GPU Acceleration of MIP Intra Prediction in VVC

Iago Storch[12], Nuno Roma[23], Daniel Palomino[4], Sergio Bampi[1]

[1]*Graduate Program in Computing, Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre - Brazil*

[2]*Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento (INESC-ID), Lisboa - Portugal*

[3]*Instituto Superior Técnico, Universidade de Lisboa, Lisboa - Portugal*

[4]*Federal University of Pelotas, Graduate Program in Computing, Video Technology Research Group, Pelotas - Brazil*

{icstorch, bampi}@inf.ufrgs.br, nuno.roma@inesc-id.pt, dpalomino@inf.ufpel.edu.br

***Abstract*—The Versatile Video Coding (VVC) standard introduced several novel encoding tools for intra frame prediction, increasing the encoder complexity when compared to previous standards. Among these novelties is the MIP tool, whose acceleration has not been tackled in the literature. Therefore, an efficient parallelization of MIP prediction targeting GPU platforms is now proposed. The presented technique makes use of alternative reference samples and computes the distortion in an approximate manner to expose and potentiate massive parallelism. Moreover, the adopted prediction scheduling and memory communication were tailored by considering the GPUs' architecture and memory hierarchy. When compared with a CPU execution, this work is capable to accelerate the MIP prediction up to 105 times at the cost of a negligible coding efficiency loss of 0.284% BD-BR.**

***Index Terms*—parallel video coding, intra prediction, OpenCL**

I. Introduction

To comply with the growing demand for coding efficiency, new video coding standards have introduced new tools to deal with several encoding aspects. A remarkable example of this trend is the Versatile Video Coding (VVC) [1], which introduced major novelties in the intra prediction, such as the Matrix-based Intra Prediction (MIP) and Intra Sub-partitions (ISP) [1]. However, video coding is known for being burdensome and more tools increases that problem. Thus, several authors have proposed distinct methods to accelerate intra prediction in VVC. The authors of [2], [3] propose sophisticated techniques to evaluate fewer angular modes; [4] uses machine-learning to discard unlikely modes (angular and ISP); while [5] uses machine-learning to accelerate ISP. However, algorithmic solutions like these do not have the same acceleration potential as the ones provided by parallel-processing.

Modern computing systems usually provide multiple processing cores alongside hardware accelerators such as GPUs. Although GPUs were originally designed to accelerate graphics, their massively-parallel architecture and the availability of convenient APIs (such as OpenCL and CUDA) led GPUs to be used in several applications – including video coding.

Galiano *et al* [6] accelerate the HEVC intra prediction by computing the distortion of each mode in GPU. However, they do not exploit the full potential of GPUs since part of the prediction is still conducted in the CPU and there is a large communication overhead. In [7], the authors propose a GPU+CPU scheme to implement the whole intra prediction of HEVC, including distortion and bitrate estimates. These authors use the original samples as references to simultaneously conduct the prediction of multiple blocks in the GPU.

In summary, current algorithmic solutions [2]–[5] to accelerate intra prediction can hardly provide the massive acceleration provided by GPUs, although the existing proposals using GPUs [6], [7] only consider the angular prediction. However, VVC introduced other intra prediction tools, including MIP. Although its execution pattern is particularly suitable for the massively-parallel architecture of GPUs, its acceleration (using GPUs) has not been covered in the literature. Thus, this work is the first to tackle the acceleration of MIP prediction by a specifically designed modeling aiming at GPU execution.

II. Matrix-based Intra Prediction (MIP) in VVC

A. VVC Partitioning and Intra Mode Decision

In VVC, a frame is divided into a regular grid composed of Coding Tree Units (CTUs) with at most 128×128 samples. Each CTU is recursively partitioned into Coding Units (CUs), by following a sequence of splits using binary, ternary, and quaternary trees [1]. Due to the novel binary and ternary splits, the CUs can be either square-shaped or rectangular. Moreover, CUs of the same size are not necessarily aligned with each other as was the case of previous standards; this is depicted in Fig. 1, which highlights 16×16 CUs produced using different split sequences. Each square with a blue or green circle represents one 16×16 CU, and the symbols "BH,BV,TH,TV,QT" represent a sequence of splits used to produce the CU: binary horizontal/vertical, ternary horizontal/vertical, or quaternary. The CUs highlighted in blue are produced using only quaternary splits (as in previous standards); all of such CUs are aligned with each other forming a grid. However, the CUs highlighted in green are produced combining binary and ternary splits, and they may not be aligned with each other and with those highlighted in blue. In summary, the possible positions for the same CU size can overlap, but it is not possible to simultaneously use overlapping positions.

