GPU BASED REAL-TIME UHD INTRA DECODING FOR AVS3

Xu Han[*], *Bo Jiang*[†], *Shanshe Wang*[†], *Lin Li*[‡], *Yi Su*[‡], *Siwei Ma*[†] *and Wen Gao*[†]

[*]School of Electronic Information and Electrical Engineering, Shanghai Jiaotong University
[†]Institute of Digital Media, Peking University
[‡]MIGU Co.,Ltd
xu.han@sjtu.edu.cn, {bjiang, sswang, swma, wgao}@pku.edu.cn, {lilin, suyi}@migu.cn

ABSTRACT

To encode ultra high definition (UHD) content such as 4K or 8K sequences, AVS3 introduced many complex coding tools, which increased the compression efficiency. However, UHD bitstreams, especially streams encoded with all intra configuration, still have quite high bit rates. To decoding such bitstreams in real-time, a large amount of computing resources are needed. In this paper, an efficient heterogeneous CPU+GPU framework is presented for AVS3 decoding under AI testing configuration. Through efficient asynchronous collaboration mechanism for CPU and GPU, the computation of different computing units and the data transfer between them are highly overlapped. Moreover, to increase GPU utilization, high parallel and low memory access latency schemes are carefully designed for each module in AVS3. The proposed framework archives 113 fps for 4K bitstreams decoding with the NVIDIA GeForce RTX 2080Ti GPU. When the bitrate is up to 300 Mbps, the decoding speed is still higher than 50 fps. For 8K bitstreams decoding, an average frame rate of 47 fps is obtained, which is 78 times faster than HPM 4.0.

Index Terms— AVS3, GPU, intra prediction

1. INTRODUCTION

AVS3 is the next generation video coding standard developed by the AVS working group. It aims at adapting to the growing demand of encoding and transmitting ultra high definition videos. Compared to its predecessor, AVS3 introduces many complex coding tools to increased the coding efficiency. Besides quad-tree (QT) partition, binary tree (BT) partition and extended quad-tree (EQT) partition are added in AVS3, which make the block partition more flexible and adapt to the content better. Intra derived tree (Intra-DT), intra prediction filter (IPF) and two step cross-component prediction mode (TSCPM) are introduced to increase the coding efficiency of intra frames. Affine inter prediction and many motion vector prediction or expression technologies are introduced to increased the codding efficiency of inter frames. AVS3 achieved 24.30% bd-rate reduction compared to AVS2 on UHD sequences and 26.88% compared to HEVC[1].

The introduced coding tools also increased the complexity of decoder, which brought new challenge for real-time decoding. To accelerate the decoding process, many methods are proposed to increase the utilization of computing resource of CPU. Single instruction, Multiple Data(SIMD) instruction is a common technology. It uses the vector register to process multiple elements with a single instruction, which increases the efficiency of each core of CPU. Chi et al.[2] explored speedup performance of different SIMD instruction sets for each module in HEVC. Another widely used technology is multiple threading, which fully utilizes multiple cores of CPU by executing multiple tasks in parallel. Duan et al.[3] used both of the methods and achieved 40-75 fps for 4K HEVC videos decoding on the Intel i7-2600.

However, with the development of coding tools and the resolution of contents increased, it is more and more difficult for CPU to handle the huge computation in decoding process. In this situation, many other platforms have been attempted, including GPU[4, 5, 6], FPGA[7] and DSP[8]. Among these platforms, GPU is the most common platforms in research and industry society. Compared to CPU, GPU have more advantages on parallel computing because of its special single instruction multiple thread architecture. De et al.[9] integrated the whole decoding pipeline except entropy decoding into a GPU decoder based on OpenHEVC, which achieved 56 fps for 4K all intra bitstreams with the NVIDIA GeForce GTX TITAN X GPU. In this paper, a new CPU+GPU heterogeneous framework for AVS3 intra decoding is proposed. Each module in the framework is high parallel and can be executed efficiently. The cooperation of CPU and GPU is totally asynchronous so that the different computation unit can perform different tasks simultaneously. Also, the data transfer is hidden in the asynchronous cooperation, which reduce the waiting time for both CPU and GPU.

