Jumbo frames	17 %
Linux, DXT compression, 1500B frames	24 %
MacOS X, no compression, Jumbo frames	160 %
MacOS X, DXT compression, Jumbo frames	9 %
MacOS X, DXT compression, 1500B frames	29 %

Table 3: Receiver CPU load evaluation results summary.

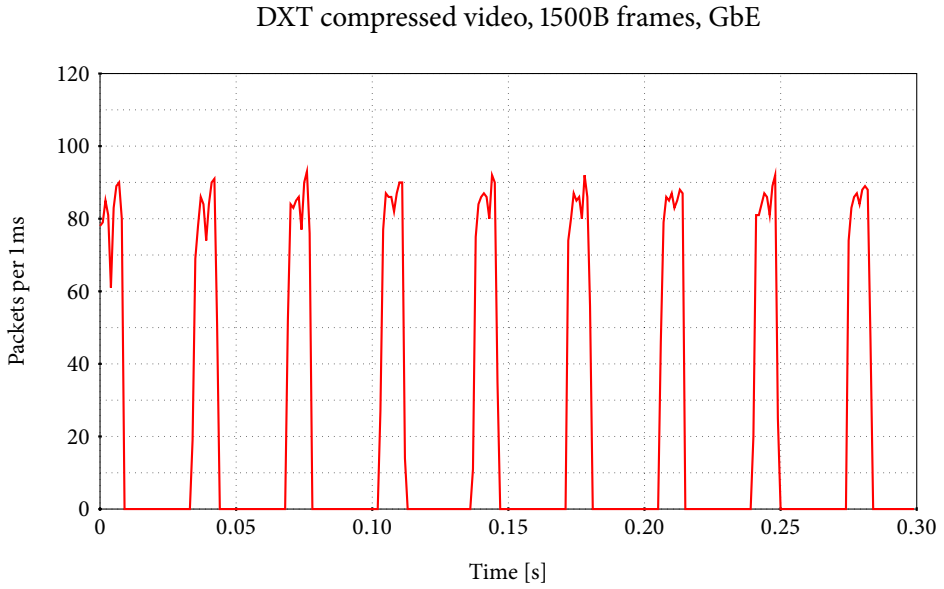

Figure 3: Traffic profiles for various UltraGrid setups.

Traffic profiles. UltraGrid is known for generating bursty traffic in order to deliver video frames to the destination as fast as possible [3]. We have compared traffic profiles for DXT compressed video with both standard sized and Jumbo frames and uncompressed video with Jumbo frames. The profiles were captured using `tcpdump` utility on the Mac Pro sender machine. Only first 68 B of each packet were captured in order to minimize performance impact of the measurement itself.

The results are shown in Figures 3, 4, and 5. The raw packet arrival times were aggregated into 1 ms time slots and which are used to plot the profiles. Most extreme bursts in terms of packet count are generated when transmitting DXT compressed video with standard-sized frames. Both compressed and uncompressed streams with Jumbo frames generate similar packet rates, but for compressed video the burst length is significantly shorter.

4 Conclusions

In this report, we describe the DXT compression implementation for UltraGrid, a low-latency high-definition collaborative system. The OpenGL front end for UltraGrid has been implemented, and, more importantly, this allowed for real-time image compression utilizing a modified version of FastDXT library and decompression using the OpenGL front end. This compressed version of UltraGrid required only 250 Mbps of bandwidth, 20 % of CPU utilization during data receiving on an Opteron 245, and provided quite usable quality and stability. Although the compressions process is computationally intensive, it is a highly parallel task, and results in minimal latency impact when compared to other compression techniques. The end-to-end latency of the uncompressed HD video was significantly lower compared to the results reported in [3], which can be attributed to optimization of the software since 2005 and utilization of the new DVS SDK.

