Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2014, Article ID 183716, 12 pages
http://dx.doi.org/10.1155/2014/183716

Research Article

4kUHD H264 Wireless Live Video Streaming Using CUDA

A. O. Adeyemi-Ejeye and S. Walker

Access Networks Laboratory, School of Computer Science and Electronic Engineering, University of Essex, Wivenhoe Park, Colchester, Essex CO4 3SQ, UK

Correspondence should be addressed to A. O. Adeyemi-Ejeye; aoteje@essex.ac.uk

Received 1 November 2013; Accepted 24 December 2013; Published 16 February 2014

Academic Editor: Martin Fleury

Ultrahigh definition video streaming has been explored in recent years. Most recently the possibility of 4kUHD video streaming over wireless 802.11n was presented, using preencoded video. Live encoding for streaming using x264 has proven to be very slow. The use of parallel encoding has been explored to speed up the process using CUDA. However there hasnot been a parallel implementation for video streaming. We therefore present for the first time a novel implementation of 4kUHD live encoding for streaming over a wireless network at low bitrate indoors, using CUDA for parallel H264 encoding. Our experimental results are used to verify our claim.

1. Introduction

Video streaming in wireless networks has witnessed improvements in recent years. The recent adoption of 4kUHD (3840 × 2160) video resolution now makes it necessary to investigate the possibilities of streaming such video resolution in a wireless environment due to its popularity. Video display at such resolutions can be done with the aid of SAGE [1, 2] (scalable adaptive graphics environment), tiled displays (using more than one display output unit to produce the required resolution), CAVE [3] (cave automatic virtual environment) a one to many presentation systems, and more recently commercial production of 4kUHD television sets. Currently streaming at this resolution is normally done using compressed formats [4–7], as the minimum requirement for uncompressed UHD video starts at 2.39 Gb/s for 8-bit 4 : 2 : 0 subsampling at 24 frames per second. Streaming uncompressed video at this resolution has been carried out in both wired networks by [8, 9] and in wireless [10], using four wireless 60-GHz parallel channels. However storage and bandwidth limitations make it harder for deployment of uncompressed video streaming applications. The analysis of H.264 video transmission over 802.11b [11, 12], 802.11g [13, 14], and 802.11n [15] has provided insight into how the H264 encoder can function in a lossy network such as wireless. Previous studies done only show video resolutions no larger than full-high definition (1080p) with the exception of our previous work [7], which studied the use of preencoded video for live streaming over 802.11n wireless network. We therefore take this further by implementing a live streaming encoder which can accept both uncompressed video and live capture from a 4kUHD camera.

2. 4kUHD Wireless Video Streaming

Quality of experience is getting more attention at the moment due to the rapid increase in high quality multimedia content especially with the adoption of higher video resolution content (such as 4kUHD); on one hand end-users expectation increases, while transmission requirements also present new problems [16]. A major problem here is how best to utilize encoding parameters for both compression efficiency and intended transmission medium. Wireless networks communication is very unstable due to the environment factors and can experience interference from other signals, especially those operating in similar frequency range. Both major factors influence the dynamic rate scaling in wireless networks. Therefore when designing a solution for wireless video streaming services the fidelity of the network must be taken into consideration.

Previous studies already show the possibilities of 4kUHD video streaming over a wireless network both in uncompressed and compressed formats. Uncompressed 4kUHD wireless video transmission and playback has already been explored in [10]. This study has showed initial possibilities of uncompressed 4kUHD playback and streaming over four commercial off-the-shelf (COTS) wirelessHD [17] 60 GHz parallel channels using a high-end PC and GPU (graphics processing unit) based on decoding using CUDA [18] (Compute unified device architecture); however, this was only done in a short range due to interference between the channels. Another study conducted showed the use of 4kUHD H264 compressed video in a live stream over 802.11n [7]. In this study, the Chroma subsampling was varied from 4 : 2 : 0 to 4 : 4 : 4 at a low bitrate of 20 Mb/s using a group of picture (GOP) length of 40. This approach did not take into consideration other video sequences due to availability and interroom transmission due to the bandwidth of 802.11n between rooms; video quality used was measured using structural similarity (SSIM) [19] and is considered to emulate the human visual system (HVS). Furthermore the studies shown in [20–22] have shown that the widely used peak signal-to-noise ratio (PSNR) and mean squared error (MSE) are flawed in differentiating structural contents of images since different types of impairments can be applied and still have the same MSE value, while PSNR is more sensitive to noise. We therefore use SSIM, since increase in spatial resolution of video enables improved satisfaction for the HVS [23], and structural impairments will be easily noticed at higher resolution which can diminish the end-user QoE.

Based on the observations stated above we therefore propose an implementation of 4kUHD live encoding for live video streaming at low bitrates. We employ CUDA in implementing this system as regular open source x264 [24] as shown to provide low bitrates for 1080p videos and 4kUHD videos.

3. Background on 802.11ac

802.11ac [25] provides high-throughput wireless local area network (WLAN) on the 5 GHz band. Its specification allows for a multistation link of 1 Gb/s and a single link throughput of 500 Mb/s due to the increase in channel width of 80 MHz. Furthermore it supports multiuser multiple input multiple output (MIMO). The standard also implements standardized beamforming technology and feedback compatibility across vendors, unlike the nonstandardization in its predecessor standard 802.11n which made it difficult to implement beamforming effectively.

4. Parallel H264 Encoder

Previous studies, showing the parallelization of H.264 encoder applications for GPUs, involved the development of different solutions for separate encoding processes such as prediction, entropy encoding, and deblocking filter. More recently there has been a study of porting these processes onto a GPU using CUDA [26].

In [27] an efficient algorithm for block-size motion estimation with fractional pixel refinement using CUDA GPU is presented. The study shows the decomposition of H264 motion estimation algorithm into 5 steps allowing for highly parallel computation with low memory transfer rates. In [28] motion estimation and discrete cosine transform (DCT) coefficient parallelization in CUDA is studied; however, the relational dependencies between the transformation of DCT and intraframe predictions were neglected. Intraframe encoding algorithm was presented in [29] which is GPU-based and does 4 × 4 block process reordering in a diagonal order using openCL (a platform dependent GPU application development environment), however, this is only one way of reordering the block for parallelism. In [26], all encoding processes are parallelized on a GPU using CUDA; however, the parallel encoder is aimed at high definition video and also for video storage as their implementation does not take into consideration real-time encoding for live video streaming due to the huge amount of memory transfer latency.

