Conference
4–8 July, 2016.

Parallel processing in GPUs for intra-picture prediction in HEVC

Vicente Galiano[1], Héctor Migallón[1], Victoria Herranz[2], Pablo Piñol[1], Otoniel López-Granado[1] and Manuel P. Malumbres[1]

[1] *Department of Physics and Computer Architecture, Miguel Hernández University*

[2] *Center of Operations Research, Miguel Hernández University*

emails: vgaliano@umh.es, hmigallon@umh.es, mavi.herranz@umh.es, pablop@umh.es, otoniel@umh.es, mels@umh.es

Resumen

The HEVC video coding standard launched on 2013, is able to reduce to the half, on average, the bit stream size produced by H.264/AVC encoder at the same video quality, but it requires nearly 70 % more time than H.264/AVC to encode a video sequence. GPUs can help to reduce this coding time considerably. In this paper, we propose the use of GPUs to perform the intra-picture prediction, explaining which steps in the coding process has been traslated to GPU and comparing the coding time with the one obtained on a CPU.

Key words: Parallel algorithms, video coding, HEVC, GPUs, performance, prediction

1. Introduction

HEVC, the High Efficiency Video Coding standard [1] launched on January 2013 by the Joint Collaborative Team on Video Coding (JCT-VC) is the newest video coding standard of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). The main video coding standard preceding HEVC is the current H.264/AVC [2] standard, but an increasing diversity of services and the emergence of 4K and 8K resolution are creating stronger needs for better coding efficiency. HEVC greatly improved this coding efficiency over its predecessor (H.264/AVC) by a factor of almost twice while maintaining an equivalent visual quality [3]. HEVC has been designed to cover all services of H.264/AVC and to focus on two key issues: higher video resolutions and increased use of parallel processing architectures. In terms of complexity, Bossen et al. [4]

 ISBN: 978-84-608-6082-2

studied the complexity of HEVC encoding and decoding software showing that the encoding process is much more challenging than the decoding process.

Despite being a recent standard, we can find in the literature several works about complexity analysis and parallelization strategies for the HEVC standard [4, 5, 6]. Most of the parallelization proposals are focused in the decoding side, looking for the most appropriate parallel optimizations at the decoder that provide real-time decoding of High-Definition (HD) and Ultra-High-Definition (UHD) video contents. In [7] and [8] the authors present a technique called Overlapped Wavefront (OWF) for the HEVC decoder which is a variant of Wavefront Parallel Processing (WPP) in which the executions over consecutive pictures are overlapped. In a multi-threaded approach of the HEVC decoder, a picture is decoded by several threads at the same time, and each thread decodes different Coding Tree Unit (CTU) rows. In these works, authors claim that a single thread may continue processing the next picture when it finishes the current one, without waiting for the other threads. These improvements allow a better parallel processing efficiency, reducing the overall decoding time. Recently, in [9] the authors combine Tiles, WPP with SIMD (Single-Instruction Multiple-Data instruction set extension to the x86 architecture) instructions to develop a real-time HEVC decoder.

However, there are less works focused at the HEVC encoder. In [10] authors propose a fine-grain parallel optimization in the motion estimation module of the HEVC encoder allowing to perform the motion vector prediction in all Prediction Units (PUs) available at the Coding Unit (CU) at the same time.

In [11], authors apply parallel processing techniques to HEVC encoder. Authors propose several, synchronous and asynchronous, parallelization approaches working at a coarse grain parallelization level, based on the Group Of Pictures (GOP), where several groups of consecutive frames are encoded simultaneously using a multicore platform with shared memory. The results show that near ideal efficiencies are obtained using up to 10 cores.

A proposal for parallelization for distributed memory systems is presented in [12]. In this paper, authors present several parallelization approaches to the HEVC encoder for distributed memory platforms which work at a coarse grain level parallelization, being one group of pictures (GOP) the basic structure. These approaches encode simultaneously several GOPs and ideal parallel behavior is shown for a right GOP conformation and distribution.

