PARALLELIZATION OF VIDEO ENCODING ON CPU+GPU

Mirko Liebender

Bsc Computer Engineering,
Technical University of Berlin, TUB
Berlin, Germany
mirko@mailbox.tu-berlin.de

ABSTRACT

Video Coding is nowadays a central technology in a wide range of systems. It is used in TVs, Smart phones, streaming platforms, video conferences, Blu-ray etc. For that purpose many different video codecs and algorithms were developed over the last decades (e.g., H.265/HEVC [1], H.264/AVC [2], MPEG-2 [3]) to address certain requirements of many applications.
Since the demand of high resolution videos is more and more increasing these codecs aim to reach a high compression rate while maintaining a good quality. The cost of this approach is always a higher computing requirement (e.g. HEVC aims to reach half the size while maintaining the same quality as AVC [4]), especially for systems which perform real time encoding.
In order to meet these requirements graphic processing units (GPUs) have emerged as a coprocessing unit for central processing units (CPUs) to accelerate certain logical parts of the encoding process.
GPUs consist of a large amount of streaming processors, which allow them to execute a lot of operations in parallel. Because they are restricted to Single Instruction Multiple Data (SIMD) operations and the fact that data synchronisation between CPU and GPU is a bottleneck, it is a rather complex task to efficiently use this immense parallel performance.
In this paper three different strategies to address this challenge are reviewed. Two of them propose parallel strategies for the H.264 codec and one for the H.265 codec. They all reach certain speed ups, but because of there different nature it is not possible to directly compare them.

1. INTRODUCTION

The increasing demand of high quality video communication in nearly every part of the electronic entertainment area and the tremendous growth of video contents on the Internet pushed the development of high efficient compression methods. Especially with High Definition (HD) video content going to be replaced by beyond-HD content like Ultra High Definition (UHD) a high compression rate is very important.
The H.264/AVC codec was designed to reach double the compression rate of every other codec at that time while remaining the same quality for the reconstructed video. It also aimed to be used on many different system types. At the time[1] parallelism of the algorithm wasn't really taken into consideration. Due to the fact that a standard personal computer around 2003 didn't have a multi core processor, parallelization wasn't really an option. The utilization of GPUs was also mostly limited to video gaming and frameworks like CUDA [2] or OpenCL[3] were just developed recently.
The H.265/HEVC codec had a similar goal as the H.264/AVC and aimed to reach half the size of its predecessor while maintaining the same quality. It is going to be the next standard for video compression and was designed to run on systems with a lot of parallel performance.
Both codecs are very similar and inherited the known hybrid Motion Estimation (ME)/Motion Compensation (MC) followed by transform and entropy coding framework adopted by H.261 since 1994 [5]. Figure 1 depicts the encoder block diagram of H.264/AVC and Figure 2 depicts the encoder of H.265/HEVC.

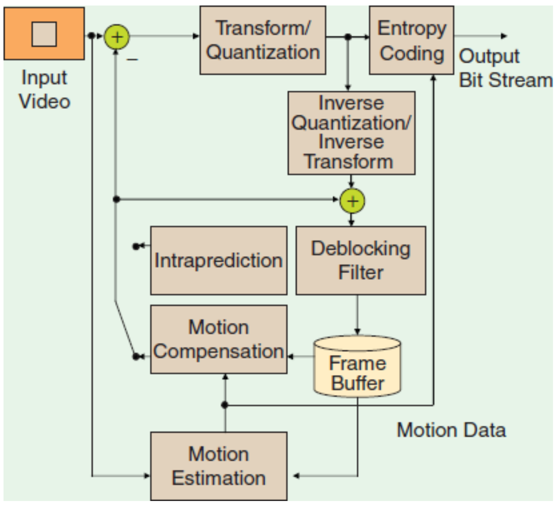


Figure 1: *H.264/AVC encoding algorithm [6]*

For H.264/AVC being available since 2003 and therefore being part of a lot of previous work, a lot of papers targeting the issue of parallelization of the algorithm exist.
Since Motion Estimation (ME) is one of the most compute intensive parts of both H.264 and H.265, a lot of these papers propose strategies to utilize GPUs to undertake ME [7–10].
Another approach [11] considered the possibility to parallelize the deblocking filter on GPU. Most of the previous work always targets a single module of the codec algorithm and tries to improve the performance by parallelizing it on GPU.
In this paper there are 3 approaches presented, which try to harness the possible parallel performance of modern computers. In section 2 a short overview on the video coding techniques is given. In the

[1]Release date: May 2003

[2]Compute Unified Device Architecture
[3]Open Computing Language

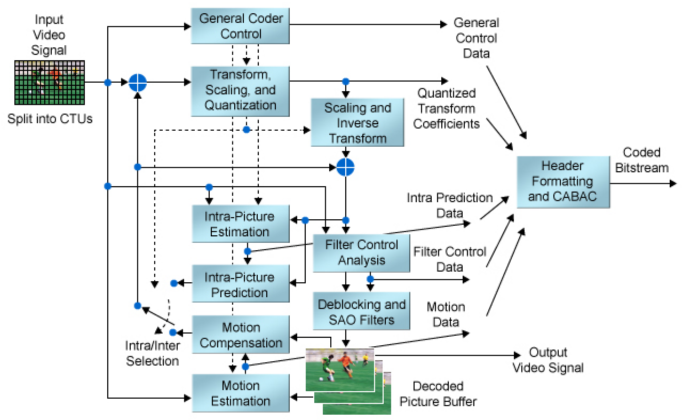


Figure 2: *H.265/HEVC encoding algorithm*

following section 3 the first paper, which proposes a GPU based fast motion estimation method, is presented. The next two sections present the remaining two papers: 4 (dynamic cpu+gpu model for encoding), 5 (cpu+gpu encoding framework).
In the last section 6 a conclusion is given.

