Using a Commodity Hardware Video Encoder for Interactive Video Streaming

Martin Alexander Wilhelmsen, Håkon Kvale Stensland, Vamsidhar Reddy Gaddam,
Asgeir Mortensen, Ragnar Langseth, Carsten Griwodz, Pål Halvorsen
University of Oslo / Simula Research Laboratory

***Abstract*—Over the last years, video streaming has become one of the most dominant Internet services. A trend now is that due to the increased availability of high-speed internet access, multimedia services are becoming more interactive and immersive. Examples of such applications are both cloud gaming [4] and systems where users can interact with high-resolution content [1]. Over the last few years, hardware video encoders have been built into commodity hardware. We evaluate one of these encoders in a scenario where we have individual streams delivered to the end users. Our results show that we can reduce almost half of the CPU time spent on video processing, while also greatly reducing the power consumption on the system. We also compare the visual video quality and the frame size of the hardware based encoder, and we find no significant difference compared to a software based approach.**

I. Introduction

Video streaming is the dominant service on the Internet today giving extreme resource requirements in order to process and deliver data to billions users. In this respect, we have researched the efficiency of the sender side processing, and in particular the encoding.

Providing efficient video processing is important in many scenarios like on-demand services such as YouTube and live services such as sport events. As a case study in this paper, we have targeted an interactive and immersive streaming scenario, enabled by the growing availability of high-speed Internet access. In this respect, we already see systems where users can interact with live high-resolution content, e.g., some systems allow end-users to pan, tilt and zoom with a virtual camera into panorama videos [1] and new cloud gaming systems run the games entirely on the server providing only the output as a video stream to the clients [4]. Thus, in such scenarios, each user is delivered a personalized video stream requiring enormous processing demands on the server.

A lot of research have focused on reducing the resource and power requirements for encoding and transcoding videos with general purpose processors [5]. Dedicated hardware solutions for encoding and transcoding video have also been investigated, but they are usually very expensive and lack flexibility. Encoding is therefore often done in software across multiple servers in clusters which consume significant amounts of resources and power [2].

Nevertheless, over the last few years, the hardware video encoders have become much more flexible, and they have been built into commodity hardware such as graphics processing units (GPUs) from Nvidia and general purpose processors (CPUs) from Intel. In this paper, we evaluate the efficiency of Nvidia's NVENC. The reason for choosing NVENC is support for Linux and a free SDK, feature wise i is very similar to the QuickSync hardware encoder on new Intel CPUs. NVENC is one of the new commodity hardware-based video encoders with support for multiple resolutions and quality settings. One of the scenarios for NVENC is ending the display output from computer games and stream it to handheld devices such as tablets. We evaluate a scenario where individual streams are delivered to the users in terms of video quality, CPU usage,interactivity/encoding latency, bandwidth requirement and power consumption. Our experiments show that video can be delivered at similar quality as with a software encoder, but with much less CPU utilization and power consumption. The hardware encoder is also able to serve more streams in parallel.

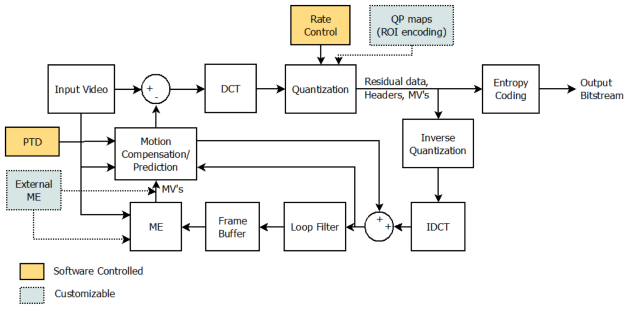

Figure 1: NVENC block diagram [3]

II. Commodity Hardware Encoding

Over the last few years, dedicated hardware H.264 encoders have appeared in commodity hardware such as CPUs and GPUs. Hardware video encoders are not new; it is commonly integrated in system on chips found in mobile phones. These encoders are often limited to encoding video from the built-in camera on the device. Video encoders are also found in the broadcasting market, however, these encoders are often expensive and not flexible with regards to encoding parameters.