Besides defining the partition sizes, it is necessary to define the intra prediction mode with the best rate-distortion (RD) tradeoff for each CU. Since conducting an exhaustive evaluation is too time-consuming, the VTM encoder uses a

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPERGS, CNPq, and Fundação para a Ciência e a Tecnologia (FCT) under project UIDB/50021/2020.

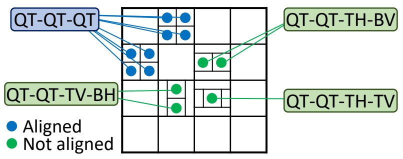

Fig. 1. Example of 16×16 CUs obtained through different split sequences.

well-known decision method based on Rough Mode Decision (RMD), Most Probable Modes (MPM), and Rate-Distortion Optimization (RDO) [8]. This scheme is presented in the upper part of Fig. 3. The RMD stage conducts a fast evaluation of several modes and creates a list of the most promising ones, called rd-list. Here, VTM conducts the prediction, computes the distortion of the predicted block, and estimates the bitrate using a simplified context. The distortion is estimated according to the Sum of Absolute Differences (SAD) and the Sum of Absolute Transformed Differences (SATD). During RMD the encoder tests most of the angular prediction modes, the most probable modes (MPM, discussed later) with multiple reference lines (MRL tool), and all MIP modes for the current CU (which can be as much as 32). Each mode is checked against the modes on the rd-list such that, after concluding RMD, the rd-list has a subset of the modes with the best rd-costs to be evaluated by the RDO [8]. Then, the MPMs without MRL are always added to the rd-list. MPM is a list of 6 very likely modes based on the adjacent blocks.

Then the encoder enters RDO procedure and conducts a thorough evaluation of the modes in the rd-list. This evaluation involves computing the distortion of the reconstructed block and computing the bitrate using the appropriate contexts and entropy encoder. After RDO, the mode with the lowest rd-cost is selected to encode the current block [8].

B. Matrix-based Intra Prediction (MIP)

MIP is a new prediction concept introduced in VVC based on data-driven models. Although the original design of MIP was based on feeding a neural network with reference samples, it was simplified into a matrix-vector multiplication due to complexity issues [1]. An overview of the MIP procedure is presented in Fig. 2, where W and H represent the CU width and height, respectively. The MIP mode is applied to CUs with dimensions equal to 64×64, 32×32, 32×16, 16×32, 32×8, 8×32, 16×16, 16×8, 8×16, 32×4, 4×32, 16×4, 4×16, 8×8, 8×4, 4×8, and 4×4. These CUs are grouped into three sets according to SizeId={0, 1, 2}. SizeId=0 comprehends 4×4 CUs, SizeId=1 comprehends 8×8 CUs and CUs with exactly one side of length equal to 4; whereas SizeId=2 comprehends the remaining sizes. The number of MIP modes for SizeId equal to 0, 1, and 2 is equal to 16, 8, and 6, respectively, and every mode has a transposed counterpart – which in practice doubles the number of modes. Each MIP mode is represented by a matrix of predefined coefficients that were obtained by neural networks aiming to minimize the prediction error.

When a CU goes through MIP, the W reconstructed samples directly above the CU (**refT**) and the H reconstructed samples directly left of the CU (**refL**) are fetched. These samples are called complete boundaries. Then, these complete boundaries are averaged to create reduced top (**redT**) and left (**redL**) boundaries. Each reduced boundary has either 2 samples (for SizeId=0) or 4 samples (for SizeId={1,2}). In sequence, the reduced top and left boundaries are concatenated to obtain **redBound**. For non-transposed MIP modes redT is taken first, whereas for transposed modes redL is taken first.