2. BACKGROUND

In the past decade new video coding standards have achieved state-of-the-art coding performance. H.264/AVC typically requires 60% or less of the bit rate compared to previous standards to achieve the same reconstruction quality [2]. In comparison, H.265/HEVC requires around 50% less then H.264/AVC. Next, a short overview on both codecs is given.
The **H.264/AVC** video coding standard is designed based on the block-based (so called macro blocks) hybrid video coding approach [12], [2], which has been used since earlier video coding standards. The coding algorithm exploits spatial correlation between neighboring pixels of the same picture. In addition, it also exploits temporal correlation between neighboring pictures in the input video sequence to achieve compression. Figure 1 depicts the encoder block diagram. The input picture is partitioned into different blocks, and each block may undergo intraprediction using neighboring reconstructed pixels in the same frame as predictor. H.264 supports intraprediction block sizes of 16 x 16, 8 x 8, and 4 x 4, and it allows different ways to construct the prediction samples from the adjacent reconstructed pixels. Alternatively, the input block may undergo interprediction using the reconstructed blocks in the reference frames as predictor. Interprediction can be based on partition size of 16 x 16, 16 x 8, 8 x 16, 8 x 8, 8 x 4, 4 x 8, or 4 x 4. Displacement between the current block and the reference block can be up to quarter-pel accuracy and is signaled by the motion vector and the reference picture index [12]. The prediction residue signal from intraprediction or interprediction would then undergo transformation to decorrelate the data. In H.264, a 4 x 4 separable integer transform is used, which is similar to 4 x 4 discrete cosine transform (DCT) but avoids the mismatch between forward and inverse transform. Then, the transform coefficients would be scalar quantized and zig-zag scanned. The context adaptive variable length coding (CAVLC) may then be employed to entropy code the scanned transform coefficients. CAVLC is an adaptive coding scheme, and it may switch between different code word tables during encoding depending on the values of the already-coded elements. Alternatively, the transform coefficients may be coded by context-adaptive binary arithmetic coding (CABAC). To mitigate blocking artifacts, an adaptive in-loop deblocking filter would be applied to the reconstruction from the feedback loop.
The **H.265/HEVC** block diagram is depicted in 2. The codec is very similar to H.264 but with the following major improvements/changes.

1. *Coding tree units and coding tree block (CTB) structure*: The core of the coding layer in the H.264 standard was the macroblock, containing a max. 16x16 block of luma samples and, in the usual case of 4:2:0 color sampling, two corresponding 8x8 blocks of chroma samples; whereas the analogous structure in HEVC is the coding tree unit (CTU), which has a size selected by the encoder and can be larger than a traditional macroblock. The CTU consists of a luma CTB and the corresponding chroma CTBs and syntax elements. The size LxL of a luma CTB can be chosen as L = 16, 32, or 64 samples, with the larger sizes typically enabling better compression. HEVC then supports a partitioning of the CTBs into smaller blocks using a tree structure and quadtree-like signaling [13].

2. *Coding units (CUs) and coding blocks (CBs)*: The quadtree syntax of the CTU specifies the size and positions of its luma and chroma CBs. The root of the quadtree is associated with the CTU. Hence, the size of the luma CTB is the largest supported size for a luma CB. The splitting of a CTU into luma and chroma CBs is signaled jointly. One luma CB and ordinarily two chroma CBs, together with associated syntax, form a coding unit (CU). A CTB may contain only one CU or may be split to form multiple CUs, and each CU has an associated partitioning into prediction units (PUs) and a tree of transform units (TUs).

3. *Prediction units and prediction blocks (PBs)*: The decision whether to code a picture area using interpicture or intrapicture prediction is made at the CU level. A PU partitioning structure has its root at the CU level. Depending on the basic prediction-type decision, the luma and chroma CBs can then be further split in size and predicted from luma and chroma prediction blocks (PBs). HEVC supports variable PB sizes from 64x64 down to 4x4 samples.

4. *TUs and transform blocks*: The prediction residual is coded using block transforms. A TU tree structure has its root at the CU level. The luma CB residual may be identical to the luma transform block (TB) or may be further split into smaller luma TBs. The same applies to the chroma TBs. Integer basis functions similar to those of a discrete cosine transform (DCT) are defined for the square TB sizes 4x4, 8x8, 16x16, and 32x32. For the 4x4 transform of luma intrapicture prediction residuals, an integer transform derived from a form of discrete sine transform (DST) is alternatively specified.

5. *Motion vector signaling*: Advanced motion vector prediction (AMVP) is used, including derivation of several most probable candidates based on data from adjacent PBs and the reference picture. A merge mode for Motion Vector (MV) coding can also be used, allowing the inheritance of MVs from temporally or spatially neighboring PBs. More-

over, compared to H.264/MPEG-4 AVC, improved skipped and direct motion inference are also specified.

