**The Journal of Supercomputing manuscript No.**
(will be inserted by the editor)

# GPU-based HEVC Intra Prediction Module

**V. Galiano · H. Migallón · V. Herranz · P. Piñol · O. López-Granado · M.P. Malumbres**

Received: date / Accepted: date

**Abstract** The HEVC video coding standard requires nearly 70% more time than H.264/AVC to encode a video sequence. Manycore architectures can considerably help to reduce the coding time. In this paper, we propose the use of GPUs to perform the intra-picture prediction without any R/D loss. We have evaluated our proposal and compared the obtained results with the ones obtained when running on a CPU. The results show that a time reduction of up 85% can be obtained without any R/D loss.

**Keywords** Parallel algorithms, video coding, HEVC, GPUs, performance, intra prediction, manycore

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

V. Galiano
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.
Tel.: +34-966658394
Fax: +34-966658814
E-mail: vgaliano@umh.es

H. Migallón
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

V. Herranz
Center of Operations Research, Miguel Hernández University, 03202 Elche, Spain.

P. Piñol
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

O. López-Granado
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

M.P. Malumbres
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

## 1 Introduction

The High Efficiency Video Coding (HEVC) standard was designed and developed by the Joint Collaborative Team on Video Coding (JCT-VC) and the first draft was completed in early 2013. The HEVC standard has been developed in order to deal with the increasing diversity of services and the emergence of 4K and 8K video resolutions. HEVC greatly improves the coding efficiency over its predecessor standard (H.264/AVC) [1] by a factor of almost twice while maintaining an equivalent visual quality [2].

In terms of complexity, Bossen et al. [3] studied the complexity of HEVC encoding and decoding software showing that the encoding process is much more challenging than the decoding process. Also in [20], Jens et al. performed a comparison of the recent video standards including the HEVC.

Despite being a recent standard, we can find in the literature several works about complexity analysis and parallelization strategies for the HEVC standard [3][4][5]. Most of the parallelization proposals are focused in the decoding side, looking for the most appropriate parallel optimizations at the decoder which provide real-time decoding of High-Definition (HD) and Ultra-High-Definition (UHD) video contents. In [6] and [7] 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.

Like most other video codecs, HEVC uses both intra and inter-frame prediction and of course, the optimal performance is achieved when both tools are combined. Nonetheless, there are a number of applications in which intra coding configuration is required. These include ultra low delay applications like video game streaming services, video editing, digital cinema, etc. Even more, by encoding video in intra mode, there is an implicit higher recovery capability from data loss in wireless networks, since every individual frame is self-contained, avoiding error propagation.

HEVC inherits the hybrid video coding framework from its H.264/AVC. However, HEVC includes new coding tools like CTU quad-tree structure, SAO [8] or Tiles [9]. Furthermore, a rate distortion optimization (RDO) method is adopted to achieve best performance, checking all the candidates of quadtree partitions and prediction modes in a brute force way. As a consequence, the intra coding complexity drastically increases[2][10]. In [10] authors perform a deep comparison between HEVC and H264 in the scope of Intra coding. The experiments reported bit rate savings of 22% on average for all tested video sequences but requiring, on average, 3 times longer than H264 to encode a video sequence in Intra mode.

