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

Category: Research

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 [7]. 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 and it has been integrated into multimedia framework FFmpeg [2].

2 Implementation

NvPipe is built on top of FFmpeg to provide compatibility across systems with or without hardware accelerated H264 compression. Although NvPipe accepts images with format RGB/RGBA/YUV, it will convert the image into NV12 to before passing it to FFmpeg/NVENC. As is the decompressed image.

2.1 Design

NvPipe could use both hardware accelerated NVIDIA codec and libx264 codec. Using widely applicable default values requires little effort to integrate into existing systems. The library simplifies exposed interfaces by providing generic default configuration. For example the default group of pictures (GOP) size is 60 and I:P frame ratio is 1:59.

Compression and decompression with NvPipe is implemented in a synchronous manner without frame latency. Each input frame/packet will return an output packet/frame. NvPipe handles only image compression/decompression. It leaves memory allocation and streaming to user for ease of integration.

2.2 Performance

The performance of the compression library is measured by 3 factors: computational efficiency, compression ratio and image quality preservation.

2.2.1 Compression Ratio

Compression ratio is determined by the average bitrate parameter of NvPipe. A guideline for bitrate setting is through Kush Gauge [4]:

$$bitrate = resolution * \text{fps} * f_m * 0.07 \tag{1}$$

Where f_m is the motion factor that defines the estimated motion of the video with value 1 to 4. While 1 indicates lowest motion and 4 the highest. NvPipe default bitrate uses eq. (1) with $f_m = 4$.

Assuming we are given 8bit RGB image. Each pixel contains 3 channels. The compression ratio could be calculated by dividing bitrate to total image size per second.

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

2.2.2 Image Quality

NvPipe could introduce artifacts during conversion and compression stage. Format conversion from RGB to YUV applies chroma subsampling which introduces artifacts on sharp edges. And NvPipe H.264 compression is configured as high performance low latency mode that is not lossless. fig. 1 shows that very fine features are decently preserved through NvPipe.

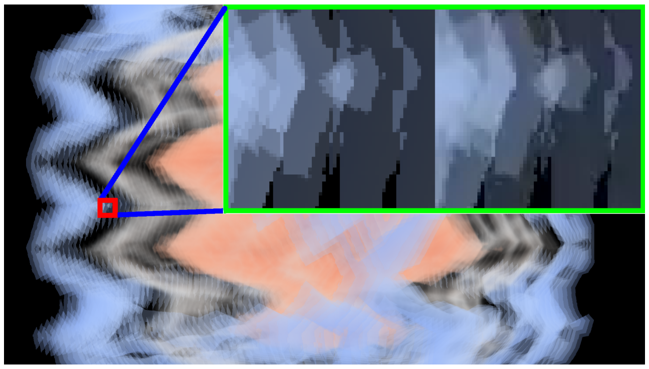

Figure 1: 1080P image rendered from randomly rotating volume data and compressed using bitrate with $f_m = 4$. Average stractural similarity (SSIM) between the images are 98% [6]. Looking at a the block of 60x60 pixels. The left block is the original image while the right block is the recovered image, we can see fine features has been well preserved.

3 Results

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

We first run the benchmark of NvPipe on RGB images. We set up experiment groups using 4 common resolutions. For each one we run 3 cases with low bitrate, medium bitrate and high bitrate settings. The medium setting uses eq. (1) with $f_m = 1$ to calculate the bitrate. While low and high uses f_m of 0.5 and 2 respectively. Each case runs 300 cycles of compressing and decompressing of consecutive images generated with moving color pattern.

The second part we integrate NvPipe into ParaView and compare the performance of NvPipe with and without hardware acceleration to built-in compressors utilizing lz4, squirt and Zlib. NvPipe uses high bitrate setting with $f_m = 4$ for best image quality. For configuration of ParaView compressors: Lz4 uses quality level 0. Squirt uses compression level 5. Zlib uses compression level of 9 for slow compression with highest compression ratio, color space reduction level of 5 and alpha channel stripping. Zlib fast uses compression level of 1 for fast compression with lowest compression ratio while other parameters remain the same. The experiments use 300 cycles of 1080P RGBA images of rotating volume data. One case uses fixed small rotation angle of 1.2° per frame. And the other uses large rotation angle of 40° per frame.

3.1 NvPipe Benchmark

The benchmark experiment shows linear scalability of NvPipe to image size. Given the total number of million pixels of the image to be n_p and motion factor used for bitrate calculation to be f_m. Linear regression analysis of compression time t_c and decompression time t_d generates the following models: $t_c = 2.27n_p + 0.16f_m + 0.86$ and $t_d = 1.60n_p + 0.02f_m + 0.89$ 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.

table 1 lists general bandwidth requirements and performance for common resolution.