These related works show improvements in using CUDA for parallel H264 encoding; however, none of these studies take into consideration live video streaming as the memory transfer latencies which occur during the pre- and postencoding process is not suitable for such an application. For example in [30] the achieved speedup of the GPU application was only double that of a single threaded CPU application performance. Their implementation was found to have architectural bottlenecks as a result of caching latencies and limitations of registers. Their solution seemed impossible using software approach due to their CUDA compute capability. In [31] A CPU/GPU parallel model for H.264 SVC encoder using CUDA was implemented to study the data transfer between host (CPU) and device (GPU). The authors used a raw frame as reference, though this increased data granularity, the resulting image quality was poor; for a resolution of 352 × 288 the performance achieved was 1.03 fps. In [26] the authors do a port of four major processes of H264 encoding to CUDA. They use data localization to organize data and threads to work efficiently on the GPU. However this approach is constrained to available GPU resources, as their approach eliminates the use of higher than HD resolutions especially on GPUs with constrained local memory. Furthermore their implementation allows for a latency bottleneck for critical video applications such as live streaming and real-time encoding/decoding processes as data must be transferred from and to the CPU memory before it can be accessed by other processes. We therefore extend the work of [26] to support zero-copy memory and also implement it as a directshow filter to enable it to accept a feed from live capture. The modules in [26] are Interprediction, intra prediction, entropy encoding, and deblocking filter. We also work on the interprediction module where we implement dynamic parallelism for motion estimation and on the intraprediction module where we reduce the predictions; however, every other module within their implementation is available to perform our experiments.

Our GPU architecture is different from previous implementations, as our GPU uses a newer streaming multiprocessor unit called SMX available through the Kepler GK110 [32] architecture with CUDA compute capability 3.5. This allows for dynamic parallelism; therefore, kernels can spawn new threads without needing to get instructions from the CPU all over again. This is very useful in modules such as interprediction where the macroblocks are subdivided into smaller units. The CPU need not issue instructions every time a subdivision is to be executed.

5. Implementation Design

Our test bed consists of two computers connected using the 802.11ac [25] network at channel 36 using buffalo [33] air station (wireless transmitter and receiver) as illustrated in Figure 1(a). Open source VLC Media player [34] was used to decode video. Our GPU platform is the NVidia Quadro 510; streaming was done using the real-time streaming protocol (RTSP) server filter provided by [35] and network statistics were collected using wireshark. We do our tests in three parts:

(1) encoding,

(2) real-time encoding for live streaming,

(3) live capture with a point grey flea 3 [36] 4k camera and encoding for live streaming.

In conducting the test numbers 2 and 3, we consider two major scenarios which depict the usage of video streaming indoors in a typical office space with lots of furniture. The two scenarios considered are as follows:

(i) change in distance intraroom (10 and 20 meters),

(ii) the use of obstruction: interroom (one room and two rooms apart) and interfloor (one floor above).

Figure 2 shows the floor plan of two floors of the office environment used for the experiments. The notations Rx (A, B, and C) show the position of receiver in other parts of the building during the experiments. In the intraroom scenario, the distance between the transmitter and receiver was 10 and 20 meters, respectively. During each experiment the following test clips were used: Sintel 4k [37], Coast, Foreman, and News [38]. These video clips were chosen based on motion with sintel having the fastest and foreman the slowest, each video clip had 500 frames. At the beginning of each experiment, the data rate was measured using LAN speed test [39] software to observe the bandwidth available. For the sake of clarity we use arbitrary numbers to denote the scenario in Table 1.

Based on the data rate observed at the beginning of each scenario we can expect good quality results for peer-to-peer video transmission. The video clips and live capture were compressed using a GOP length of 40 (using I and P frames only), with a bitrate of 20 Mb/s ABR (average bitrate); the designated packet size used was 1500 bytes.

5.1. Directshow Filter Implementation. Microsoft directshow filter [40] implementation provides an architecture for high-quality video capture, streaming, and playback. It supports a wide variety of formats including Motion Picture Experts Group (MPEG) and audio-video interleaved (AVI). Furthermore, it supports video capture from analogue and digital devices supported by the windows driver model (WDM).

Table 1: List of scenarios and designated arbitrary numbers.

Scenario	Arbitrary number
10 meters	1
20 meters	2
Interroom (Rx A)	3
Interroom (Rx B)	4
Interfloor (Rx C)	5

The main aim of using directshow is to simplify the task of creating multimedia applications on windows platform irrespective of hardware differences, data transport, and synchronization of individual components of the entire framework.

Filters are normally divided into three main categories, namely, source, transform, and render. For this filter we implement a transform filter, with the base class CTransform. The filter itself implements the following functions in order to achieve its core purpose.

5.1.1. Check Input Type. This function checks whether the specified media type provided by the upstream filter is acceptable to the encoder filter or not. In our case the major type of the media should be video. The filter supports the following media subtypes:

(a) WMMEDIASUBTYPE_I420

(b) MEDIASUBTYPE_IYUV

(c) MEDIASUBTYPE_YV12

(d) MEDIASUBTYPE_YUY2.

5.1.2. Get Media Type. This function provides the supported media type of the output pin of the encoder filter to the downstream filter during the connection negotiation process. The major type of the output pin is video while the subtype it supports is custom defined for the CUDA H264 encoder. It is defined as a global unique identifier (GUID) in the following manner.

5.1.3. DEFINE_GUID (MEDIASUBTYPE_H264CUDA, 0x55B845A5, 0x8169, 0x4BE7, 0xBA, 0x63, 0x6C, 0x4C, 0x2C, 0x01, 0x26, 0x6D). This function also makes sure that the input pin of the encoder filter is connected before the output pin attempts any connection with the downstream filter. This is because most of the properties (width, height, frame rate, etc.) between input and output do not change and these must be known beforehand so that they can be conveyed to the downstream filter.

5.1.4. Decide Buffer Size. This function negotiates memory requirements between the output pin of the encoder filter and the input pin of the downstream filter. This function asks for memory that can hold the largest possible encoded frame. The input pin of the encoder filter must be connected for this function to succeed because it needs to estimate the size of the output frame based on the properties of the raw incoming video.

