EFFICIENT REALIZATION OF PARALLEL HEVC INTRA ENCODING

Yanan Zhao1, Li Song1, Xiangwen Wang2, Min Chen2, Jia Wang1

1Institute of Image Communication and Network Engineering, Shanghai Jiao Tong University
2Shanghai University of Electric Power
{ya_nanzhao, song_li}@sjtu.edu.cn, wxw21st@gmail.com, chenm003@163.com, jiawang@sjtu.edu.cn

ABSTRACT

The emerging next generation video coding standard – HEVC (High Efficiency Video Coding) achieves more than 50% bitrate reduction in comparison with its predecessor H.264/AVC under the same visual quality, but at the expense of a substantial complexity increase. As a result, a traditional single thread encoder which operates in sequential way is difficult to be optimized to a satisfactory level, for example, a real-time HEVC codec for Ultra High Definition resolution (4Kx2K@50fps and beyond). Thus parallelism in the encoding stage must be extensively utilized to meet such requirements. In this paper, we give a full analysis of fine granularity parallelism for HEVC intra coding and exploit its maximum parallel degree in context of coding unit dependency. Then an efficient parallelization encoding strategy is proposed and implemented on x265, the first open source HEVC encoder targeted to real-time application, to demonstrate its potentiality. Experimental results show that our scheme could achieve average 502% speedup gains for different video sequences without performance loss.

Index Terms— HEVC; Intra Coding; Parallelism; Multi-thread;

1. INTRODUCTION

With the fast upgrading of video capture and display devices, HD (High Resolution) and UHD (Ultra High Resolution) format videos get a growing popularity in daily lives. However, the increased resolution poses great challenge on video storage, transmission and real-time processing, as the state-of-the-art video coding standard H.264/AVC could no longer provide satisfactory compression efficiency. Thus, the ITU-T VCEG and ISO/IEC MPEG together formed the Joint Collaborative Team on Video Coding (JCTVC) for developing the next generation video coding standard, called High Efficiency Video Coding (HEVC) [1]. HEVC aims at providing a 50% bitrate reduction compared with H.264/AVC under the same visual quality, and the first version has been finalized on Feb. 2013.

HEVC inherits the hybrid video coding framework since H.261, but obtains substantial performance improvement than its predecessor H.264. The achieved coding benefits are mainly due to works of two aspects: new coding tools, including quad-tree structure, SAO [5], Tiles [6]; extension of existed structures, such as larger block sizes. A rate distortion optimization (RDO) method is adopted to achieve best performance, which checks all the candidates of quad-tree partitions and prediction modes in a brute force way. As a consequence, the intra coding complexity increases dramatically [2] [4].

In tackling with this problem, lots of proposals have been put forward for HEVC [6][7][8][9]. These tools exploit a similar routine to achieve parallelism: divide the picture into independent regions and encode them simultaneously. These methods exploit the large scale parallelism and their performance at decoder side have been demonstrated in several works [8][9][10]. Parallel encoding performance on more *realistic* encoder rather than *reference software* HM, has not yet been reported in literatures. Beside these high level tools, fine granularity parallelism exists in intra coding, but the study on this topic is rare. This article aims at filling this gap. In this paper, we analyze the fine granularity parallelism in HEVC intra coding, and derive the maximum parallel extent in context of coding unit dependency. A parallelization encoding scheme is proposed and implemented on x265, an ongoing open source realtime HEVC encoder, to show significant parallel acceleration gain can be obtained.

This paper is organized as follows. Section 2 presents a brief review of the HEVC intra coding. Section 3 analyses the detailed coding unit dependency. In Section 4, a parallel intra coding scheme is proposed to give a glance at the parallel processing potential. Experimental results are given in section 5. Finally, section 6 concludes this paper.

2. OVERVIEW OF HEVC INTRA CODING

This work was supported by National 863 project (2012AA011703), National Key Technology R&D Program of China (2013BAH53F04), NSFC (61221001, 61271221), the 111 Project (B07022) and the Shanghai Key Laboratory of Digital Media Processing and Transmissions.