6. Motion compensation (MC): Quarter-sample precision is used for the MVs, and 7-tap or 8-tap filters are used for interpolation of fractional-sample positions (compared to six-tap filtering of half-sample positions followed by linear interpolation for quarter-sample positions in H.264/MPEG-4 AVC). Similar to H.264/MPEG-4 AVC, multiple reference pictures are used. For each PB, either one or two motion vectors can be transmitted, resulting either in unipredictive or bipredictive coding, respectively. As in H.264/MPEG-4 AVC, a scaling and offset operation may be applied to the prediction signal(s) in a manner known as weighted prediction.
7. *Intrapicture prediction*: The decoded boundary samples of adjacent blocks are used as reference data for spatial prediction in regions where interpicture prediction is not performed. Intrapicture prediction supports 33 directional modes (compared to eight such modes in H.264/MPEG-4 AVC), plus planar (surface fitting) and DC (flat) prediction modes. The selected intrapicture prediction modes are encoded by deriving most probable modes (e.g., prediction directions) based on those of previously decoded neighboring PBs.
8. *Quantization control*: As in H.264/MPEG-4 AVC, uniform reconstruction quantization (URQ) is used in HEVC, with quantization scaling matrices supported for the various transform block sizes.
9. *Entropy coding*: Context adaptive binary arithmetic coding (CABAC) is used for entropy coding. This is similar to the CABAC scheme in H.264/MPEG-4 AVC, but has undergone several improvements to improve its throughput speed (especially for parallel-processing architectures) and its compression performance, and to reduce its context memory requirements.
10. *In-loop deblocking filtering*: A deblocking filter similar to the one used in H.264/MPEG-4 AVC is operated within the interpicture prediction loop. However, the design is simplified in regard to its decision-making and filtering processes, and is made more friendly to parallel processing.
11. *Sample adaptive offset (SAO)*: A nonlinear amplitude mapping is introduced within the interpicture prediction loop after the deblocking filter. Its goal is to better reconstruct the original signal amplitudes by using a look-up table that is described by a few additional parameters that can be determined by histogram analysis at the encoder side.

3. GPU BASED FAST MOTION ESTIMATION

In [6] the authors designed a GPU-based fast Motion Estimation (ME) (the GPU ME code was developed by the authors based on the H.264 JM 14.2 reference software). Their goal was to harness the GPU parallel capabilities and on how to trade-off the speed-up with Rate Distortion (RD) performance.
For that purpose they implmented Motion Estimation on the GPU based on *simplified unsymmetrical multihexagon search (smpUMHexagonS)* [14], which is one of the fast ME algorithms adopted by the H.264 JM reference software. They selected smpUMHexagonS because it can archieve very good tradeoff between computational complexity and coding efficiency. It is not only reported to achieve up to 94% redution in ME execution time with comparable RD efficiency, when compared with the fast full search in the JM software [14] but also to be very compact and therefore meeting the memory constraint of the GPU.

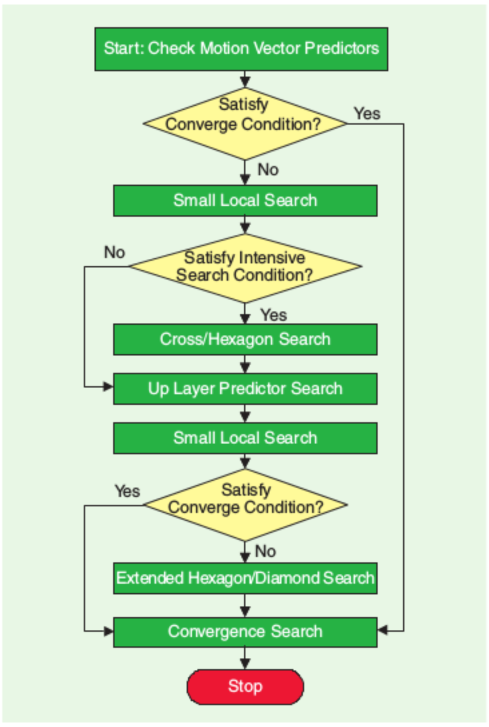

Start: Check Motion Vector Predictors
Satisfy Converge Condition?
Yes
No
Small Local Search
No
Satisfy Intensive Search Condition?
Yes
Cross/Hexagon Search
Up Layer Predictor Search
Small Local Search
Yes
Satisfy Converge Condition?
No
Extended Hexagon/Diamond Search
Convergence Search
Stop

Figure 3: *Fast ME using smpUMHexagonS [14]. The figure depicts the steps for integer-pel search for an MB partition.*

Figure 3 depicts the flow chart of smpUMHexagonS.For each MB, smpUMHexagonS computes the MVs for all the MB partitions (16 x 16, 16 x 8, . . . 4 x 4). MVs are selected by minimizing the Lagrangian cost $D + \lambda R$, where D is the SAD between the current block and the candidate, and R is the bit- rate to encode the MV. In smpUMHexagonS, computation reduction is achieved mainly by sampling the search space judiciously, using several techniques including motion vector pre- diction, different search patterns (cross, hexagon, and diamond) and early termination. In particular, MVs from spatially adjacent blocks and from other MB partitions are used to initialize the search for the current partition. Notice that as depicted in Figure 3, smpUMHexagonS uses several tests to determine if the search (of the current partition) can be terminated based on the mini- mum cost computed so far. As a result, different MBs with different contents may undergo different processing paths (which is typical in many fast ME algorithms [15]), and this may affect the performance of the GPU implementation. To utilize the parallelism in the GPU the authors used tiling, which means they partitioned the current frame into multiple tiles with each tile containing K (height) x L (width) MBs.