In [13], authors combine a GPU-based motion estimation algorithm with two different parallelization techniques: WPP and group of pictures (GOP). This approach allows a multicore system to process multiple Coding Tree Units (CTUs) by splitting the frame in rows or the sequence in GOPs, respectively. In either case, the motion estimation of these regions is issued to the GPU device obtaining speed-ups of up to 3.93x for 4 processes.

In [14] authors propose a parallelization inside the intra prediction module that consist on removing data dependencies among subblocks of a CU, obtaining interesting speed-up results.

 ISBN: 978-84-608-6082-2

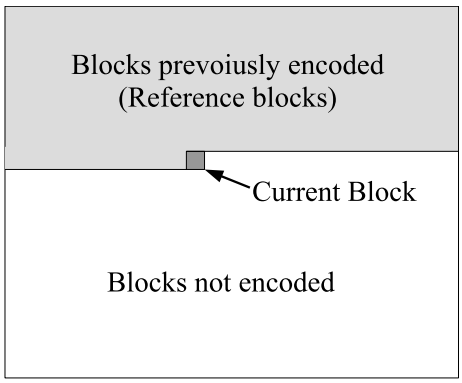


Figura 1: Avalaible blocks on decoder side for intra-prediction

Recently, in [15], authors present an hybrid approach combining WPP and intra prediction on GPUs obtaining reductions on the total encoding time of up to 62 % with a lower coding performance loss.

In this paper, we will focus on applying parallel processing in Graphic Processors Units (GPUs) to the intra-picture prediction process of the HEVC encoder. The remainder of this paper is organized as follows, in Section 2 an overview of intra-picture prediction in HEVC and an introduction to the main algorithms used are presented. Section 3 present the parallelization strategy proposed for using GPUs parallelism in the intra-picture prediction process, while in Section 4 an evaluation of the proposed architecture and parallel strategies is presented. Finally, in Section 5 some conclusions are drawn.

2. Intra-picture Prediction in HEVC

The basic source-coding algorithm is a hybrid of inter-picture prediction to exploit temporal statistical dependences, intra-picture prediction to exploit spatial data dependences, and transform coding of the prediction residual signals to further exploit spatial data dependences. These three elements provide an improvement in compression efficiency when compared to the previous video coding standard H.264/AVC. In particular, the intra-picture prediction process consists on the prediction of a block of pixels of current frame, using the information from neighbouring blocks pixels. It supports three different modes, the angular mode with 33 different directions, the planar mode and the DC mode.

The values of the samples in a frame are often similar to their adjacent neighbour samples' values; this is called spatial redundancy or intra-frame correlation. This redundant information in the spatial domain can be exploited to compress the image. Each frame is partitioned in blocks of pixels and for each block the prediction of its pixels is done using the pixels belonging to previously encoded adjacent blocks. So, at the decoder side, only the

 ISBN: 978-84-608-6082-2

pixels along the upper and/or left edges can be used to create the prediction block as shown in Figure 1. Once the prediction has been generated, it is subtracted from the current block to form a residual signal. The residual signal is transformed into the frequency domain and binary arithmetic encoded, together with the selected prediction mode.

In HEVC, a frame is split into one or several slices and an intra slice contains a number of consecutive CTUs, which are partitioned into CUs. The maximum CU size is 64x64 (the size of one CTU), and the minimum size is 8x8. A CU is considered as a whole or partitioned into 4 smaller CUs (forming a quadtree). Whether to further split the current CU depends on its Rate-Distortion (RD) cost and the total RD cost of the 4 smaller CUs.

HEVC employs 35 different intra modes to predict a CU, compared to the 8 modes available in H.264/AVC. The video encoder will choose the intra prediction mode that provides de best RD performance. The prediction modes are organized into three categories:

- Planar prediction: the value of each sample of the prediction CU is calculated assuming an amplitude surface with a horizontal and vertical slope derived from the boundary samples of the neighbouring blocks (mode 0) .
- DC prediction: the value of each sample of the prediction CU is an average of the boundary samples of the neighbouring blocks (mode 1).
- Directional prediction with 33 different directional orientations: the value of each sample of the prediction CU is calculated extrapolating the value from the boundary samples of the neighbouring blocks as shown in Figure 2 (mode $2 \dots 34$).