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

2.1. New block concepts

Three independent block concepts are defined in HEVC: CU (Coding Unit), PU (Prediction Unit) and TU (Transform Unit). CU is the basic unit of coding, the largest CU is called CTU (Coding Tree Unit). A CU can be predicted by one PU or many non-overlapped PUs, each of which with individual prediction parameters. In intra coding, only the smallest CU is allowed to be predicted by four squares PUs. TU is the basic unit for transform [2]. Separating the block structure into three different concepts allows each to be optimized according to its role, which results in improved coding efficiency.

2.2. Quadtree-based splitting structure

A quadtree-based structure is introduced to HEVC for CU splitting. Two configuration parameters, the CTU size and maximum split depth, are defined by the encoder. A CTU can be recursively divided into four sub-CUs and each sub-CU may be further split into four smaller sub-CUs, until reaching the maximum allowed splitting depth. This splitting structure allows the partition to be adapted to the specific characteristics of the picture.

2.3. Up to 36 prediction modes

Intra prediction modes are extended from at most 9 in H.264/AVC to at most 36 in HEVC[1][3], constituted by 33 directional modes (shown in Fig.1), one DC mode, one planar mode, and one special mode used for chroma only. The finer angle divisions make the prediction more accurate, resulting in less prediction residuals for the following transform coding.

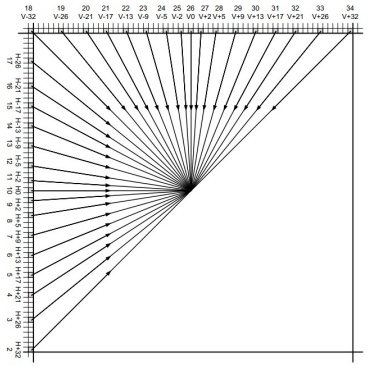

Fig.1. HEVC Intra prediction modes

HEVC intra coding achieves remarkable performance improvement compared with H.264/AVC, but with the expense of several times complexity increase [4]. For each CTU, the encoder decides the best splitting and prediction mode by a rate-distortion optimization process, which enumerates all the candidates of CU partition and intra modes. Although certain fast algorithms have been proposed, which simplify the process by skipping some unlikely candidates, the remaining complexity is still far from satisfactory. Besides, these fast algorithms often contribute to a decreased performance as the candidates removed may still be the best choice. Parallel processing, however, proves to be a nice solution to this speed-efficiency dilemma.

Several works haven targeted to parallel processing for H.264/AVC. Ref [11] analyzed the intra coding dependency in H.264/AVC and AVS, and simulated with a Graphic Processing Unit (GPU). Recently many parallel tools have also been proposed for HEVC [6][7][9]. Ref [6] proposed a parallel method named Tile, which divides a picture into regular rectangular regions and encode each separately. This division manner greatly simplifies the codec design. Ref [7] described WPP (Wavefront Parallel Processing), an algorithm that treats each CTU row as independent regions. The most remarkable benefit is the little RD performance loss. Ref [9] analyzed and evaluated the HEVC parallel tools exhaustively in terms of scalability and efficiency. It also proposed OWF (Overlapped Wavefront) to further improve the performance of WPP by allowing frame-level parallelism. These algorithms suffer certain amount of performance penalty in RD sense, resulted from the damaging of spatial correlation, or the discontinuity of context model updates. In the next section, we analyze the spatial correlation and intra dependency in detail.

3. DEPENDENCY CONSTRAINTS BETWEEN NEIGHBORING CUS IN INTRA CODING

In HEVC intra coding, all the prediction modes utilize the same basic set of reference samples, constituted by the reconstructed pixels of left-bottom and left columns, top and top-right rows and left-top point. It should be noted, that in addition to left, above, and above-right reconstructed samples used for H.264/AVC intra prediction, below-left side of samples are also used for HEVC. These samples were excluded from the H.264/AVC process as they were rarely available in the traditional macroblock based coding structure, but the hierarchical quad-tree architecture makes these candidates available more frequently [4]. The intra coding in HEVC is performed by unit of CTU in raster scan order. Within each CTU, CUs are processed in quad-tree traverse order. As a consequence, two levels of CU dependency exist. We discuss them separately.

