Fine Granular Parallel Algorithm for HEVC Encoding Based on Multicore Platform

Yi Li[3(✉)], Dong Hu[1,2,3(✉)], Chuanwei Yin[3], and Yingcan Qiu[3]

[1] Education Ministry's Key Lab of Broadband Wireless Communication and Sensor Network Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[2] Education Ministry's Engineering Research Center of Ubiquitous Network and Health Service, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
[3] Jiangsu Province's Key Lab of Image Procession and Image Communications, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
{1217012311,hud}@njupt.edu.cn

Abstract. Compared with the previous standards, the coding efficiency and complexity of High Efficiency Video Coding (HEVC) have been greatly improved. Parallel encoding scheme based on CTU rows like wavefront parallel processing (WPP) and inter-frame wavefront (IFW) can efficiently reduce the encoding time of HEVC. However, due to the coding complexity of CTU within various rows may be quite different, WPP and IFW have the problem of unbalanced load among threads for parallel encoding tasks. To address this issue, in this paper, factors affecting coding efficiency are found by analyzing the data dependence and load relationship of intra- and inter-frame CTUs, and we propose a fine granular parallel strategy accordingly. In the meanwhile, refine the parallel granularity while maintaining the accuracy of symbol prediction requires additional context information in CABAC encoding, which leads to higher bit rate, and will reduce the efficiency of CABAC encoding. In order to decrease the bit rate without affecting the quality, we also making some modifications for the CABAC encoding. The proposed method is implemented on the Tilera-GX36 multicore platform. Experiment results show that our algorithm achieves up to 1.6 and 2.8 times speedup improvement compared with IFW and WPP respectively.

Keywords: HEVC encoding · CTU · IFW · WPP · CABAC · Multicore platform

1 Introduction

In order to achieve a range of 50% bit-rate reduction for equal perceptual video quality [1] and about 40% bit-rate reductions at similar PSNR [2] compared to its previous standard–H.264/AVC, some new features have been introduced into HEVC, which also bring great increments of computational complexity [3]. HEVC provides several parallel schemes to cope with the high-speed processing demands, which make parallel coding on multicore platform possible. A representative method is slice-based

Y. Zhao et al. (Eds.): ICIG 2019, LNCS 11903, pp. 59–69, 2019.
https://doi.org/10.1007/978-3-030-34113-8_6

parallelism, which has been introduced in H.264/AVC [4]. Two new parallel methods are introduced in HEVC, which are tiles [5] and wavefront parallel processing (WPP) [6]. Tiles splits a picture horizontally and vertically into multiple rectangular regions for parallel processing, which is removing the parsing and prediction dependency. WPP splits a picture into coding tree unit (CTU) rows, and processes these rows in parallel.

Several related works focus on improving parallelism of HEVC based on WPP have been proposed, Chi et al. [7] presented an approach called overlapped wavefront (OWF) to address ramping inefficiencies in WPP for HEVC decoding. Chen et al. [8] proposed a novel parallel encoding scheme called Inter-frame wavefront (IFW) using the dependence of inter-frame CTU, realizing the multi-frame parallel encoding. Although these methods well exploit parallelism in HEVC, since the coding complexity of CTUs within various CTU rows may be quite different, WPP and IFW have the problem of unbalanced load among threads for parallel encoding tasks. To handle this problem, in this paper, we focus on exploring the data dependence and load balancing between different CTUs within intra- and inter-frame, and propose a fine granular parallel encoding based on the exploration. To avoid the possible loss in coding efficiency, CABAC module is modified accordingly. Therefore, better parallel speedup ratio can be achieved while the image quality can be guaranteed.

The rest of this paper is organized as follows. Section 2 introduces the CTU row-level parallel schemes and analyzes its parallelism. In Sect. 3, our new fine granular parallel method is described in details. We evaluate performance of the proposed algorithm in Sect. 4, followed by a short conclusion in Sect. 5.