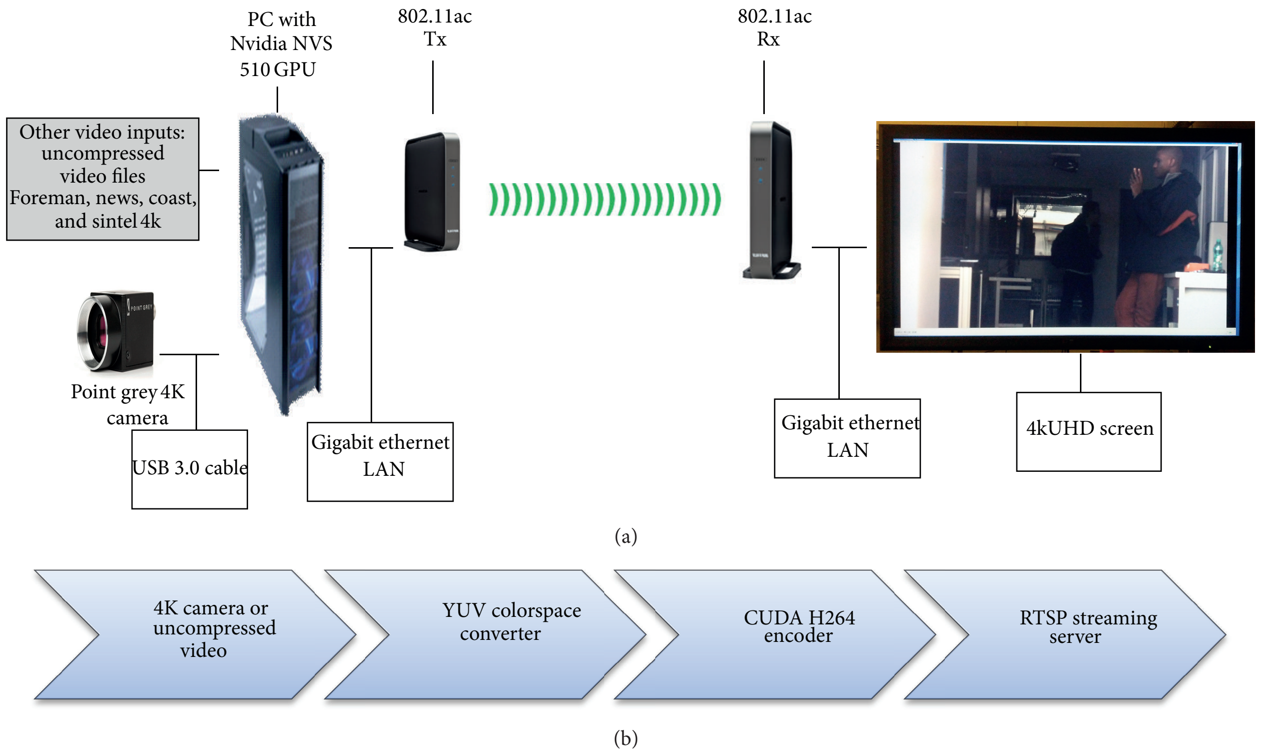


Figure 1: Implementation design for 4kUHD live video streaming: (a) shows an illustration of the hardware implementation while (b) shows the directshow implementation.

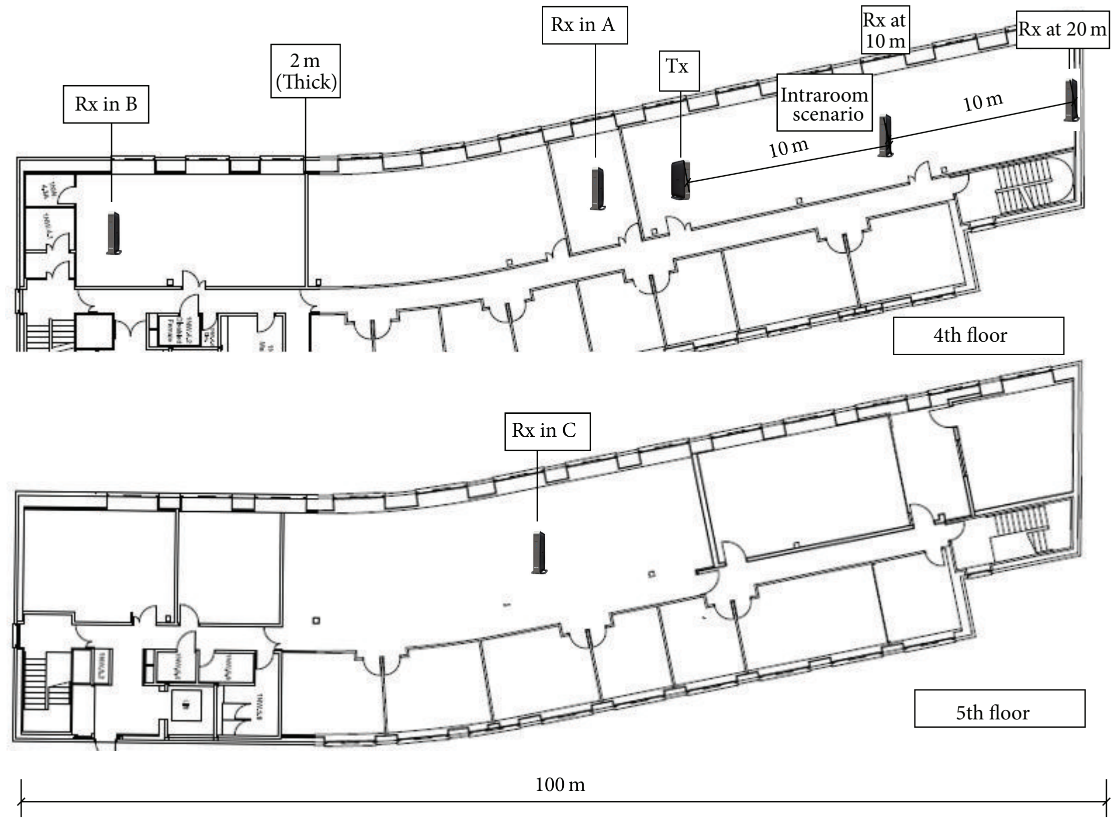


Figure 2: Floor plan of the 2 areas used for experimental video streaming of 4kUHD video.

5.1.5. Transform. This function receives raw YUV video frame by frame from the input pin and passes it to the encoder instance. During this process, the CPU pointer allocated to the input buffer of the encoder allows the filling of this buffer, when the buffer is full it is unlocked for the GPU to access. If any output is available from the encoder, it sends that output to the downstream filter. This function does not return the encoded frame. The encoded frame is returned by a call-back function that is registered with the encoder during initialization. Since this becomes an asynchronous process, a queue is needed to hold the encoded frames being output by the encoder using the call-back function. Whenever the "transform" function is called it checks the queue; if some encoded frame is available, it sends it to the downstream filter.

5.1.6. Start Streaming. This function is called when the stream is being executed. It initializes the encoder using the API already provided by the work of [26]. After the default settings are fetched, some changes are made to customize the encoding process according to user requirements (e.g., bitrate, fps, and so forth). A call to the following function returns the encoder object that is used for the actual encoding. A call to the following functions instantiates the object with the memory buffer and a call-back function that is used to output encoded bit stream by the encoder. The encoded bit stream is then allocated a CPU pointer. At this point the encoded bit stream is pushed to the RTSP server for packetization and streaming.

5.1.7. Stop Streaming. This function is called when the streaming is stopped. This function frees any memory allocated for the encoder and destroys the encoder object.

5.2. Zero-Copy Memory Mapping. Irregular memory access patterns can be successfully captured by CPUs due to its design that allows for reduction in memory latency access through extensive caching. However, the same patterns may prevent the efficient utilization of GPU memory bandwidth because the restrictions on access patterns must be met in order to achieve good memory performance, which are stricter on GPUs than they are on CPU.

When running a typical CUDA application, memory is allocated as pageable. Therefore, the memory is only allocated when needed. The use of pageable memory allows for increase in memory access latency as this memory will page out, and therefore it will only be reallocated when there is a need for it. The major disadvantage of this is that CPU↔GPU memory transactions are slower due to the bandwidth of peripheral component interconnect express expansion bus (PCIe) which cannot be fully exploited. Since the paged memory can be swapped or reallocated, the PCIe driver needs to access every single page, copy it to a buffer, and pass on to direct memory access (DMA) thus a synchronous page by page copy. This is the only way PCIe transfers can occur.