This rest of the paper is organized as follow: the intra decoding of AVS3 is introduced in Section 2. The proposed parallel algorithms are presented in Section 3. The experimental results and conclusions are shown in Sections 4 and 5.

978-1-7281-1485-9/20/$31.00

2. INTRA DECODING IN AVS3

2.1. Block Partition Structure

In AVS3, a more flexible block partition structure is adopted. A frame is divided into several largest coding units (LCU). The size of LCU is configurable and the maximum size is 128x128. As it is shown in Fig.1, a LCU is further divided into multiple coding units (CU) by QT, BT and EQT partition. Once the coding unit is divided into BT or EQT, QT cannot be used in the subsequent partition process. Especially, the size of CU in intra frame cannot exceed 64x64. A CU may be further divided into 4 prediction units (PU) or 4 transform units (TU) according to its prediction partition mode. The size of TU may be different from that of PU.

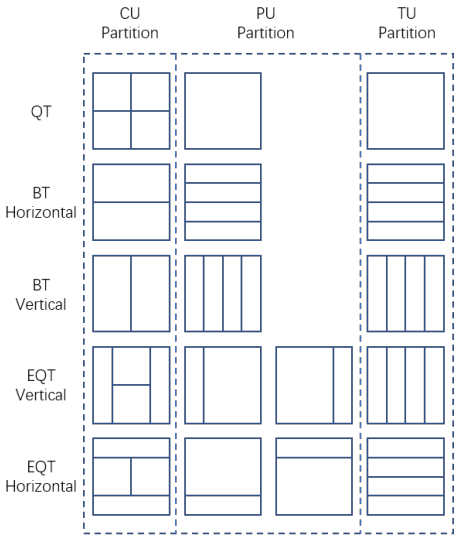


Fig. 1. Block partition in AVS3

2.2. Dequantization and Inverse Transform

Dequantization and inverse transform (DIT) is the first stage in the process of intra decoding. It is applied to the transform coefficients of each transform blocks (TB). The coefficients are obtained by entropy decoding and converted to residual data in DIT. If the coded block flag (CBF) of a TB is set to 0, it indicates that the TB is an all zero block and does not need to perform DIT process. If CBF is set to 1, the coefficients are de-quntized first. Then, the inverse secondary transform enable flag will be checked to determine whether inverse secondary transform will be applied on the top left 4x4 low frequency coefficients. At last, inverse DCT is performed on the coefficients according to the size of TB.

2.3. Intra Prediction

Similar to TU, a PU also consists of three luma and chroma prediction blocks (PB). The intra prediction (IP) is performed on each PB. As Fig.2 shows, the input data of prediction, also called reference pixels are taken from the top, left, top right and left bottom neighbor reconstructed blocks. When the left or top neighbor block is unavailable, the corresponding pixels are set to 128 or 512 according to the bit depth. When the top right or left bottom neighbor block is unavailable, the top or left reference pixels will be padded to fill the pixels. Because the prediction are depended on the reconstructed pixels of previous blocks, the intra prediction must follow the z-scan order strictly. After reference pixels are generated, the intra prediction is performed according to the intra prediction mode of CU. Then the IPF enable flag is checked to determine whether the prediction pixels will be filtered. Especially, for the chroma TB, if the prediction mode is TSCPM, a linear transform will perform on the luma reconstructed pixels and the chroma reconstructed pixels will be generated by downsampling of the transform result.