The residual block is obtained as the difference between the original CU and the prediction CU, or:

$$residualAngularBlock = OriginalCU - PredictionAngCU$$

Finally the Sum of Absolute Differences (SAD) of the residual block is calculated as:

$$SADmodeAngular = sum(abs(residualAngularBlock))$$

In HEVC, the increase in the number of intra prediction modes provides substantial coding performance gain over H.264/AVC, but it also makes the RD optimization process more complex. The fast encoding algorithm of HEVC reference software includes two phases.

In the first phase, called Rough Mode Decision (RMD), the N most promising candidate modes are selected. In this process, all candidates (35 modes) are evaluated with respect to the following Rate/Distortion cost function:

$$C = D_{Had} + \lambda \cdot R_{mode}$$

where the D_{Had} represents the absolute sum of Hadamard transformed residual signal for a CU, λ is the Lagrange multiplier that determines the trade-off between rate and distortion,

 ISBN: 978-84-608-6082-2

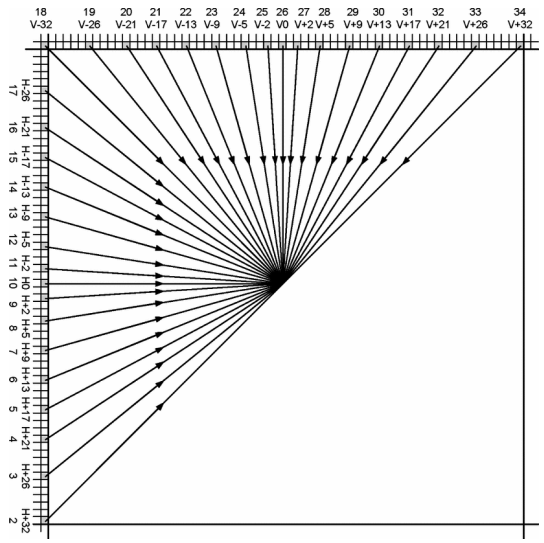


Figura 2: HEVC angular intra prediction modes

and R_{mode} represents the estimaded number of bits of the block when it is encoded with CABAC. Then, up to three modes with the lowest costs are added to a subset of candidates (SC).

In the second phase, the full RD optimization process is performed on all the candidates in set SC, and the intra prediction mode with the minimum RD cost is selected. The total complexity of this step depends on the number of modes in set SC.

3. Moving Hadamard transformed to GPUs

The release of NVIDIA CUDA API [16, 17, 18] to the developers has led to an spectacular increase of interest in using the GPU capabilities towards faster and more efficient parallel algorithms. The GPU serves as a coprocessor to the CPU through the CUDA API and exploits the massive data parallelism on the Single Instruction Multiple Data (SIMD) architecture of the GPU. Independently operating threads executing CUDA kernels while efficiently sharing high speed memory can be implemented with a set of threads being organized into blocks. It should be noted that to obtain higher bandwidth and overall performance gains, memory sharing between threads must be optimized with very careful programming to ensure very low latency between reads/writes.

At the moment, we have introduced the algorithm used in the intra-picture prediction which is based in a Single Instruction Single Data (SISD) architecture. SAD computing is implemented to work with blocks of size up to 8 x 8. So we need to perform 8 x 8 transforms