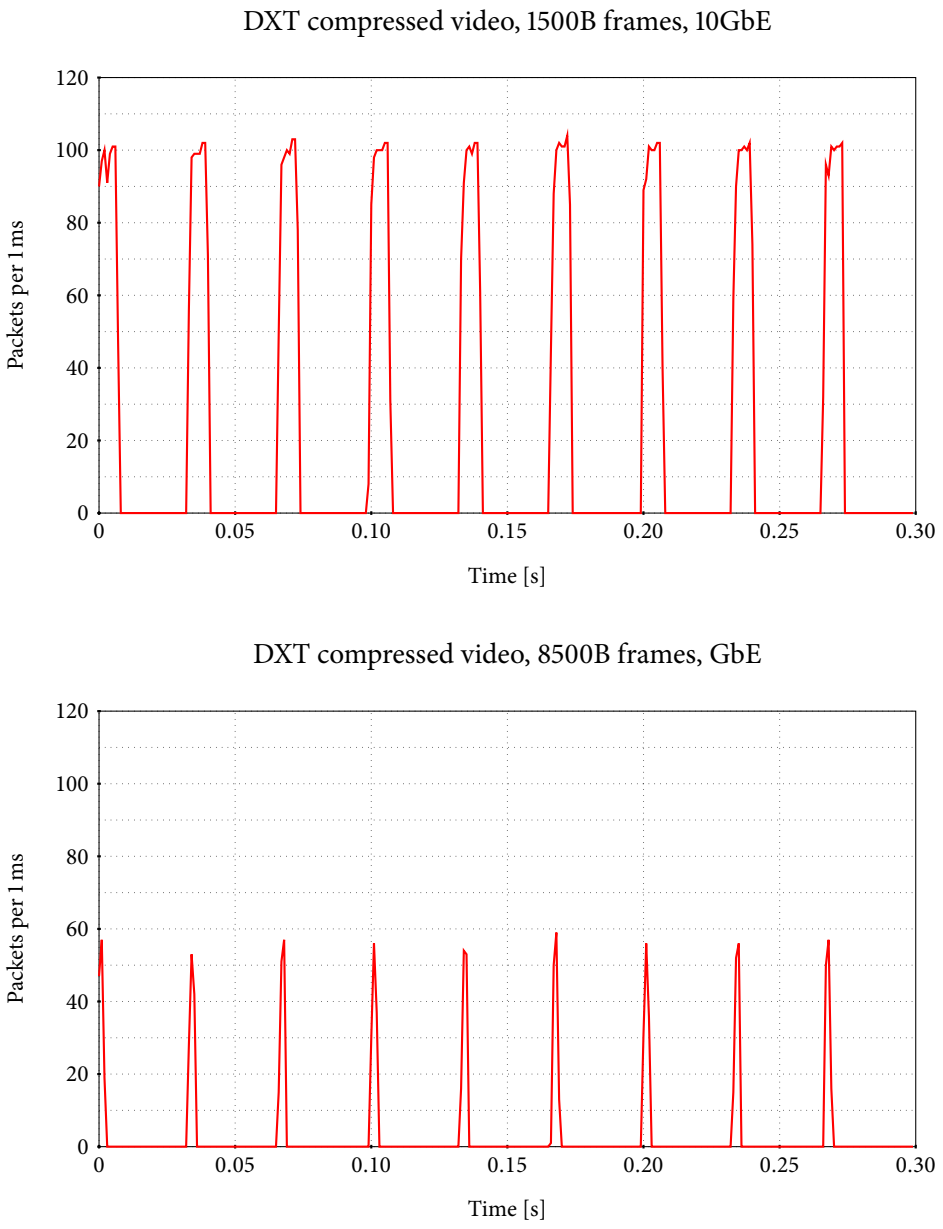

Figure 4: Traffic profiles for various UltraGrid setups.