3.1. CTU-Level dependency

In CTU level, each CTU must waits until its left and top-right neighbor CTUs finish reconstruction. This constraint

makes the current CTU row always two-CTU latent than its adjacent upper row, and the latency maintains until the upper row finishes its right-most CTU. The maximum parallelism is achieved if each CTU starts encoding whenever its two dependent CTUs finish. Assume each CTU needs similar amount of time for processing (as the processing time and pixel numbers have a roughly linear relationship); we could assert the processing threads will form an inclined frontier which keeps a constant slope during the encoding period. Fig.3 depicts an example with 8 threads launched. As the frontier moves forward like a wavefront, this method is often called wavefront processing.

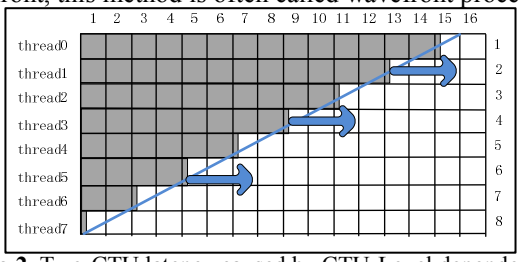

Fig.2. Two-CTU latency caused by CTU-Level dependency, at the moment thread7 starts processing its first CTU

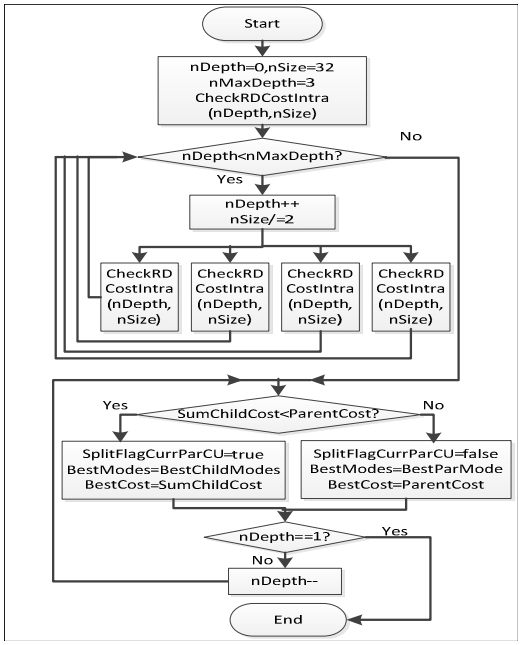

Fig.3. HEVC intra partition deciding process

3.2. CU-Level dependency

Intra dependency not only exists on CTU level, but also inner-CTU level. Assume CTU size is 32x32, it can be further split into 16x16, 8x8 CUs and 4x4 PUs in quad-tree manner. For simplicity, in sequel we use CU_nSizexnSize(x,y) denoting a CU with offset (x, y) in a CTU and size nSize, and refer a PU_4x4 as CU_4x4. For each CTU, the HEVC encoder decides the best partition and a mode in a brute force way, a flowchart is given in Fig.4. And for each CU, the encoder first decides the best mode by performing intra prediction with 35 modes, storing the best cost in RD sense; then splits it into four sub-CUs and repeats the process until reaching the maximum allowed depth. This calculation is processed from top to bottom in quad-tree traverse manner. When the best modes and costs for CUs in each level have been acquired, the encoder decides the best partition by checking whether the cost of a parent CU is smaller than the cost sum of its children's. The process is from bottom to top. Considering the sequential manner, this is actually high time-consuming. We discuss it in terms of iterations.

In HEVC, the minimum intra prediction block is 4x4, so we denote the time performing intra prediction on one CU_4x4 with 35 modes as one iteration. Due to the roughly linear relationship between intra prediction and pixel numbers, it is easy to deduce that process a CU_8x8, CU_16x16, CU_32x32 each need 4, 16, and 64 iterations. Then we can get the specific iteration each CU starts processing, since the whole process is sequential. Here we particularly fill the starting moments of each CU_4x4 in Fig.5 (a) (we ignore the cost comparing time as it is trivial compared with processing time).