2 Related Works

In this section, the CTU data dependency and parallelism of CTU row-level parallel schemes are introduced and analyzed. In order to denote the dependencies between CTUs, $C_{i,j,k}$ is used here to represent the k-th CTU of the j-th CTU row in the i-th frame in the coding order, $\mathrm{Dep}_{IFW,inter}(C_{i,j,k})$ and $\mathrm{Dep}_{IFW,intra}(C_{i,j,k})$ is used to represent the CTU on which $C_{i,j,k}$ depends performing parallel inter- and intra-frame encoding using IFW respectively.

2.1 Analysis of CTU Data Dependency

The intra-frame CTU dependency of IFW is same as WPP, so that the current CTU will rely on the information of CTUs in its top, top right, top left and left directions, as is illustrated in Fig. 1, the specific dependence can be expressed as (1). The encoding of each CTU rows start only after coding from the second CTU of the previous row. As is indicated in Fig. 2, a picture is separated to multiple CTU row partitions, each CTU row is allocated to different threads for parallel processing.

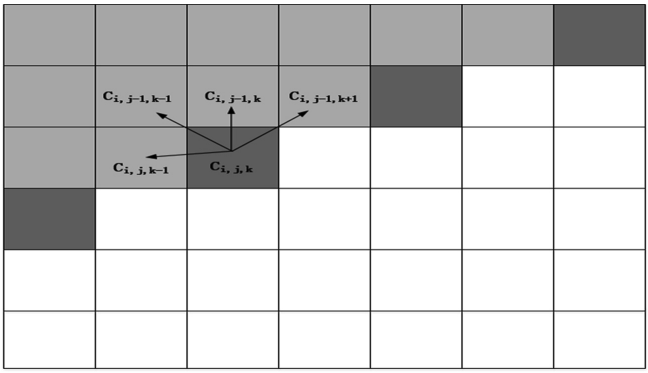

Fig. 1. The CTU data dependency of current CTU within a frame

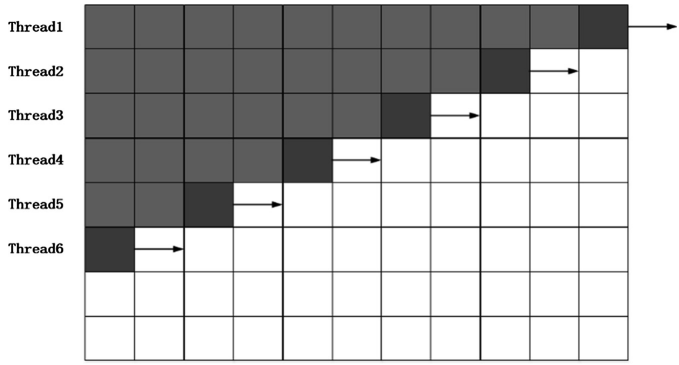

Fig. 2. Parallel processing of intra-frame in IFW

$$\mathrm{Dep}_{IFW,intra}\left(C_{i,j,k}\right) = \mathrm{Dep}_{wpp}\left(C_{i,j,k}\right) = \left\{C_{i,j-1,k-1}, C_{i,j-1,k}, C_{i,j-1,k+1}, C_{i,j,k-1,}\right\} \quad (1)$$

The motion estimation is to find the best matching pixel block in the search range and get the best motion vector. All CTUs in the search range corresponding to the current coding CTU in the reference frame must be encoded before the optimal motion vector of the current CTU can be determined. The dependency relationship of inter-frame CTU in IFW is given by (2) and Fig. 3 shows the detail.

$$\mathrm{Dep}_{IFW,inter}\left(C_{i,j,k}\right) = \left\{C_{i1,j1,k1} | \mathrm{i1} \in \mathrm{ref(I)}, 0 \leq \mathrm{j1} < j + l_h, 0 \leq k1 < l_w\right\} \quad (2)$$

l_h indicates the downward vertical component of the motion vector, it is usually set to 0 or 1 in IFW. l_w indicates the horizontal component of motion vector. The encoding process of $C_{i,j,k}$ will be started immediately after the encoding of $C_{i1,j1,k1}$ is completed,