 ISBN: 978-84-608-6082-2

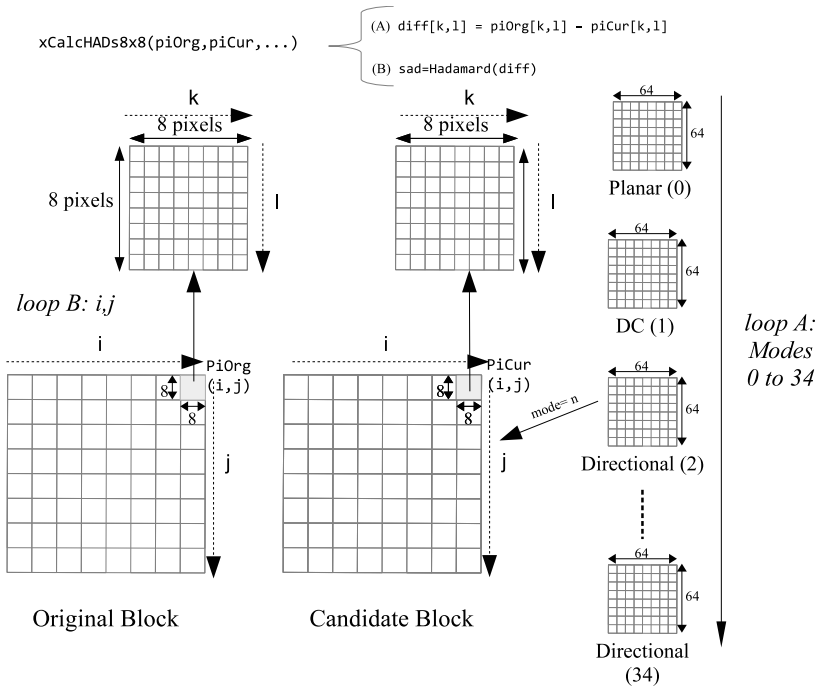


Figura 3: Algorithm for calculate D_{Had} for each mode

of the prediction residuals resulting from each intra-prediction mode, in a CTU of 64 x 64 pixels. In this section, we explain how Hadamard transformed is implemented in the HM 16.3 reference software, and how we have moved this massive computing to GPUs in order to accelerate the prediction algorithm using the data parallelism.

In Figure 3, a representation of the algorithm used to compute the SAD of each mode is illustrated. First, we have the original block on the left and the best candidate block must be searched for modes 0 to 34 (loop A). For the candidate block corresponding to the intra-prediction mode n, the difference with the original block is Hadamard transformed in blocks of 8×8 pixels, so we must get the transformed to 64 subblocks of size 8×8 (loop B). In the HM 16.3 reference software, the function `xCalcHAD8x8` is called for each original subblock (i, j) and each candidate subblock (i, j) with size 8×8. This function has two steps: (1) the difference between predicition and original pixels is calculated, and (2) the Hadamard transformed from the resulting difference matrix is obtained. Inside this function, a new iteration (loop C) is done to calculate the difference for each matrix element (k, l) and its Hadamard transformed. As we can note, there are not data dependency between adjacent subblocks and all SAD values for each CTU could be calculated simultaneously. But in each subblock there are dependencies on the Hadamard transformed computing, and there are

 ISBN: 978-84-608-6082-2

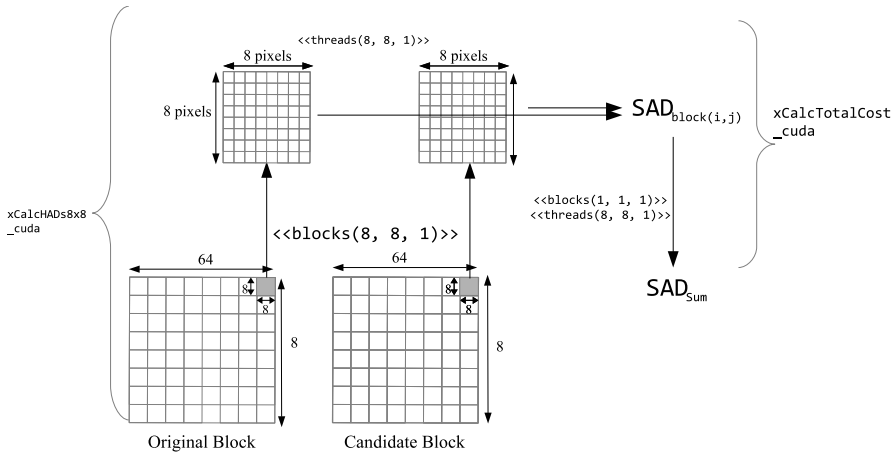


Figura 4: Concurrent SAD computation in GPU

dependencies in the necessary reduction processes to compute the sublock differences.

