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

Jie Jiang [*] University of Illinois at Chicago
Thomas Fogal[†] NVIDIA
Cliff Woolley[‡] NVIDIA
Peter Messmer[§] NVIDIA

ABSTRACT

NvPipe is a lightweight image compression library that utilizes H.264 standard and NVIDIA hardware accelerated API. Our benchmark results reveal that the compression and decompression time for 1080P resolution image is around 5 ms while the compression ratio could be around 0.7%. We integrated NvPipe with ParaView's distributed rendering system. NvPipe outperforms ParaView's built-in compressors compression ratio by 30 times while staying competent in efficiency. This high bandwidth saving indicates great potential in remote visualization system 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.

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

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 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. 2.2.2 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.

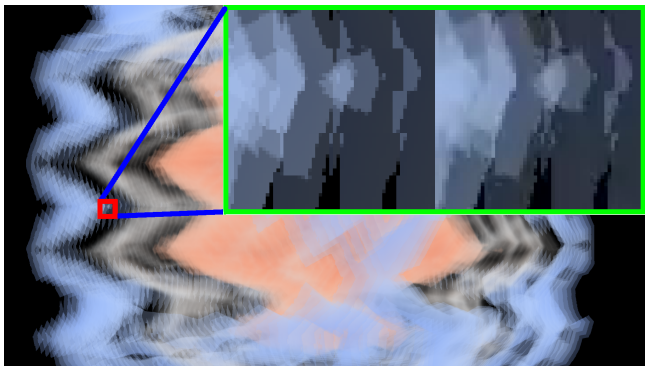

Figure 1: Image quality reservation, 1080P image with area of 60x60 pixels zoomed in. Left block is the original while the right block is the recovered from a compression using $f_m = 4$. Although there is errors between two images, we can see features inside image has been well preserved.

3 RESULTS

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

use case and compared the performance of NvPipe with ParaView's native built-in compressor.

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

3.1 NvPipe Benchmark

We have 4 experiment groups using different image resolutions as 1. For each resolution we run 3 cases with different 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 images of a rotating wavelet cube.

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

Resolution	Bitrate(mbps)	Compress(ms)	Decompress(ms)
1024x768	1.651	2.8057	2.1170
1280x720	1.935	3.4793	2.4304
1920x1080	4.355	5.4358	4.2876
4096x2160	18.579	21.2504	15.0976

3.1.1 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.

Given the total number of million pixels to be n_p and bitrate multiplication factor to be f_b. Linear regression analysis of compression time t_c and decompression time generates the following models: $t_c = 2.27n_p + 0.16f_b + 0.86$ and $t_d = 1.60n_p + 0.02f_b + 0.89$ with $R_c^2 = 99.92\%$ and $R_d^2 = 99.97\%$. Performance is linear to total number of pixels in the image with trivial influence from the bitrate multipliccation factor used. 1 gives an indication of recommended bitrates and average performance for different resolution.

3.2 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.

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

Table 2: Average compression ratio of 1080P images with randomly rotating volume data. The lower ratio means better compressed.

	LZ4	Squirt	Zlib	Zlib_fast	NP_hw	NP_sw
ratio(%)	79.4	69.8	30.7	30.4	0.70	0.65
std dev	0.061	0.061	0.033	0.034	0.001	0.001

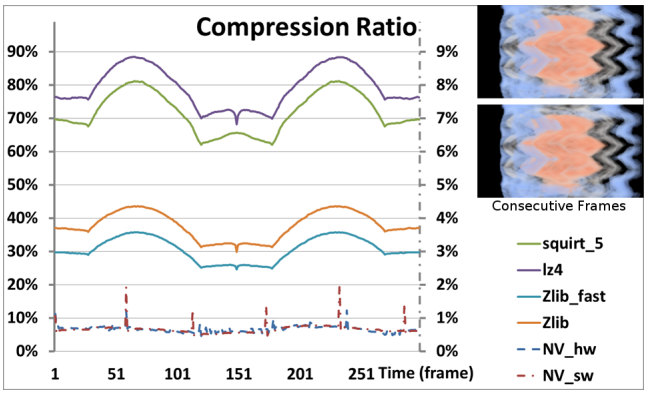

Figure 2: In-situ compression ratio during 300-frame of 1080P visualization of volume data rotating at 1.2° per frame: i. NvPipe compressor has a huge compression ratio advantage of below 1% versus ParaView compressors staying between 25% to 90%. ii. ParaView compressors are highly sensitive to data, notice the symmetric pattern due to the 360° rotation, while NvPipe compression ratio stays stable except for the peak due to the key frame every 60 frames.

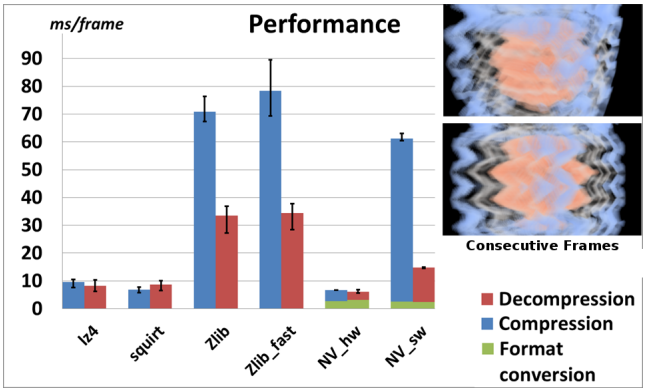

Figure 3: Average compress and decompress time of 1080P image with randomly rotating volume. For NvPipe we stack the time of format conversion and compress/decompress time. NvPipe software approach takes much longer time for image compression in exchange for superior compression ratio. While NvPipe hardware compressor outperforms other compressors in both perspective 2.

NvPipe provides more stable output rate. While images processed by other compressors tends to have varying sizes between frames.

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].

4 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*.
[6] *ssim*.