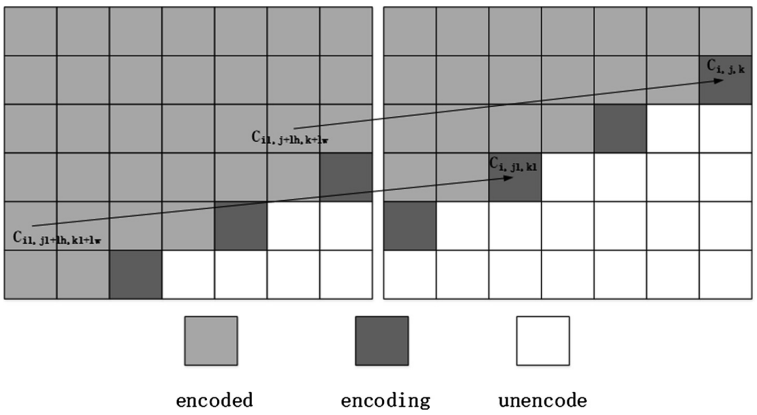

Fig. 3. Dependency of inter-frame CTU in IFW

instead of frame-by-frame CTU row-level parallel encoding in WPP. That is, CTUs in multiple frames can be encoded parallel after the dependencies are satisfied.

2.2 Parallelism of CTU Row-Level Parallel Schemes

Both the IFW and WPP using the CTU row as parallel unit. As shown in the Fig. 4, the dependencies between CTUs are represented by a directed acyclic graph (DAG), there are 4 × 5 CTUs in this example, and each CTU row is assigned a thread. It can be seen that most of the time between threads is idle, so the scheme using CTU rows as parallel unit wastes thread resources. Therefore, we propose a parallel scheme based on CTU. In this paper, using parallelism (PL) to evaluate the improvement of coding speed by parallel schemes and thread resource strategies. As shown in (3).

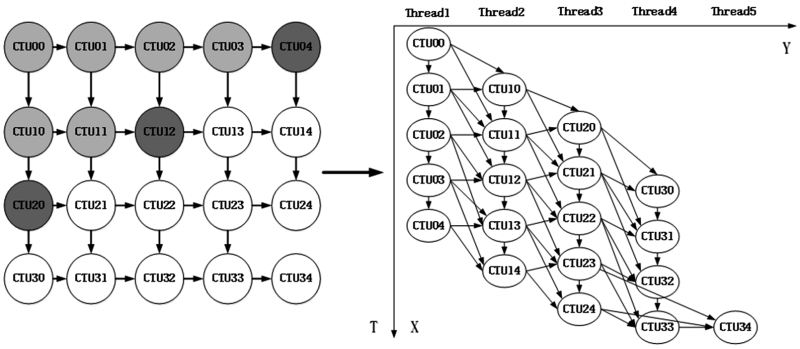

Fig. 4. DAG used to express the dependency between CTUs

$$\mathrm{PL} = \frac{EncTime_{1,F}}{EncTime_{N,F}} \tag{3}$$

$EncTime_{N,F}$ refers to the time spent using N cores to encode in the parallel algorithm F. We dividing a frame horizontally and vertically into (W × H) CTUs, assuming that the average encoding time of each CTU is a unit of time. According to the Fig. 4, there will be n units of time parallel processing when the thread resources are sufficient, n is given as (4).

$$\mathrm{n} = 2 \times (\mathrm{H} - 1) + \mathrm{W} \tag{4}$$

According to the relationship between the number of thread and CTU, we designing the scheduling strategies in two cases. Firstly, if the threads resources are sufficient, the H-th row begins to encode while the first row of CTU has not encoded yet, indicating that each CTU row has a designated thread to process. Therefore, when a thread encoding task is completed, the next frame will not be read due to the frame-level task has not been completed, so the core thread is not fully utilized and the parallelism is relatively low. (5) can express the PL.