Several attempts have been made in order to speed-up the intra coding process[11][12][13]. In [11] authors propose a parallelization inside the intra prediction module that consist on removing data dependencies among sub-blocks of a CU, obtaining interesting speed-up results. Recently, in [12], authors present an hybrid approach combining WPP and intra prediction on Graphic Processors Units (GPUs) obtaining reductions on the total encoding time of up to 62% with a reduced coding performance loss. In [13] authors propose a two-stage parallelization scheme over GPUs using an open source realtime implementation of the HEVC encoder called X265. In the first stage several threads are launched to perform intra prediction, with each thread processing one CTU row taking into account the CTU dependencies. In a second stage, one additional thread is used to encode all CTUs within the whole picture in raster scan order. In [14] an hybrid CPU+GPU intra prediction scheme is presented. Firstly, a fast intra prediction scheme at the GPU is proposed, where for one slice the best intra prediction mode with minimum Sum of Absolute Differences (SAD) cost for every possible blocks of all CTUs is determined concurrently at the GPU, and then transmitted back to the host CPU. For each intra block, instead of the Rough Mode Decision (RMD) process, the intra modes returned by GPU along with the Most Probable Modes (MPMs) are directly used for RDO decision. Secondly, a fast CU splitting and pruning algorithm is proposed based on parallel texture gradient measuring using classic Sobel operator. In a similar way, authors in [15] presented an heterogeneous CPU+GPU system. The main contributions of this work are based on Prediction using Original Samples (POS) for RMD process, which is entirely executed on the GPU, as well as the analysis of the impact of POS to the RD results. Moreover, a Parallel RDO (PRDO) scheme that entirely runs on the GPU has been adapted to GPUs and does not depend on Context-Adaptive Binary Arithmetic Coding (CABAC).In [16] the authors use the combination of two types of hardware architectures (GPUs and CPUs). GPUs are used to parallelize the Motion Estimation algorithm used for inter-picture prediction, while CPUs are used to add a higher level of parallelism (by using WPP or a GOP-based algorithm).

Most of the previous works modify the behavior of the HEVC encoder in order to avoid the inherent dependencies at CTU and slice levels at the cost of a certain Rate Distortion (R/D) performance loss. In this paper, we will focus on applying parallel processing in GPUs to the intra-picture prediction process of the HEVC encoder. Our proposal does not perform changes in the dependencies of the CTUs, providing the same R/D performance than the original HEVC encoder.

The remainder of this paper is organized as follows, in Section 2 an overview of intra-picture prediction in HEVC is presented. Section 3 present the parallelization strategy proposed for using GPUs 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.

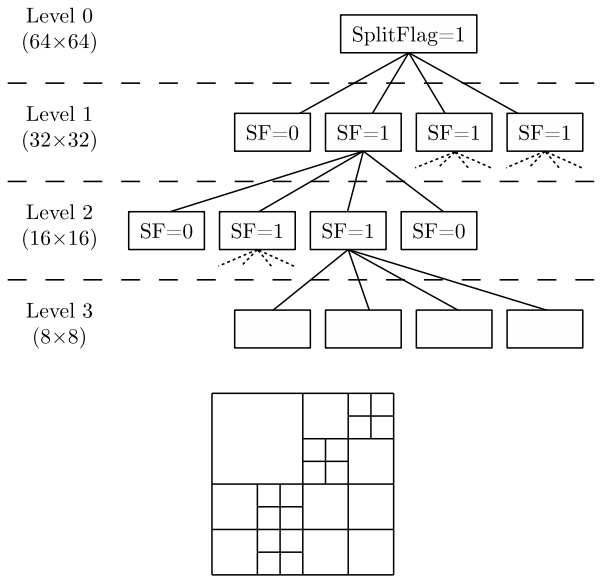



**Fig. 1** Example of CTU quadtree structure defined in HEVC

## 2 Intra-picture Prediction in HEVC

Intra coding in HEVC can be viewed as an extension of H.264/AVC, as both approaches are based on spatial sample prediction followed by transform coding [3]. But HEVC includes several new features to improve the coding efficiency.

One of the most important changes affects the picture partitioning, which is now performed following a quadtree structure as shown in Fig. 1. Each picture is partitioned into 64×64 pixel square regions called CTUs, which can be recursively divided into 4 smaller sub-regions simply called CUs, whose size range from 64×64 to 8×8. These structures can be partitioned, in turn, into the so-called PUs, whose size and shape have been defined in the standard, and TUs, which are also organized in a tree structure named residual quadtree (RQT) and have a size that ranges from 32×32 to 4×4.

For intra-picture prediction, only square PUs of the same size as the parent CU can be used (2N×2N), with the exception of CUs at the maximum depth level, which can be split into four square PUs (N×N). Therefore, the PU size can range from 64×64 to 4×4.

