A TWO STAGE PARALLEL ENCODER SCHEME FOR REAL TIME VIDEO ENCODER

Shengfu Dong[1], *Zhihang Wang*[2], *Ronggang Wang*[1], *Zhenyu Wang*[1], *and Wen Gao*[2], *Fellow, IEEE*

[1]School of Computer & Information Engineering, Peking University Shenzhen Graduate School, Shenzhen 518055, China (e-mail: {dongsf, rgwang, wangzhenyu}@pkusz.edu.cn)
[2]School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: {zhihangwang, wgao}@jdl.ac.cn)

ABSTRACT

With the development of hardware technologies, multi-core processor becomes more and more popular. At the same time, the performance of video encoding technologies is greatly improved at the cost of complexity. How to exploit the computing capacity of the multi-core processors efficiently to meet the demand of video encoding technology is a challenging task. Traditional work usually split the frame into slices to enable parallel encoding. But the coding efficiency will lose especially when there are more slices. In this paper, first we propose a two stage real time encoding scheme without splitting the frame into slices. Secondly, some modules in the hybrid video coding scheme, such as interpolation and loop filter, are split into independent modules that can be processed in parallel. A simple and effective task scheduling strategy is also proposed to fully exploit the CPU capacity. Finally, fast algorithms suitable for this architecture are developed. Experimental results show that we can encode the 720p sequence at about 50 fps with little loss in the rate distortion performance.

***Index Terms*—**AVS, Real time video encoder, Fast mode decision, Parallel encoding

1. INTRODUCTION

With the development of hardware development of hardware technologies, multi-core processor becomes more and more popular. Intel has release quad-core, six-core and ten-core processors [1]. Tilera released their multi-core processor with 64 cores [2]. These new hardware technologies provide us great computing capacity. On the other hand, the performance of video encoding technologies, such as H.264/AVC [3] and AVS [4], are greatly improved at the cost of complexity. How to exploit the computing capacity of the multi-core processors efficiently to meet the demand of the new video encoding technology is a challenging task.

Traditionally there are four parallelism schemes used in the real time encoder. The first one is GOP (group of picture) level parallelism architecture [5]. This method is very simple and effective. But it will cause high latency. The second one is frame level parallelism [6]. The B frame is regarded as the independent frame. Thus it can be parallel encoded with I and/or P frames. But the B frame can be used as the reference frame in H.264/AVC. If there is no B frame in the sequence such as the baseline profile of H.264/AVC, there are no independent frames. The most commonly used architecture is the slice level parallel encoding architecture [7]. Each frame is split into many slices. Because every slice is independent, they can be encoded in parallel. The main drawback of this scheme is the degradation in the coding efficiency. When the slice number is up to 9 or 18, the degradation in the performance is up to 1-2dB [6]. Some researchers merged the GOP level and slice level parallel architecture to get a good trade off between the speed and coding efficiency [8]. The last one is the macroblock level parallel scheme [6] [9]. Considering the interdependencies among the macroblocks, the encoding task is done diagonally [6]. The main advantage of this scheme is that there is almost no degradation in the coding performance. But the implementation of this scheme is more complex than the others.

In this paper, we propose another macroblock level parallel encoding scheme. The macroblocks are encoded row by row. Because of the interdependencies among the macroblock rows, some threads have to wait until all the dependent encoding threads finish their task. In order to fully exploit the CPU computing capacity, the two stage architecture is proposed. A fast mode decision algorithm is also developed based on this architecture.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the AVS coding standard. In Section 3, we develop a two stage parallel encoding scheme suitable for multi-core processors. The modules of AVS are split row by row in Section 4. A simple and effective scheduling strategy is proposed in Section 5. In Section 6, the fast mode decision algorithm based on this architecture is proposed. Experimental results are given in Section 7. Conclusions are drawn in Section 8.