One of the first commodity hardware H.264 encoder was released by Nvidia in 2012 [3] when the Kepler GPU architecture was launched. This first generation encoder was named NVENC. The second generation was released in

2014 with the Maxwell GPU architecture. NVENC is a fully dedicated hardware encoder which does not use the 3D engine on the GPU, leaving it free to perform other compute or 3D tasks. The encoder described in figure 1 supports resolutions up to 4096x4096 (4K resolution) at 30 frames per second, and can encode multiple streams in parallel. Support for advanced parts of H.264 such as B-frames, MVC and CABAC entropy coding are also present, and the hardware have the possibility of using an external motion estimation. The encoder API has several presets defined for different encoding scenarios. The two presets we will use in this papers are the high quality (HQ) preset and the low latency (LL) preset. The main differences between HQ and LL is that HQ enables CABAC and B-frames.B-frames usually enable better compression since they are generated using both preceding and following frames. However, they add additional latency before a frame can be delivered to the client.

Since the rendering often takes place on the GPU, having the H.264 encoder close saves time otherwise spent transferring the image over the PCIe bus, it also reduces the size of the transfers, since a 1920×1080 24bit RGB image is 6MB uncompressed, while with NVENC, we only need to transfer the encoded image which is about 13KB. This enables us to send the image to the client earlier reducing the interaction latency.

III. System Design

The video delivery system is interactive, and the individual users can have a personalized virtual camera, enabling the users to manually control the view. The data required for viewing the entire panorama of a match is over 6GB compressed H.264 video making it inaccessible to the end-users. In addition, a high-end CPU or GPU is also required to decode the 4450x2000 panorama video stream. Our solution is to make the delivery pipeline execute on the server side. All we need on the client is a web-browser with support for WebSockets and HTML5 video elements. The resolution of the virtual camera that are delivered to the browser on the end-device can be individually adjusted to adapt to either bandwidth requirements or screen limitations if the device is a smartphone or tablet device. In this study, we use 720p and 1080p resolution for video delivery to the clients.

A detailed overview of the delivery pipeline can be seen in figure 2. We use NVENC to encode the personalized H.264 streams. The panorama is rendered to framebuffer using OpenGL. Next, the color buffer is processed by an OpenGL Compute Shader transforming it from standard RGB to NV12 (YUV 4:2:0 with interleaved chroma channels), which in turn is used as input for NVENC to generate the final H.264 video stream.

The clients connect to a standard HTTP server that returns an HTML page containing a JavaScript for creating and establishing a WebSocket between the client and the server

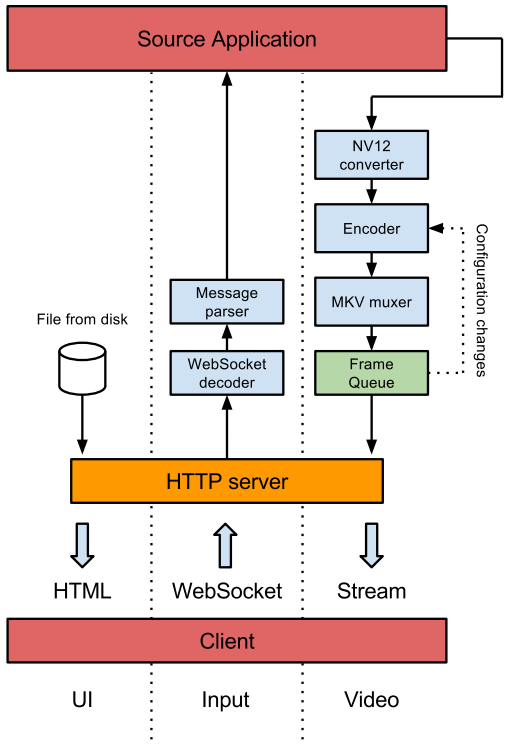

Figure 2: Block diagram of delivery pipeline

for streaming mouse and keyboard input directly to the visual view application. The HTML5 video element on the page connects to the H.264 video stream. When the client connects, it will receive a general matroska video (MKV) header containing the frame rate, format and the size. As the NVENC video encoder outputs video, the video frames will be added to a frame queue, where the client controller can fetch the frames. If the client is on a high latency and/or low bandwidth network, the client controller may ask the stream generator (frame queue) to tune the bit rate. It is also possible to ask for a key frame if a frame is lost or the client is lagging behind, clearing the frame queue. The NVENC encoder has support for dropping frames, however, we do not currently utilize this feature.

IV. Experiments

The computer used for our experiments has a quad-core Intel Core i7-2600 processor clocked at 3.40 GHz, 16 GB dual-channel main memory, and an Nvidia GeForce GTX 750 Ti with 2 GB memory, based on the Maxwell GPU architecture. The x264 encoder used is the latest version that comes with Ubuntu 12.04. All the x264 encoding results are tuned for zero latency to match our interactive scenario.