Due to this there is the need for an independent host controlled memory management allocation unit (MMU) especially for applications such as live video streaming or video conferencing which have strict latency requirements.

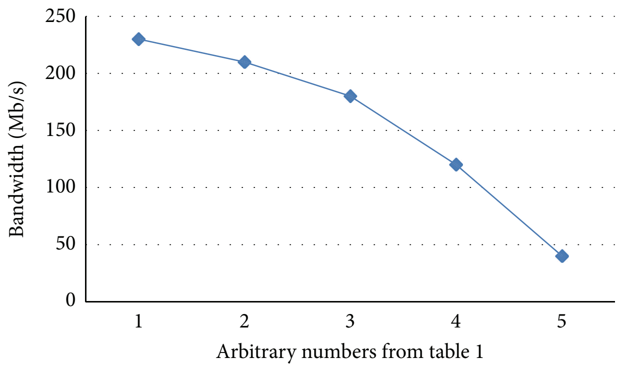


Figure 3: Bandwidth available with each scenario.

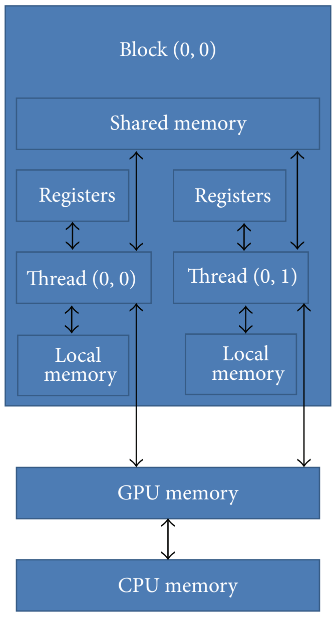
At initialization, host mapping is enabled and the MMU allocates CPU pinned memory for input data. It is assumed that the maximum memory size for this application is 2 GB; this is based on experiments performed with x264 video streaming. The CUDA kernel pointers are produced to allow access to slash from the GPU. Finally the kernels were allocated pointers to the host memory as if it were the GPU's global memory for the encoding process as the memory is already mapped to CUDA unified virtual address space. This technique of memory allocation allows for the overlap of encoding and packetization for video transport, as data is accessed directly via DMA (residing on the GPU) from the CPU without any explicit data transfer to the GPU memory. Figure 3 shows an illustration of this.

At runtime the CUDA kernels are normally asynchronous to the CPU; therefore, each block also issued an atomic counter for synchronization. Therefore all blocks can be executed in a sequential manner with nondivergent branching [41] and data can also be read from previous threads. It should be noted that the CPU does wait until the encoding process is complete before releasing the memory buffers for packetization and buffer refill of unprocessed frames.

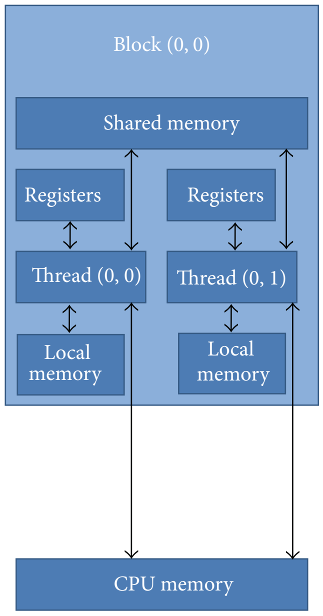
5.3. Interprediction. Interprediction is the most demanding process in x264 encoder [24]. Previous profiling in [26] indicates that interprediction accounts for approximately 70% of the total encoding process time and is done with the help of motion estimation.

To obtain accurate prediction values, H264 standard makes it possible to partition variable size MB, which is a 16 × 16 macroblock (MB). Each macroblock for encoding a frame is split into 4 × 4, 4 × 8, 8 × 4, 8 × 8, 16 × 8, 8 × 16 submacroblocks, an illustration can be seen in Figure 4. We therefore implement dynamic parallelism. Using dynamic parallelism the 16 × 16 macroblock acting as a parent kernel can spawn the thread blocks of the subblocks (which act as child kernels) without needing any extra instructions from the CPU; dynamic parallelism drastically reduces the execution control from the CPU (Figures 5(b) and 6). In deciding the final encoding mode, two steps are involved.

The first step involves the calculation of the best motion vectors (MV) for each possible mode within the reference

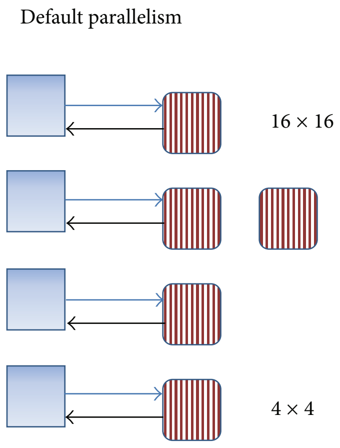


(a) Typical procedure

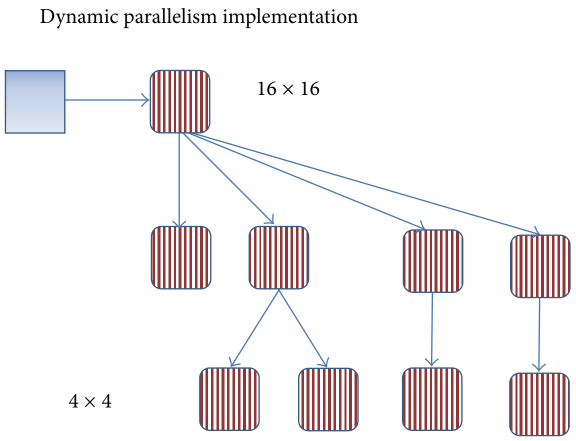


(b) Modified procedure

FIGURE 4: It shows an illustration of the memory access structure for (a) a typical H264 CUDA encode and (b) using zero-copy.



(a)



(b)

FIGURE 5: It shows the difference between the default parallelism and dynamic parallelism used in motion estimation process.

image. Based on the matching criterion the sum of absolute differences (SAD) is matched. The second step involves the evaluation of ratedistortion of each mode; the final mode is determined from the selection of the best mode.

Since a macroblock is divided into sixteen 4×4 blocks, the SAD value is calculated for each 4×4 block in parallel for all candidate motion vector positions within the reference search range. Since 960×540 blocks exist in a 4kUHD frame, 1024 candidate positions (MVs) per block exist in a 32×32 search range. Each SAD candidate is computed by one thread and 512 threads are executed in one thread block; we allocate more threads to increase processing granularity. For example, since 16 threads can be executed in a 4×4 block, therefore 32 MBs can be processed per execution. Therefore the total number of blocks is based on.

$$B(m,n) = \frac{Fw}{m} \times \frac{Fh}{n} \times R^2 \times \frac{1}{N_{\text{thread}}}, \quad (1)$$

where $B(m,n)$ denotes the number of blocks per subblock, with possible combinations of m and n, (4×4, 4×8, 8×4, 8×8, 16×8, 8×16, 16×16), where m and n are the subblock dimensions. Fw and Fh are the frame dimensions, and R denotes the search range while N_{thread} denotes the number of threads allocated per block. Based on this calculation every 1024 candidates of one 4×4 block are assigned to a thread block, which is then allocated to the shared memory therefore shared by all threads executed within a block.

