HEVC DECODER ACCELERATION ON MULTI-CORE X86 PLATFORM

Bingjie Han, Ronggang Wang, Zhenyu Wang, Shengfu Dong, Wenmin Wang, Wen Gao

School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School

ABSTRACT

In this paper, we propose a hybrid parallel decoding strategy for HEVC which combines task-level parallelism and data-level parallelism based on CTUs. The data-level parallelism makes the execution time distribution of different decoding stages more balanced, and makes the task-level parallelism more efficient. Our approach imposes no constraint on bit streams that they shall be generated by optional parallel coding tools such as tiles or WPP, so it can be applied for all kinds of HEVC bit streams. Furthermore, SSE, a typical SIMD instruction set on X86 platform, is utilized to accelerate time-consuming modules, which shortens the execution time gaps between different stages and make them in favor of parallel processing. We have implemented these acceleration strategies on HM-10.0 decoder, and a great speed-up ratio is achieved.

Index Terms— HEVC, video decoder, parallel processing, SIMD

1. INTRODUCTION

High Efficiency Video Coding (HEVC) is the latest joint video coding standardization project of the Joint Collaborative Team on Video Coding (JCT-VC) which is established by ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group. The first edition of the HEVC standard is finalized in January 2013, and it achieves about 50% lower bit rate than H.264/AVC for the same subjective quality [1]. The HEVC test Model (HM) decoder is an example implementation following the HEVC decoding standard. Aimed at correctness, completeness and readability, it doesn't use any parallelization techniques. Nowadays it is common that a PC has a dual-core CPU or quad-core CPU which supports Simultaneous Multithreading (SMT) meanwhile so that a suitable parallel decoding strategy is expected to achieve significant performance improvement on PCs. Besides, since Intel introduced the Streaming SIMD Extensions (SSE) on the Pentium III, the SIMD instructions have been supported well on PCs.

Parallel decoding strategies can be classified into two categories: task-level parallelism and data-level parallelism. Task-level parallelism is to divide a decoder into several sub-tasks and to attach each sub-task to a separate thread. To maximize the degree of parallelism, the execution time of all the sub-tasks is expected to be as close as possible. A task-level parallelism strategy which shortens execution time gap between different sub-tasks by adjusting size of blocks that a sub-task processes is proposed in [2], but it does not resolve the problem that the second sub-task is always consuming more time than other sub-tasks. Data-level parallelism is to process multiple data units in parallel by attaching each data unit to a separate thread. The data unit can be group of picture (GOP), frame, slice, slice segment, tile, coding tree unit (CTU) and so on. The granularities of GOP and frame are so large that parallelism based on them will lead to a long delay. The slice, slice segment and tile may be suitable parallelism granularities, but boundaries of them break up the connection of context models in entropy decoding and may also cut off the prediction dependency, which decreases the coding efficiency. Besides slice segment and tile, Wavefront Parallel Processing (WPP) is also adopted in HEVC, and it achieves a better balance between parallel granularity and coding performance loss than slice- and tile-level parallelism. Several approaches have been proposed to decode HEVC bit streams in parallel. For example, [3] proposes a parallelization strategy based on entropy slices which is similar to slice segments, and [4] introduces a parallelization approach called Overlapped Wavefont based on WPP. But, these approaches can only be applied for specific bit streams with corresponding parallel decoding mechanism support. Furthermore, some other approaches utilize data-level parallelism based on self-defined blocks. In [5], a data-level parallelism strategy based on inverted Z-shaped blocks is used on H.264/AVC decoders. This method simplifies dependencies between different threads, but it doesn't make full use of parallelism between different stages.

In this paper, we propose a new parallel decoding strategy for HEVC which combines task-level parallelism and data-level parallelism based on CTUs. Data-level parallelism makes execution time of different stages close and task-level parallelism makes full use of parallelism between different stages. This strategy can be applied for all kinds of HEVC bit streams without any constraint on coding tools. What's more, SSE optimization on time-consuming modules is utilized, and the execution time of different stages is more balanced after SSE optimization.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

