PARALLEL VIDEO DECODING IN THE EMERGING HEVC STANDARD

Mauricio Alvarez-Mesa[1,2] *, Chi Ching Chi*[1]*, Ben Juurlink*[1]*, Valeri George*[2]*, Thomas Schierl*[2]

[1]Embedded Systems Architectures, Technische Universität Berlin, Berlin, Germany.
[2]Multimedia Communications, Fraunhofer HHI, Berlin, Germany.

ABSTRACT

In this paper we propose and evaluate a parallelization strategy for the emerging HEVC video coding standard. The proposed strategy is based on entropy slices which allows exploiting parallelism in the entropy decoding stage while maintaining high coding efficiency. Our approach requires to encode videos with one entropy slice per LCU row in order to decode multiple LCU rows in a wavefront parallel manner. Evaluations performed on a PC with 12 Intel Xeon cores running at 3.3 GHz show that it is possible to achieve real-time performance for 1920×1080p50 (53.1 fps) and 2560×1600 (29.5fps) video resolutions with speedups of 5.2× and 6.3× compared to sequential execution, respectively.

Index Terms— HEVC, video codecs, parallel processing, high definition video.

1. INTRODUCTION

Recent demands on video coding support for high resolutions such as 4k or UHD in consumer devices have further driven the video coding development. Therefore, the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T and ISO/IEC MPEG has started a new project to develop a new video coding standard aiming to reduce the bitrate of H.264/AVC state-of-the-art High Profile [5] by another 50%. The target application is beside 4k resolution, also the support of native HD and mobile resolutions. The standard further aims to support high quality color depth at 8 and 10 bit. Some of the application use cases, which have been selected for the first test model evaluation, are random access, such as used in Video-on-Demand or Broadcast applications as well as low delay for conversational applications. In order to take into account the variety of user devices, high efficiency and low complexity test cases have been defined, where the former targets highly processing-capable devices and the latter targets low-complexity such as embedded devices. The HEVC project started in 2010 and is scheduled for finalization in 2012/2013. The project development is implemented into the HEVC test Model (HM), which is the reference software following the standard developments. In this paper, we propose parallelization strategies and improvements for an HEVC software realization to support real-time HD and near real-time 4k on a standard PC platform.

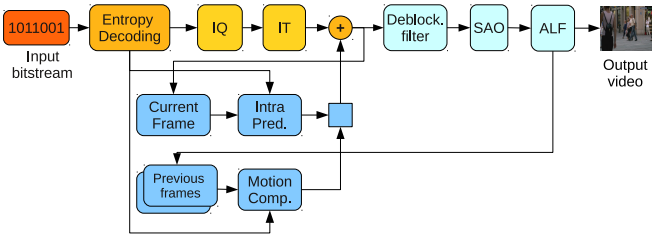

Fig. 1: General diagram of HEVC decoder

2. OVERVIEW OF HEVC

HEVC is based on the same structure as prior hybrid video codecs like H.264/AVC but with enhancements in each coding stage [8]. HEVC includes a prediction stage composed of motion compensation and spatial intra-prediction, an integer transform applied to prediction residuals, and an entropy coding stage that uses either arithmetic coding or variable length coding. Also, as in H.264/AVC, an in-loop deblocking filter is applied to the reconstructed frame. Fig. 1 depicts a general diagram of the HEVC decoder and its coding stages.

An important difference of HEVC compared to H.264/AVC is the frame coding structure. In HEVC each frame is divided into Largest Coding Units (LCUs) that can be recursively split into smaller Coding Units (CUs) using a generic quadtree segmentation structure. CUs can be further split into Prediction Units (PUs) used for intra- and inter-prediction and Transform Units (TUs) defined for transform and quantization.

HEVC also includes two new filters that are applied after the deblocking filter: Sample Adaptive Offset (SAO) and Adaptive Loop Filter (ALF). In the SAO filter the entire picture is considered as an hierarchical quadtree. For each subquadrant in the quadtree the filter can be activated by transmitting offset values that can either correspond to the intensity band of pixel values (band offset) or the difference compared to neighboring pixels (edge offset). ALF is designed to minimize the distortion of the decoded frame compared to the original one. It uses a Wiener filter that can be activated at the CU level using coefficients encoded at the slice level.

3. PARALLELIZATION OPPORTUNITIES

3.1. Slice- and Block-level Parallelism

Previous video codecs, e.g. H.264/AVC, have been parallelized using slice-level or block-level parallelism. In case of slice-level parallelism, a frame is split in several slices which are completely independent from each other. Multiple threads can be used to process the slices of a single frame in parallel, increasing the throughput and decreasing the frame latency at the same time.