$$\mathrm{PL} = \mathrm{W} \times H/(2H + W - 2) \tag{5}$$

Secondly, if the threads resources are limited, in this case, $(2\mathrm{H} - 2\mathrm{X} - 2) < \mathrm{W} < (2\mathrm{H} - 2\mathrm{X})$, that is there are still X CTU rows that remain to be encoded when the first row encoding has been completed. As long as there are threads in the H − X CTU row threads has completed the corresponding row encoding, these threads will be transferred to deal with the remaining X rows, where the X threads are reused, the remaining H − X threads are not scheduled, so it is a waste of thread resources. (6) give the PL:

$$\mathrm{PL} = \mathrm{W} \times H/(2W + 2X - 2) \tag{6}$$

2.3 Existing Problems

The possibility of load unbalance among CTU row threads exists in CTU row-level parallel schemes, like WPP and IFW, since the encoding complexity of CTUs within various CTU rows may be quite different. When these encoding threads are scheduled with each CTU row as the basic processing task, the encoding complexity of a CTU in the previous row is too high, which will lead to the blocking of the coding threads in the subsequent CTU rows, due to the data dependence between CTUs. This will cause considerable latency among encoding threads. Moreover, according to the two cases discussed in part 2.2, the parallelism are relatively low and thread resources has not be fully utilized. Discussion of more details and our method are in the Sect. 3.

3 Proposed Method

3.1 Analysis of CTU Load Balancing

As is shown in Fig. 5, suppose there are only two threads currently used to encode CTU03 and CTU11 respectively. Because the complexity of CTU03 is higher than CTU11, as a result, the second thread is blocked in CTU11, making it impossible to code CTU12. If using the CTU rows as parallel granularity and the threads are allocated to encode the CTU rows in parallel, the load of the coding threads will be unbalanced, which will lead to the blocking of the coding threads. Dependency DAG and (First in First out) FIFO queue are introduced to solve this issue. When the second thread is blocked, the coding condition of CTU20 has met in the meanwhile, thus, the second thread can be used to encode CTU20, thereby reducing the coding delay caused by load unbalance of encoding based on CTU rows. Similarly, if the CTU20 has a