Since the SADs of the variable block sizes 4×8, 8×4, 4×8, 8×8, 16×8, 8×16 and 16×16 are a consolidation of 4×4 SADs, the 4×4 SADs are merged to obtain the SAD values of all combinations possible. Each thread retrieves sixteen 4×4 SADs of one macroblock obtained at a candidate position and

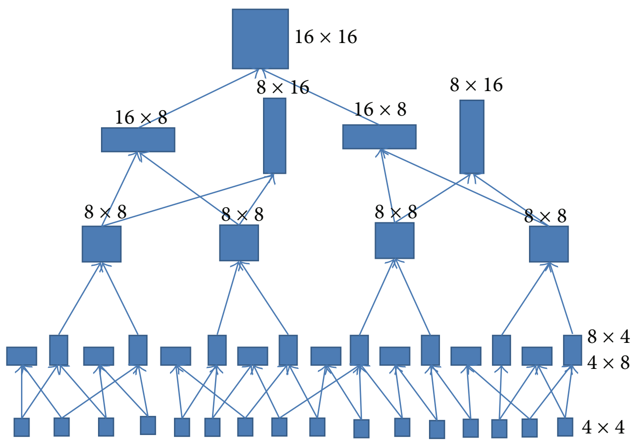


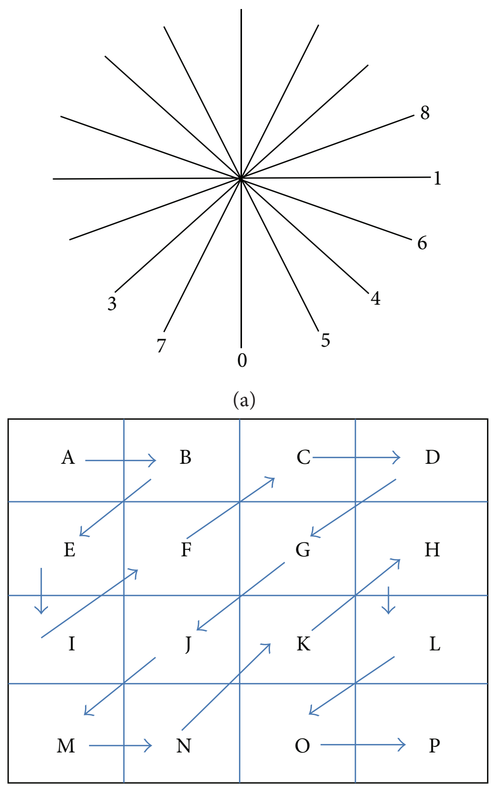
Figure 6: An illustration of division of a 16 × 16 macroblock kernel into smaller unit kernels using dynamic parallelism.

combines them in different ways to determine the SADs of all these block sizes.

After all the SADs of the block sizes are generated, the least of them is chosen as the best motion vector. Based on our implementation the SADs are compared in one thread block. During this process each thread pulls in 4 SADs from the host memory allocated to the parent kernel to produce the least SAD value. This value is then stored temporarily in the shared memory alongside their indexes. The SADs are then compared and the least value is stored back in the memory. This process helps with reducing the system memory access. Therefore the total number of thread blocks equals the block number of a frame. When the smallest SAD which is the Fractional Pixel motion pixel MV (FMV) of that block is identified and indexed it is then stored in the mapped memory allocated to it.

5.4. Intraprediction. In intraprediction the reconstructed pixels are needed as reference pixels. Strong dependencies between neighbouring MBs, enable multiple prediction modes. This makes it quite different from interprediction and thus presenting a fundamentally low parallel execution. A typical example is in 4 × 4 intraprediction scenario in which the blocks will refer to the 4 × 4 blocks in both diagonally and vertically (in form of a zig zag manner). This in itself limits it from high performance in a parallel implementation as the degree of parallel execution is limited to multiples of 2 subblocks within a 4 × 4 macroblock. For a 4kUHD image resolution, the maximum degree of parallel execution for a 4 × 4 intraprediction mode is limited by 270 (using diagonal processing for a wave front algorithm).

Other prediction modes which have been proposed rely on the recommendations of [42, 43], where the number of predictions is decreased for a better degree of parallelism. However during the test experiments it was noticed that the degree of parallel execution could be increased by reducing the direction of predictions even further and not compromising on video quality. Therefore horizontal approach for parallel execution is proposed.



(b)

Figure 7: Typical 4 × 4 intramode predictions.

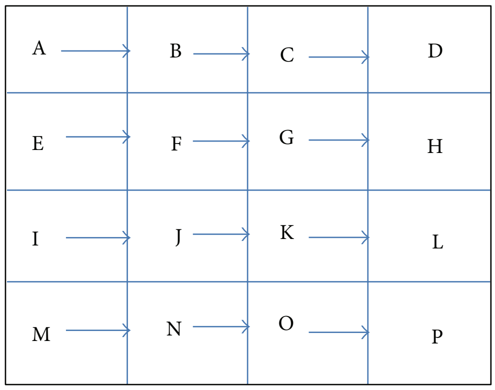
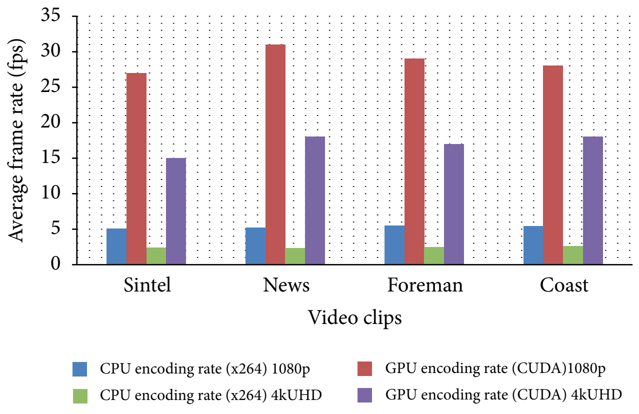


Figure 8: Horizontal mode for Intraprediction.

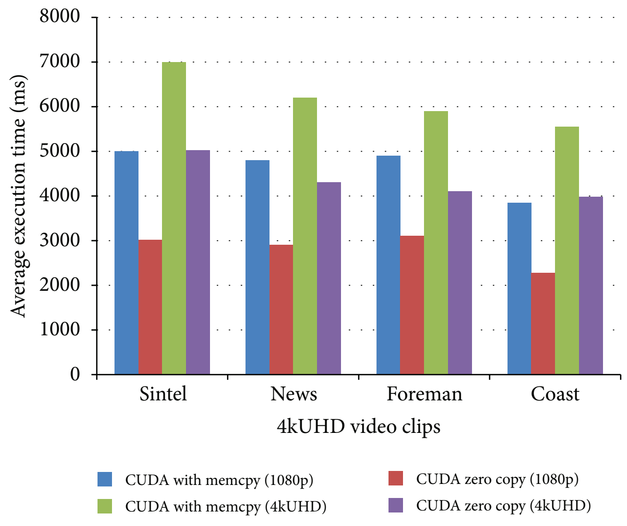
The numbers of predictions within the 4 × 4 block are decreased by computing the prediction directions (Figure 7). In doing this, the wave parallelism in the blocks is simplified such that the blocks on the top and lower ends are dependent on each other horizontally while the other blocks between them are vertically independent; the illustration can be seen in Figure 8. This simplification is aimed at reducing decoding complexity after a transmission through a noisy channel such as a wireless network.