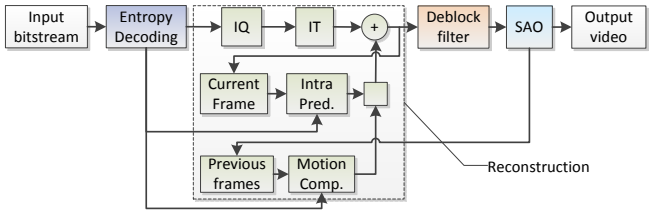

Fig. 1. The decoding process in HEVC

2. ARCHITECTURE OF HEVC DECODER

HEVC is based on the same architecture as prior video codecs like H.264/AVC but with enhancements in each coding stage. As shown in Fig. 1, decoding process of HEVC can be divided into four stages. The first stage is entropy decoding in which the relevant data to be used in the later stages are extracted. The second stage is reconstruction which includes inverse quantization (IQ), inverse transform (IT), and a prediction process that may be intra prediction or motion compensation. Then, in the third stage, a deblocking filter which is similar to that in H.264/AVC is applied to the reconstructed frame. Finally, a new filter called Sample Adaptive Offset (SAO) is applied in the fourth stage. The SAO filter simply adds offset values which are obtained by indexing a lookup table to certain sample values [6].

Compared with H.264/AVC, HEVC supports larger transform sizes such as 16x16 and 32x32, which are more difficult to be implemented. For intra prediction, HEVC support up to 35 prediction modes and wider range of prediction unit sizes than H.264/AVC. As to motion compensation, the use of a separable 8-tap filter for luma sub-pel positions and larger intermediate storage buffers make the implementation cost increase [7]. Furthermore, as an additional module, the SAO filter adds complexity inevitably.

3. CTU-LEVEL PARALLEL DECODING

As depicted in section 2, the decoding process in HEVC can be divided into four stages: entropy decoding, reconstruction, deblocking and SAO. The first three stages consume most of the decoding time as shown in Fig 5(a). Therefore, it will be efficient to use a task-level parallelism in which the three stages are attached to separate threads. As the different stages for a CTU shall be processed in order, threads that process different modules cannot be executed in full parallel. So it is necessary to set a synchronization mechanism between those threads. When entropy decoding of a CTU is finished, reconstruction of it can be executed instantly, which means the reconstruction thread has to delay at least one CTU relative to entropy decoding thread. For the deblocking thread, synchronization with the reconstruction thread is more complex. We have to satisfy dependencies that horizontal filtering should be prior to vertical filtering, and horizontal filtering of the current CTU cannot be executed until samples in the top, the current and the bottom CTUs have been reconstructed. In our approach, a synchronization mechanism is used first to guarantee that horizontal filtering will satisfy the dependencies, and vertical filtering is executed with one CTU delay relative to horizontal filtering.

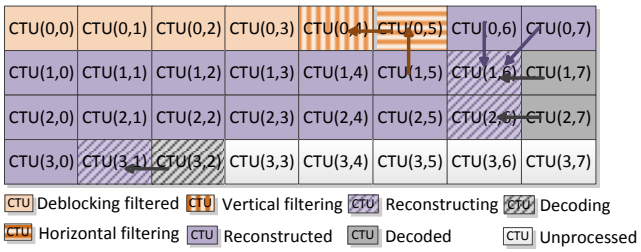

Fig. 2. An example of execution states when number of reconstruction threads is set to 3.

Reconstruction is a performance bottleneck in the task-level parallelism, as it is the most time-consuming stage. Considering that dependencies between CTUs in reconstruction are much weaker than dependencies in entropy decoding, we apply a data-level parallelism on it to improve execution speed of this stage. The dependencies in reconstruction are mainly reflected in intra prediction in which current CTU may need information from left, top-left, top and top-right CTUs. A common parallelism is that a CTU in the current row (except for the first row) does not start to be processed until the top-right CTU in the upper row is processed [8]. This diagonal wavefront parallelism satisfies that CTUs in different rows are processed in parallel under the dependencies between CTUs. However, the synchronous operation to be executed before each CTU is processed may be a significant cost and reduce the degree of parallelism especially for low resolution sequences.

In HEVC, CTUs can be divided into multiple coding units (CUs) through a recursive quad-tree partitioning and the decision whether intra or inter prediction is used is based on CUs. Considering that a inter mode CU need not to wait the finishing of reconstruction of any CUs in the upper CTU row, we can set a condition to decide whether the current CTU needs to wait the CTUs in upper row. We can find that the current CTU needs information from the top-right CTU only when the top-right CU in the current CTU is intra mode, and the current CTU needs information from the upper CTU when one of the top CUs is intra mode. To simplify the synchronization mechanism, in our design, the current CTU cannot be processed until reconstruction of the top-right CTU is finished, when one of the top CUs in the current CTU is intra mode; otherwise, the current CTU can be processed immediately after the left CTU finishes the reconstruction stage. As intra mode CUs account for a quite low ratio, most of the CTUs in different rows can be processed in parallel in B-frames and P-frames. Fig. 2 shows

a possible execution states when the hybrid parallelization strategy is applied.