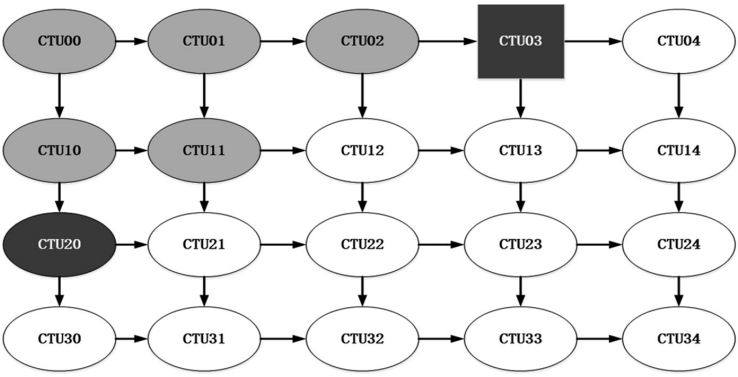

Fig. 5. Analysis of CTUs load balancing

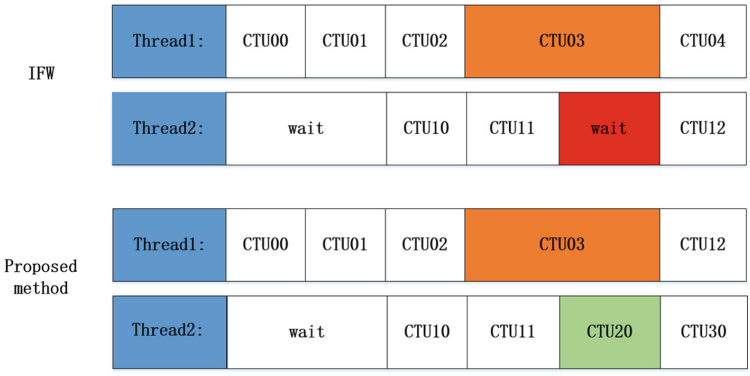

Fig. 6. The actual threads load

larger complexity, the thread that complete tasks first can be used to encode the CTU, which has no dependency in the queue. A main thread operates on the queue to schedule the threads pool; the DAG dependency matrix will be updated when a CTU encoding is completed. Figure 6 shows the actual thread load of IFW and our algorithm.

3.2 Specific Proposed Method

Threads scheduling strategy is described by adjacency matrix and in-degree matrix to solve the problem of CTU load unbalance. The dependency is expressed by the in-degree matrix, and the matrix will be updated every time a CTU has been encoded, when the dependencies of CTU are satisfied, idle threads can be invoked to realize multi-frame parallel encoding, thus solving the problem of thread blocking and low parallelism of frame-by-frame coding. DAG represents the dependency between CTUs, m is the serial number of CTUs, M is the number of CTUs within a frame, floor and mod are representing downward integral and modular function respectively.

The location of each CTU is represented by (i, j), that is:

$$\mathrm{i} = \mathrm{floor}\left(\frac{m}{M}\right) j = \mathrm{m} \bmod \mathrm{M} \tag{7}$$

For $\mathrm{DAG} = (\mathrm{V}, \mathrm{E})$, if the set of boundaries E contains $\left(V_{i,j}, V_{m,n}\right)$, indicating that the CTU (m, n) depends on the CTU (i, j). When the CTU represented by $V_{i,j}$ is encoded, removing the boundaries containing $V_{i,j}$ from DAG. The entry of CTU, which has a boundary relationship with $V_{i,j}$, will be reduced by 1, and indicates a dependency is satisfied. When the entry of vertices becomes 0, the corresponding vertices can be processed in parallel. Therefore, the DAG and the in-degree matrix will be updated each time a CTU is encoded. Since the entrance values of different CTUs in the in-degree matrix are obtained from the DAG, the adjacency matrix is used to represent the DAG, and W and H are used to represent the number of CTUs in the horizontal and vertical directions of a frame respectively. The entry value D of each CTU is obtained by adjacent matrix A:

$$A_{(i,j),(m,n)} = \begin{cases} 1, \left(v_{i,j}, v_{m,n}\right) \in E \\ 0, else \end{cases} \quad 1 \le i, m \le H, 1 \le j, n \le W \tag{8}$$

$$D_{m,n} = \sum_{i=1}^{H} \sum_{j=1}^{w} A(i,j),(m,n) \quad 1 \le m \le H, 1 \le n \le W \tag{9}$$

Parallel granularity refinement makes CABAC encoding require additional context information if symbol prediction accuracy is to be maintained, which leads to higher bit rate. In order to reduce the bit rate, we divide the coding process at CTU level into two consecutive modules. The first module makes intra- and inter-frame prediction at CTU level according to the rate-distortion cost, and obtains the best partition and prediction mode of CTU. The second module to encode CTU according to the partition and prediction mode information. In the optimal mode decision-making stage, the best

partition and optimal mode are obtained by parallel processing of multiple threads, the syntax information, however, will not be encoded by CABAC immediately, but will continue the encoding process of other CTUs. Independent CABAC encoding threads will encode the syntax elements of this optimal information. Consequently, the proposed scheme can be divided into two task levels, frame-level and CTU-level, as shown in Fig. 7. Frame-level tasks request threads to process the CTU-level task queue of the current frame for parallel processing. When the encoding process of a frame is finished, the thread resource for this frame is released and the code stream after CABAC is stored