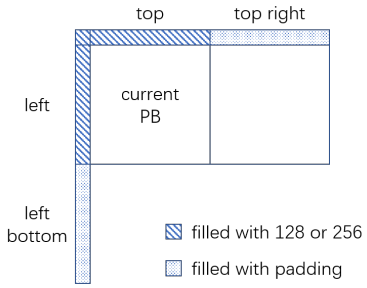


Fig. 2. Reference pixels in intra prediction

3. GPU-BASED INTRA DECODER

To fully exploit the power of GPU, there are three key points should be noticed. Firstly, in GPU, threads are organized as thread blocks and thread blocks are further organized as thread grids. A functional module (called kernel) will be executed by the threads in a grid parallel and each thread will be responsible for a sub-part of the module. Normally, the more threads can execute simultaneously, the faster the kernel will be executed. Secondly, wraps consist of 32 threads are the basic units of thread scheduling. Threads in a wrap must execute the same instruction. If the threads in a wrap diverge via a conditional branch, both branch paths will be executed by the wrap, which decrease the efficiency. Thirdly, the memory hierarchy of GPU is more complicated than CPU. The main memory of GPU is also called global memory. An global memory access needs about hundreds of cycles for GPU, which will result in a long latency of execution. There is another programmable memory called shared memory. It has much higher bandwidth and lower latency than global memory. Loading those frequently accessed data into shared memory will increase the memory access efficiency. However, shared memory is limited and once a section of shared

memory is allocated for a thread block, other thread blocks cannot access it anymore. So assign too much shared memory for a thread block will limit the number of thread blocks that can execute simultaneously.

3.1. Dequantization and Inverse Transform

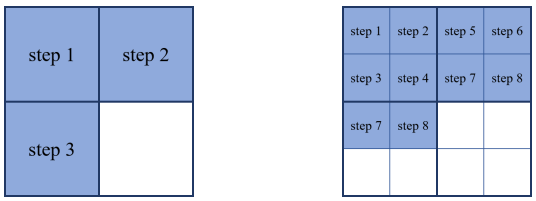
In the proposed dequantization and inverse transform (DIT) module, a compact data structure is designed to store the necessary info which includes the position and size of block, nonzero flag, secondary transform related flags and the position of corresponding coefficients. Different from [9], the transform info is not assigned for each 4x4 block but for each TB. This structure has many advantages. First, it reduces the global memory access for transform info when process large blocks. Secondly, it allows the coefficients of blocks stored in a frame-level buffer continuously and the coefficients of the all zero blocks can be dropped. Reducing the coefficients size will help to reduce the pressure of data transmission between CPU and GPU. Because all of the TBs in a LCU share a same QP, the QPs are stored in a LCU level buffer.

The DIT of each TB in a LCU is performed by a thread block. For luma TB, the thread block is consist of 4 wraps and for chroma TB, the wrap number is 2. If the nonzero flag of a block is not set, the thread block can bypass the subsequent computational steps. If the nonzero flag of TB is set, each threads in a wrap will fetch a coefficient from global memory and de-quantize it. The de-quantized coefficients are stored in shared memory to accelerate the later access. Then, the threads will check the secondary transform flags and the block size to determine whether the secondary transform is performed. The results are still stored in shared memory. Finally, matrix multiplication based inverse transform is applied to the coefficients. The intermediate result of the first inverse transform is also stored in shared memory. Although butterfly-based IDCT reduces the amount of computation and memory access greatly, the matrix multiplication is still more efficiency for GPU because it increases the number of threads executing simultaneously.

3.2. Intra Prediction

Similar to the data structure of TU, prediction info element is assigned for each PU. The structure contains the position and size of block, availability of neighbors blocks and IPF enable flag. Especially, if Intra-DT is enabled, the PU info will be split into multiple sub-PU info. To avoid global memory access for transformation info during reconstructing, nonzero flag is also included in prediction info.

Due to the data dependency between intra PUs, the intra prediction must follow the z-scan order strictly. [9] use a thread block consist of 8 wraps to perform the intra prediction and reconstruction of a LCU row and each wrap is responsible for a 8 pixel row. To confirm whether the dependency is satisfied, they use a buffer in shared memory to track the status of the PU in the LCU and a buffer in global memory to track the status of LCU in the frame. The frequent dependency detection will introduce large amount of memory access and decrease the efficiency of entire kernel. In AVS3, the LCU size has been increased to 128, which further increased the cost of dependency detection. So in our proposed intra prediction module, the reconstruction of a frame is split into many steps similar to wavefront parallel processing (WPP). Considering the maximum size of PU will not exceed 64x64, a LCU is split into four 64x64 sub-blocks and the reconstruction of a sub-block will not start until its top and top right neighbor sub-blocks are finished. As showed in Fig.3, the delay of two LCU in the same position of two adjacent rows is 8 sub-blocks in origin WPP and decreases to 6 sub-blocks after partition. For each step, a kernel is launched and each sub-block row will be assigned a thread block to performs the reconstruction of PU in z-scan order. Dependency check is unnecessary in this method. Moreover, considering TSCPM, the luma and chroma component of the same LCU are handled in different kernels, which ensures the luma pixels is reconstructed before the corresponding chroma PU starting.