The video delivery pipeline used for experiments in this paper is the pipeline in our Bagadus soccer analysis system. The Bagadus soccer analysis system integrates a video pipeline, sensor system and annotation system. The system

is described in detail in [6]. In this paper we will focus on the delivery part of the video pipeline. The video system in Bagadus is described in [1] delivers a stitched panorama video. The panorama consists of five 1080p cameras that are mounted vertical in a matrix. The stitched panorama resolution is 4450x2000, which in turn is used to create virtual cameras for the users of the system.

A. Video quality

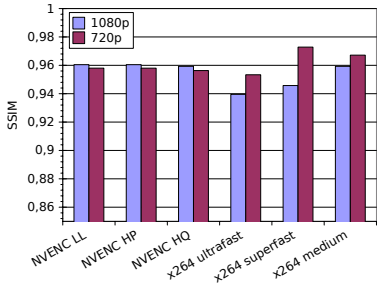

Figure 3: Video quality measured with SSIM.

In the first experiment, we are comparing the quality of different presets with SSIM as the quality metric (figure 3). We found that the quality of all three NVENC presets are similar, and they perform between x264 superfast and medium. We also used subjective testing of the videos, and we found that x264 ultrafast did not produce adequate video quality compared to the other presets. There was no noticeable difference between the other presets.

B. Frame size and bandwidth

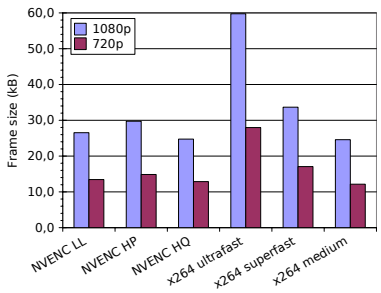

Figure 4: Size of the frames produced by the encoder.

In figure 4, we compare the size of every frame encoded and found that the bandwidth savings of using NVENC compered to x264 ultrafast were significant. The results produced by NVENC are again comparable with x264 medium and superfast. When we compare NVENC LL and HQ, we see that the usage of B-frames in HQ does not result in notably changes in file size. Based on the frame size finding and the video quality tests, we use the superfast preset of x264 and the LL preset of NVENC for further tests.

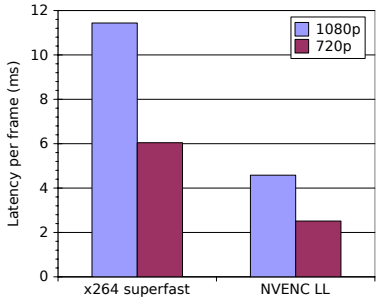

Figure 5: Latency of encoding a single image

C. Encoding latency

The latency of the encoder is measured inside the running system, i.e., it comprises the time it takes from when a frame is sent into the encoder component until it is ready. In figure 5, we observe, as expected, that NVENC is more than twice as fast compared to x264 superfast. The times for NVENC include downloading the encoded stream while x264 does not include the time for downloading the NV12 to system memory as we were able to hide in an earlier pipeline stage. Typically, downloading an OpenGL buffer is done using `glMapBuffer` which is known for being notoriously slow and stall the pipeline. Our readings show this to be around 9 ms. Similarly, NVENC will need to synchronize OpenGL buffers with CUDA buffers, but we found this to be as low as 45 µs.

D. Energy Usage

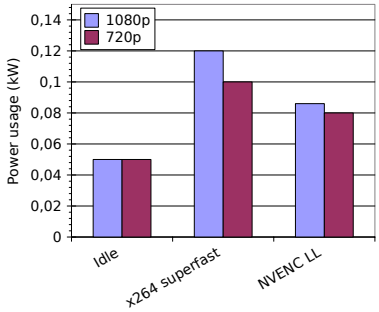

Figure 6: Power usage of system while running pipeline

The power measurements can be seen in figure 6. The data is collected by monitoring the entire system while running our delivery pipeline. To measure the power consumption, we use an APC Rack PDU 2G specified to have an accuracy of $\pm$ 3% of the reading. From the measurements, we observe that the NVENC pipeline uses considerably less energy compared to the software encoder. We also observe that the power usage using NVENC does not increase much when we change the resolution.