6. Evaluation

As explained in implementation section we evaluate our implementation in three stages. The metrics used for the two



(a)



(b)

Figure 9: (a) Average frame rate comparison with x264 and (b) average execution time comparison between CUDA memcpy and CUDA zero copy.

other stages are structural similarity index metric (SSIM), network delay, and packet loss. Since at all stages the experiments were conducted ten times, an average of each metric is used for each video clip and test. Since the encoder is only running in parallel we have access to the same input parameters as any other H264 encoder.

6.1. Encoding Performance. Figure 9(a) shows a comparison of attained average frame rate between x264 and its parallel implementation on CUDA for both 1080p videos and 4kUHD videos. Across board the results show that there is a significant speedup of the GPU implementation over the CPU implementation. We cannot compare directly to previous work as CUDA parallel processing cores differ between GPU generations due to several important architectural changes that exist between streaming multiprocessor designs. However, we only compare based on the approach used. In [26], the authors use memory copy between the GPU memory and the host (CPU) memory. Figure 9(b) shows the amount of latency avoided. On average a staggering 2000 ms of latency is avoided due to the implementation of zero-copy in our implementation. Therefore, we can assume that based on 500 frames the additional memory copy latency incurred during the encoding process for each frame will, on average, be 4 ms. This might look negligible for a small number of frames; however, for a longer sequence this will be more pronounced.

Table 2: Standard deviation and variance of the SSIM values at 10 m (intraroom).

	Sintel	News	Foreman	Coast
Variance	0.000426	0.000593	0.002145	$8.67111E-06$
Standard deviation	0.020642	0.02435	0.046318	0.002944675

6.2. Real-Time Encoding for Live Streaming Performance. Based on these observations we proceed to the live encoding for streaming experiments. Figure 10 shows an illustration of the interior of two floors within the networks building at the University of Essex which is used for the experiments.

We show the variance and standard deviation of video quality results at 10 meters.

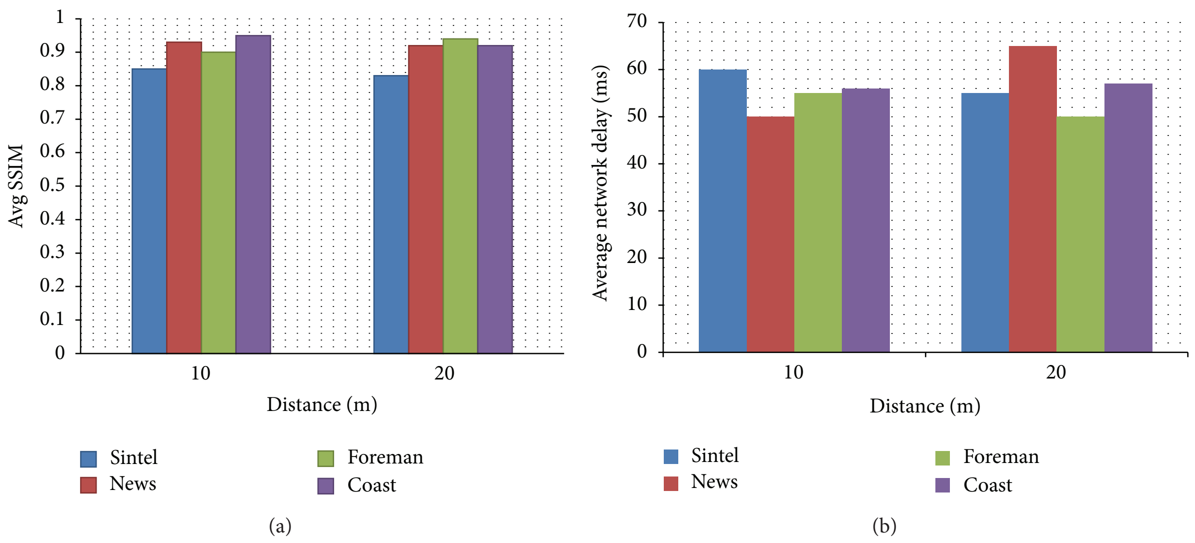
Table 2 shows the standard deviation (SD) and variance in values of the video quality for each video clip used at 10 meters. All SD values show slight deviation from the mean SSIM value in each case with foreman having the largest deviation and coast the smallest deviation.

Figures 11 and 12 show that across board for a peer-to-peer transmission the 802.11ac network provides sufficient bandwidth for a 20 Mbps ABR stream. The least average SSIM value 0.73 occurs in the interfloor scenario for sintel 4k; however, the video quality is still acceptable. The average network delay spans between 59 and 92 ms in all cases. The packet loss is impressive as the maximum value for average packet loss is only 0.72%.

6.3. Live Capture Stream. We experiment using a point grey flea 3 4k camera (blue arrow) connected to a PC which is connected to the 802.11ac WLAN transmitter (green arrow). We use the buffalo air station receiver within the room in the upper diagram of Figure 2 and use a netgear A6200 [44] usb 2.0 dongle in the room on the upper floor (lower diagram in Figure 2). The Air station was connected to the 4kUHD screen from [39] which still performs better compared to the netgear USB dongle. This is typical of the hardware design as we did not expect a very good quality video from the netgear. However, when the air station was used in the same scenario as the netgear, we had better quality video.

7. Conclusion and Future Work

In this paper, we have demonstrated a novel implementation of 4kUHD live encoding for streaming over a wireless network using CUDA parallel H264. We extended previous research by implementing zero-copy to reduce memory copy latencies between host memory and GPU memory; a