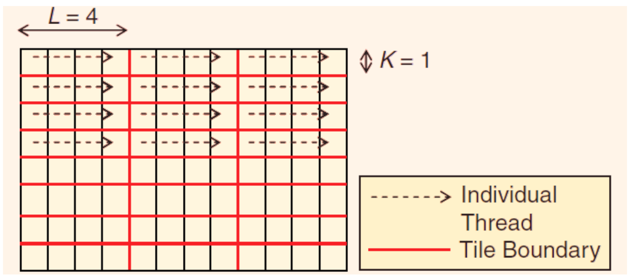


Figure 4: *GPU-based fast ME: the current frame is divided into multiple tiles to facilitate parallel processing in ME. Here each square represents an MB.*

Figure 4 shows the case with K = 1, L =4. Each tile is processed by a single GPU thread, i.e., each thread processes K *x* L MBs in a tile sequentially, and different tiles are processed by different independent threads concurrently on the GPU. As mentioned in 2, individual MBs are not independent under smpUMHexagonS. In particular, MBs depend on their neighbours. That means for the approach of tiling there will be a quality loss, due to the fact, that the authors fixed the dependency problem by assuming all MVs on the border of a tile are equal to zero.

3.1. Experimental Setup

To examine the performance of the GPU-based fast ME using tiling, the authors conducted experiments on PCs equipped with one GeForce 8800 GTS PCIe graphics card with 96 stream processors [16], and an Intel Core 2 Quad Q9400 2.66 GHz CPU with 3.23 GB of RAM. To implement the GPU code, they used NVIDIA's Compute Unified Device Architecture (CUDA) [17].
They first evaluated how the tiling may affect the RD performance. They used JM 14.2 to encode HD 720p sequences (1280 x 720 pixels, 60 frames per second) *Crew, City, Habor* and *Night*. They used the H.264 high profile with a search range of 64. All pictured were encoded as P-frames except the initial I-frame. For every experiment they used a width K=1 and up to 5 different values for the height L (1, 4, 16, 40, 80). They also measured the execution time of the GPU implementation (with tiling and using parallel processing on multicore) with different tile sizes and compared it to a Single Core (without tiling and using sequential processing) processor.

3.2. Experimental Results

As for the evaluation of the RD performance table 1 shows the average peak signal-to-noise-ration (PSNR) degradation and the average increase in bit rate using different tile sizes, measured by Bjontegaard Delta PSNR (BDPSNR) and Bjontegaard Dekta bit rate (BDBR)[4] respectively.

The results suggest tiling may lead to average degradation between 0.08 dB to 0.4 dB for these sequences with the tile size K=1, L=1.
Regarding the speed-up of the fast ME with tiling, figure 5 shows the speed-up of the GPU implementation compared to a Single-Core CPU implementation (smpUMHexagonS). The authors indicate that they took the overhead to transfer the video frames from system memory to GPU memory into account.They also mention, that both the GPU/CPU code have room for further improvement, especially the GPU code, because it stores the pixel data in global memory (off-chip memory), which has considerable access latency [17].

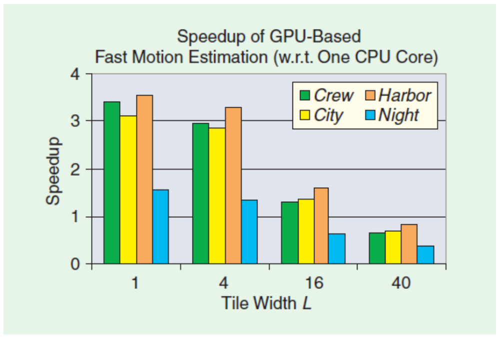


Figure 5: *Trade-off between tile width and speed-up*[5] *(tile height K is equal to one).*

Figure 5 suggests speed-ups of 1.5-3.5 with a tile size of K=1 and L=1.

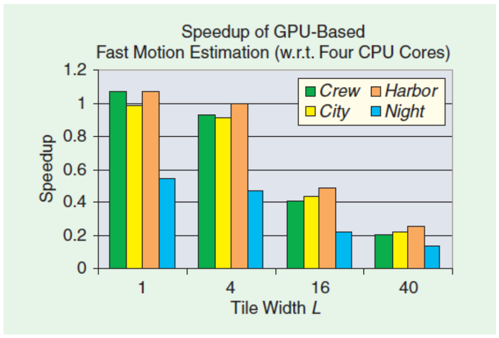


Figure 6: *Trade-off between tile width and speed-up*[5] *(tile height K is equal to one).*

Figure 6 shows the speedup between the GPU implementation and a parallel CPU implementation using the four CPU cores on the Intel Core 2 Quad. To achieve parallel CPU processing, the current frame is partitioned into four tiles of equal number of MB rows (i.e., L = width of the video frame in MB, K = height of the video frame in MB/4), and each tile is processed by an independent thread running on a CPU core. They observed the parallelization reduces the CPU running time by a factor of three approximately and therefore the previous speed-up in the comparison with the single-core processor is gone.

4. DYNAMIC MODEL FOR PARALLEL H.264/AVC VIDEO ENCODING ON HYBRID GPU+CPU

The authors of [19] proposed a dynamic model for parallel H.264/AVC video encoding on hybrid GPU+CPU. First they anal-

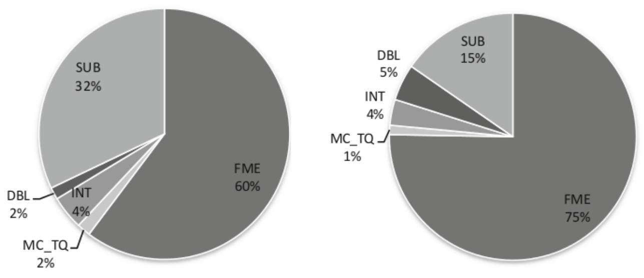
[4]BDPSNR and BDBR are used frequently in the video standardization community [18]

[5]Speed-up is the ratio of the CPU running time (sequential program code on one CPU core) to the GPU running time (including data transfer overhead).