85	86	93	94	133	134	141	142
87	88	95	96	135	136	143	144
101	102	109	110	149	150	157	158
103	104	111	112	151	152	159	160
181	182	189	190	229	230	237	238
183	184	191	192	231	232	239	240
197	198	205	206	245	246	253	254
199	200	207	208	247	248	255	256

(a)

1	2	5a	6	9b	10b	15a	16a
3	4	7	8a	13b	14b	17a	18a
5b	8b	11a	12a	15b	18b	21a	22a
9a	10a	13a	14a	19a	20a	23a	24a
11c	14b	17b	20b	25b	26b	30a	31
15c	16b	21b	22b	28b	29b	32	33a
17c	23b	26a	27a	30b	33b	34b	35b
24b	25a	28a	29a	34a	35a	36	37

(b)

Fig.4. The starting time of each CU_4x4:
(a) Sequential, (b) with maximum parallelism

Take CU_4x4(8,0) as example. Before reaches this CU, the encoder needs 64 time iterations for CU_32x32(0,0), 16 for CU_16x16(0,0), 4 for the CU_8x8(0,0), and 1 for each CU_4x4 with offsets (0,0), (4,0), (0,4), (4,4), and 4 for CU_8x8(8,0). This sums up to be: 64+16+4+1+1+1+1+4 =92. So CU_4x4(8,0) starts at 93th iteration. In this sequential manner, the completion of one CTU needs total 257 iterations.

The intra dependency guaranties that the current CU starts after the reconstruction of its dependent CUs. At the start moment of one CTU, the neighboring CTUs have been constructed, so the CU_32x32, CU_16x16, CU_8x8 and CU_4x4 with offset (0,0) are all available for processing. Further, CU_4x4(8,0) and CU_4x4(0,8) could start simultaneously at 93th iteration. Ref [11] discussed this topic for H.264/AVC and AVS, and used a directed acyclic graph (DAG) to visualize the dependency relationship and parallel execution. Here we also utilize this tool to analysis the HEVC parallel intra processing. This is more complicated than the previous work because of the introduced larger block sizes and quad-tree structure. For space limit, we only show part of the whole graph in Fig.6.

