NvPipe: A Lightweight H264-based Hardware Accelerated Image Compression Library

Jie Jiang *
University of Illinois at Chicago

Tom Fogal†
NVIDIA

Cliff Woolley‡
NVIDIA

ABSTRACT

NvPipe is a lightweight image compression library that utilizes H.264 standard and NVIDIA hardware accelerated API. Our benchmark results and experiments with ParaView integration shows superior compression ratio gain and performance of NvPipe over ParaView built-in compressors. It reveals great potential of its application in image streaming over network with limited bandwidth.

1 INTRODUCTION

Image compression is commonly used in distributed visualization systems to overcome limitations of available bandwidth. Conventional image compression approach focuses on removing redundant information within each image without necessarily utilizing the inter-frame coherency.

H.264 is a block-oriented motion-compensation-based video compression standard. It provides high compression capability at the cost of significant computational expenses. NVIDIA provides hardware accelerated H.264 encoder since Kepler archetecture. It enables real-time encoding with minimum overhead through NVIDIA codec API and it has been integrated into multimedia framework FFmpeg.

2 IMPLEMENTATION

NvPipe is built on top of FFmpeg to provide compatibility on systems with either CPU or accelerated GPU.

2.1 Design

NvPipe is implemented on top of FFmpeg to provide compatibility to both CPU-only and GPU-accelerated systems. It uses libx264 codec on CPU and NVIDIA codec on GPU.

The principle of the interface design of NvPipe is simplicity. It expects the user with minimum knowledge about video encoding and requires little effort to set up.

Encoding and decoding with NvPipe is implemented in a synchronous manner without frame latency. Each input frame/packet guarantees a corresponding output packet/frame. Both input and output buffer are allocated and provided by user on host memory.

2.2 Performance

The performance of the compression library is measure by 3 factors which are compression ratio, computational efficiency and image quality preservation.

2.2.1 Compression Ratio

Average bitrate parameter for encoder determines the bandwidth of encoded data, i.e. compression ratio. A guideline for bitrate setting is through formula:

$$bitrate = resolution * \text{ fps} * f_m * 0.07 \quad (1)$$

Where f_m is the motion factor that defines the estimated motion of the video with value of [1...4]. While 1 stands for low motion and 4 for high motion.

Assuming we are given 8bit RGB image. Each pixel contains 3 channel of image data. The compression ratio could be defined by dividing the bitrate to the total generated image data per second.

$$r = \frac{\text{bitrate}}{\text{resolution} * \text{ fps} * f_m * 3 * 8} = 0.07/3/8 = 0.29\% \quad (2)$$

2.2.2 Computational Expenses

Each image compression or decompression call to NvPipe consists of format conversion and encoding or decoding API calls. Computational complexity for both tasks are linear to the totally number of pixels of the input or output image. So we could expect the overall complexity of our compression and decompression call remains linear to the image size.

2.2.3 Image Quality

NvPipe could possibly deteriorate image quality at both stage. The image format conversion applies chroma subsampling that could introduce error on sharp edges. Meanwhile H.264 is also lossy compression. 2.2.3 shows the image recovery quality. For normal visualization images using recommended encoding bitrate only introduces negligible difference. The Structural Similarity (SSIM) between two image on the left is 0.9986. Even after siginificant white noise being added to the original image we start to have perceivable quality drops. The SSIM between two images on the right is 0.7861.

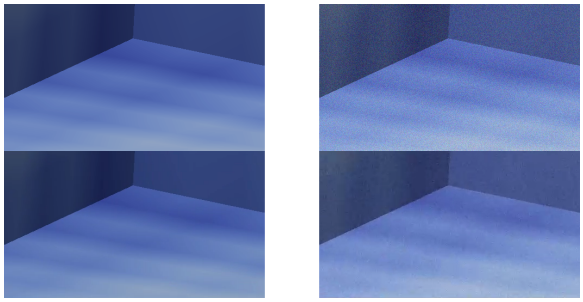

Figure 1: Image quality reservation: images on top are the original while the bottom ones are the recovered. The right column uses the left column image with artificially added white noise.

3 EXPERIMENT

Our experiment setup contains two parts. In part one we benchmark the performance of our library given different resolutions and bitrates. In part two we implement vtkNvpipeCompressor to integrate NvPipe into Paraview. We create a distribute visualization

*e-mail: jjiang24@uic.edu
†e-mail:tfogal@nvidia.com
‡e-mail:jwoolley@nvidia.com

Table 1: Computational Scalability Experiment measuring encoding and decoding time with RGB images

Format	Resolution	Number of pixels	Medium Bitrate (mbps)
XGA	1024x768	786432	1.651
720P	1280x720	921600	1.935
1080P	1920x1080	2073600	4.355
4k	4096x2160	8847360	18.579

use case and compared the performance of NvPipe with ParaViewś native built-in compressor.