Besides, intra prediction modes are extended from 9 in H.264/AVC to 35 in HEVC, constituted by 33 directional modes (shown in Fig. 2), one DC mode and one planar mode. A special mode is also used for chroma components. Due to the finer angle divisions a more accurate prediction is obtained, resulting in a reduction of prediction residuals that will be fed into the transform coding stage. For each CTU, the encoder decides the best splitting and pre-

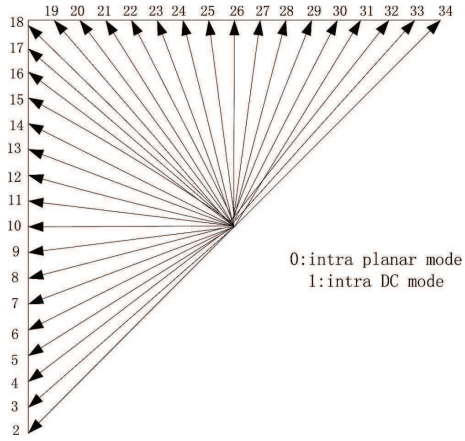



**Fig. 2** HEVC angular intra prediction modes

diction mode by a rate-distortion optimization process, which enumerates all the candidates of CU partition and their prediction intra modes.

The prediction modes are organized into three categories:

- Planar prediction: the value of each sample of the predicted CU is calculated assuming an amplitude surface with a horizontal and vertical slope derived from the boundary samples of the neighboring blocks (mode 0) .
- DC prediction: the value of each sample of the predicted CU is an average of the boundary samples of the neighboring 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 neighboring blocks as shown in Figure 2 (modes $2 \ldots 34$).

The fast encoding algorithm of HEVC reference software includes two steps. In the first one, 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, $\lambda$ is the Lagrange multiplier that determines the trade-off between rate and distortion, and $R_{mode}$ represents the estimated 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 the $SC$ set, 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 the $SC$ set.

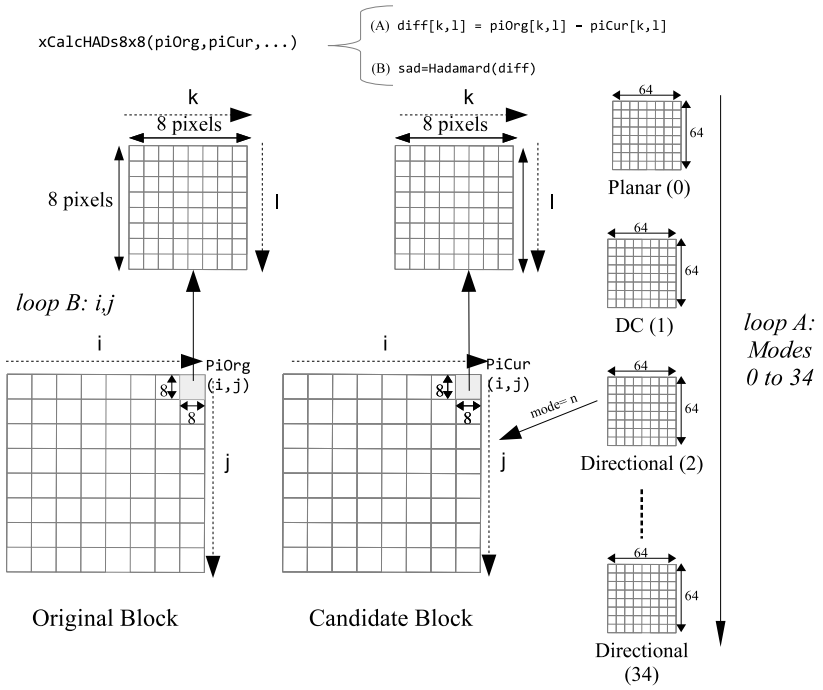



**Fig. 3** Algorithm for calculating $D_{Had}$ for each mode

## 3 Moving intra prediction cost function to GPUs

The release of NVIDIA CUDA API [17–19] provided to the developers a way to easily use 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) or Single Instruction Multiple Thread (SIMT) GPU architecture. Independent operating threads that execute CUDA kernels may efficiently share high speed memory with a set of threads organized into blocks. It should be noted that to obtain higher bandwidth and overall performance gains, memory sharing between threads must be optimized to ensure very low latency in read/write operations.