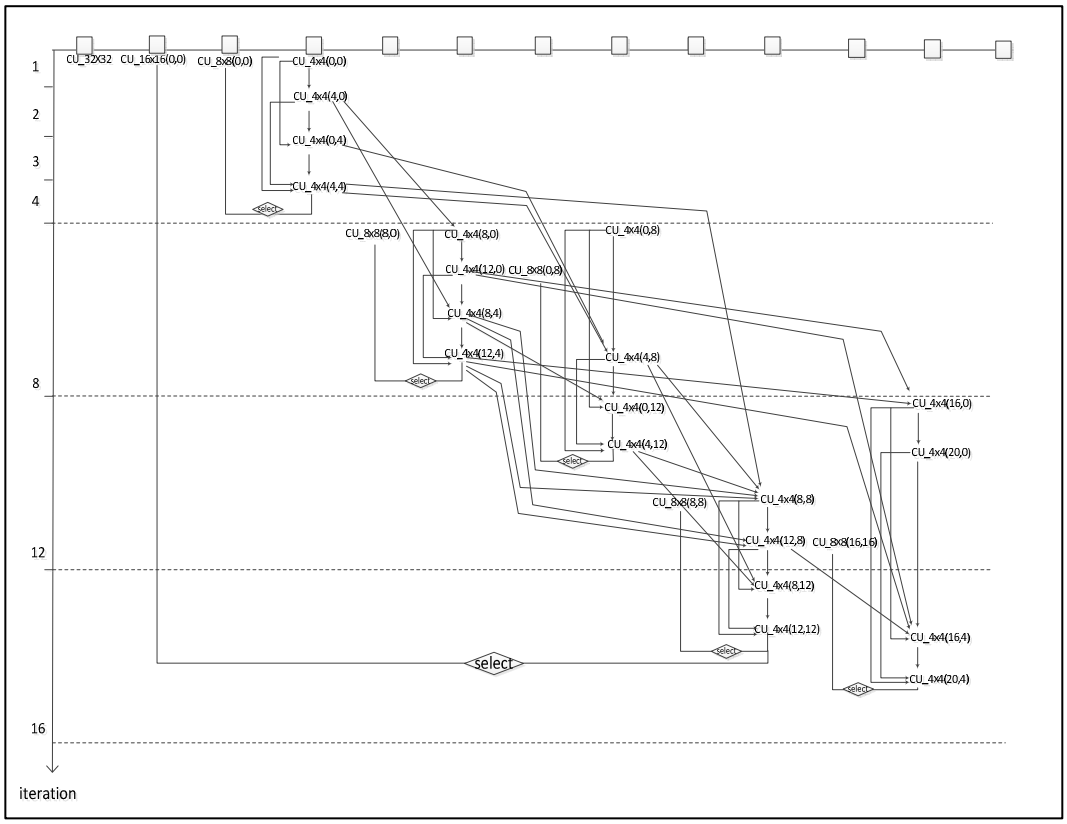

Fig.5. Part of the DAG graph for one CTU encoding

Fig.6 shows iterations of the upmost 20 CU_4x4s in quad-tree traverse order and their corresponding parent CUs. Vertical axis stands for iteration, while horizontal axis is parallel processing thread. The lines connecting two CUs indicate that the lower CUs rely on the reconstruction of the upper CUs.

Fig.5 (b) is the new time table for each CU_4x4 in this parallel processing manner. Note that although the last CU_4x4 ends at iteration 37, the entire processing for one CTU ends at 64, which is the finishing time of the largest CU. Obvious, processing time of all CUs have been hided in the largest CU's processing time. If parallelism is fully utilized, the theoretical speedup gain for one CTU can be as high as: (257-64)/64 x 100% = 301.56%.

4. PROPOSED PARALLELIZATION SCHEME

In this section, we propose a two-stage parallelization speedup scheme exploiting CTU level parallelism. The design of the scheme takes two major aspects into consideration: maximizing encoding speed and minimizing compression performance loss. The overall structure strikes a good balance between design effort, parallelism degree and RD performance.

The first stage, called parallel processing stage, numbers of threads are launched to perform intra prediction, with each thread processing one CTU row and proceeding under the same dependency constraint described in Section 3.1. Once one CTU is over, the intra information is stored first. The second stage, call entropy coding stage, one additional thread is used to encode all CTUs within the whole picture in raster scan order. This scheme achieves three benefits in terms of encoding speed and compression efficiency:

-*Maximizing acceleration ratio*: In the first stage, a wavefront processing method is used to get the split and best mode information for each CTU. This enables maximizing the processing speed, considering the fact that major time is consumed by the brute force selection.

-*Minimizing performance loss*: One independent thread encodes all CTUs in the second stage. The entropy coder proceeds in raster scan order within the whole picture, so the continuity of context state updating is guaranteed. This avoids the coding efficiency decrease introduced by WPP, which assigns each row an entropy coder and initializes each by inheriting context from the upright CTU.

-*Increasing speedup gain*: The entropy coding of a CTU can begin if this CTU has been processed by stage one and all its preceding CTUs (in raster scan order within the whole picture) have been encoded. Thus the two stages could start simultaneously. Fig.7 exhibits two different encoding moments. The gray squares are the CTUs finished processing, and blue ones are those

completed entropy coding. The threads are captioned on the left of each CTU row, with *thread0* to *thread3* for processing, *thread4* for entropy coding.

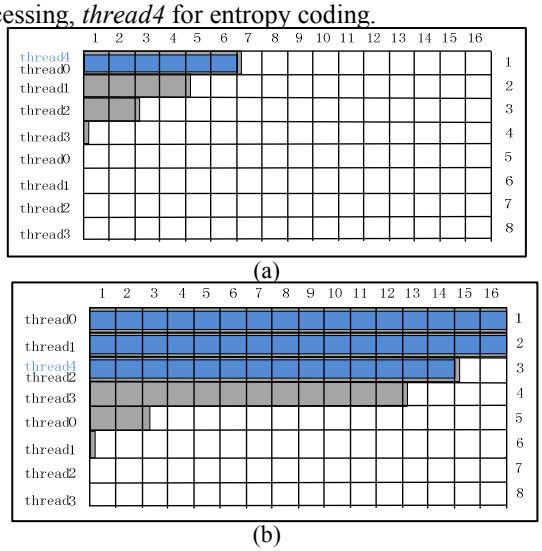

Fig.6. The proposed parallelization scheme at different moment: (a) thread3 starts processing, (b) thread1 starts processing 6th CTU row.