2. BRIEF OVERVIEW OF THE AVS STANDARD

978-1-4673-2193-8/12/$31.00 ©2012 IEEE

The AVS provides similar compression efficiency to H.264/AVC [3]. Similar to H.264/AVC and MPEG-2 [10], AVS adopts the hybrid video coding framework. The intra prediction technology is used in AVS to improve the intra frame compression efficiency. There are total 5 prediction modes for the luminance component and 4 prediction modes for the chroma component.

There are 5 inter modes in AVS for P frame. The maximal reference number is 2 for P frame in progressive video coding mode. Besides the modes used in P frame, AVS adopts DIRECT and SYMMERY mode [11] to encode the B frame. Quarter-pixel accuracy motion vectors are used. The prediction motion vector of current macroblock is predicted by the motion vector of left, up and up-right macroblocks. The intra prediction and the motion vector prediction make current macroblock depend on the left, up, up-left and up-right macroblocks as show in Fig. 1. The spatial macroblock dependency is the same as that in H.264/AVC [6].

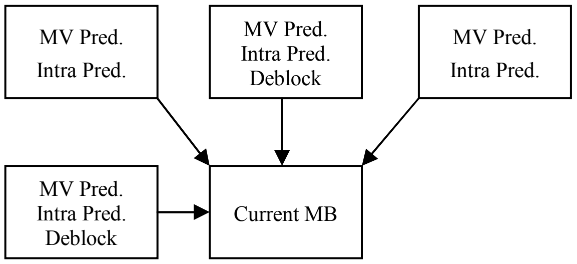


Fig. 1 Spatial data dependencies for a macroblock

3. THE SCHEME OF THE TWO STAGE ENCODER

The parallel encoding scheme composes of two level parallelisms: frame level and macroblock level. The frame level parallel structure is show in Fig. 2. There are two stages in the encoder. In the first stage, we use the original frame as the reference frame. The intra and inter prediction signal are formed based on the original frame. The SAD of every intra or inter mode are calculated as

$$SAD = \sum_{y=0}^{h} \sum_{x=0}^{w} (Org(x,y) - \hat{P}red(x,y)) \qquad (1)$$

Note that $\hat{P}red(x,y)$ is the prediction signal and formed based on the *original frame*. h and w are the height and width of the block. Detailed function about the two stage encoder is show in **Error! Reference source not found.**. The main function of the first stage is a motion estimator. In the first stage, it needn't to reconstruct the frame. DCT and quantization operation are not needed. The interpolation is also avoided to reduce the complexity in the first stage. The second stage utilizes the results of the first stage saved in the buffer to speed up the encoding process.

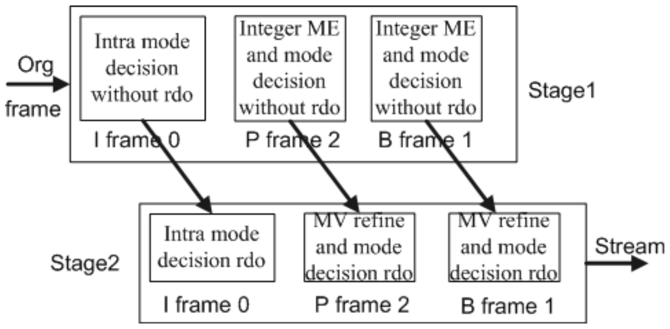

Org
frame
Intra mode decision without rdo
I frame 0
Integer ME and mode decision without rdo
P frame 2
Integer ME and mode decision without rdo
B frame 1
Stage1
Stage2
Intra mode decision rdo
I frame 0
MV refine and mode decision rdo
P frame 2
MV refine and mode decision rdo
B frame 1
Stream