$A_{-1,0}$				$A_{0,0}$	$a_{0,0}$	$b_{0,0}$	$c_{0,0}$	$A_{1,0}$				$A_{2,0}$
$d_{-1,0}$				$d_{0,0}$	$e_{0,0}$	$f_{0,0}$	$g_{0,0}$	$d_{1,0}$				$d_{2,0}$
$h_{-1,0}$				$h_{0,0}$	$i_{0,0}$	$j_{0,0}$	$k_{0,0}$	$h_{1,0}$				$h_{2,0}$
$n_{-1,0}$				$n_{0,0}$	$p_{0,0}$	$q_{0,0}$	$r_{0,0}$	$n_{1,0}$				$n_{2,0}$
$A_{-1,1}$				$A_{0,1}$	$a_{0,1}$	$b_{0,1}$	$c_{0,1}$	$A_{1,1}$				$A_{2,1}$

Fig. 3. Integer and fractional sample positions for luma interpolation.

$A_{-3,0}$	$A_{2,0}$	$A_{-1,0}$	$A_{0,0}$	$A_{1,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{-2,0}$	$A_{-1,0}$	$A_{0,0}$	$A_{1,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{5,0}$
$A_{-1,0}$	$A_{0,0}$	$A_{1,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{0,0}$	$A_{1,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{7,0}$
$A_{1,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{7,0}$	$A_{8,0}$	$A_{2,0}$	$A_{3,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{7,0}$	$A_{8,0}$	$A_{9,0}$
$A_{3,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{7,0}$	$A_{8,0}$	$A_{9,0}$	$A_{10,0}$	$A_{4,0}$	$A_{5,0}$	$A_{6,0}$	$A_{7,0}$	$A_{8,0}$	$A_{9,0}$	$A_{10,0}$	$A_{11,0}$
-1	4	-11	40	40	-11	4	-1	-1	4	-11	40	40	-11	4	-1

Fig. 4. Samples to be filtered and filter coefficients in the registers.

4. SSE OPTIMIZATION OF KEY MODULES

The streaming SIMD extensions (SSE) is a set of processor instructions for the x86 architecture designed by Intel to boost performance of multimedia and Internet applications, and it is introduced in Intel Pentium III series processors [9]. SSE is subsequently expanded by Intel to SSE2, SSE3, SSSE3, and SSE4. AMD also adds support for SSE instructions, starting with its Athlon XP and Duron (Morgan core) processors. SSE supports 128-bit wide vector operations so that it can greatly increase performance at the situation when exactly the same operations are to be performed on multiple data objects. Fortunately, most time-consuming modules in HEVC decoders own this feature, such as fractional sample interpolation in motion compensation, inverse transform, inverse quantization, deblocking filter, SAO and so on, so the decoder can be greatly accelerated if SSE is applied appropriately. To illustrate how to achieve performance improvement by using SSE, we take luma fractional sample interpolation as an example.

In motion compensation, if a motion vector has a fractional value, the reference block needs to be interpolated accordingly [10]. In HEVC, an 8-tap filter for the half-sample positions and a 7-tap filter for the quarter-sample positions are applied separately in fractional sample interpolation for luma samples. As shown in Fig. 3, Ai,j represents the luma sample at integer sample location (i, j), and symbols with lowercase letters represent samples at non-integer sample locations. The fractional samples are all calculated from the integer samples by applying the interpolation filter. Taking the samples $b_{0,j}$ as an example, the computation expression is as follows:

$$b_{0,j} = (\Sigma_{i=-3..4} A_{i,j} filter[i]) >> (B - 8)$$

The array *filter* stores coefficients of the 8-tap filter. The constant $B \geq 8$ is the bit depth of the reference samples. For most applications, the bit depth is set to 8. With this setting, we can load 16 serial samples ($A_{0,-3}\ldots A_{0,12}$) in a register first, and then trim and store them in four 128-bit registers by using PSRLDQ and PUNPCKLQDQ instructions as shown in Fig. 4. The filter coefficients are loaded twice to a 128-bit register shown in Fig. 4. Then 8 fractional samples ($b_{0,0}\ldots b_{7,0}$) will be calculated in few steps by using PMADDUBSW and PMADDW instructions. As values of fractional samples may be larger than 255, the results will be 16-bit numbers and the eight values calculated in parallel can be stored by just an instruction. But, for some fractional samples, such as $e_{i,j}$, they are generated by applying the filter to other fractional samples resulting that 8 samples to be filtered at most can be loaded to a 128-bit register and intermediate results may be beyond range of 16-bit numbers, so only 4 values can be computed in parallel. Similarly, we can process multiple samples together in other modules where the same operation is applied on different samples. As operations are always used based on blocks, it is critical to note that samples to be processed are not beyond the current block. By this method, we apply SSE in fractional sample interpolation, inverse transform, inverse quantization, weighted average computing in MC, and partial modules in deblocking filter and SAO.