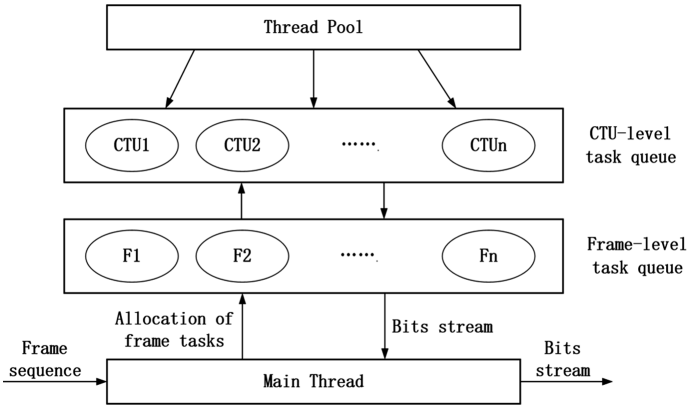

Fig. 7. Two task levels of the proposed scheme

4 Experiment

4.1 Experiments Design

The algorithm is implemented on the Tilera-GX36 multicore platform, and using x.265 as the reference software. Two sets of comparison experiments are set up in the meanwhile. The first one is the WPP that does not take into account the inter-frame correlation, and the second one is IFW, which takes into account the inter-frame correlation. Since the experiment involves the dependency of the CTU in the time domain, we dividing the coding frame type into two cases: IPPPP and IBBBP. The indicators used for evaluating the experimental performance are bit rate, PSNR, and parallel acceleration ratio. With the increase in the number of threads, the trend of the parallel acceleration of the three algorithms is compared. The size of LCU is 32×32 in the experiment, and the video sequence used in our experiment are 1600p Traffic, 1080p Kimono and 720p FourPeople. Parallel acceleration ratio is used to express the acceleration of parallel algorithm relative to serial encoding, which is represented by speedup.

$$\text{Speedup} = \frac{EncTime_{serial}}{EncTime_{parallel}} \tag{10}$$

$EncTime_{serial}$ is the time required for single-core serial encoding and $EncTime_{parallel}$ is the time spend on multi-core parallel encoding.

4.2 Results and Discussion

Experimental results in Table 1 are obtained with the thread number is 32. It can be found from these data that compared with the two comparison algorithms, the PSNR of our method decreases and the parallelism degree improves. Furthermore, compared with the WPP, the IFW scheme also increases the bit rate slightly, because the improvement of the algorithm increases the syntax elements, but compared with IFW, our parallel algorithm reduces the bit rate due to the addition of the CABAC optimization scheme. The parallel acceleration ratio of our algorithm is much higher, which is achieves up to 1.6 and 2.8 times speedup improvement compared with IFW and WPP respectively for 720P FourPeople test sequence (Figs. 8, 9 and 10).

Table 1. Comparison results of three algorithms

Frame type	Sequences	WPP			IFW			Our method		
		PSNR (dB)	Bit rate (kbps)	Speed up	PSNR (dB)	Bit rate (kbps)	Speed up	PSNR (dB)	Bit rate (kbps)	Speed up
IPPPP	Traffic	31.422	3111.93	3.56	31.235	3286.68	6.31	31.081	3231.24	9.42
	Kimono	33.086	3098.74	3.37	32.863	3178.55	6.14	32.673	3145.16	9.13
	FourPeople	23.167	712.09	2.92	22.841	796.34	5.69	22.682	764.24	8.24
IBBBP	Traffic	32.087	3243.55	5.45	31.765	3395.56	7.43	31.584	3326.13	9.85
	Kimono	35.126	3122.88	4.39	34.839	3209.08	6.98	34.691	3186.24	9.61
	FourPeople	26.253	792.03	3.09	26.033	834.11	5.03	25.876	821.67	8.22