		CREW		*CITY*		*HARBOR*		*NIGHT*	
TILE SIZE	NUMBER OF TILES	BDBR (%)	BDPSNR (DB)	BDBR (%)	BDPSNR (DB)	BDBR (%)	BDPSNR (DB)	BDBR (%)	BDPSNR (DB)
$K = 1,\ L = 1$	3,600	3.135	−0.082	12.933	−0.407	5.578	−0.221	4.636	−0.17
$K = 1,\ L = 4$	900	3.081	−0.079	11.115	−0.352	2.385	−0.094	3.546	−0.13
$K = 1,\ L = 16$	225	3.116	−0.08	11.171	−0.35	2.246	−0.089	3.415	−0.125
$K = 1,\ L = 40$	90	3.224	−0.083	10.821	−0.339	2.205	−0.087	3.4	−0.124
$K = 4,\ L = 80$	12	0.63	−0.016	1.412	−0.044	0.57	−0.022	1.19	−0.043
$K = 16, L = 80$	3	0.094	−0.003	0.261	−0.008	0.07	−0.003	0.161	−0.006

Table 1: *Tradeoff between tile size and RD perforance. Average increase in bit rate and average PSNR degradation are computed with respect to the reference software (i.e., without tiling).*

ysed the dependencies of the encoding process of H.264/AVC 1. This dependencies analysis showed, that parallel processing can only be considered within the scope of a single frame, since inter-prediction can not start before the list of Reference Frames (RF) is updated. Moreover, due to the intra-frame dependencies in the deblocking filter, this module can not be concurrently applied on two different regions of each slice. Hence, the conjunction of all these observations exclude the possibility of dividing each slice in several independent parts to be simultaneously processed in a pipeline fashion. Considering all the functional dependencies between the H.264/AVC modules, it can be concluded that only the interpolation and the ME can be processed in parallel, while the rest of the modules have to be sequentially processed. To simplify the authors decided that a single-slice frame configuration will be assumed for the paper, without any loss of generality.



(a) CPU. (b) GPU.

Figure 7: *Breakdown of the H.264/AVC interprediction-loop processing time (FME: full- pixel ME; SUB: sub-pixel ME; INT: interpolation; DBL: deblocking filter; MC TQ: di- rect transform, quantization, dequantization and inverse transform)*

Figure 7 represents a breakdown of the H.264/AVC processing time for both CPU and GPU implementations, regarding the several encoding modules. These profiling results were obtained for highly optimized implementations on an Intel Core i7 CPU and on an NVIDIA GeForce GTX 580 GPU. As it can be seen, ME is the dominant module, with more than 60% and 75% of the total processing time on CPU and GPU, respectively.
Because the heterogeneous structure of the H.264/AVC encoder includes modules with very different characteristics regarding the data dependencies and parallelization potential the authors analysed the possibility of minimizing the encoding time by efficiently distributing the several tasks on the CPU and on the GPU.

They created an algorithm which evaluates the performance of the GPU and CPU for certain parts of the encoding algorithm. They used the algorithm to determine during the encoding process which device has the best performance in which step of the process. With that knowledge always being updated during encoding they developed a dynamic load distribution model which distributes the different tasks of the encoding process to the CPU or GPU and keeps track of there individual execution times.

The proposed distribution procedure is illustrated in Figure 8. A data-flow diagram for all the encoding modules, considering both the CPU and GPU, is initially constructed. The transform and quantization tasks, as well as the dequantization and inverse transform, are presented together, due to the low computational requirements and simpler parallelization model. When the measured execution times are considered as a parameterization of each task, a weighted Directed Acyclic Graph (DAG) is obtained. The several nodes of such a graph (A, B ... H) are the decision points, where each task can be submitted to any of the two processing devices. The edges represent the individual task transitions, weighted by the respective computing and data transfer times. The shortest path between the starting and ending nodes of this graph represents the minimum encoding time.

4.1. Experimental Setup

To validate and evaluate the proposed model they used the H.264/AVC encoder implemented by the JM 17.2 reference software [20]. The video sequences considered for the experiments were *blue_sky, rush_hour* and *river_bed* (all at 1920 x 1080 pixels). The ME module was parametrized with a search area of 32x32 pixels. The used evaluation platforms adopted Linux operating system, CUDA 4.1 framework, icc 12.0 compiler and OpenMP 3.0 API to parallelize the video encoder. Platform 1 has an Intel Core i7 with 3Ghz and Platform 2 has an Intel Core 2 Quad with 2 Ghz. Both Platforms have 4GB memory and use a Geforce 580GTX.

4.2. Experimental Results

Figure 9 presents the achieved encoding performance of the experiments for both hybrid platforms. The diagrams compares five different scheduling strategies:

- **CPU-only**: the whole encoder is implemented in the CPU
- **GPU-only**: the whole encoder is implemented in the GPU
- **Chen_original**: method proposed by Chen [21], where the ME, SME and interpolation modules are statically offloaded to the GPU (the rest are kept in the CPU)