Fig. 2 The two stage encoding scheme

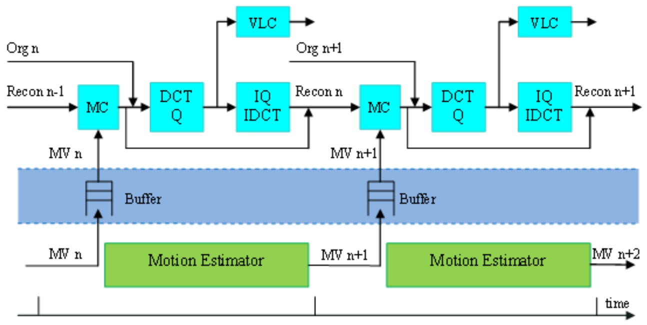
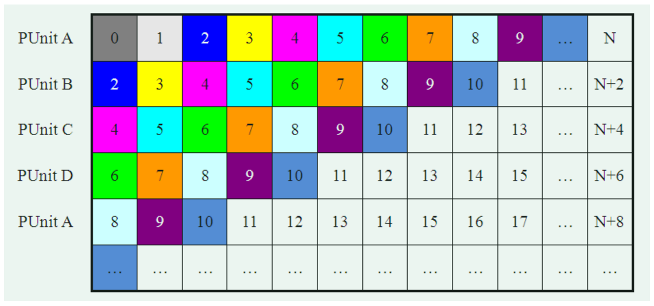


Fig. 3 Detailed structure of the two stage parallel encoder



PUnit A	0	1	2	3	4	5	6	7	8	9	…	N
PUnit B	2	3	4	5	6	7	8	9	10	11	…	N+2
PUnit C	4	5	6	7	8	9	10	11	12	13	…	N+4
PUnit D	6	7	8	9	10	11	12	13	14	15	…	N+6
PUnit A	8	9	10	11	12	13	14	15	16	17	…	N+8
	…	…	…	…	…	…	…	…	…	…	…	…

Fig. 4 Macroblock level parallel encoding scheme

Since the first stage uses the original frame as the reference frame, it can parallel encoding with the second stage. Thus the frame level parallel encoding can be achieved. This parallel scheme has no relation to the frame type. In order to alleviate the thread waiting event, the first stage encodes a GOP pictures using all the cores at beginning. Then the two stages work simultaneously.

Because of the spatial dependencies as shown in Fig. 1, the macroblock level parallel encoding is achieved as shown in Fig. 4. Every macroblock (except those in the first row) will wait until all its dependent macroblocks are finished encoding. Traditionally the frame is split into slices or split as in [6] to enable macroblock level parallel encoding, but experiment results in Section 7 show that our method can also work very well. This is a very different view point as before. The main advantage of this macroblock parallel scheme is its implementation simplicity.

4. MODULES DECOMPOSITION FOR AVS

The interpolation and loop filter operation are split to suit for the second stage encoding. They can be done row by row to take the advantage of the multi-core processor. The interpolation operation is divided into two stages: half-pixel interpolation and quarter-pixel interpolation. The half-pixel interpolation is finished based on the integer-pixel values as shown in Fig. 6.

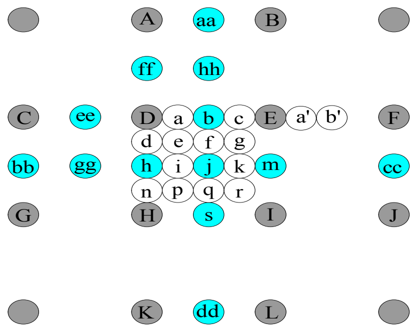


Fig. 5 The AVS interpolation operation

PADDING EDGE
INTERPOLATING ROW 0
ROW 1

Fig. 6 Macroblock level sub-pixel interpolation

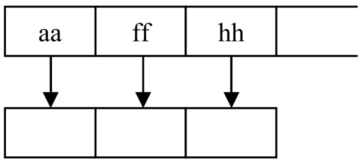


Fig. 7 The data structure for the interpolated reference frame

When the second macroblock row is encoded, we can interpolate the half-pixel values of the first row as shown in Fig. 6.