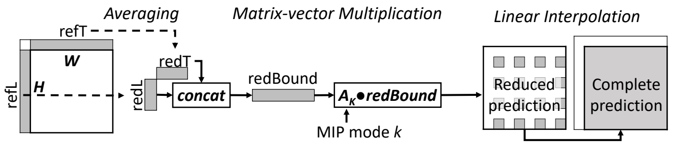

Fig. 2. Overview of MIP prediction.

In the next stage, a reduced prediction is obtained through a matrix-vector multiplication. The inputs are redBound (a vector) and the current MIP mode (matrix A_K). The reduced prediction has dimensions 4×4 for SizeId={0,1} and 8×8 for SizeId=2. For SizeId=0, the obtained reduced prediction has the same dimension as the original CU; for SizeId={1,2}, the reduced prediction must be upsampled to match the CU size.

The final upsampling is conducted using linear interpolation, first in the horizontal and then in the vertical direction. refL and refT are used as the leftmost and uppermost references, respectively. A thorough discussion of MIP is available in [1].

III. Proposed Parallelization and Implementation

The RMD stage of intra prediction requires testing all MIP modes for each CU. It involves the computation of up to 32 MIP predictions for each CU. Furthermore, the rd-list cannot be completed and sent to RDO until all MIP modes (and the other tools inside RMD) are evaluated. However, the MIP prediction stage presents a great parallelization opportunity that can be explored to accelerate the mode decision and the generation of the rd-list. In this context, this work focuses on exploiting a GPU to accelerate the evaluation of MIP modes during RMD, aiming a faster processing of the rd-list.

An overview of the proposed GPU parallelization model, alongside the main intra prediction stages, is depicted in Fig. 3, where the numbers in red circles represent the order in which the data is exchanged between the host and the GPU, and inside the GPU – these will be referred to as "**data #**", where # is a number. The main idea is that prior to RMD the reference samples are fed to the GPU (**data 1**), and the GPU produces the distortion for all MIP modes of all CUs. When the encoder has to check a MIP mode during RMD, it can simply fetch the corresponding distortion (**data 8**) and determine if it should be in the rd-list. The intermediate exchanges (**data 2-7**) occur inside the GPU during the prediction process.

Three GPU kernels are proposed to conduct the major MIP steps. The *produceBoundaries* kernel produces the complete and reduced boundaries; the *reducedPred* kernel obtains the reduced prediction for all modes; and the *upsampleDist* kernel upsamples the reduced prediction and computes the distortion of each mode. The GPU main memory serves as an interface between the host and GPU, and between the GPU kernels.

Hence, the main contributions of this work comprehend the following aspects related to the GPU parallelization of MIP prediction using OpenCL API, which will be further detailed in the next subsections.

- Definition of alternative reference samples for MIP prediction to expose massive parallelism (Section III-A);
- Usage of specific SIMD instructions to efficiently conduct the prediction (Section III-C);
- Parallelization model at the 4×4 Hadamard-based SATD computation (Section III-D);
- A comprehensive set of memory optimizations to alleviate the communication overhead and to better explore the memory hierarchy.

A. *Top-level Scheduling and Parallelism Exposition*

Although GPUs are designed to execute massively-parallel tasks, the dependencies between adjacent blocks during intra prediction turn it inherently sequential. To break these dependencies, this work employs the same technique also used by Radicke [7] for angular prediction by adopting the original samples of the frame (that are always available) as reference samples for MIP prediction during RMD. This way, any CU can be predicted without depending on other CUs.