5. EXPERIMENTAL RESULTS

To achieve higher execution efficiency, we converted the HM-10.0 decoder into an edition in C programming language instead of C++ programming language and the optimized edition is used as the baseline. The baseline edition achieves a little performance improvement relative to the HM-10.0 decoder. All the following experiment results are obtained on a PC with a quad-core Intel Core i7-3770k processor clocked at 3.5GHz. The test sequences cover 5 resolutions with 2 kinds of QP setting as shown in Table 2. We assign one sub-thread for deblocking filter, a configurable number of sub-threads for reconstruction, and the main thread for entropy decoding and other operations.

As shown in Table 3, a significant performance boost is achieved after SSE is applied on time-consuming modules. Since we don't apply SSE on entropy decoding and intra prediction, the improvement in all-intra configuration is small compared with LB and RA configurations. After SSE optimization, decoding time distributions of the four stages change significantly for random-access and low-delay configurations. Take the random-access configuration as an example, as shown in Fig. 5, after SSE optimization, the decoding time distribution of different stages is more balanced.

Sequence	QP	AI [Mbit/s]	LB [Mbit/s]	RA [Mbit/s]
Traffic(2560x1600,30Hz)	29	45.50	7.133	7.441
	33	29.07	1.759	2.213
ParkScene(1920x1080,24Hz)	29	21.98	2.262	2.336
	33	12.86	1.122	1.236
FourPeople(1280x720,60Hz)	25	22.57	1.204	1.670
	29	15.22	0.650	0.999
PartyScene(832x480,50Hz)	25	33.20	4.790	4.454
	29	22.21	2.508	2.475

Table 2. Bit rate of test sequences. AI is all-intra. LB is low-delay using B slices and RA is random-access.

Sequence	AI	LB	RA
Traffic QP=29	1.140	1.825	1.983
Traffic QP=33	1.159	1.933	2.087
ParkScene QP=29	1.141	1.854	2.027
ParkScene QP=33	1.165	2.020	2.165
FourPeople QP=25	1.053	1.455	1.538
FourPeople QP=29	1.092	1.577	1.675
PartyScene QP=25	1.162	1.377	1.599
PartyScene QP=29	1.190	1.387	1.650

Table 3. Speed-up ratios obtained by using SSE

Sequence	AI2	AI3	LB2	LB3	RA2	RA3
Traffic QP=29	2.838	2.842	2.961	2.969	2.781	2.825
Traffic QP=33	3.063	3.084	2.873	2.882	2.774	2.799
ParkScene QP=29	2.664	2.613	2.916	2.872	2.723	2.765
ParkScene QP=33	3.076	3.131	2.850	2.823	2.732	2.827
FourPeople QP=25	2.676	2.683	2.237	2.264	2.361	2.361
FourPeople QP=29	2.857	2.815	2.248	2.240	2.362	2.370
PartyScene QP=25	1.813	1.800	2.130	2.171	2.126	2.114
PartyScene QP=29	2.031	2.060	2.300	2.333	2.272	2.273

Table 4. Speed-up ratios obtained by using multiple threads. AI2 means two reconstruction threads are used in all-intra configuration.

Sequence	AI		LB		RA	
	BL	HM	BL	HM	BL	HM
Traffic QP=29	3.236	3.711	5.406	5.754	5.516	5.805
Traffic QP=33	3.550	4.136	5.553	5.969	5.789	6.129
ParkScene QP=29	3.038	3.495	5.404	5.535	5.520	5.520
ParkScene QP=33	3.582	4.161	5.758	5.930	5.916	5.998
FourPeople QP=25	3.109	3.571	3.079	3.767	3.776	4.464
FourPeople QP=29	3.401	3.926	3.117	3.935	3.896	4.692
PartyScene QP=25	1.908	2.141	3.100	3.270	3.269	3.400
PartyScene QP=29	2.219	2.465	3.628	3.815	3.805	3.963

Table 5. Speed-up ratios obtained by using SSE and multiple threads compared to the baseline edition (BL) and the HM10.0 decoder (HM). The number of reconstruction threads is set to two.

