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

Category: Research

Abstract

Hardware video encoding can lower the perceived latency for remote visualization. NvPipe is a lightweight library that simplifies the use of NVIDIA's video compression hardware. We achieve overall latencies below 15ms with compression ratios of approximately 140:1 and is theoretically capable of supporting 60 FPS streaming over 10mbps network. To verify its applicability in real world scenarios, we integrated NvPipe into ParaView. NvPipe offloads the encoding within ParaView to the GPU and compresses images 30 times better as well.

1 Introduction

Image compression is commonly used in remote visualization systems to create a smoother user experience over low-bandwidth links. Current systems generally consider each image in isolation even though image differencing approaches can yield considerable data savings.

H.264 is a block-oriented motion-compensation-based video compression standard [7]. H.264 provides high compression ratios at the cost of significant computational expense. Since the Kepler architecture, NVIDIA provides a hardware-accelerated H.264 encoder. It enables real-time encoding with minimal overhead and it has been integrated into the multimedia framework FFmpeg [2].

2 Design

NvPipe can utilize NVIDIA's hardware encoder or the libx264-based software encoder. We specifically designed NvPipe to abstract away most video compression configuration. A detailed discussion of H.264 parameters is covered in [4]. For NvPipe we have configured the encoder to use settings reasonable for visualization. For example, video compression normally has a lag time on the order of tens of frames. For NvPipe we have reduced the lag time to a single frame and guaranteed that the encoder produces a single buffer for every input image.

NvPipe handles only image compression and decompression. It leaves streaming to user for ease of integration.

2.1 Performance

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

2.1.1 Compression Ratio

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

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

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 Equation (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 total image size per second to bitrate.

Compression ratio:

$$r_c = \frac{resolution * \text{framerate} * 3 * 8}{bitrate} = \frac{3 * 8}{0.07 f_m} = \frac{342}{f_m} : 1 \quad (2)$$

2.1.2 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. Figure 1 shows that very fine features are decently preserved through NvPipe.

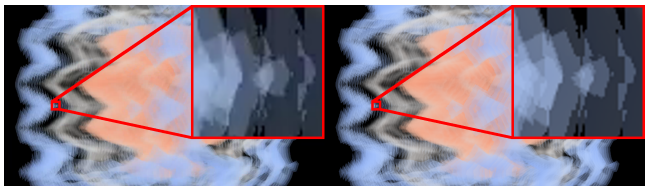

Figure 1: 1080P image rendered from randomly rotating volume data and compressed using bitrate calculated with $f_m = 4$. Average structural similarity (SSIM [6]) between the images are 98%. Looking at a the block of 60x60 pixels. The right block is the original image and the left block is the recovered 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. Our benchmark is measure using NVIDIA profiling tools.

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 Equation (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 uses 1080P RGBA images generated from rotating volume data. The high coherency case applies a fixed small rotation angle of 1.2° to the volume data per frame. The low coherency uses larger rotation angle of 45° per frame. For each case we run 5 full test each with 300 cycles.

3.1 NvPipe Benchmark

Our experiment data shows that NvPipe compression computational complexity is lineary towards the total number of pixels. And the compression time is independent of image data. 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

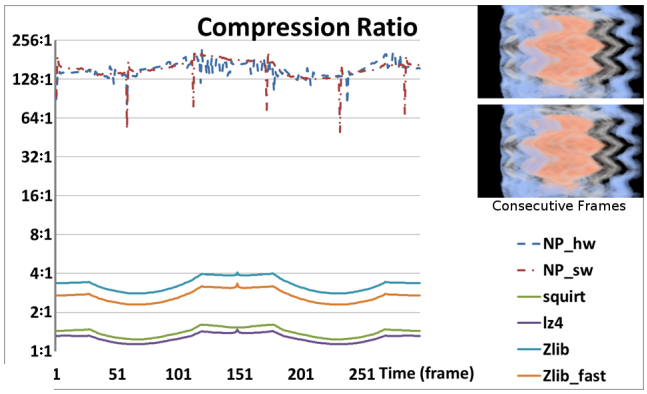

Figure 2: Compression ratio during high coherency experiemnt: i. NvPipe compressor has a huge compression ratio advantage of above 128:1 versus ParaView's' compressors staying between 1.3:1 to 3.4:1.

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.

Figure 2 shows the compression ratio over time of our high coherency experiment. It exhibits the image compression ratio benefit and stability of NvPipe over built-in compressors in ParaView under normal interactive visualization.

For more general case, our low coherency experiment shows the computational efficiency in Figure 3. 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 Equation (3), where communication time $t_{network} = \frac{\text{compressed image size}}{BW}$.

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

Assume we have 10mbps available bandwidth, with average compressed image size $\frac{1920*1080*8*3}{r_c}$ we could calculate the communication time for each compressor. Combining data from Figure 3 we get the overall latency in Table 2.

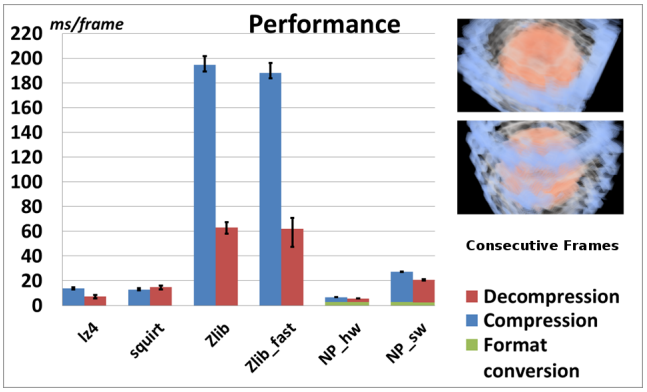

Figure 3: Average compress and decompress time of large rotation experiment. For NvPipe we stack the time of format conversion and compress/decompress time. NvPipe software approach takes much longer time for image compression. While NvPipe hardware compressor outperforms other compressors.

Table 2: Average compression ratio and theoretical latency using 10mbps network of low coherency 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
r_c	1.29:1	1.43:1	3.43:1	2.78:1	144:1	152:1
$t_{network}$(ms)	483.2	434.1	181.1	223.7	4.3	4.1
$T_{latency}$(ms)	504.2	461.8	438.8	473.8	16.6	51.9
FPS	2.0	2.2	2.3	2.1	60.1	19.3

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 NvENC API. Other areas of investigations are towards more asynchronocity in this compression process: NvENC is entirely asynchronous to the rest of the GPU, with the potential to completely hide the encoding latency. On the other hand, software only approaches will not be able to fully hide that cost.

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.