At the moment, we have started with the algorithm used in the HEVC standard for the intra-picture prediction which is based in a Single Instruction Single Data (SISD) architecture. The SAD computing is implemented to work with blocks of size up to 8 x 8. So, for a CU of 64 x 64 pixels, we need to perform 8 x 8 block transforms of the prediction residuals resulting from a particular intra-prediction mode. Remark that a CTU of 64 x 64 which corresponds to the maximum CU size, is recursively divided into 4 smaller sub-regions called CUs forming a quadtree structure (see Fig. 1). In this section, we explain how Hadamard transform is implemented in the HM 16.3 reference software, and

how we have programmed the heterogeneous platform CPU+GPU in order to accelerate the prediction algorithm.

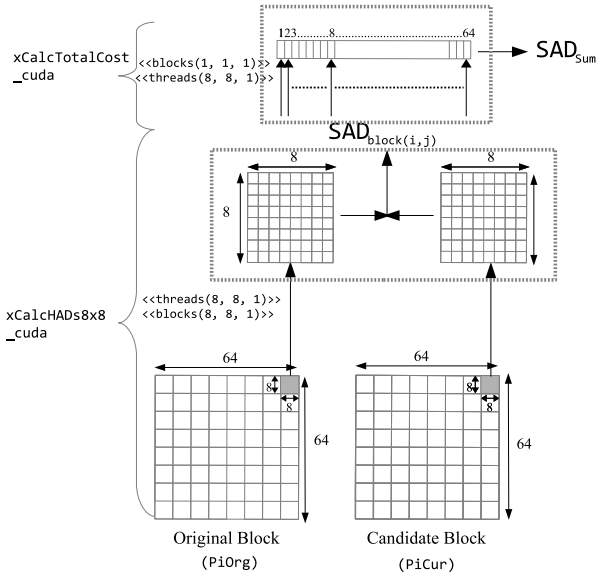
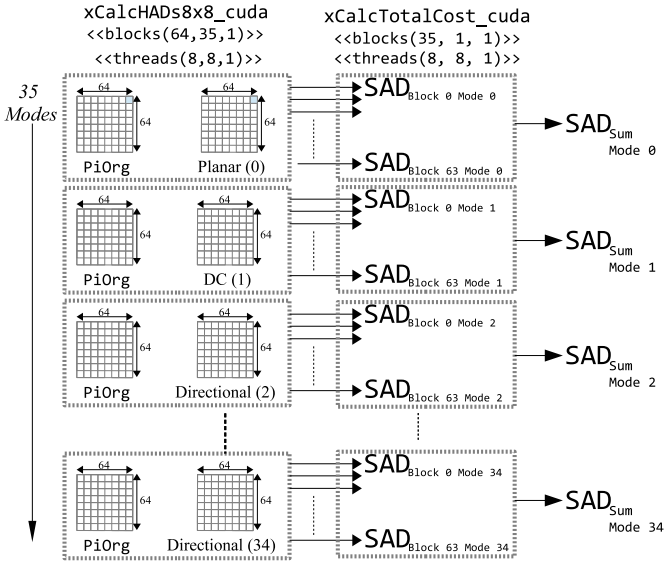
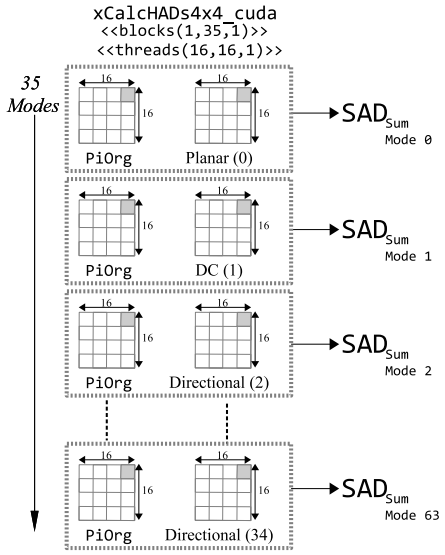