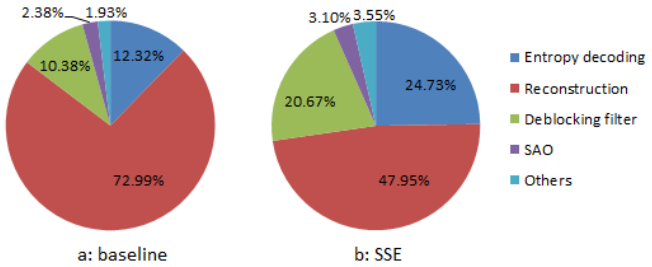

Fig. 5. Decoding time distributions for random-access configuration in the baseline edition (a) and SSE optimized edition (b). The part of others includes reading bit stream, initializing and destroying some structures.

Sequence	QP=29		QP=33	
	LB	RA	LB	RA
Traffic(2560x1600)	50.092	49.834	61.249	57.937
ParkScene(1920x1080)	78.701	78.303	98.320	97.087

Table 6. The decoding speed (frame rate) achieved by using SSE and multiple threads for Full HD (1920x1080) and WQXGA (2560x1600) sequences

Table 4 shows the speed-up ratios for sequences of different resolutions using our parallel decoding strategy. Larger speed-up ratios are achieved for sequences of higher resolutions owing to more CTUs that can be processed in parallel alleviating synchronization delay cost. A significant acceleration is not achieved when the number of reconstruction threads is bigger than two. Table 5 shows the speed-up ratios using multiple threads and SSE compared to the baseline edition and the HM10.0 decoder.

6. CONCLUSIONS

In this paper, we propose a hybrid parallelization strategy for HEVC decoder combining task-level parallelism and data-level parallelism on CTUs, and SSE optimization on time-consuming modules is utilized. Experimental results demonstrate that a great speed-up ratio can be achieved by using above two optimization methods. As shown in Table 6, Full HD and even higher definition (WQXGA) bit streams can be decoded in real-time on a quad-cores PC.

7. ACKNOWLEDGE

This work was partly supported by the grant of National Science Foundation of China 61370115, and Shenzhen Basic Research Program of JC201104210117A, JC201105170732A, and JCYJ20120614503016 23.

8. REFERENCES

[1] G.J. Sullivan, Woo-Jin Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE*

Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[2] Won-Jin Kim, Keol Cho and Ki-Seok Chung, "Stage-based frame-partitioned parallelization of H.264/AVC decoding," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 1088-1096, May 2010.

[3] M. Alvarez-Mesa, C.C. Chi, B. Juurlink, V. George and T. Schierl, "Parallel video decoding in the emerging HEVC standard," *IEEE International Conference on Image Processing (ICIP)*, pp. 213-216, Sept. 30 2012-Oct. 3 2012.

[4] C.C. Chi, M. Alvarez-Mesa, B. Juurlink, V. George and T. Schierl, "Improving the parallelization efficiency of HEVC decoding," *IEEE International Conference on Speech and Signal Processing (ICASSP)*, pp. 1545-1548, March 25-30 2012.

[5] B. Erik, van der Tol, Egbert G.T. Jaspers and Rob H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," *Image and Video Communications and Processing, Proc. SPIE*, vol. 5022, May 7 2003.

[6] Chih-Ming Fu, E. Alshina; A. Alshin, Yu-Wen Huang, Ching-Yeh Chen, Chia-Yang Tsai, Chih-Wei Hsu, Shaw-Min Lei, Jeong-Hoon Park and Woo-Jin 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.

[7] F. Bossen, B. Bross, K. Suhring and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.22, no.12, pp.1685,1696, Dec. 2012.

[8] R.G. Wang, J. Wan, W.M. Wang, Z.Y. Wang, S.Y. Dong and W. Gao, "High Definition IEEE AVS Decoder on ARM NEON Platform," IEEE International Conference on Image Processing (ICIP), 2013

[9] S.K. Raman, V. Pentkovski and J. Keshava, "Implementing streaming SIMD extensions on the Pentium III processor," *IEEE Micro*, vol.20, no.4, pp.47,57, Jul/Aug 2000.

[10] Alexander Alshin, Elena Alshina, Jeong Hoon Park and Woo-Jin Han, "DCT based interpolation filter for motion compensation in HEVC," *Applications of Digital Image Processing XXXV, Proc. SPIE*, vol.8499, Oct. 15 2012.