Table 1: Computational scalability benchmark measuring average compression and decompression time per frame during 300 frames of RGB images with moving pattern

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

3.2 ParaView Integration

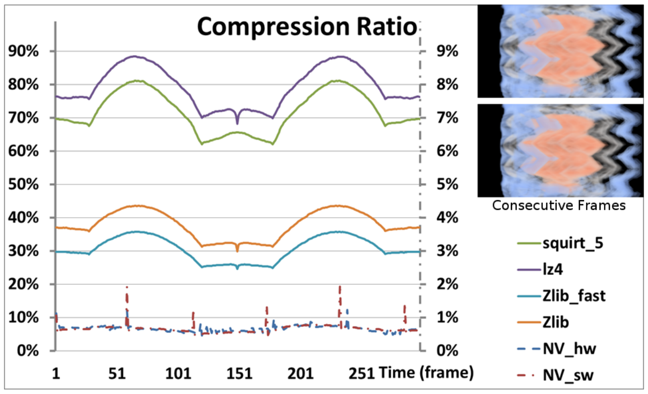

Figure 2: Compression ratio during small rotation experiemnt: i. NvPipe compressor has a huge compression ratio advantage of below 1% versus ParaView compressors staying between 25% to 90%. ii. ParaView compressors are highly sensitive to data, notice the symmetric pattern due to the 360° rotation, while NvPipe compression ratio stays stable except for the peak due to the key frame every 60 frames.

Distributed ParaView server uses NvPipe to compress RGBA image buffer and streams compressed image to display client. Our experiment launches both server and client on the same machine to eliminate network latency.

fig. 2 shows the compression ratio over time of a slowly rotating volume data. It exhibits the image compression ratio benefit and stability of NvPipe over built-in compressors in ParaView under normal interactive visualization.

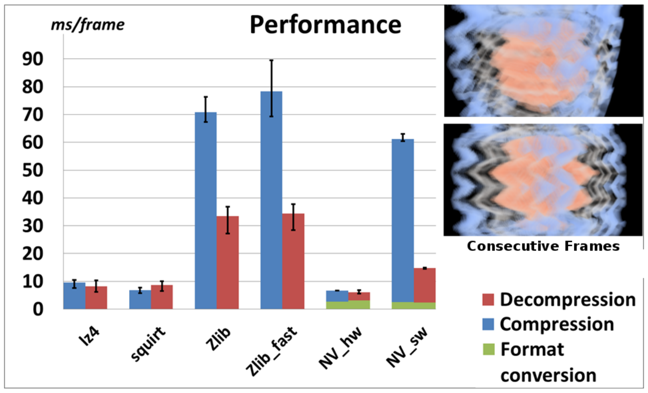

Figure 3: Average compress and decompress time of large rotation experiment. For NvPipe we stack the time of format conversion and compress/decompress time. NvPipe software approach takes much longer time for image compression. While NvPipe hardware compressor outperforms other compressors.

For more generic case, our experiments runs 5 sets of 300 frames of randomly rotating volume data at resolution of 1080P. The efficiency in fig. 3 shows that although software-based NvPipe suffers from slow compression, the hardware accelerated NvPipe compressor outperforms other tested compressor in both encoding and decoding time.

The compression ratio on the first 2 rows in table 2. NvPipe has 43 times better compression ratio comparing to fast Zlib, which is the best compressor tested in Paraview. The theoretical aggregated latency could be calculated through eq. (3), where communication time $t_{network} = compressed\, image\, size / BW$.

$$T_{latency} = t_{compression} + t_{decompression} + t_{network} \quad (3)$$

Assume we have 100mbps available bandwidth, for each compressor image size $s_i = 1920 * 1080 * 8 * 3 * ratio$ we could calculate the communication time for each compressor. Combining data from fig. 3 we get the overall latency in table 2.

Table 2: Average compression ratio and theoretical latency of large rotation experiment. The lower ratio means better compressed. H.264 has much better compression ratio as well as higher computational complexity as can be seen from the encoding time of NvPipe software. For the overall latency NvPipe hardware approach has 4 times speed up comparing to the second fasted compressor LZ4

	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
comm(ms)	39.49	34.71	15.15	15.28	0.35	0.32
Latency(ms)	57.23	50.22	119.45	128.02	13.14	76.26

4 Conclusion

NvPipe demonstrates great potential with its image quality preservation, compression ratio and efficiency. It could reduce overall latency and improve framerate for distributed visualization system with limited bandwidth. In future research it will be intereting to investigate the performance of lossless H.264 configuration and HEVC standard provided with NVIDIA codec 7.0 API.

REFERENCES

[1] *nvidia codec*.

[2] Ffmpeg, 2016. [Online; accessed 22-August-2016].

[3] A. Henderson, J. Ahrens, C. Law, et al. *The ParaView Guide*. Kitware Clifton Park, NY, 2004.

[4] I. Iszaidy, R. Ahmad, N. Kahar, M. Rahman, and S. N. Yaakob. The investigation of bitrate effects on the video quality of plecord system.

[5] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[6] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[7] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.