5. EXPERIMENTAL RESULTS

We implemented our algorithm on x265[12], an open source HEVC encoder with same ambition as famous x264 - achieving dramatic performance in realtime on a single consumer-level computer. Its current version is only a highly simplified encoder compared to the full-featured HM reference implementation. For example, SAO, Tile and non-square PU is not supported yet. However, for intra coding, x265 does support most of the HEVC key features, so we choose it as our benchmark to develop more advanced encoder.

Firstly, we provide a performance comparison between x265 and HM9.0 at intra-only configuration to give a glance at the current x265 progress. The encoder is configured as follows: all intra, CTU size 32x32, maximum allowed split depth 3, QP=22, 27, 32, 37, no fast split decision algorithm or fast intra mode selection algorithm enabled. The benchmark is x265 default encoder which runs at single thread. All tests run on a HP workstation which contains 8 cores: Intel(R) Xeon(R) CPU E5-2670@2.6 GHz, with 24GB memory, 64bit Windows7 operating system. It should be noted that, currently x265 uses prediction error rather than reconstruction error for RD cost estimation and no context is maintained in this estimation process, so there is no performance loss for our scheme compared to the single-threading one.

5.1. Performance comparison between x265 and HM9.0 at intra-only encoding

In this experiment, Both HM9.0 and x265 encode 100 frames with sequence *Cactus_1920x1080* under the same configuration: intra_main, SAO off, QP 22, 27, 32, 37. From the results shown in Fig.8 and Table 1, we can see, roughly x265 causes roughly 0.3dB PSNR loss under the same bitrate compared to HM9.0, but achieves 11.86 times speedup on average.

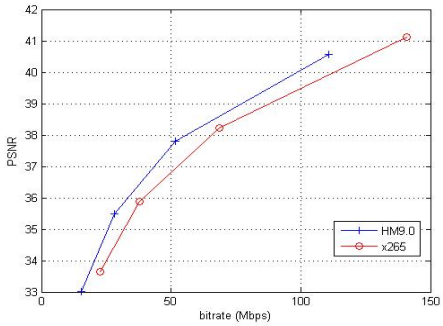

Fig.7. RD curve of HM9.0 and x265

Table 1. Performance comparison of HM9.0 and x265

	HM 9.0			x265		
QP	PSNR	Bitrate	Time(s)	PSNR	Bitrate	Time(s)
22	40.5579	110.650	2351.62	41.117	140.549	159.44
27	37.8080	51.628	1828.74	38.219	68.101	150.36
32	35.4934	28.224	1559.86	35.884	37.436	146.42
37	33.0021	15.391	1413.98	33.640	22.295	143.23

5.2. Speedup gain with thread numbers

This experiment studies relationship between speedup ratio and thread numbers. Intuitively, adding a thread could accelerate the program execution. But no application can be accelerated limitlessly. When thread number exceeds a specific threshold, instead of promoting the speed, it is even harmful. Fig.9 shows this tendency. It depicts the encoding times of a *832x480* video with different number of threads.

The speedup gain is calculated through

$$\frac{T_{anchor} - T_{proposed}}{T_{proposed}} \times 100\%$$

where T_{anchor} and $T_{proposed}$ are the encoding time of the anchor and proposed method. From the figure we can see, the speedup gain is 80% when two threads are used. The gain becomes larger as thread number increases, and reaches maximum value 398% at 7 threads. If thread number continues growing, the benefit shrinks, eventually maintains at a roughly constant value. The reason is too many threads lead to additional latency for bottom CTU rows.