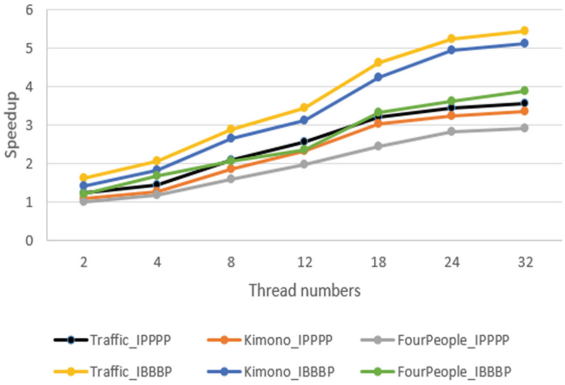

Fig. 8. Parallel acceleration ratio of WPP

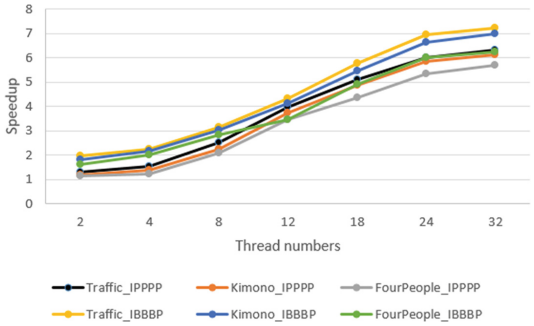

Fig. 9. Parallel acceleration ratio of IFW

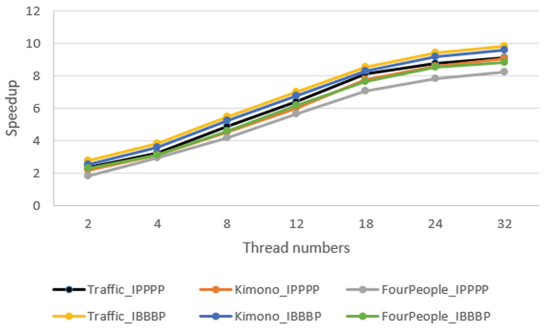

Fig. 10. Parallel acceleration ratio of proposed method

It can be found from these line charts that the parallel acceleration ratio for three methods increase gradually with the increasing thread numbers, the addition of B frames can improve the parallel acceleration ratio, because B frame is bidirectional predictive frame, which can enhance the efficiency of video sequence compression.

5 Conclusion

In this paper, we introducing the parallel algorithms based on CTU row firstly. Then, the dependency of CTUs on intra- and inter-frame and the load balancing among threads of CTU row-level parallel schemes are analyzed. A fine granular parallel coding model is established for the dependency relationship between CTUs, in-degree matrix and adjacent matrix are used to represent the update of dependency. Finally, we implementing the proposed method on the Tilera-GX36 multi-core platform. Experimental results show that the parallel acceleration ratio is further improved by our method.

References

1. Sullivan, G.J., et al.: Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1649–1668 (2013)
2. Ohm, J., et al.: Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC). IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1669–1684 (2012)
3. Bossen, F., et al.: HEVC complexity and implementation analysis. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1685–1696 (2012)
4. Zhao, L., et al.: A dynamic slice control scheme for slice-parallel video encoding. In: IEEE 19th International Conference on Image Processing 2012, pp. 713–716. IEEE, Florida (2012)
5. Baik, H., Song, H.: A complexity-based adaptive tile-partitioning algorithm for HEVC decoder parallelization. In: IEEE International Conference on Image Processing 2015, pp. 4298–4302. IEEE, Quebec (2015)
6. Radicke, S., et al.: A multi-threaded full-feature HEVC encoder based on wavefront parallel processing. In: 11th International Conference on Signal Processing and Multimedia Applications 2014, pp. 90–98. IEEE, Vienna (2014)
7. Chi, C.C., et al.: Improving the parallelization efficiency of HEVC decoding. In: IEEE 19th International Conference on Image Processing 2012, pp. 213–216. IEEE, Florida (2012)
8. Chen, K., et al.: A novel wavefront-based high parallel solution for HEVC. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 181–194 (2016)