(a) 128x128 blocks (b) 64x64 blocks

Fig. 3. Reconstruction process of blocks in a frame.

A thread block in IP module is consist of 4 wraps. At the beginning, the 4 wraps fetch the top, top right, left and left bottom reference pixels from global memory to shared memory respectively. A block synchronization is performed before prediction starting to ensure all reference pixels are fetched. Then each warp performs the intra prediction of quarter of the PB. The prediction result will be stored in shared memory. If the IPF flag is set to 1, a block synchronization is required and then each pixel will be filtered. If the nonzero flag is set to 1, the reconstruction is performed by adding the residual. The reconstructed pixels are finally write back to global memory. For chroma component, if TSCPM is enable, the reference pixels of luma component will be fetched for the parameters of the linear function. The parameters are calculated by only one thread and broadcast to other threads by shared memory. Then the transformed luma pixels are stored into shared memory. and down-sampled to chroma reconstructed pixels.

3.3. Deblocking Filter

The deblocking filter (DBF) is applied for 8x8 block top and right boundaries in a frame if they are TB edges. The edge filtering parameters of each 8x8 block is calculated on CPU.

GPU cannot handle the process efficiently because the calculation of whether the edge should skip filtering contains too much condition judgement and needs much related info. The two 4-pixels edge of a 8-pixels edge may have different filter type. Each 4-pixels edge need 2 bit to represent the luma component edge and chroma component edge filtering situation. So the filtering parameters of each 8x8 block needs 1 byte memory to store.

Similar to [5], we use two wraps to process an 8x8 block and each wrap is assigned for a 4-pixels edge. Through offsetting the 8x8 grid by 4 pixels, the vertical and horizontal filtering of each block is not effected by other blocks. At first, the QP of the 4x4 blocks beside the vertical boundary are fetched from the LCU QP buffer to calculate the average QP. Then the reconstructed pixels of the 8x8 blocks are fetched from global memory and stored in the shared memory. Each wrap is responsible to fetch 8x4 pixels and filter the 4 pixels edge. The vertical filtering result is stored in shared memory. At last, the horizontal filtering can start after both wraps finished their vertical filtering and the result will be written back to global memory.

3.4. Sample Adaptive Offset

The sample adaptive offset (SAO) is performed in LCU. The necessary parameters include the offset mode index and the offset values of each band. Each component has its own parameters, so the SAO parameters of a frame are distributed into three buffers. In order to not transfer the patch info, the availability of neighbors is also calculated on CPU.

We use a thread block only having one wrap to perform the SAO process of a LCU. To avoid redundant compunction, the comparing between a pixel and its neighbor is calculated only once and stored in the sign buffer in shared memory. This introduces the data dependency between the adjacent pixels rows. To avoid block synchronization, only one wrap is assigned for a unit and each thread is responsible for a column in the unit. For EO_45 and EO_135, the sign buffer for the first row is constructed according to its top neighbor LCU availability.

3.5. Adaptive Loop Filter

In adaptive loop filter (ALF) module, a thread block with 4 wraps is assigned for the filtering of a LCU. First, the LCU ALF switches are checked. If ALF is enabled in current LCU, the filter coefficients are fetched from global memory and stored in shared memory. Then each wrap will perform the filtering of 32 columns pixels. We did not cache the pixel in shared memory because the 128x128 shared memory will limit the number of thread blocks executing simultaneously. Also, we did not padding each LCU to reduce condition judgement because padding operation in global memory is inefficient.