The proposed parallelization model conducts the MIP prediction of all CTUs and CU sizes concurrently, at the same time. In particular, each OpenCL workgroup is scheduled to conduct the prediction of a whole CTU, composed of multiple CUs with a specific dimension. However, the novel binary and ternary splits introduced by VVC [1] allow some CUs to be in positions that are not aligned with the quadtree structure, as discussed in Fig. 1. For CU sizes with this property, multiple workgroups are assigned to process the same CU size, where each workgroup deals with CUs with a different alignment. This improves the processing regularity and promotes a better memory communication. Finally, the proposed modeling considers that CUs are always indexed in raster order to improve data locality in memory accesses.

B. *Boundaries Computation*

This kernel fetches the samples of the current CU from global memory (**data 2**), produces the complete and reduced boundaries, and stores them in global memory again (**data 3**).

The *produceBoundaries* kernel processes CUs in batches. First, it obtains the top and the left boundaries of the CUs in the current batch, then proceeds to the next batch, where each batch is composed of an integer number of CU rows/columns. When computing the top boundaries of CUs that are horizontally adjacent, a single row of CUs is processed in each batch. When there are horizontal gaps between the CUs, such as “QT-QT-TV-BH” and “QT-QT-TH-TV” (see Fig. 1), two rows are processed in each batch – only half of the samples in each row are relevant. The same strategy is employed for the left boundaries, where the number of considered columns depends on the vertical gaps between CUs.

Starting with the top boundaries, each workitem fetches exactly one sample to compose the complete boundary of one

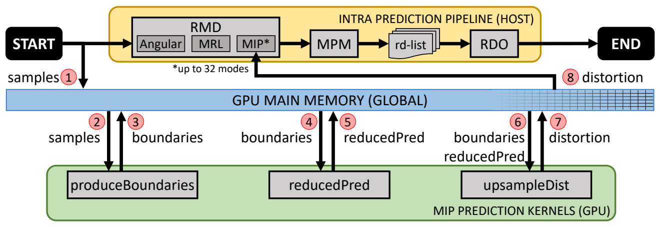

Fig. 3. Overview of the proposed modeling.

CU in the current batch. Workitems WI_i and WI_{i+1} fetch the complete boundary samples $refT_j$ and $refT_{j+1}$, such that the global memory reads are coalesced. These complete boundaries are stored in the local memory of the workgroup. In sequence, the reduced top boundaries are computed and also stored in local memory. Since the reduced boundaries have either 2 or 4 samples, a different schedule is employed. For 4×4 CUs, workitems WI_0 and WI_1 compute the reduced boundaries for CU_0, WI_2 and WI_3 compute the reduced boundaries for CU_1, and so on such that the work is focused on as few warps/wavefronts as possible (i.e., fewer multiprocessors). For the remaining CU sizes a similar scheduling is used. However, 4 workitems are assigned to each CU. In summary, only 2 or 4 workitems per CU are active in this stage. Once the reduced top boundaries for the current batch are computed, the top boundaries for the next batch can be computed.

After computing all top boundaries of the CTU, the data is offloaded to the global memory. The boundaries of sequential CUs are stored in sequential positions in memory to make the data layout friendlier in the next kernels. Considering that GPU memories are optimized to move chunks of data in each access, offloading all boundaries at once into a contiguous memory region is more efficient than offloading the boundaries individually, which would cause bursts of memory accesses.

The procedure that computes the left boundaries is similar, but each batch is composed of one or two columns of CUs. Due to the CTU dimensions, each row/column of samples has at most 128 elements; therefore, this kernel uses 128 workitems per workgroup in the proposed implementation.

C. *Reduced Prediction Computation*

The *reducedPred* kernel fetches the reduced boundaries (**data 4**) and produces the reduced prediction (**data 5**). For each prediction mode, the prediction signal of a whole batch of CUs is produced in parallel. Hence, once the current prediction mode is applied to all CUs of the current batch, the next batch can start. Once a mode is applied to all batches of CUs, the next mode is tested. The reduced prediction of CUs with SizeId={0,1} has 4×4 samples, while for CUs with SizeId=2 it has 8×8 samples (16 and 64 samples, respectively). Each predicted sample is produced by one workitem. Thus, 16 or 64 workitems work together to predict each CU. Sequential workitems process sequential samples in each CU.