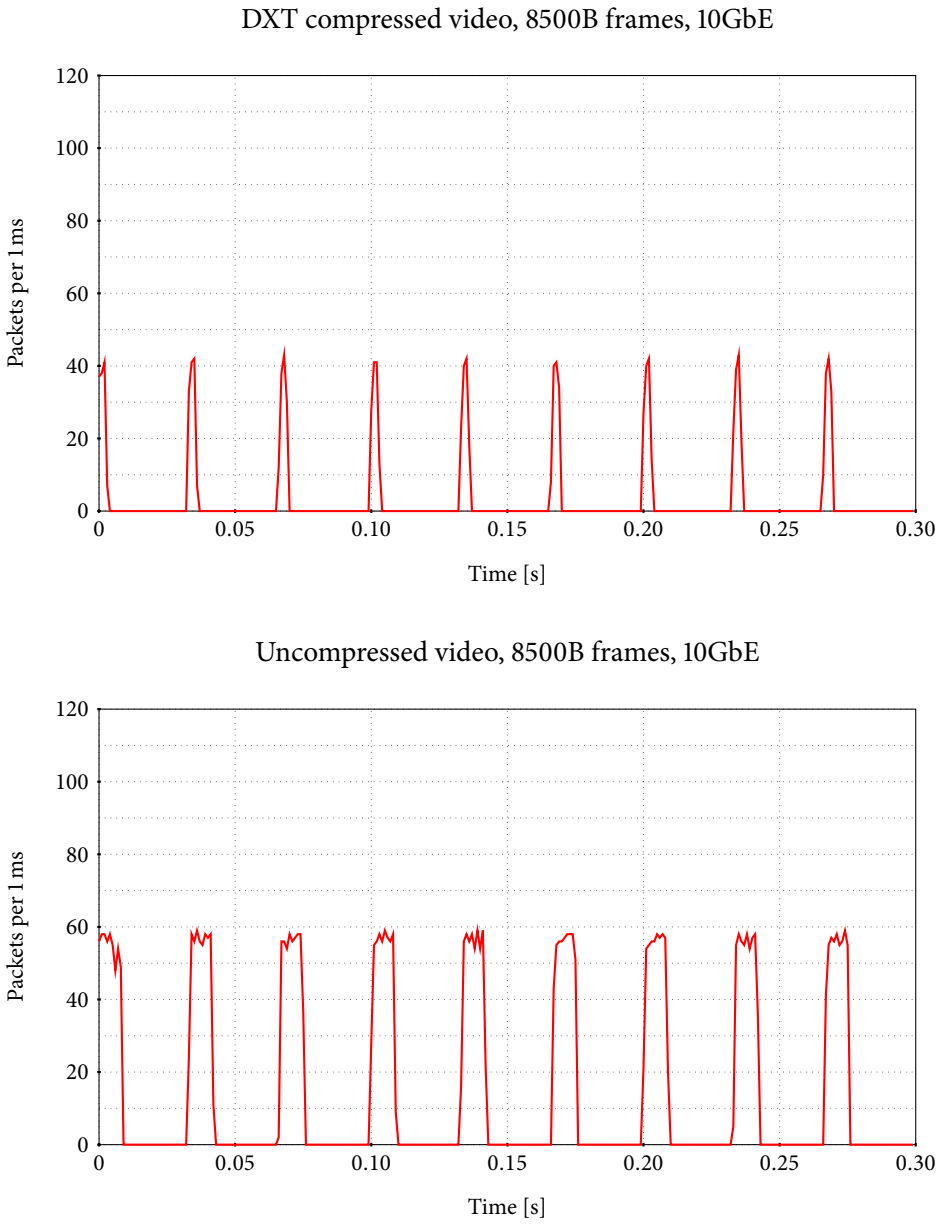

Figure 5: Traffic profiles for various UltraGrid setups.

The whole system was demonstrated at the Global Lambda Infrastructure Forum (GLIF) workshop in Prague (September 2007) as part of the CoUniverse demonstration organized by Masaryk University and during the Center for Computation and Technology demo at Super Computing 2007 (November 2007).

As for the future work, we would like to focus more on real-time compression techniques using GPUs, namely on the new generation of NVidia GPUs that will allow us to use the CUDA programming model. As those newer cards are designed with general purpose calculations in mind, we expect much better performance in terms of read-back latency.

References

[1] Colin Perkins, Ladan Gharai, Tom Lehman and Allison Mankin, "Experiments with Delivery of HDTV over IP Networks", Proceedings of the 12th International Packet Video Workshop, Pittsburgh, PA, USA, April 2002. http://ultragrid.east.isi.edu/publications/pv2002.pdf

[2] UltraGrid—A High Definition Collaboratory, http://ultragrid.east.isi.edu/

[3] P. Holub, L. Matyska, M. Liška, L. Hejtmánek, J. Denemark, T. Rebok, A. Hutanu, R. Paruchuri, J. Radil, and E. Hladká, High-definition multimedia for multiparty lowlatency interactive communication, Future Generation Computer Systems 22 (8) (2006) 856–861.

[4] UltraGrid by Laboratory of Advanced Networking Technologies (ANTLab), https://www.sitola.cz/igrid/index.php/UltraGrid

[5] iHDTV project, ResearchChannel, http://ihdtv.sourceforge.net/

[6] K. Harada, T. Kawano, K. Zaima, S. Hatta, and S. Meno. Uncompressed HDTV over IP Transmission System using Ultra-high-speed IP Streaming Technology. NTT Technical Review, 2003. http://www.ntt.co.jp/tr/0304/files/ntr200304084.pdf

[7] P. Brown, *S3 Texture Compression*, NVidia Corporation, November 2001. http://oss.sgi.com/projects/ogl-sample/registry/EXT/texture_compression_s3tc.txt

[8] S3 Texture Compression, http://en.wikipedia.org/wiki/Texture_compression

[9] Legacy:DXT, http://wiki.beyondunreal.com/Legacy:DXT

[10] J.M.P. van Waveren, Real-Time DXT Compression, Id Software, Inc., 2006. http://cache-www.intel.com/cd/00/00/32/43/324337_324337.pdf

[11] NVIDIA Cg Toolkit, http://developer.nvidia.com/page/cg_main.html

[12] NVIDIA CUDA, http://www.nvidia.com/cuda

[13] GPGPU - General-Purpose computation on GPUs, http://www.gpgpu.org/

[14] FastDXT Library, http://www.evl.uic.edu/cavern/fastdxt/

[15] L. Renambot, B. Jeong, and J. Leigh. Real-Time Compression For High-Resolution Content. *Proceedings of the Access Grid Retreat 2007*, Chicago, IL http://www.evl.uic.edu/files/pdf/ag2007-renambot.pdf