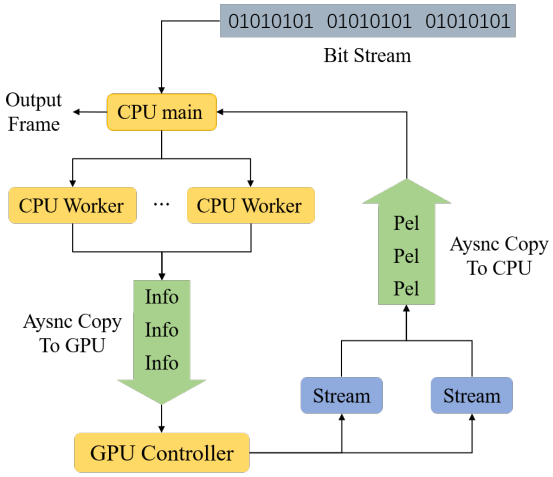


Fig. 4. Overall Framework

4. FRAMEWORK OF GPU-BASED INTRA DECODER

4.1. Asynchronous cooperation between CPU and GPU

In order to achieve excellent subjective quality, 4K or 8K bitstream usually has a high bit rate. So that the entropy decoding, especially the coefficients decoding puts a lot of computational pressure on the CPU. In our heterogeneous system, CPU is mainly responsible for entropy decoding and construct the information need by each module. Moreover, CPU needs to control the execution of GPU modules and the data transfer between CPU and GPU.

As Fig.4 shows, three kinds of CPU threads are created to perform different tasks. First is the main thread. It segments the bitstream of sequence by frames and assigns the stream to a free worker thread. A worker thread is an entropy decoder. When it receives a bitstream, it decodes the bitstream and constructs the info needed by the GPU modules. After the reconstruction info is ready, the worker thread will launch the asynchronous data transfer of the info from CPU to GPU. To reduce the memory usages, the thread will not be available for next frame until the transmission finished. Considering the waiting time caused by inconsistent decoding speeds of CPU and GPU, after the transmission, the reconstruction info will be added into a info list. Besides, due to the different content, the entropy decoding of different frames may cost different times and be added into the list randomly. Therefore, the list will be sorted in decoding order after the new info added. The last kind is GPU controller thread. It fetches info of frames from the info list and launch several GPU modules to finish the reconstruction. After a frame is decoded, its reconstructed pixels will be transferred to CPU asynchronously.

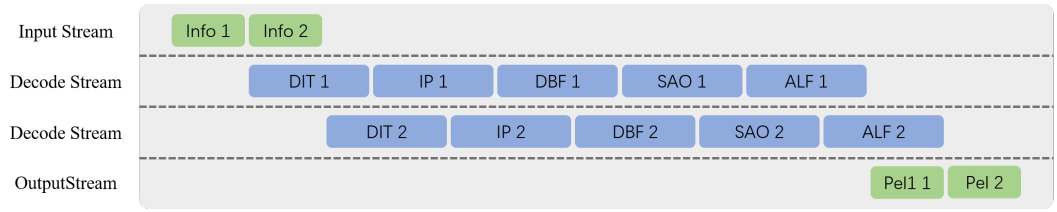


Fig. 5. Overall Framework

4.2. Concurrency Modules Execution

As Fig.5 shows, to overlap the memory operation and the computation time, the data transmission and modules are organized in different CUDA streams. The data transmission from CPU to GPU are executed in input stream and the data transmission from GPU to CPU are executed in output stream. All modules are organized in two decoding stream to overlap computation time of different frames. Due to the low parallelism, intra prediction module only consumes a little computational resources. Therefore, a large part of the intra prediction computing time of different frames can overlap, which increases the average decoding speed.

The communication between different are achieved by CUDA events. For each frame, a copy event will be created after the info transfer starting and a decoding event will be created after all modules are launched. The modules in decoding stream will not start until the transmission are finished. The transmission in output stream will not start until all of the reconstruction modules in corresponding frame finished. In this way, the GPU can execute the reconstruction task efficiently.

5. EXPERIMENT RESULT

To evaluate the efficiency of the proposed GPU-based decoder, we use open source encoder uavs3e [10] to generate the test bitstreams. uavs3e has basically consistent performance with AVS3 reference software HPM 4.0 but supports SIMD instruction and multi-threads encoding, which improved coding efficiency greatly. The AVS3 common test conditions (CTC) were adopted with All Intra configuration in our experiments. Only the four 4K sequences in the CTC are considered. Moreover, to evaluate decoding speed of 8K bitstream, we also used a non-CTC 8k 10-bit sequence. The sequence has 4000 frames and is spliced with multiple 8K sequences so it contains different scenes which can test the average performance of our decoder when decoding bitstream of different content.