Having more slices in a frame, however, reduces coding efficiency significantly due to three reasons. First, the entropy coding is less efficient due to breaking up the training of the context models and the inability to cross slice boundaries for context selection. Second, in the prediction stage the pixels from neighboring slices cannot be used. Finally, for each slice an additional slice header and start code needs to be present in the bitstream [9].

Block-level parallelism does not rely on having multiple slices in a frame, and does not have the associated coding losses. Instead coding blocks (macroblocks in H.264/AVC and LCUs in HEVC) inside a frame can be reconstructed in parallel using a wavefront approach to satisfy the prediction and filtering dependencies. The entropy decoding, however, cannot be parallelized on the block-level, and has to be performed sequentially for an entire frame. Multiple frames, however, can be entropy decoded in parallel [4]. This approach, however, introduces the need for frame buffers to hold the entropy decoded syntax elements, and can only reduce the frame latency of the reconstruction and filtering stages.

3.2. Entropy Slices

HEVC introduces a new coding tool, entropy slices [7], and different from regular slices, entropy slices have been designed for parallelism instead of error resilience. In both entropy and regular slices, context models are initialized at the beginning of each slice. The main difference is that in the reconstruction and filtering phases, it is allowed to use data of neighboring blocks across slice boundaries. Also entropy slice headers are smaller than regular slice headers, because common header data is only sent in the first slice header of a frame.

Until now entropy slices have only been considered as a parallelization tool for the entropy decoding stage. With entropy slices multiple threads can entropy decode the same frame which is beneficial to lower the frame latency in designs using block-parallelism. A large frame buffer is still required to store the entropy decoded data, since the entropy decode stage is still decoupled from the reconstruction and filtering. In our approach this frame buffer is not required as we combine the entropy decode stage with the reconstruction and filtering stages, without reducing parallelism or coding efficiency.

4. PARALLEL DECODING WITH ENTROPY SLICES

To combine the entropy decoding with the reconstruction and filtering phases, the entropy decoding dependencies must match the dependencies of the reconstruction and filtering phases. The reconstruction and filtering phases in HEVC exhibit the same wavefront dependencies as in H.264/AVC, and only differ in the coding block size. The wavefront dependencies restrict the parallelism to one block per block row. Currently, the number of entropy slices per frame is chosen arbitrary based on a fixed number of LCUs or byte size. This results in irregular slice shapes, that do not match the wavefront dependencies. Instead to match the wavefront dependencies a one entropy slice per row encoding approach must be enforced.

The BD-rate [2] losses that we have obtained (for the luma component) using one entropy slice per row are 5.4% and 6.3% for 2560×1600 and 1920×1080 resolutions, respectively. Enforcing a one slice per row encoding approach, allows context propagation between LCU rows in a wavefront manner, which was not present in our HM base code. Results show that the BD-rate losses are reduced to 1.7% and 1.3% for the same resolutions when using context propagation [6].

In our approach, the HEVC decoder can be parallelized by assigning one thread per LCU row. In each, so-called, line decoder thread the LCUs in a row are processed one-by-one. The entropy decode, reconstruct, and deblock vertical edge filter can be performed for the current LCU. In HM-3.0 the deblocking of the horizontal edges must overtake the deblocking of the vertical edges, and, therefore, has to be delayed by one LCU. This also in turn delays the SAO filter as it operates on the deblocked output image and, therefore, cannot proceed until the lower and right edges are deblocked. The SAO filter has to be performed on the upper left LCU, for which all the deblocked image data is available. The decoding order of the stages and the corresponding modified pixels for one LCU are illustrated in Fig. 2.

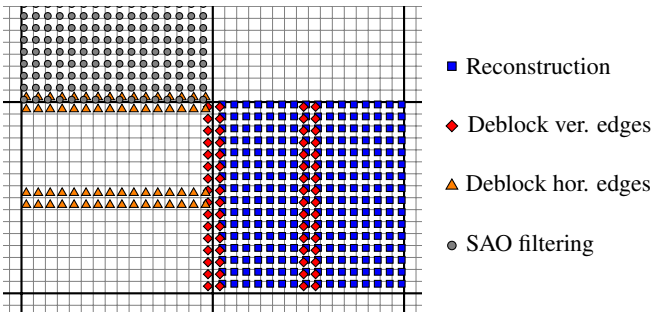

Fig. 2: The decoder stages are applied on different adjacent LCUs to maintain the kernel dependencies. Each square represent a 4×4 pixels.

To maintain the wavefront dependencies the line decoder

threads are synchronized using the Ring-Line strategy [4]. Using the Ring-Line strategy an arbitrary number of line decoders threads can be used to decode the picture in a line interleaved manner. The dependencies are maintained efficiently using a ring synchronization approach. Fig. 3 shows the wavefront progression when using four line decoder threads.