**Fig. 4** Parallel SAD computing for each 8×8 subblock (GPU-SAD)

In Fig. 3, a representation of the reference SAD algorithm for a CU size of 64 x 64 is presented. As can be seen, we have the original block on the left and the best candidate block must be found among all spatial prediction modes as shown in Fig. 2 (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 SAD of the corresponding 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 prediction and original pixels is calculated, and (2) the Hadamard transform from the resulting difference matrix is obtained. Inside this function, a new iteration (loop C) is done to calculate the difference of each matrix element $(k, l)$ and its Hadamard transform. As we can note, there is no data dependency between adjacent subblocks of size 8 x 8 that belongs to a CU and, therefore, all SAD values for each CU may be calculated at the same time. But, inside each subblock there are dependencies on the Hadamard transform computing, and there are also dependencies in the necessary reduction processes to compute the subblock differences (xCalcTotalCost_cuda in Fig. 4).




(a) Parallel cost computing for all intra prediction modes (GPU-MMode) in 8 ×8 subregions




(b) Parallel cost computing for all intra prediction modes (GPU-MMode) in 4×4 subregions

**Fig. 5** Parallel intra-picture prediction computing using GPUs

### 3.1 Moving intra prediction SAD function to GPU

In this work we propose the concurrent calculation of SAD values (including the Hadamard transform) for all subblocks belonging to a CU. The parallel

algorithm which calculates the SAD in the GPU (we have called it GPU-SAD) is shown in Fig. 4. The sum of SAD values is obtained in two steps. In a first step, we define a kernel grid of at most 8×8 blocks of 8×8 threads. Remark that the number of blocks will depend on the CU size determined by the depth level in the quadtree structure. 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 between the pixels (i,j) of the original and candidate subblocks, and (b) its corresponding Hadamard transformed value. At the end of this step, we get a vector of at most 64 SAD values corresponding to each subblock. As each SAD value has been computed by a block of threads, to add these values, a new kernel in the GPU is called and it performs an optimized reduction process based on the use of shared memory. The result is the $SAD_{Sum}$ associated with the prediction mode $n$. We must note that before the first computation step, both CUs (original and candidate) must be transferred from host memory to the GPU global memory using asynchronous transfers. As stated before, the amount of memory transfered will depend on the CU size determined by the quadtree structure, being the maximum memory size transfered 8192 pixels corresponding to both original and candidate CUs of 64 x 64 pixels. 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 the uncached global memory latency.

### 3.2 Computing 35 intra prediction modes simultaneously

As explained in the previous section, in order to calculate the SAD of a given prediction mode, kernels with grid of at most 64 blocks by 64 threads (for a CU size of 64x64) are launched in the GPU. But this grid size which varies as the CU size does, entails a low computing load of the GPU. In order to increase the performance of the GPU it is necessary to increase the GPU computational load. We can increase the GPU workload by computing all the prediction modes for a given CU (from 0 to 34) at the same time in the GPU. In Fig. 5(a), we show the algorithm used to reach this target which we have called GPU-MMode. In this case, the original and all the prediction candidates CUs must also be transferred to GPU memory, but in this case, before calling the kernel. So, the SAD for each prediction mode is also computed in two steps. As in GPU-SAD, we define a maximum of 8×8 blocks of 8×8 threads. Nevertheless, in GPU-MMode we define 35 blocks of GPU-SAD units. This will lead to a maximum of 8×8×35 blocks of 8×8 threads, depending on the CU size. As shown in Fig. 5(a), we call a GPU kernel with a mesh arrangement of at most `(64,35)` thread blocks. As in GPU-SAD, we need a second step to add all SADs resulting of the at most 64 subblocks 8×8 in a CU, but in this case the 35 SAD values are obtained at the same time.