E. CPU usage on delivery pipeline

The final test in figure 7 is the CPU usage of the entire delivery pipeline. The most obvious gain from using an

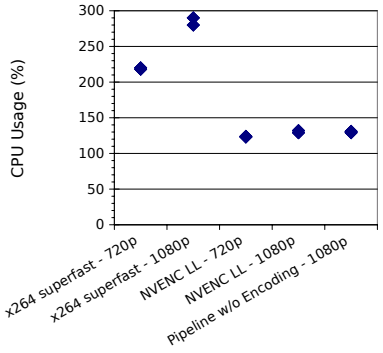

Figure 7: Total CPU usage of delivery pipeline

external video encoder such as NVENC would be to lower the CPU usage. We have done a single test where the entire encoding part of the pipeline was removed from the code. We can see that this gives the same CPU usage as when using NVENC, meaning that NVENC does not load the CPU during encoding.

V. Discussion

We have also tried to test scaling of the system, even though our test setup only has a single GPU. The x264 software encoder was able to deliver two 1080p streams in parallel while maintaining the 30 frames per second framerate. On the GPU with NVENC, we were able to deliver four concurrent session while still maintaining the framerate. The tests were simply done by running several instances of the application and does not utilize that a lot of resource sharing is possible in the video source application. During raw performance tests of NVENC we found that it should be able to encode more than 200 1080p frames per second. In our test case, CPU resources for creating the virtual view were actually the limiting factor.

For scaling individual streams to millions of users concurrently, hardware video encoding with commodity hardware is a step in the right direction. Encoding of video is a compute intensive workload, and not all the parts of such workload have potential for parallelization. New video codecs such as HEVC will increase this complexity even further. Dedicated hardware can help offload parts of this compute intensive workload. Another advantage with using hardware is the reduced energy consumption. This means that the data center is cheaper to run, and requires less effort to cool, which again leads to reduced maintenance.

We have used interactive streaming as our case study, but our solution can be used in several other scenarios as well. For example, adaptive HTTP streaming is frequently used today for both live and on-Demand video delivery. The adaptiveness is achieved by encoding the video stream in multiple bit-rates, temporal and spatial resolutions, as well as conversion from one video coding to another, i.e., each video is encoded multiple times. For example, taking into account that one hour of video to is uploaded YouTube every second (January 2012) and that YouTube alone delivers more than four billion video views globally every day [7], one can clearly see the need for efficient video encoding systems.

VI. Conclusion

Our experiments show that NVENC is able to deliver a 30Hz stream at 1080p for real time application streaming. The resulting latency from interaction to display of the decoded image at a client using the Google Chrome web browser is about 100 ms + network latency. Here, most of the time is spent in the browser's buffer, creating a subjectively smooth experience for the end-user.

Using NVENC, we both save power and are able to run more instances of the same application on a single computer. The visual quality and the compression rate produced by NVENC are comparable to x264 at the superfast preset mode, which is acceptable for the live streaming use-case. For other cases where encoding latency is not important, x264 at higher presets may still be preferred.

Further and ongoing work includes increasing the frame rate at which the virtual view is rendered to 60 Hz which will lower the user interaction latency, but will also increase the performance requirements. We are also investigating how our pipeline can scale over multiple GPUs on multiple computers.

Acknowledgements

This work has been performed in the context of the *iAD* centre for Research-based Innovation (project number 174867) funded by the Norwegian Research Council.

References

[1] V. R. Gaddam, R. Langseth, S. Ljødal, P. Gurdjos, V. Charvillat, C. Griwodz, and P. Halvorsen. Interactive zoom and panning from live panoramic video. In *Proc. of NOSSDAV*, pages 19–24. ACM, 2014.

[2] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai. Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices. In *Proc. of NOSSDAV*, pages 33–38. ACM, 2012.

[3] NVIDIA. Nvenc – nvidia kepler hardware video encoder, 2012.

[4] OnLive Cloud Gaming. http://games.onlive.com, 2014.

[5] M. Song, Y. Lee, and J. Park. Cpu power management in video transcoding servers. In *Proc. of NOSSDAV*, pages 91:91–91:96. ACM, 2014.

[6] H. K. Stensland, V. R. Gaddam, M. Tennøe, E. Helgedagsrud, M. Næss, H. K. Alstad, A. Mortensen, R. Langseth, S. Ljødal, O. Landsverk, C. Griwodz, P. Halvorsen, M. Stenhaug, and D. Johansen. Bagadus: An integrated real-time system for soccer analytics. *ACM Trans. Multimedia Comput. Commun. Appl. (TOMCCAP)*, 10(1s):14:1–14:21, Jan. 2014.

[7] YouTube statistics. http://youtube.com/t/press_statistics, 2012.