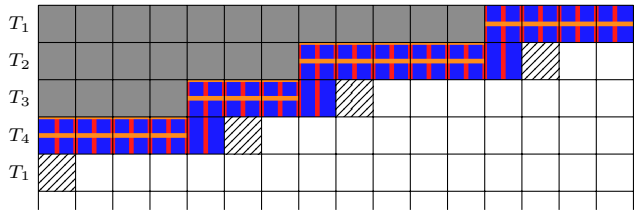

Fig. 3: Wavefront progression of the combined stages. The colors denote the same stages as in Fig. 2 and show the decoding progress of each stage, before starting the entropy decode of the hatched blocks.

The ALF is the last filtering step and is performed for the entire picture in a separated pass, and could not be be combined due to a misplacement of the ALF syntax elements. The ALF is LCU independent and can be executed for each block in parallel. In our implementation, to reduce cache line conflicts and synchronization overhead, eight consecutive LCUs are grouped in a work unit and processed by a single core.

5. EXPERIMENTAL RESULTS

5.1. Experimental Setup

We have implemented our parallel HEVC decoder on top of the HM-3.0 reference decoder [1]. We selected the Random Access High Efficiency (RA-HE) "profile" which targets the most demanding application scenarios of the current HEVC proposal.

Table 1 shows the main encoding parameters of the JCT-VC common conditions [3]. All the videos from the HEVC test sequences are encoded using these parameters with the HM-3.0 reference encoder. Due to space reasons, and because we are mainly interested in high definition applications, we only present results for class A (2560×1600 pixels) and class B (1920×1080 pixels) sequences. Additionally, we also evaluated 4K videos (3840×2160) from the SVT High Definition Multi Format Test Set. We will refer to these as class S sequences.

For our parallel decoding experiments we used a dual socket machine based on the Intel Xeon X5680 processor that has 6 cores per chip for a total of 12 cores. Main parameters of the architecture and software environment are listed in Table 2.

Options	Value
Max. CU Size Width	64×64
Max. Partition Depth	4
Period of I-frames	32
Number of B-frames (GOPSize)	8
Number of reference frames	4
Motion Estimation Algorithm	EPZS
Search range	64
Entropy Coding	CABAC
Adaptive Loop Filter (ALF)	enabled
Sample Adaptive Offset (SAO)	enabled
Quantization Parameter (QP)	22, 27, 32, and 37

Table 1: Coding Options

System		Software	
Processor	Intel Xeon X5680	Boost C++	1.42.1
ISA	X86-64	Compiler	GCC-4.5.2
μarchitecture	Westmere	Optimization level	-O3
Sockets	2	Operating system	Ubuntu 11.04
Cores/socket	6	Kernel	2.6.38-8
SMT	disabled	HEVC base software	HM-3.0 [1]
Clock frequency	3.33 GHz		
Level 3 cache	12MB / socket		
TurboBoost	disabled		

Table 2: Experimental setup

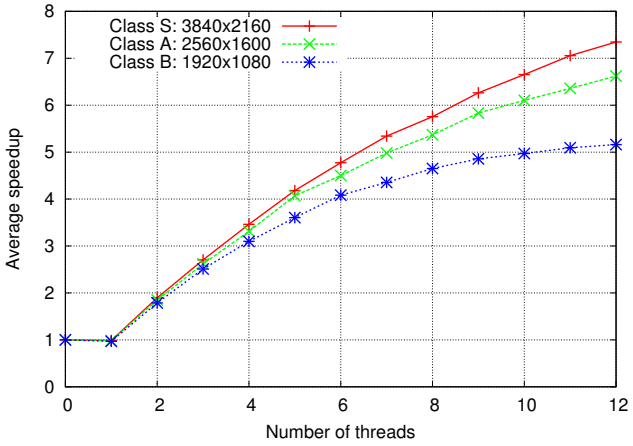

Fig. 4: Speedup. Baseline (0) is the sequential code

5.2. Speedup

Fig. 4 shows the average speedup for the three sequence classes under study. The average speedup represents the average of the speedups of the individual sequences in the class with 4 different QP values, each executed 5 times. The speedup of the individual sequences deviates at most 6% from the average. The speedup is computed against the original sequential code (thread 0) and is presented along with the parallel code using one core (thread 1). The speedup curves show that the parallel efficiency is relatively high for low core counts (82% for 4 cores) and decreases with a high core count (53% for 12 cores). With higher resolutions higher speedups are achieved.