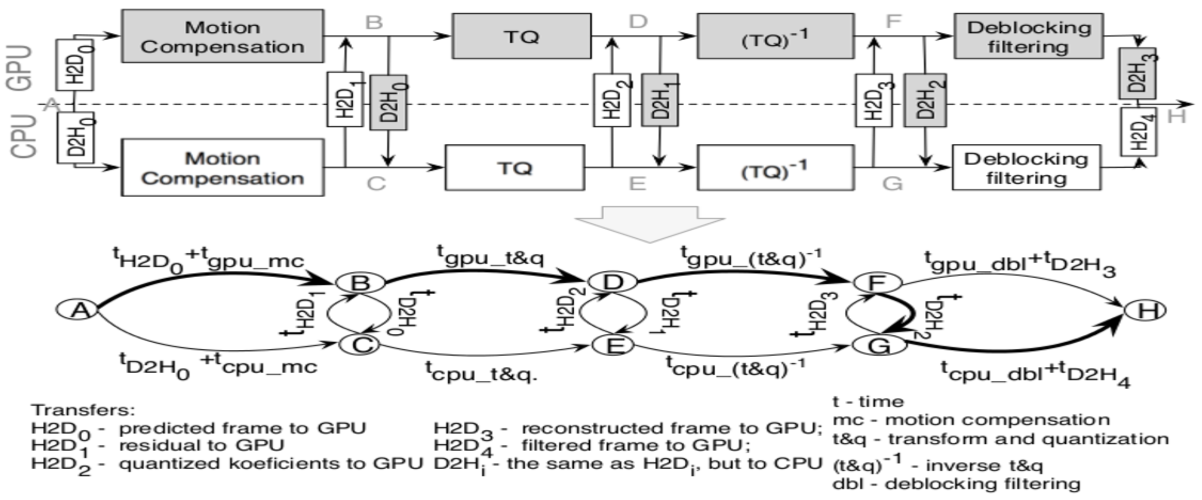


Figure 8: *Construction of weighted DAG from the data-flow diagram of H.264/AVC and a possible minimal path (represented in bold)*

- **Chen_optimized**: Chen's encoder [21], optimized with OpenMP and SSE4 vec- torization techniques
- **Proposed**: presented dynamic load distribution strategy

In 9 the performance regarding the frames per second considering different number of reference frames for both platforms are presented. As it can be seen, the Proposed method achieves speedup levels of up to 1.5 and 2, when compared with the GPU-only implementation and with the Chen optimized strategy, respectively, and a speedup of up to 5 when compared with the CPU-only implementation. Due to the fact that Platform 1 has a significantly slower CPU, the Chen optimized strategy achieves a lower performance when compared to the GPU-only implementation, while in the case of Platform 2 their performances are very similar. However, GPU-only requires a CUDA implementation of all the H.264/AVC modules of the inter-loop. The impact of the considered OpenMP and SSE4 vectorization [22] optimizations of the CPU code are emphasized when comparing the results obtained for the Chen original and Chen optimized approaches. Contrasting with Chen optimized and GPU-only approaches, the proposed algorithm achieves a higher performance that is less dependent on the adopted hybrid platform. Contrasting with Chen optimized and GPU-only approaches, the Proposed algorithm achieves a higher performance that is less dependent on the adopted hybrid platform. In particular, by simultaneously using the computational resources of the CPU and the GPU devices, in an adaptive and dynamic load balanced fashion, both processing devices participate in a much better distribution of the computational load, based on a constantly updated prediction of the offered performance by each device. Finally, it can be seen that a real time encoding with more than 20 fps is achieved on both platforms for a single RF.

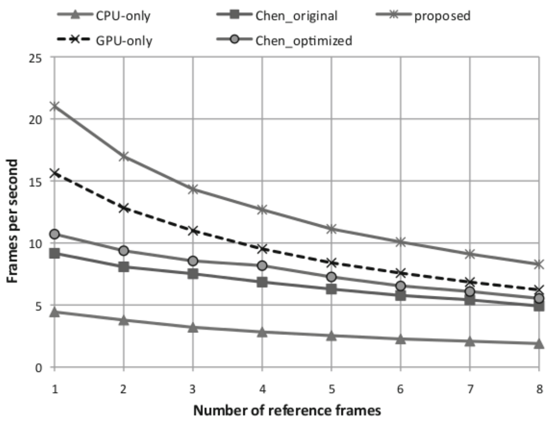
5. CPU PLUS GPU PARALLEL ENCODING FRAMEWORK FOR H.265/HEVC

In [23] the authors used (unlike in the other two papers) for their approach the H.265/HEVC codec. They present in their work a CPU plus GPU parallel encoding framework which targets the critical bottlenecks like Variable Block Size Motion Estimation (VBSME) of the encoding algorithm and optimises the data flow between the devices. The proposed framework is shown in figure 10

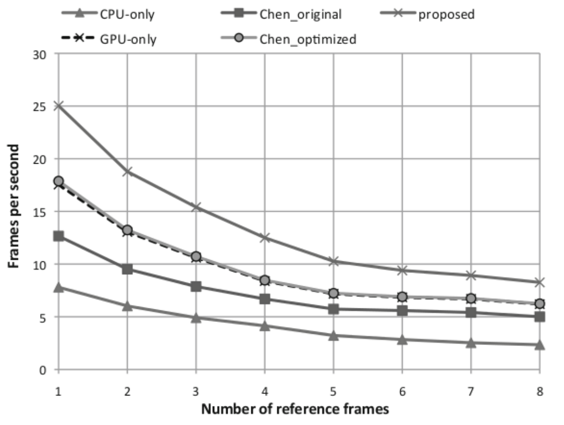
The HEVC encoder is mainly divided into six components: VBSME, Mode Decision, MC, Interpolation, Encode plus Reconstruction and Entropy coding. The VBSME and Interpolation components, i.e. the right part in figure 10, run on GPU, the others run on cpu. There are two loops in the parallel encoding framework, the CTU line loop and the frame loop as shown in the right part of figure 10. The input sequence is coded frame by frame in the frame loop. One frame is divided into a series of non-overlapped CTU lines which are coded line by line in the CTU line loop. The Mode Decision, VBSME, MC and Encode plus Reconstruction components process all the data of one CTU line in one pass, while Entropy coding and Interpolation components process all the data of one frame in one pass.
The components on GPU and the components on CPU run in parallel by two Synchronization components: Sync to Interpolate and Sync to Last CTU Line ME. To calculate the RD cost for mode decision, PU reconstruction and entropy coding of all syntaxes are necessary. Mode decision becomes the bottleneck in the presented parallel encoding framework. Therefore the authors proposed a fast PU partition scheme to speed up mode decision in 6 steps. This scheme target is to be fast with the cost of an as little as possible quality loss.