### 3.3 Computing SAD for 4× 4 subregions

As shown in Fig. 1, each CTU can be recursively divided into 4 smaller subregions simply called CUs, whose size range from 64×64 to 8×8. When the CU size is 16×16 or 8×8, the HEVC reference software calculate the SAD values by calling the function `xCalcHAD4x4` instead `xCalcHAD8x8`. This new function computes the SAD values for a region of 4×4 pixels. So, we must redefine the GPU execution kernel to 1×35 blocks for each prediction mode of n×n threads, being $n$ 16 or 8 (see Fig. 5(b)). As explained in section 3.1, we require two steps to compute and reduce the sum of SAD values. However, due to the lower size of CUs, we can perform the computation of the sum of the at most sixteen partial SADs in the same kernel using the shared memory.

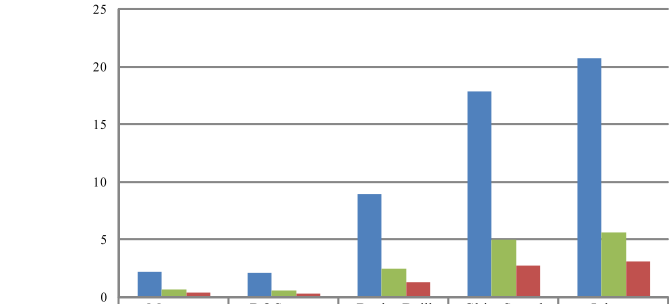
## 4 Performance Evaluation

In this section we present the performance evaluation of our GPU-based SAD computing algorithms: GPU-SAD and GPU-MMode. 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 laptop with an Intel i7-2670QM at 2.2GHz and 8GB of RAM memory.

### 4.1 Computational times comparison CPU vs GPU

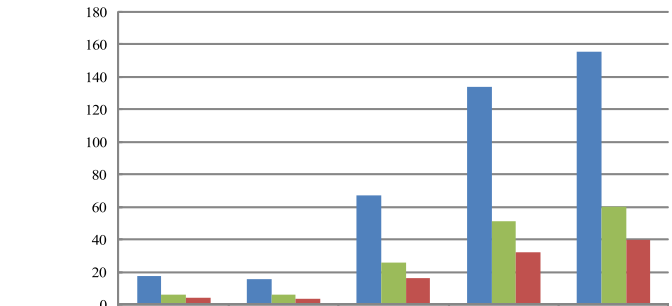
In Fig. 6, we present the computational times required for computing SADs in both GPU platforms (Tesla M2050 and Gforce GT540M, respectively), for five different video sequences: Monaco (352×288), BQSquare (416×240), BasketballDrill (832×480), ChinaSpeed (1024×768) and Johnny (1280×720). In all video sequences, we have encoded 50 frames in All Intra mode. In Fig. 6, *CPU* stands for the sequential computational time required to compute the SAD values, and *GPU-SAD* and *GPU-MMode* indicates the computational time of both proposed GPU-based algorithms. As we can see, we obtain significant time reductions with respect to the sequential CPU algorithm, being this reduction above 70% for GPU-SAD and 85% for GPU-MMode in the M2050 GPU. We also obtain considerable time reductions in the GT540M GPU platform, being above 60% in the GPU-SAD algorithm and above 70% in GPU-MMode. Note that the achieved time reduction does not depend on the video resolution. Obviously, due to the fact that the M2050 is more powerful than the GT540M, we obtain better computational results using the first one. In Table 1, the speedups for *GPU-MMode* versus *CPU* are shown for both platforms. As can be seen, we obtain speed-ups of up to 6.5x in the Tesla

M2050 platform, whereas we get a maximum speed-up of 3.9x in the GForce GT540M platform. On the other hand, note that both the Hadamard transform and the SAD computations 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 and properly mapping the kernel grid. As we expected, computing times obtained by GPU-MMode algorithm are better than the ones obtained by the GPU-SAD algorithm. The improvement achieved by the GPU-MMode algorithm is due to both a reduction in the kernel calls (remaining unchanged the workload assigned to the GPU), and a higher percentage of occupancy of the GPU which allows a more efficient GPU execution.




| | Monaco | BQSquare | BasketDrill | ChinaSpeed | Johnny |
|---|---|---|---|---|---|
| CPU | 2.20 | 2.11 | 8.92 | 17.86 | 20.78 |
| GPU-SAD | 0.64 | 0.60 | 2.41 | 4.96 | 5.62 |
| GPU-MMode | 0.36 | 0.33 | 1.32 | 2.75 | 3.06 |