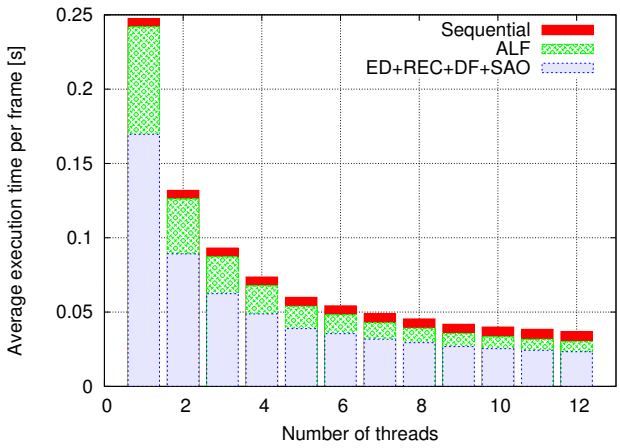

Fig. 5: Breakdown of execution time for class A sequences

5.3. Execution Profile and Performance

Fig. 5 shows the execution time contribution to the average frame execution time of the sequential part, the wavefront part (Entropy Decoding (ED), Reconstruction (REC), Deblocking Filter (DF) and SAO), and the ALF part. The contribution to the total execution time with 12 cores is 19%, 63% and 18%, respectively. Due to its massively parallel nature the ALF part reduces almost linearly with the number of threads. The wavefront part also reduces but reaches a saturation point. The sequential part, consisting mostly of the bitstream parsing and header decoding, stays constant, but increases its fraction of the total execution time according to Amdhal's law.

Table 3 shows the speedup of the parallel parts and the performance in frames per second at the highest core count. The ALF section exhibits almost linear speedup (efficiency of 88%). The wavefront part has a lower efficiency (57%). With 12 cores our parallel decoder achieves real-time performance for 1080p50 and is close to achieve it for 2560×1600p30. Additional optimizations (such as SIMD vectorization) can be applied to increase the single threaded performance to reduce the number of cores needed to achieve real-time performance.

The parallel efficiency of the wavefront stage can also be improved significantly by overlapping execution of consecutive frames. This requires the ALF to be included in the wavefront to be able to reference the partially completed previous frame. In HM-3.0 the ALF could not be combined in the wavefront, because the complete ALF flag array is transmitted in the first slice header of the frame, requiring the CU index relative to the start of the frame for indexing. In the most recent HM this is solved by transmitting the ALF flag array partitioned over the entropy slice headers, allowing the flag array to be indexed with the CU index relative to the start of the slice.

Video Class	'S'	A	B
Num. entropy slices	34	25	17
Max. processors	12	12	12
ED+REC+DF+SAO speedup	7.94	7.24	5.35
ALF speedup	11.15	10.62	9.98
Total speedup	7.35	6.62	5.20
Frames per second	15.38	29.54	53.15

Table 3: Speedup and frames per second at highest core count

6. CONCLUSIONS

In this paper we have proposed and evaluated a parallelization strategy for the emerging HEVC video codec. The proposed strategy requires that each LCU row constitutes an entropy slice. The LCU rows are processed in a wavefront parallel fashion by several line decoder threads using a ring synchronization. The presented implementation achieves real-time performance for 1920×1080 (53.1 fps) and 2560×1600 (29.5 fps) resolutions on a 12-core Xeon machine.

The proposed parallelization strategy has several desirable properties. First, it achieves good scaling efficiency at moderate core counts. Second, the number of line decoders can be chosen to match the processing capabilities of the computing hardware and the performance requirements. Third, using more cores increases the throughput and at the same time reduces the frame latency, making the implementation both suitable for low delay and high throughput use scenarios.

A limitation is the scaling efficiency at higher core counts. This is caused by the sequential part and the ramp-up and ramp-down efficiency losses of the wavefront parallel part. In future work this can be solved by pipelining the sequential part and overlapping the execution of consecutive frames.

7. REFERENCES

[1] HM-3.0 Reference software. `https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-3.0/`, 2011.

[2] G. Bjontegaard. Calculation of average PSNR differences between RD-curves. Technical Report VCEG-M33, ITU-T Video Coding Experts Group (VCEG), 2001.

[3] F. Bossen. Common test conditions and software reference configurations. Technical Report JCTVC-E700, Jan. 2011.

[4] C. C. Chi and B. Juurlink. A QHD-capable parallel H.264 decoder. In *Proc. of the Int. Conf. on Supercomputing*, pages 317–326, 2011.

[5] Advanced Video Coding for Generic Audiovisual Services. ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), 2003.

[6] F. Henry, K. Misra, S. Pateux, and A. Segall. Combined proposal JCTVC-E196 and JCTVC-E409. Technical Report JCTVC-E470, March 2011.

[7] K. Misra, J. Zhao, and A. Segall. Entropy slices for parallel entropy coding. Technical Report JCTVC-B111, July 2010.

[8] G. J. Sullivan and J.-R. Ohm. Recent developments in standardization of high efficiency video coding (HEVC). In *Applications of Digital Image Processing XXXIII. Proceedings of SPIE Volume: 7798*, 2010.

[9] V. Sze and A. P. Chandrakasan. A high throughput CABAC algorithm using syntax element partitioning. In *Proceedings of the 16th IEEE international conference on Image processing*, pages 773–776, 2009.