For a given prediction mode (and batch of CUs), each workitem fetches the top and left reduced boundaries of one CU and concatenates them accordingly. The computation pattern of MIP prediction allows breaking the matrix-vector multiplication ($A_K \cdot redBound$ in Fig. 2) into several dot product operations, where each predicted sample is a dot product between the reduced boundaries (identical to all samples in the same CU) and a line of coefficients extracted from A_K (distinct for each sample). Therefore, all predicted samples of a given CU are obtained in parallel by a single dot product operation of each workitem [8]. Since GPUs have special SIMD instructions to conduct dot product, it is performed very efficiently. The reduced prediction of all CUs with a particular prediction mode is stored in local memory, and when all CUs are predicted, the prediction signal is offloaded to global memory. Then, the next prediction mode can be considered and the same process is repeated, reusing the local memory. The proposed implementation of the *reducedPred* kernel uses 256 workitems per workgroup, and either 4 or 16 CUs per CTU are concurrently processed, depending on SizeId.

D. Upsample and Distortion Computation

The *upsampleDist* kernel evaluates the reduced prediction (**data 6**) and computes the distortion (**data 7**), processing CUs in batches. First, the original samples and boundaries for CUs in the current batch are transferred to local memory. This data is used to compute the distortion of all modes for each CU.

For one mode, the reduced prediction of all CUs in the batch is fetched into local memory. During horizontal upsampling, sequential workitems compute the upsampled signal for samples in sequential positions, and the result is stored in local memory. The complete left boundaries are used as leftmost references. When the horizontally-upsampled prediction has more samples than the number of workitems assigned to such CU, multiple passes are required. When the current CU size does not require horizontal upsampling (i.e., CUs 4×16), this stage copies the reduced prediction into a temporary buffer. The vertical upsampling is similar, but the complete top boundary is used as uppermost reference to match the CU width. Whenever the block does not require vertical upsampling, the horizontally-upsampled signal is bypassed into the final buffer.

Then, the distortion is estimated using the SAD and SATD metrics. For SAD, sequential workitems compute the absolute difference for sequential samples inside the CUs, and each workitem stores the intermediary result in local memory. When a CU has more samples than workitems, each workitem computes the absolute difference of multiple strided samples and accumulate them. When all samples in each CU are processed, the SAD for each CU is obtained by parallel reduction, and the result is stored in local memory.

The SATD distortion adopted by VTM uses Hadamard transforms of variable sizes, such that larger blocks use larger transforms [8]. However, this computation presents large data dependencies since several samples are required at once. The proposed parallelization employs an approximate SATD based solely on the 4×4 Hadamard transform. The CU is divided into 4×4 patches. Then, similar to SAD, each workitem process one or more patches and the intermediary values are accumulated in local memory. The SATD for each CU is obtained by parallel reduction of the intermediary SATDs. Finally, when the SAD and SATD distortion of all CUs is computed, the distortion is offloaded to global memory.

The VTM encoder estimates the bitrate for MIP prediction during RMD stage based on two features: (i) the current split sequence and context, and (ii) the current MIP mode and SizeId. Since the proposed parallelization conducts MIP for all possible CUs (irrespective of the split sequence) and it does not have access to the host's previous decisions, this bitrate is hardly computed on the GPU side. However, since each pair of SizeId and MIP mode always yield the same bitrate, this bitrate can be easily estimated with an if-else structure on the host, thus avoiding the communication overhead that would be observed if this computation was performed at the GPU.

The proposed implementation of *upsampleDist* uses 256 workitems per workgroup. Moreover, CUs with SizeId equal to 0, 1, and 2 are processed by 16, 32, and 128 workitems – this equals the number of samples of the smaller CU size in each SizeId. Therefore, 16, 8, and 2 CUs are concurrently processed for CUs with SizeId equal to 0, 1, and 2.

IV. Experimental Validation

The proposed parallelization was implemented in a standalone module to validate its performance. For the sake of obtaining a compliant bitstream, the two proposed modifications over the original encoding (alternative references and distortion) were applied on a specially devised encoder, also based on VTM [8]. The encoding followed the Common Test Conditions [9] of VTM with the *all_intra* configuration. The first 64 frames are encoded and the subsample ratio is 1. The performance is assessed with speedup and coding efficiency.