(a) Nvidia Tesla M2050




| | Monaco | BQSquare | BasketDrill | ChinaSpeed | Johnny |
|---|---|---|---|---|---|
| CPU | 17.33 | 15.79 | 66.88 | 133.90 | 155.76 |
| GPU-SAD | 6.24 | 6.07 | 25.72 | 51.50 | 59.91 |
| GPU-MMode | 4.03 | 3.76 | 16.31 | 31.88 | 39.94 |

(b) Nvidia GForce GT540M

**Fig. 6** Computational times for GPU-based Hadamard transform for intra-picture prediction in HEVC

**Table 1** Speed-ups of the proposed GPU-based intraencoder versus reference CPU intraencoder

(a) Nvidia Tesla M2050

| Sequence | CPU | GPU | Speed-up |
|---|---|---|---|
| Monaco | 2.20 | 0.38 | 5.81 |
| BQSquare | 2.11 | 0.36 | 5.78 |
| BasketDrill | 8.92 | 1.55 | 5.77 |
| ChinaSpeed | 17.86 | 2.75 | 6.50 |
| Johnny | 20.78 | 3.39 | 6.12 |

(b) Nvidia GForce GT540M

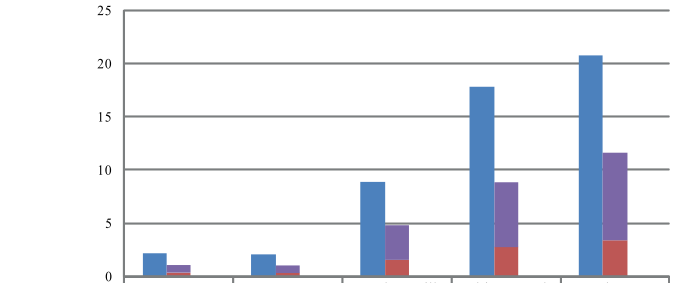
| Sequence | CPU | GPU | Speed-up |
|---|---|---|---|
| Monaco | 17.33 | 4.23 | 4.10 |
| BQSquare | 15.79 | 4.05 | 3.90 |
| BasketDrill | 66.88 | 17.60 | 3.80 |
| ChinaSpeed | 133.90 | 34.69 | 3.86 |
| Johnny | 155.76 | 42.44 | 3.67 |

### 4.2 Communication CPU-GPU

One important issue not analyzed in Section 4.1 is the communication time required to transfer both original and candidates CUs from the global memory on host to the global memory on GPU, and also to transfer the cost array for the 35 prediction modes, from GPUmemory to the global memory on host. These transfers are performed for the original CU which size varies from 64×64 to 8×8 and for all the 35 candidate prediction modes of the same size. These communications are massively performed previous to and post to the computation of the SADs for each CU. Remark that the way that HEVC standard determines the final CTU partitioning is obtained spliting the CTU in a recursive way. For each CU, the encoder first decides the best mode by performing intra prediction with 35 modes, storing the best R/D cost; then the CU is split into four sub-CUs and repeats the process until reaching the maximum allowed depth. This calculation is processed from top to bottom following the quad-tree structure. When the best modes and costs values for CUs at each level have been acquired, the encoder decides the best partition by checking whether the cost of a parent CU is smaller than the cost sum of its children's CUs. The process is from bottom to top. These dependencies inside a CU do not allows to group the communicactions in order to perform transfers with more volume of data. In Fig. 7 we show the effect of the communications required over the GPU-MMode algorithm. As can be seen, the memory transfers introduce a penalty in the computational time reduction, which is now around 60%.