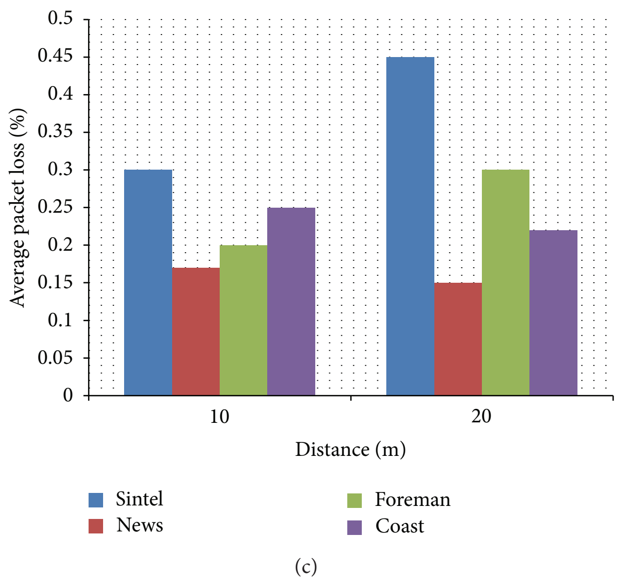


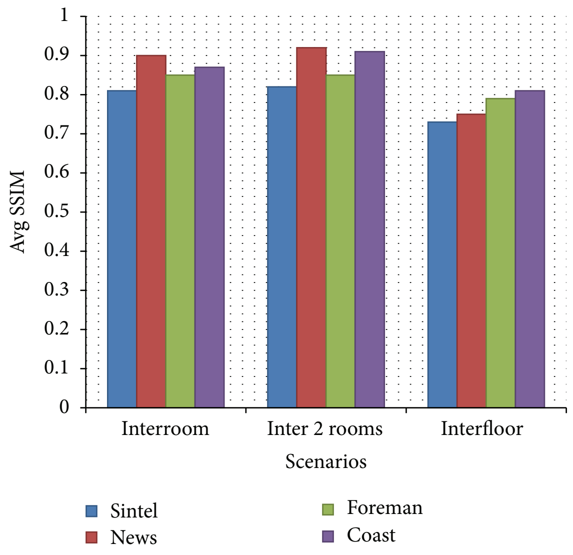
Figure 10: Evaluation of (a) video quality, (b) network delay, and (c) packet loss for change in distance using 802.11ac WLAN for a peer-to-peer transmission.

dynamic parallel motion algorithm and a novel intraprediction mode. Finally we demonstrate proof of concept by streaming 4kUHD video content using uncompressed video sequences and a live capture device in a peer-to-peer network, making this is a significant improvement to the state of the art. Having reached these results (Figures 8, 9, and 10), we can now conclude that 4kUHD real-time encoding for live streaming at low bitrates is possible and can be implemented in real-world applications particularly in one-way video streaming applications where delay is not a major issue.

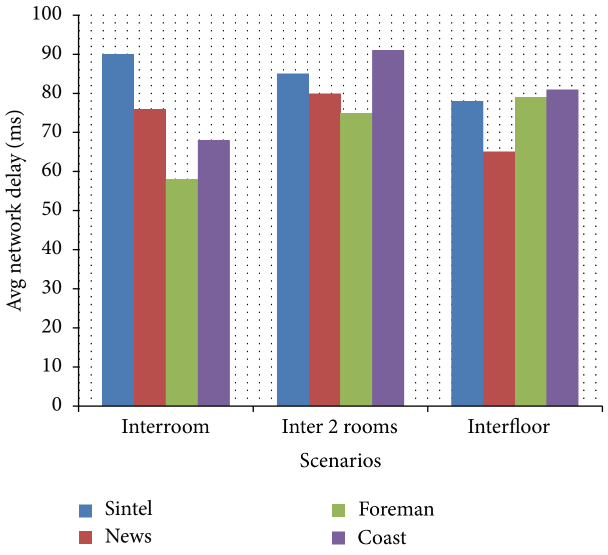
The findings of this research show the possibilities of reduced bitrate for 4320p (8KUHD) video at reduced bitrates. Moreover, since GPU generational changes will bring about changes in parallelization, therefore improved interprediction algorithms will be investigated as soon as those changes happen. Furthermore, H.264 is currently limited to 4kUHD; the major focus of future research will be on the new high efficiency video codec (HEVC). Currently HEVC shows a huge potential for video streaming since it can provide up to 50% the bandwidth needed compared to its predecessor. However, the issue which we have come across is the time taken to encode video files, as it takes longer to encode due to its threading issues. We will therefore be working towards the implementation of the CUDA parallel encoder for HEVC thereby enabling and improving the quality of experience for UHD raw videos being encoded and streamed in realtime.

Conflict of Interests

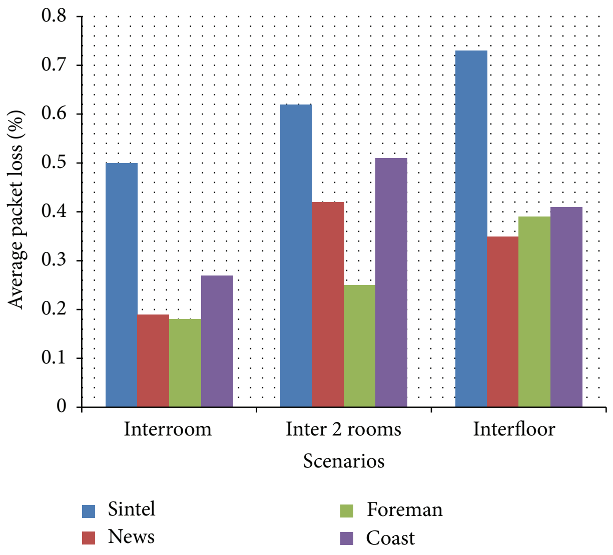
The authors declare that there is no conflict of interests regarding the publication year.



(a)

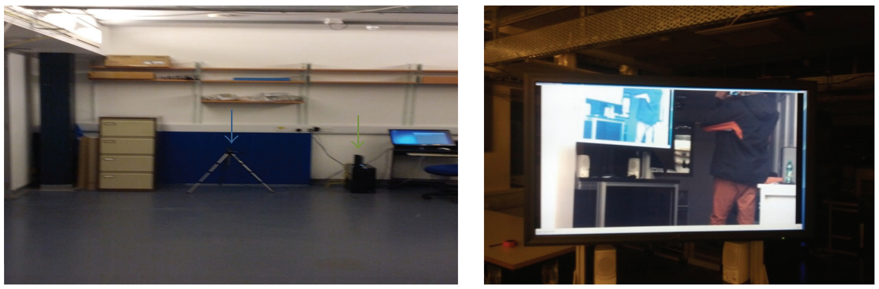


(b)



(c)

FIGURE 11: Evaluation of (a) video quality, (b) network delay, and (c) packet loss for other scenarios using 802.11ac WLAN for a peer-to-peer transmission.



(a) (b)

FIGURE 12: Live capture experiment (a) shows the 4kUHD web camera while (b) shows the 4kUHD screen used.

References

[1] A. R. L. Renambot, R. Singh, B. Jeong et al., “SAGE: the scalable adaptive graphics environment,” in *Proceedings of World Conference on Cooperative & Work-Integrated Education (WACE ’04)*, 2004.

[2] K. Ponto, K. Doerr, and F. Kuester, “Giga-stack: a method for visualizing giga-pixel layered imagery on massively tiled displays,” *Future Generation Computer Systems*, vol. 26, no. 5, pp. 693–700, 2010.

[3] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, “Surround-screen projection-based virtual reality: the design and implementation of the CAVE,” in *Proceedings of the ACM Conference on Computer Graphics (SIGGRAPH ’93)*, pp. 135–142, August 1993.

[4] K. Jarrett, “Beyond broadcast yourselfŮ: the future of YouTube,” *Media International Australia*, no. 126, pp. 132–144, 2008.

[5] N. Electronics, http://www.ntt-electronics.com/.

[6] IntoPIX, http://www.intopix.com/.