In this work we propose the concurrent calculation of SAD values (including the Hadamard transformed) for all subblocks belonging to the CTU. In Figure 4, the concurrent algorithm to calculate the SAD in GPU is showed. The sum of SAD values is obtained in two steps. In a first step, we define a kernel grid of 8×8 blocks of 8×8 threads. Each block of threads must calculate the SAD value of each subblock. Inside each block of threads, we organize them in a 8 x 8 mesh, so thread (i,j) should compute (a) the difference of the pixels (i,j) of the original and the candidate subblocks, and (b) its corresponding Hadamar transformed value. At the end of this first step, we get a vector of 64 elements with the SAD value for each subblock. Due to each value has been computed by a block of threads, to sum these values, a new kernel in the GPU is called performing an optimized reduction process based on the use of shared memory. The result is the SAD_{Sum} associated with the mode n. We must note that before the first computation step, both CTUs (original and candidate) of size 64×64 must be transfered from host memory to the global memory of the device using asyncrhonous transfers. Transfer times can be overlapped with computing times for previous CTUs. Thereby, communication times between global memory and device memory should not represent a significant penalty. On the other hand, the computation of sum values in the first and second steps is implemented using the shared memory. This shared memory is allocated per thread block, so all threads in the block have access to the same shared memory, which latency is roughly $100x$ lower than uncached global memory latency.

 ISBN: 978-84-608-6082-2

4. Performance Evaluation

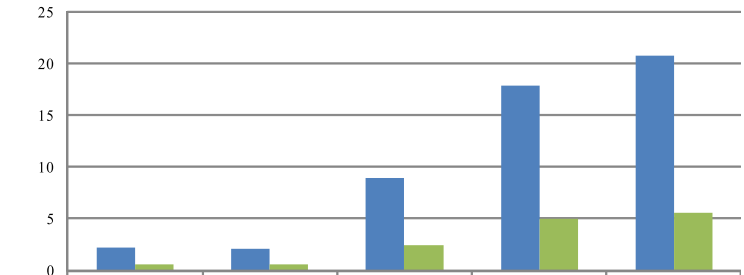
In this section we present the performance evaluation of our GPU-based SAD computing algorithm. Two different platforms have been used in this work. The first one is a Nvidia Tesla M2050 which contains 448 CUDA cores with 3 GB of dedicated video memory. The second one is a laptop GPU Geforce GT540M with 96 CUDA cores and 2 GB of video memory. The sequential algorithm of the HEVC reference software has been also executed in two platforms: first, a node with two processors Intel Xeon X5660 and 48 GB of RAM memory and second a labtop with an Intel i7-2670QM at 2.2GHz and 8GB of RAM memory.

In Figure 5, we present the computational times required for computing SADs in both GPU platforms (Tesla M2050 and Gforce GT540M, respectively) and for five different video sequences: Monaco (352×288), BQSquare (416×240) , BasketballDrill (832×480), ChinaSpeed (1024×768) and Johnny (1280×720). In all video sequences, we have encoded 50 frames. In both figures, *CPU* means the sequential computational time required to compute the SAD values and *GPU* indicates the time for the proposed GPU based algorithm. As we can observe, we obtain remarkables time reductions, above 70 % for the M2050 and above 60 % for the GT540M. Note that the timer reduction does not depend on the resolution image. Obviously, due to M2050 is more powerful than the GT540M, we obtain better computational results using the first one. On the other hand, note tha both the Hadamard transformed and the SAD compunting include reduction operations. These reduction operations decrease substantially the inherent parallelism. To be exploited, we execute these operations on the GPUs managing efficiently the shared memory abd mapping suitably the kernel grid.