We redesigned the overall decoder structure of HPM 4.0 and make it support multi-thread entropy decoding. The GPU modules are implemented with CUDA 10.1 and integrated into the optimized decoder. As is mentioned above, reading bitstream, entropy decoding and flow control are performed on CPU and the rest of modules are executed on GPU. We evaluated our algorithms on a device with Intel 8700K and NVIDIA GeForce RTX 2080Ti. Intel 8700K is a 6 cores 12 threads CPU, whose max turbo frequency is 4.70 GHz. NVIDIA RTX 2080Ti has 4352 CUDA cores and 1545 boost clock. The baseline is HPM 4.0. We also test the open source decoder uavs3d[11]. Compared to HPM 4.0, it is optimized with SIMD instruction and supports multi-threads decoding so that it is far more efficient.

Table 1 shows the decoding performance of different decoders. As can be observed, the proposed framework achieve significant faster decoding speed compared to uavs3d. When only 1 CPU thread is used, our decoder is 13.6 times faster than HPM and 2.5 times faster than uavs3d on 4K bitstreams. On 8K bitstreams, our decoder is 17.8 times faster than HPM and 2.76 times faster than uavs3d. It indicates that when the CPU resource is limited, the decoding speed can be increased greatly by migrating a part of computation to GPU. When 12 threads are used, our decoder archives 113 fps on 4K bitstream, which is 1.5 times faster than uavs3d. On 8K bitstreams, the average decoding speed is 47 fps, which is 2.0 times faster than uavs3d. Especially, even with a bit rate of 331Mbps (ParkRunning3, QP=22), our decoder can still decode the bitstream in real time. For the same bitstream, uavs3d can decode only 34.3 frames per second on average, even if the computing resources of CPU are used up.

6. CONCLUSION

This paper proposes a heterogeneous CPU+GPU decoder for intra decoding of AVS3. In our framework, CPU concentrates on entropy decoding and all other modules are executed on GPU. The collaboration of different computing units is asynchronous so that the time of computation and data transfer is well overlapped. Through proper thread organization and careful management of memory access, modules on GPU are high parallel and have low memory access latency. Experiment results show that the proposed decoder archived 113 fps on 4K bitstreams and 47 fps on 8K bitstreams.

7. ACKNOWLEDGEMENT

This work was supported by Key-Area Research and Development Program of Guangdong Province

Table 1. Performance of proposed GPU-based decoder

Sequence		QP	Bit-rate (Mbps)	HPM	uavs3d 1T		uavs3d 12T		Proposed 1T		Proposed 12T	
				FPS	FPS	Speedup	FPS	Speedup	FPS	Speedup	FPS	Speedup
4K	Campfire	27	124.7	1.8	6.6	3.67	45.4	25.22	9.9	5.50	67.2	37.33
		32	56.7	2.2	9.4	4.27	65.1	29.59	18.2	8.27	85.7	38.95
		38	25.7	2.5	12.8	5.12	86.1	34.44	32.9	13.16	120.5	48.20
		45	12.2	2.8	15.8	5.64	102.7	36.68	52.2	18.64	151.2	54.00
	DaylightRoad2	27	178.0	1.7	7.8	4.59	56.0	32.94	11.7	6.88	76.3	44.88
		32	89.7	2.0	10.9	5.45	76.2	38.10	19.6	9.80	99.1	49.55
		38	51.2	2.2	14.1	6.41	94.1	42.77	29.0	13.18	124.5	56.59
		45	26.5	2.5	18.1	7.24	116.4	46.56	44.3	17.72	158.0	63.20
	ParkRunning3	27	331.4	1.5	5.4	3.60	34.3	22.87	7.6	5.07	50.9	33.93
		32	211.4	1.6	6.7	4.19	47.6	29.75	10.8	6.75	70.5	44.06
		38	120.2	1.8	8.5	4.72	61.3	34.06	16.1	8.94	100.9	56.06
		45	55.5	2.1	12.6	6.00	82.4	39.24	26.9	12.81	124.4	59.24
	Tango2	27	79.2	2.0	10.5	5.25	67.7	33.85	23.4	11.70	108.1	54.05
		32	42.0	2.2	12.6	5.73	81.8	37.18	36.6	16.64	138.0	62.73
		38	24.9	2.3	14.8	6.43	92.2	40.09	50.2	21.83	156.6	68.09
		45	14.0	2.5	17.0	6.80	106.9	42.76	68.2	27.28	178.4	71.36
	Average			2.1	11.5	5.45	76.0	36.09	28.6	13.58	113.1	53.72
8K	final-8k	27	413.7	0.5	2.8	5.60	19.0	38.00	5.9	11.80	32.4	64.80
		32	254.1	0.6	3.4	5.67	22.2	37.00	8.3	13.83	41.4	69.00
		38	146.4	0.6	4.1	6.83	25.2	42.00	11.6	19.33	51.5	85.83
		45	75.4	0.7	5.1	7.29	30.2	43.14	16.8	24.00	62.9	89.86
	Average			0.6	3.9	6.42	24.2	40.25	10.7	17.75	47.1	78.42