The speedup compares the time to execute MIP during RMD of VTM (running in the CPU) against the time to execute MIP on GPU. The processing time in GPU encompasses the kernel execution and the data transfer between host and GPU. The coding efficiency was measured in terms of BD-BR [10], which represents the bitrate increase required by the proposed implementation to achieve the same quality as the reference. Larger BD-BR represents worse coding efficiency.

The experimental validation was conducted on the hardware platforms specified in Table I. Here, *"Cores"* refers to the number of processing units on the devices: physical cores for Intel CPUs, CUDA cores for NVIDIA GPUs, and Streaming Processors for AMD GPUs. All VTM encodings are conducted with SIMD optimizations enabled in **CPU I**, running CentOS 7 with the AVX2 instruction set. The proposed parallelization implementation was executed on three GPUs: **GPU I** runs Ubuntu 16.04 while **GPU II** and **III** run CentOS 7.

The experimental results are presented in Table II, where "GPU I, GPU II, GPU III" represent the speedup achieved by each GPU. The *"Base."* column presents the BD-BR of the proposed modeling when compared to the unmodified reference encoder. An ablation study was also conducted to assess the coding efficiency when disabling MIP completely; the BD-BR of this encoder is presented in the *"Ablat."* column.

The results from Table II show that the proposed parallelization model and implementation achieve a significant speedup

TABLE I
HARDWARE USED FOR EXPERIMENTAL EVALUATION.

System	Device (RAM)	Boost freq.	Cores
CPU I	Intel Core i9-7900X (64 GB)	4.3 GHz	10
GPU I	NVIDIA GTX 1080 (8 GB)	1.733 GHz	2560
GPU II	NVIDIA Titan V (12 GB)	1.455 GHz	5120
GPU III	Radeon RX 6900 XT (16 GB)	2.250 GHz	5120

when compared to the encoding on the CPU. When considering 1080p videos, the average speedup obtained by GPU I, GPU II and GPU III is 33.6, 79.6, and 75.3, respectively. For 4k videos, the corresponding average speedup is 25.7, 53.3, and 66.9. The observed variation between video sequences occurs because the reference encoder in the CPU applies early termination heuristics that make the encoder test a different number of CUs depending on the video content. In contrast, the proposed modeling always processes all blocks, leading to a regular processing time. In addition to that, GPU II and GPU III have significantly more computing power than GPU I. Therefore, they are able to present a higher speedup. This is only possible because the proposed modeling exposes more opportunities to exploit the parallelism by using alternative reference samples and processing multiple CU sizes and modes concurrently, alongside a memory communication designed for GPU architectures. Finally, it is observed that GPU III performs slightly worse than GPU II when processing 1080p videos, but GPU III outperforms GPU II when considering 4k videos. This occurs because GPU III has an extra cache level when compared to GPU II. Since the MIP prediction kernels require moving large amounts of data, this extra cache plays an important role when considering larger resolutions.

With respect to the coding efficiency, it is observed that the proposed parallelization model achieves an average coding degradation of 0.231% and 0.328% BD-BR for 1080p and 4k videos, respectively. Although the coding efficiency is identical irrespective of the platform, these penalties come from the two approximations introduced in the proposed model: the use of original samples as references and the estimation of the SATD based solely on the 4×4 Hadamard. These approximations slightly interfere with the distortion estimate and may lead the encoder to occasionally select MIP modes that may not be optimal. Finally, the presented ablation study shows that when the MIP mode is completely disabled, the average coding degradation for 1080p and 4k videos is 0.558% and 0.858% BD-BR, respectively. These results show that although the proposed modeling slightly penalizes the coding efficiency, the proposed GPU-accelerated MIP decision still remains advantageous over the alternative of not using MIP at all.