5. Conclusions and Future Work

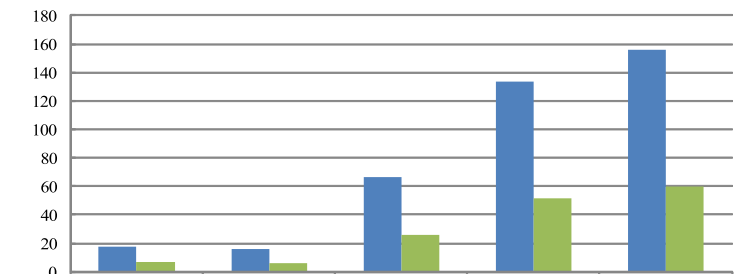
In this paper we have proposed the use of GPUs for computing the SAD values used in intra-picture prediction in HEVC. The search of the best prediction block implies a gready search of candidates among 35 modes for each block. On the other hand, this prediction algorithm is massively used in all CTUs that belongs to a frame. Besides, we have detailed how the algorithm is implemented in the 16.3 HM reference software and how concurrency can be highly exploited when computing the sum of CTU's SAD, spliting it in blocks and threads. Considering that the time reduction are up to 73 %, we value these result as the first step to substantially improve the intra-picture prediction algorithm using GPUs. For future work, we plan to increase the concurrent task in GPUs computing all modes at the same time. Even further work will lead us to make a prediction of the entire frame concurrently.

 ISBN: 978-84-608-6082-2



	Monaco	BQSquare	BasketballDrill	ChinaSpeed	Johnny
CPU	2.20	2.11	8.92	17.86	20.78
GPU	0.64	0.60	2.41	4.96	5.62

(a) Nvidia Tesla M2050



	Monaco	BQSquare	BasketballDrill	ChinaSpeed	Johnny
CPU	17.33	15.79	66.88	133.90	155.76
GPU	6.80	6.07	25.72	51.50	59.91

(b) Nvidia GForce GT540M

Figura 5: Computational times for GPU-based Hadamard transformed for intra-picture prediction in HEVC

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness under Grant TIN2015-66972-C5-4-R co-financed by FEDER funds.

Referencias

[1] B. Bross, W. Han, J. Ohm, G. Sullivan, Y.-K. Wang, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 10," *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.

 ISBN: 978-84-608-6082-2

[2] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012*, 2012.

[3] G. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1648 –1667, December 2012.

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.

[5] M. Alvarez-Mesa, C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *International Conference on Acoustics, Speech, and Signal Processing, Kyoto*, March 2012, pp. 1–17.

[6] E. Ayele and S.B.Dhok, "Review of proposed high efficiency video coding (HEVC) standard," *International Journal of Computer Applications*, vol. 59, no. 15, pp. 1–9, 2012.

[7] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, , and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1827 –1838, 2012.

[8] C. C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl, "Parallel HEVC decoding on multi- and many-core architectures," *Journal of Signal Processing Systems*, vol. 71, no. 3, pp. 247 –260, 2013.

[9] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, and T. Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10," Joint Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), Tech. Rep. JCTVC-L1003, January 2013.

[10] Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.

[11] H. Migallón, J. Hernández-Losada, G. Cebrián-Márquez, P. Piñol, J. Martínez, O. López-Granado, and M. Malumbres, "Synchronous and asynchronous {HEVC} parallel encoder versions based on a {GOP} approach," *Advances in Engineering Software*, pp.–, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S096599781630028X

 ISBN: 978-84-608-6082-2

[12] H. Migallón, V. Galiano, P. Piñol, O. López-Granado, and M. P. Malumbres, "Distributed memory parallel approaches for hevc encoder," *The Journal of Supercomputing*, pp. 1–12, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11227-016-1666-2

[13] G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, M. Tang, and J. Wen, "Accelerating HEVC using heterogeneous platforms," *Journal of Supercomputing*, vol. 71, no. 2, pp. 613 –628, February 2015.

[14] J. Jiang, B. Guo, W. Mo, and K. Fan, "Block-based parallel intra prediction scheme for HEVC," *Journal of Multimedia*, vol. 7, no. 4, pp. 289 –294, August 2012.

[15] S. Radicke, J. U. Hahn, Q. Wang, and C. Grecos, "A parallel hevc intra prediction algorithm for heterogeneous cpu+gpu platforms," *IEEE Transactions on Broadcasting*, vol. 62, no. 1, pp. 103–119, March 2016.

[16] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *Queue*, vol. 6, no. 2, 2008, pp. 40–53.

[17] N. Corporation, "Nvidia cuda c programming guide. version 3.2."

[18] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," in *IEEE Micro*, vol. 28, no. 2, 2008, pp. 39–55.

 ISBN: 978-84-608-6082-2