5.3. Speedup gain of videos with different resolutions

High-level parallelization tools, like WPP, tiles, as well as the one proposed in this paper, are useful with large resolutions. At small image sizes the parallelization

overhead might be too high for there to be any meaningful benefit. For this reason, two classes of video sequences with resolutions *1920x1080* and *2560x1600* were used in this experiment. By experiments, we found that fixed 16 threads could get fair speedup gain for these high resolution sequences. Table 2 exhibits the outcomes. Clearly, speed boosts for all video sequences are significant, with an average of 5 times faster, and the gain is larger as resolution increases.

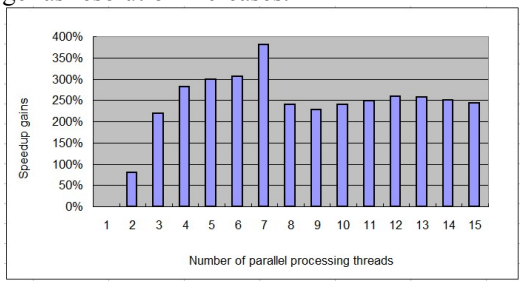

Fig.8. Speedup gains with thread numbers

Table 2. Speedup gains on different video sequences

sequences	QP	Anchor (s)	Proposed (s)	gain
PeopleOnStreet_2560x1600_30	22	384.212	61.729	522%
	27	375.194	58.203	545%
	32	367.106	57.268	541%
	37	366.805	57.205	541%
Traffic_2560x1600_30	22	386.036	58.391	586%
	27	374.760	57.704	549%
	32	370.128	57.642	542%
	37	372.294	57.143	512%
BQTerrace_1920x1080_60	22	822.581	156.312	426%
	27	806.491	132.708	508%
	32	782.312	131.881	493%
	37	757.416	131.694	475%
Cactus_1920x1080_50	22	646.017	127.295	407%
	27	644.591	109.981	486%
	32	638.755	110.055	480%
	37	632.842	110.055	475%
ParkScene_1920x1080_24	22	327.787	55.972	486%
	27	310.908	52.728	490%
	32	310.066	52.759	488%
	37	304.854	52.759	478%
Average				502%

6. CONCLUSION

This paper gives a detailed analyses of the fine granularity parallelism exists in HEVC intra coding. As a preliminary result, we show the speedup gain by exploiting the CTU-level parallelism. If inner-CTU parallelism is also utilized, current speed can be doubled as the parallelism in two levels will be magnified by multiplication. This will be part of our future work.

REFERENCES

[1] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 9," ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1003, October 2012.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Emerging Research and Standards in Next Generation Video Coding, to appear 2012.12.

[3] F. Boss, B. Bross, K. Suhring, D. Flynn, "HEVC complexity and implementation analysis", IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Emerging Research and Standards in Next Generation Video Coding, to appear Dec.2012.

[4] J. Lainema, F. Bossen, W-J. Han, J. Min, K. Ugur, "Intra coding of the HEVC standard", IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on the Emerging Research and Standards in Next Generation Video Coding, to appear Dec. 2012.

[5] C-M. Fu, E. Alshina, A. Alshin, Y-W. Huang, "Sample adaptive offset in the HEVC standard", IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on the Emerging Research and Standards in Next Generation Video Coding, to appear Dec.2012.

[6] Fuldseth, M. Horowitz, S. Xu, M. Zhou, "Tiles", ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-E408, March 2011.

[7] F. Henry, S. Pateux, "Wavefront parallel processing" ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-E196, March 2011.

[8] V. Jerome, T. Jean-Marc, "On tiles and wavefront tools for parallelism" ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-I0198, April 2012.

[9] C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, "Parallel Scalability and efficiency of HEVC parallelization approaches", IEEE Transactions on Circuits and Systems for Video Technology, Special Issue on Emerging Research and Standards in Next Generation Video Coding, to appear Dec. 2012.

[10] M. Alvarez-Mesa, C. Chi, B. Juurlink, V. George, "Parallel video decoding in the emerging HEVC Standard", in Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2012), Kyoto, Japan, March 2012.

[11] N. Cheung, O. Au, M. Kung, "Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors", IEEE Transactions on Circuits and Systems for Video Technology, vol. 19, pp. 1692-1073, Nov. 2009.

[12] x265 project, http://code.google.com/p/x265/