5.1. Experimental Setup

For their experiments the authors used the workstation Z620 produced by Hewlett-Packard. It contains one NVIDIA Tesla C2050 GPU with 448 CUDA cores at a clock rate of 1.15Ghz. The used CPUs weren't specified.
They evaluated the acceleration performance of the improved VBSME on GPU and the RD in comparison to the reference encoder (HM9.0 [24]). The improved VBSME runs on one CPU core and GPU, in parallel and serially, respectively, for acceleration performance comparison. The search range is 64x64 with the full search strategy for integer motion vector (IMV) and 24 fractional-pixel positions around the IMV. Two test sequences with different resolutions *BasketballDrive* (1920x1080) and *Traffic* (2560x1600) were used.



(a) Platform 1.



(b) Platform 2.

Figure 9: *Encoded frames per second (fps) for a varying number of RFs, using the 1920x1080 video format. Comparison of the proposed approach with CPU-only, GPU-only and the approach proposed by Chen [21].*

5.2. Experimental Results

In table 2 the results of the speed evaluation are presented. For the 2560x1600 resolution sequence, the GPU version of VBSME achieves 23.77 fps, while the single CPU thread version only process 0.21 fps. The speedup ratio is about 113 times. The same acceleration performance is obtained for the 1920x1080 resolution sequence. The ME time is picture independent due to full search. The results are only valid for a specific configuration of the proposed and the HM encoder.

The diagram 11 shows the RD curves of the two encoders. From these RD curves, it can be seen that the RD performance degradation of the proposed encoder with fast CU partition decision algorithm is about 0.7 dB degradation which is calculated followed by BDPSNR [18]. The encoding speed of the HM encoder is about 0.03 fps while the average speed of the proposed encoder is 14.5 fps.

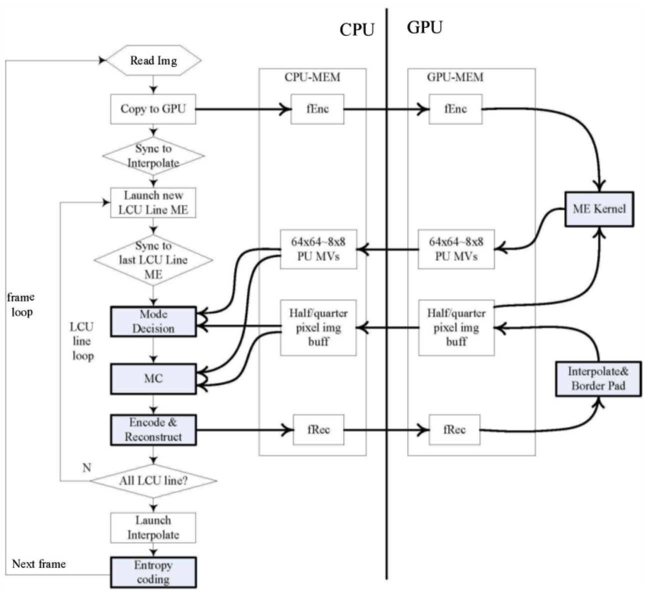


Figure 10: *Parallel encoding framework on CPU plus GPU platform. The thick lines with arrows represent the dataflow directions.*

Sequence	CPU(fps)	GPU(fps)	Speedup ratio
Traffic_2560x1600	0.21	23.77	113.2
ParkScreen_1920x1080	0.69	77.76	112.7

Table 2: *Speedup gains on different video sequences*

6. CONCLUSIONS

This paper reviewed three different approaches to utilize parallel power of modern computers for the means of reaching a better encoding performance using either the previous (H.264/AVC) or the upcoming (H.265/HEVC) standard video codec.
In particularly the first paper [6] proposed a GPU based fast motion esitmation framework for H.264/AVC with the main idea of tiling a frame. The ME of each tile is then to be calculated on a different GPU thread to reach a high level of parallelization. For this approach was only a case study, the results only indicated, that it is possible to reach a high level of parallelization, but with the implemented framework didn't prove to be efficient in speed or quality.
The second work [19] proposed a dynamic model for parallel H.264/AVC video encoding on hybrid GPU+CPU. Their main idea was to evaluate at runtime which device is more suitable for different tasks of the encoding algorithm. The implemented framework used the dynamic model and distributed every task according to the runtime analysis. Given the experimental results and the speed-ups they reached this dynamic model showed a lot of potential.
The last paper [23] was the only reviewed work, which used the H.265/HEVC codec. A CPU+GPU parallel encoding framework was proposed, which aimed to use the high parallel performance power of GPUs for the compute intensive tasks of the algorithm that can be parallelized and to efficiently direct the data flow between CPU and GPU. The results presented the speed-ups in comparison to a reference encoder and showed a huge increase using the proposed framework.