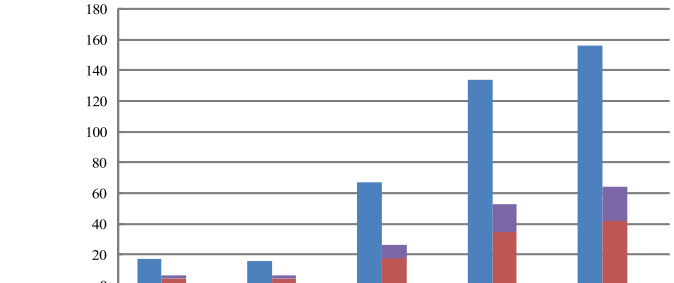
## 5 Conclusions and Future Work

In this paper we have proposed the use of GPUs for computing the distortion values needed in the intra-picture prediction of the HEVC standard. The search of the best prediction block implies a great search of candidates among 35 modes for each CU. Besides, this intra prediction algorithm is massively used in all CTUs that belong to a frame. In this paper we have detailed how the intra prediction algorithm is implemented in the 16.3 HM reference software and how concurrency can be highly exploited with two proposed algorithms specially suited for GPUs. The proposed GPU-SAD algorithm introduces the




| | Monaco | BQSquare | BasketDrill | ChinaSpeed | Johnny |
|---|---|---|---|---|---|
| CPU | 2.20 | 2.11 | 8.92 | 17.86 | 20.78 |
| GPU-MMode | 0.38 | 0.36 | 1.55 | 2.75 | 3.39 |
| GPU-Comm | 0.69 | 0.71 | 3.30 | 6.14 | 8.26 |

(a) Nvidia Tesla M2050




| | Monaco | BQSquare | BasketDrill | ChinaSpeed | Johnny |
|---|---|---|---|---|---|
| CPU | 17.33 | 15.79 | 66.88 | 133.90 | 155.76 |
| GPU-MMode | 4.23 | 4.05 | 17.60 | 34.69 | 42.44 |
| GPU-Comm | 2.26 | 2.16 | 9.40 | 18.52 | 22.66 |

(b) Nvidia GForce GT540M

**Fig. 7** Comparison between CPU vs GPU including transfer times

parallel computing in the sum of CU's SAD, splitting it in blocks and threads. In an upper parallelism level, the proposed GPU-MMode algorithm allows parallel cost function computing for the all intra prediction modes in a single kernel call. From the two proposed algorithms, the GPU-MMode one obtains the best results, achieving computing time reductions of up to 85%. However, both algorithms are highly affected by the memory transfers, because of the recursive way the HEVC standard determines the CTU partitioning. As future work, we will try to avoid CU dependencies inside the quadtree structure and also to avoid dependencies between neighboring CTUs. All these approaches will allow to increase the GPU computatinal load at the expense of a R/D penalty.

## References


1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. C. M. Fu, E. Alshina, A. Alshin, Y. W. Huang, C. Y. Chen, C. Y. Tsai, C. W. Hsu, S. M. Lei, J. H. Park, and W. J. Han, "Sample adaptive offset in the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1755–1764, Dec 2012.
9. Fuldseth, M. Horowitz, and M. Z. S. Xu and, "Tiles," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-E408, Tech. Rep., March 2011.
10. J. Lainema, F. Bossen, W. J. Han, J. Min, and K. Ugur, "Intra coding of the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1792–1801, Dec 2012.
11. 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.
12. 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.
13. Y. Zhao, L. Song, X. Wang, M. Chen, and J. Wang, "Efficient realization of parallel hevc intra encoding," in *Multimedia and Expo Workshops (ICMEW), 2013 IEEE International Conference on*, July 2013, pp. 1–6.
14. J. Ma, F. Luo, S. Wang, N. Zhang, and S. Ma, "Parallel intra coding for hevc on cpu plus gpu platform," in *2015 Visual Communications and Image Processing (VCIP)*, Dec 2015, pp. 1–4.
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. 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.
17. J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *Queue*, vol. 6, no. 2, 2008, pp. 40–53.
18. N. Corporation, "Nvidia cuda c programming guide. version 3.2."
19. 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.
20. Jens Rainer Ohm, Gary J. Sullivan, Heiko Schwarz, Thiow Keng Tan and Thomas Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards-Including High Efficiency Video Coding (HEVC)," *Circuits and Systems for Video Technology, IEEE Transactions on*, Vol. 22, no. 12, 2012, pp. 1649–1668