3.1 Experiment Setup

Our experiment runs on a linux machines powered by NVIDIA GFORCE GTX 1080 (PASCAL) graphics card.

3.1.1 NvPipe Benchmark

To verify the linear computational complexity property, we have 4 experiment groups set up as 1. For each resolution we also have 3 cases with various bitrate. The medium bitrate is the bitrate calculated using 1 with $f_m = 1$. While the low bitrate and high bitrate are calculated using a factor of 0.5 and 2 respectively. Each session runs 300 circles of consecutive motion image.

3.1.2 ParaView Integration

In this distributed visualization ParaView server use NvPipe to compress rendered image before sending it to the client GUI. Then it decompress the received packet to restore the image. Both server and client are launched on the same machine and sharing resources.

For each execution, we visualize the wavelet dataset and rotate it to render 100 frames image. We compare NvPipe with 3 other libraries. Each native compressor use its optimal compression setting that yields maximum compression ratio. While NvPipe is set to use the default bitrate.

3.2 Results

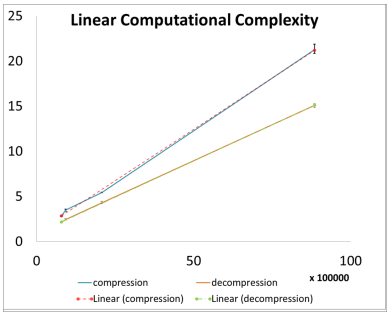

Figure 2: X-axis is the number of pixels while Y-axis is the processing time in milliseconds

From our experiment result 3.2, we verify that compression and decompression time stay linear to the size of image and remains independent of bitrate usage.

NvPipe demonstrates significant advantage over all ParaView built-in image compressor for bandwidth reduction. 2 shows that NvPipe achieves an average compression ratio of 0.32%. While the next best compressor Zlib could only achieve 4.87% compression ratio. From 3.2 and the standard deviation from 2 we observe that NvPipe provides more stable output rate. While images processed by other compressors tends to have varying sizes between frames.

Table 2: Average Compression Ratio

Compressor	NvPipe	LZ4	Squirt	Zlib
r (%)	0.32	15.49	22.31	4.87
std	0.0001	0.0358	0.0662	0.0092

NvPipe exhibits competitive efficiency among other compressors in 3.2. The average overall compression and decompression time is only 12.78ms. And the average encoding and decoding time for FFmpeg API call are 3.938ms and 3.029ms. They yield framerates of 254fps and 330fps respectively. Because of the overhead introduced through FFmpeg wrapper they are expected to be lower than 398fps/658fps released by [4].

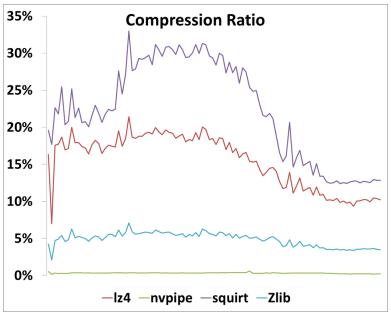

Figure 3: Real-time compression ratio during 100-frame visualization of rotating wavelet data at 1080P

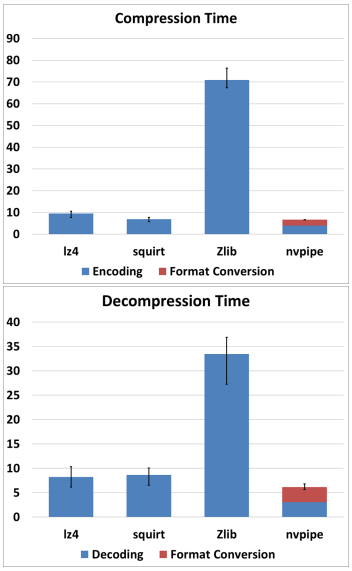

Figure 4: Average compress and decompress time per frame in milliseconds. Image resolution is 1920x1080. For NvPipe we stack the time for image conversion and encoding/decoding time

3.3 Conclusion

NvPipe demonstrate enormous advantage on compression ratio over other image compressors at no extra computational time. It could greatly reduce the communication latency and increase the achievable framerate of remote visualization system with limited bandwidth.

Although for scientific visualization, the compression process produces no perceivable errors. In future research it will definitely be interesting to investigate the lossless HEVC codec available with NVIDIA codec 7.0 API.

REFERENCES

[1] *ffmpeg*.
[2] *h.264*.
[3] *h.265*.
[4] *nvidia codec*.
[5] *ParaView book*.