To the best of authors' knowledge, there are no other works in the literature accelerating the MIP prediction, either with CPU, GPU, or dedicated accelerators. Nonetheless, some works use GPUs to accelerate the angular intra prediction. Galeano *et al* [6] use GPUs to compute the distortion of angular modes in HEVC, accelerating this process by 6 times. Since they do not interfere with the encoder decisions, there is no coding efficiency variation. Radicke *et al* [7] use a CPU+GPU scheme to perform angular intra prediction in HEVC and compute the respective costs in RMD and RDO stages. This work accelerates the whole intra prediction between 1.9 and 4.44 times with an average BD-BR degradation between 2.797% and 3.981%, depending on the operating point. Although our work cannot be directly compared to [6] and [7], it is clear that it provides competitive results. Furthermore, our proposal can be combined with related works to accelerate the angular prediction as well.

TABLE II
RESULTS OF THE PROPOSED MODELING AND IMPLEMENTATION.

	Speedup			BD-BR[%]	
Video	**GPU I**	**GPU II**	**GPU III**	**Base.**	**Ablat.**
BasketballDrive	31.5	74.5	71.1	0.219	0.515
BQTerrace	41.6	99.0	94.4	0.105	0.375
Cactus	44.5	105.4	97.0	0.184	0.563
MarketPlace	34.4	81.4	77.6	0.258	0.697
RitualDance	16.1	38.0	36.4	0.387	0.641
Average 1080p	**33.6**	**79.6**	**75.3**	**0.231**	**0.558**
Campfire	29.5	60.3	77.4	0.297	0.733
CatRobot	26.4	54.3	69.4	0.314	0.658
DaylighRoad	37.4	77.2	98.5	0.141	0.192
FoodMarket	8.8	18.1	23.0	0.249	1.939
ParkRunning	30.9	63.9	79.5	0.229	0.561
Tango	21.2	45.9	53.9	0.736	1.064
Average 4k	**25.7**	**53.3**	**66.9**	**0.328**	**0.858**

V. CONCLUSION

This work proposed the first parallelization model and the respective implementation for the VVC MIP prediction aiming at GPU execution. This modeling exposes the opportunity to exploit the massive parallelism from the MIP tool by using alternative reference samples and concurrently processing several blocks and modes, which allows to extract the most throughput out of GPUs. The obtained experimental results showed that this proposal is capable of accelerating the MIP prediction in up to 105 times at the cost of a minor coding efficiency penalty of around 0.284% BD-BR.

REFERENCES

[1] B. Bross et al, “Overview of the versatile video coding (vvc) standard and its applications,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 3736–3764, 2021.
[2] A. Gou et al, ”Fast Intra Mode Decision for VVC Based on Histogram of Oriented Gradient,” 2022 IEEE International Symposium on Circuits and Systems (ISCAS), Austin, TX, USA, 2022, pp. 3028-3032.
[3] Y. Chen et al, ”A novel fast intra mode decision for versatile video coding,” in Journal of Visual Communication and Image Representation, vol. 71, no. 1, pp. 1-11, 2020.
[4] X. Dong et al, ”Fast Intra Mode Decision Algorithm for Versatile Video Coding,” in IEEE Trans. on Multimedia, vol. 24, pp. 400-414, 2022.
[5] J. Park et al, ”Machine Learning-Based Early Skip Decision for Intra Subpartition Prediction in VVC,” in IEEE Access, vol. 10, 2022.
[6] V. Galiano et al., ”GPU-based HEVC intra-prediction module,” in Journal of Supercomputing, vol. 73, pp. 455-468, 2017.
[7] S. Radicke et al, ”A Parallel HEVC Intra Prediction Algorithm for Heterogeneous CPU+GPU Platforms,” in IEEE Transactions on Broadcasting, vol. 62, no. 1, pp. 103-119, 2016.
[8] ”VVC Test Model (VTM),” available in https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM.
[9] F. Bossen et al., ”JVET-T2010: VTM common test conditions and software reference configurations for SDR video,” Document, Oct. 2020.
[10] G. Bjøntegaard, ”VCEG-M33: Calculation of average PSNR differences between RD-curves,” Document, Mar. 2001.