[7] A. O. Adeyemi-Ejeye and S. D. Walker, “Ultra-high definition Wireless Video transmission using H. 264 over 802. 11n WLAN: challenges and performance evaluation,” in *Proceedings of the 12th International Conference on Telecommunications (ConTEL ’13)*, pp. 109–114, 2013.

[8] M. K. J. Halák, S. Ubik, P. Žejdl, and F. Nevřela, “Real-time long-distance transfer of uncompressed 4K video for remote collaboration,” in Future Generation Computer Systems 27(2011) 886–892, 2011, http://www.elsevier.com/wps/find/journaldescription.cws_home/505611/description#description.

[9] D. Shirai, T. Kawano, T. Fujii et al., “Real time switching and streaming transmission of uncompressed 4K motion pictures,” *Future Generation Computer Systems*, vol. 25, no. 2, pp. 192–197, 2009.

[10] A. O. Ejeye and S. D. Walker, “Uncompressed quad-1080p wireless video streaming,” in *Proceedings of the 4th Computer Science and Electronic Engineering Conference (CEEC ’12)*, pp. 13–16, 2012.

[11] C. T. Calafate, M. P. Malumbres, and P. Manzoni, “Performance of H.264 compressed video streams over 802.11b based MANETs,” in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops*, pp. 776–781, March 2004.

[12] S. P. A. J. V. K. Soroushian, “H. 264 parameter optimizations for internet based distribution of high quality video,” in *Proceeding of the SMPTE Annual Technical Conference*, October 2008.

[13] K. Gatimu, T. Johnson, M. Sinky, Z. Jing, L. Ben, K. Myungchul et al., “Evaluation of wireless high definition video transmission using H. 264 over WLANs,” in *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC ’12)*, pp. 204–208, 2012.

[14] D. Li and J. Pan, “Performance evaluation of video streaming over multi-hop wireless local area networks,” *IEEE Transactions on Wireless Communications*, vol. 9, no. 1, pp. 338–347, 2010.

[15] R. Stapenhurst, D. Agrafiotis, J. Chung-How, and J. Pledge, “Adaptive HRD parameter selection for fixed delay live wireless video streaming,” in *Proceedings of the 18th International Packet Video Workshop (PV ’10)*, pp. 142–149, Hong Kong, December 2010.

[16] V. Menkovski, G. Exarchakos, A. Liotta, and A. C. Sánchez, “Quality of experience models for multimedia streaming,” *International Journal of Mobile Computing and Multimedia Communications*, vol. 2, no. 4, pp. 1–20, 2010.

[17] L. WirelessHD, “WirelessHD Specification Version 1. 0 Overview,” 2010, http://www.wirelesshd.org/pdfs/WirelessHD_Full_Overview_071009.pdf.

[18] Nvidia, “Nvision 08: the world of Visual Computing,” 2011, http://www.nvidia.com/content/cudazone/download/Getting_Started_w_CUDA_Training_NVISION08.pdf.

[19] 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*, vol. 13, no. 4, pp. 600–612, 2004.

[20] A. Horé and D. Ziou, “Image quality metrics: PSNR vs. SSIM,” in *Proceedings of the 20th International Conference on Pattern Recognition (ICPR ’10)*, pp. 2366–2369, Istanbul, Turkey, August 2010.

[21] Z. Wang and A. C. Bovik, “Mean squared error: lot it or leave it? A new look at signal fidelity measures,” *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[22] S. Winkler, “Video quality and beyond,” in *Proceedings of the European Signal Processing Conference*, pp. 3–7, 2007.

[23] T. Murakami, “The development and standardization of ultra high definition video technology,” in *High-Quality Visual Experience*, pp. 81–135, Springer, 2010.

[24] Reference software X264-060805, http://www.videolan.org/developers/x264.html.

[25] I. w. group, “OFFICIAL IEEE 802. 11 WORKING GROUP PROJECT TIMELINES—2013-09-21,” 2013, http://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm.

[26] N. Wu, M. Wen, H. Su, J. Ren, and C. Zhang, “A parallel H. 264 encoder with CUDA: mapping and evaluation,” in *Proceedings of the IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS ’12)*, pp. 276–283, 2012.

[27] W.-N. Chen and H.-M. Hang, “H.264/AVC motion estimation implmentation on compute unified device architecture (CUDA),” in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME ’08)*, pp. 697–700, Hanover, Germany, June 2008.

[28] G. Zhiyong, W. Shuang, S. Zhenyu, and L. Haihua, “Design and implementation of H. 264/AVC video encoding based on cuda,” *Journal of South-Central University for Nationalities*, vol. 28, pp. 67–72, 2009.

[29] M. C. Kung, O. Au, P. Wong, and C. H. Liu, “Intra frame encoding using programmable graphics hardware,” in *Proceedings of the Advances in Multimedia Information Processing (PCM ’07)*, pp. 609–618, Springer, 2007.

[30] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-M. W. Hwu, “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” in *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP ’08)*, pp. 73–82, February 2008.

[31] Y.-L. Huang, Y.-C. Shen, and J.-L. Wu, “Scalable computation for spatially scalable video coding using NVIDIA CUDA and multi-core CPU,” in *Proceedings of the 17th ACM International Conference on Multimedia (MM ’09)*, pp. 361–370, October 2009.

[32] Nvidia, “Nvidia Kepler GK110 Next-Generation CUDA Compute Architecture,” 2012.

[33] Buffalo, “Networking at Gigabit Speeds,” 2012, http://www.buffalotech.com/resource_center/wireless_technologies.

[34] V. Lan, “Video Lan,” 2012, http://www.videolan.org/.

[35] nanocosmos, "Nanocosmos," 2012, http://www.nanocosmos.de/v4/documentation/live_video_encoder_-_playback.

[36] P. G. F. camera, "Flea 3 USB camera," 2012, http://ww2.ptgrey.com/USB3/Flea3.

[37] Sintel, "Sintel 4K," 2011, http://www.sintel.org/news/sintel-4k-version-available/.

[38] Elemental, "4K Test sequences," 2013, http://www.elemental-technologies.com/resources/4k-test-sequences.

[39] Totusoft, "LAN Speed Test," 2012, http://www.totusoft.com/lanspeed.html.

[40] Microsoft, "Directshow," 2012, http://msdn.microsoft.com/en-us/library/windows/desktop/dd375454(v=vs.85).aspx.

[41] M. Harris, "Optimizing parallel reduction in CUDA," *NVIDIA Developer Technology*, vol. 2, 2007.

[42] W. Lee, S. Lee, and J. Kim, "Pipelined intra prediction using shuffled encoding order for H.264/AVC," in *Proceedings of the IEEE Region 10 Conference (TENCON '06)*, pp. 1–4, Hong Kong, November 2006.

[43] G. Jin and H.-J. Lee, "A parallel and pipelined execution of H.264/AVC intra prediction," in *Proceedings of the 6th IEEE International Conference on Computer and Information Technology (CIT '06)*, Seoul, South Korea, September 2006.

[44] NETGEAR, "A6200 802. 11ac Wifi USB Adapter," 2013.