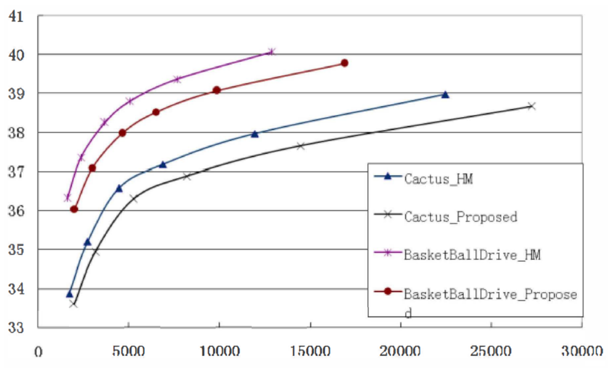


Figure 11: *RD curve of the HM encoder and the proposed encoder.*

All three presented works showed a potential speed-up in video encoding using CPU+GPU. It has to be taken into account, that especially for the last work presented the comparison of the result is rather poor, because the authors only compared their optimized framework with a badly implemented encoder on slow hardware. Besides that fact it can be seen in all 3 works, that there is a lot of potential in using parallel frameworks for the encoding of modern video codecs like H.264/AVC and H.265/HEVC. Regarding the increasing demand and the upcoming standard resolution (Ultra-HD) of video content more and better solutions towards using the parallel performance of modern computers are needed.

7. REFERENCES

[1] Gary J Sullivan, Jens Ohm, Woo-Jin Han, and Thomas Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012.

[2] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra, "Overview of the h. 264/avc video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[3] Barry G Haskell, *Digital Video: An Introduction to MPEG-2: An Introduction to MPEG-2*, Springer, 1997.

[4] Mahsa T Pourazad, Colin Doutre, Maryam Azimi, and Panos Nasiopoulos, "Hevc: the new gold standard for video compression: how does hevc compare with h. 264/avc?," *Consumer Electronics Magazine, IEEE*, vol. 1, no. 3, pp. 36–46, 2012.

[5] Ming Liou, "Overview of the p× 64 kbit/s video coding standard," *Communications of the ACM*, vol. 34, no. 4, pp. 59–63, 1991.

[6] Ngai-Man Cheung, Xiaopeng Fan, Oscar C Au, and Man-Cheung Kung, "Video coding on multicore graphics processors," *Signal Processing Magazine, IEEE*, vol. 27, no. 2, pp. 79–89, 2010.

[7] Yu-Cheng Lin, Pei-Lun Li, Chin-Hsiang Chang, Chi-Ling Wu, You-Ming Tsao, and Shao-Yi Chien, "Multi-pass algorithm of motion estimation in video encoding for generic gpu," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. IEEE, 2006, pp. 4–pp.

[8] Chi-Wang Ho, Oscar C Au, S-HG Chan, Shu-Kei Yip, and Hoi-Ming Wong, "Motion estimation for h. 264/avc using programmable graphics hardware," in *Multimedia and Expo, 2006 IEEE International Conference on*. IEEE, 2006, pp. 2049–2052.

[9] Martin Schwalb, Ralph Ewerth, and Bernd Freisleben, "Fast motion estimation on graphics hardware for h. 264 video encoding," *Multimedia, IEEE Transactions on*, vol. 11, no. 1, pp. 1–10, 2009.

[10] Svetislav Momcilovic and Leonel Sousa, "Development and evaluation of scalable video motion estimators on gpu," in *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*. IEEE, 2009, pp. 291–296.

[11] Bart Pieters, C-FJ Hollemeersch, Jan De Cock, Peter Lambert, Wesley De Neve, and Rik Van de Walle, "Parallel deblocking filtering in mpeg-4 avc/h. 264 on massively parallel architectures," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 1, pp. 96–100, 2011.

[12] Thomas Wiegand, "Joint final committee draft (jfcd) of joint video specification (itu-t rec. h. 264| iso/iec 14496-10 avc)," *JVT-D157, Aug*, 2002.

[13] Hanan Samet, "The quadtree and related hierarchical data structures," *ACM Computing Surveys (CSUR)*, vol. 16, no. 2, pp. 187–260, 1984.

[14] Xiaoquan Yi, Jun Zhang, Nam Ling, and Weijia Shang, "Improved and simplified fast motion estimation for jm," in *JVT-P021. doc, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, 16th meeting, Pozan, Poland*, 2005.

[15] Iain E Richardson, *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*, John Wiley & Sons, 2004.

[16] NVIDIA GeForce, "8800 gpu architecture overview," *Technical Brief TB-02787-001 v0*, vol. 9, 2006.

[17] Nvidia, "Cuda programming guide," Tech. Rep., 2009.

[18] G Bjøntegaard, "document vceg-m33: Calculation of average psnr differences between rd-curves," in *ITU-T VCEG Meeting, Austin, Texas, USA, Tech. Rep*, 2001.

[19] Svetislav Momcilovic, Nuno Roma, and Leonel Sousa, "Multi-level parallelization of advanced video coding on hybrid cpu+ gpu platforms," in *Euro-Par 2012: Parallel Processing Workshops*. Springer, 2013, pp. 165–174.

[20] "Itu-t: Jvt reference software unofficial version 17.2," 2010, `http://iphome.hhi.de/suehring/tml/download`.

[21] Wei-Nien Chen and Hsueh-Ming Hang, "H. 264/avc motion estimation implmentation on compute unified device architecture (cuda)," in *Multimedia and Expo, 2008 IEEE International Conference on*. IEEE, 2008, pp. 697–700.

[22] "Intel: Sse4 programming reference," 2007, `http://edc.intel.com/Link.aspx?id=1630`.

[23] Xiangwen Wang, Li Song, Min Chen, and Junjie Yang, "Paralleling variable block size motion estimation of hevc on cpu plus gpu platform," in *Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1–5.

[24] "Hm 9.0 encoder," `https:llhevc.hhi.fraunhofer.de/svnlsvn_HEVCSoftware`.