(2019B010133001), MoE-China Mobile Research Fund Project (MCM20180702) and High-performance Computing Platform of Peking University, which are gratefully acknowledged.

8. REFERENCES

[1] Jiaqi Zhang, Chuanmin Jia, Meng Lei, Shanshe Wang, Siwei Ma, and Wen Gao, "Recent development of avs video coding standard: Avs3," in *2019 Picture Coding Symposium (PCS)*. IEEE, 2019, pp. 1–5.

[2] Chi Ching Chi, Mauricio Alvarez-Mesa, Benjamin Bross, Ben Juurlink, and Thomas Schierl, "Simd acceleration for hevc decoding," *IEEE Transactions on circuits and systems for video technology*, vol. 25, no. 5, pp. 841–855, 2014.

[3] Yizhou Duan, Jun Sun, Leju Yan, Keji Chen, and Zongming Guo, "Novel efficient hevc decoding solution on general-purpose processors," *IEEE Transactions on Multimedia*, vol. 16, no. 7, pp. 1915–1928, 2014.

[4] Bo Jiang, Falei Luo, Shanshe Wang, Xiaoqiang Guo, and Siwei Ma, "Efficient gpu-based inter prediction for video decoder," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 1109–1113.

[5] Diego F De Souza, Nuno Roma, and Leonel Sousa, "Cooperative cpu+ gpu deblocking filter parallelization for high performance hevc video codecs," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014, pp. 4993–4997.

[6] Diego F de Souza, Aleksandar Ilic, Nuno Roma, and Leonel Sousa, "Towards gpu hevc intra decoding: Seizing fine-grain parallelism," in *2015 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2015, pp. 1–6.

[7] Maleen Abeydeera, Manupa Karunaratne, Geethan Karunaratne, Kalana De Silva, and Ajith Pasqual, "4k real-time hevc decoder on an fpga," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 1, pp. 236–249, 2015.

[8] M Chavarrias, Fernando Pescador, Matías J Garrido, Eduardo Juarez, and Mickaël Raulet, "A dsp-based hevc decoder implementation using an actor language dataflow model," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 839–847, 2013.

[9] Diego F de Souza, Aleksandar Ilic, Nuno Roma, and Leonel Sousa, "Ghevc: An efficient hevc decoder for graphics processing units," *IEEE Transactions on Multimedia*, vol. 19, no. 3, pp. 459–474, 2016.

[10] Zhenyu Wang, Bingjie Han, Jiang Du, Kui Fan, Xi Xie, Shengyuan Wu, Tong Wu, Shiyi Liu, Jin Lin, Guisen Xu, Xufeng Li, Yangang Cai, Hao Lv, and Ronggang Wang, "uavs3e," https://code.ihub.org.cn/projects/1014/repository/uavs3e.

[11] Bingjie Han, Zhenyu Wang, and Ronggang Wang, "uavs3d," http://bggit.ihub.org.cn/p10472598/uavs3d.