After encoding the reference frame, we can get the reconstruction frame with half-pixel values. All the quarter-pixel values can be interpolated row by row simultaneously because they only depend on the integer and half-pixel values.

In order to use SIMD (Single Instruction Multiple Data) instructions, we split the reference frame into 16 frames. Each frame corresponds to one position in the reference frame as shown in Fig. 7.

The loop filter operation can also be split macroblock by macroblock. When one macroblock is encoded, the left, up macroblock edge and the 8x8 block edges in this macrobloc can be filtered. One advantage of deblocking the macroblock as soon as possible is that the cache hit possibility can be improved.

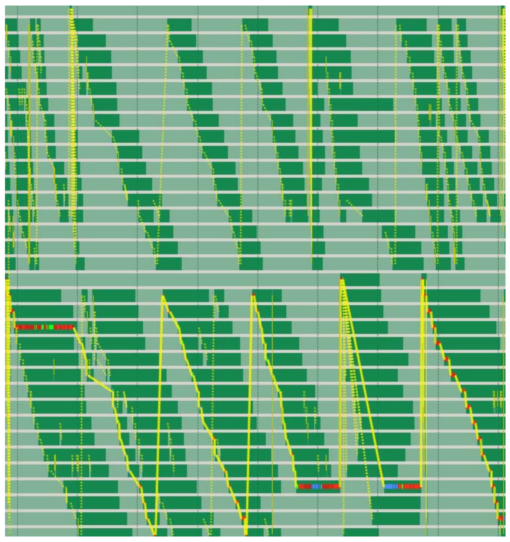


Fig. 8 Thread scheduling of the two stage scheme

5. TASK SCHEDULING STRATEGY

The dependencies among the macroblocks are not removed in our scheme. So the thread has to wait for the other dependent threads. Suppose the thread number in the second stage is T. The macroblock number in one row is M. The percentage of the wasted CPU time is

$$\eta = \frac{T-2}{M} \times 100\%$$

Obviously, η is proportional to T and is inverse proportional to M. The half-pixel interpolation and loop filter operation are split row by row and are done after encoding one macroblock row. This is equal to enlarge M and will decrease η. For example, we set 7 threads in the first stage and 9 threads in the second stage on the 16 core server to encode the 720p sequence. The CPU usage is only between 70%-80%. Furthermore, if the fast algorithm is adopted in the second stage, this corresponds to decrease M and more CPU time will be wasted.

In order to tackle this problem, we find that if we set the same thread number in the first stage and the second stage, usually the CPU usage can be higher than 95%. Sometimes the CPU usage can be up to 100%. Fig. 8 is part of the Intel vtune thread profiler results when we encode the 720p city sequence. The above 16 threads are the first stage threads. The bottom 16 threads are the second stage threads. We allocate the encoding task of every macroblock row as follows. Thread 0 in the second stage will encode row 0, 16, 32. Thread1 will encode row 1, 17, 33. The other threads follow the same rule.

It can be seen from Fig. 8 that in the first stage is higher than in the second stage. This is because the task load in the second stage is higher than that in the first stage. When the thread in the first stage is waiting for the other thread, the corresponding core will be empty. The OS (operating system) will wake up the thread in the other stage to work. The spare time in one stage is consumed by the thread in the other stage. Thus the CPU usage is improved.

We can also see from Fig. 8 that the macroblock row waiting time is very little. This is because the encoding speed difference of adjacent macroblock rows is very small.

6. THE FAST MODE DECISION ALGORITHM

The possessing results in the first stage can be used in the second stage. Here we only propose a fast mode decision algorithm. Other preprocessing technologies can also be used in this scheme.

In the first stage, the SAD costs of all modes are saved and sorted from small to big. The best integer motion vectors of every inter mode are also saved. In the second stage, the mode decision is made according to the cost order. Only part of the mode needs to be refined. The eight points around the best integer motion vector are checked in the second stage to find the best integer motion vector again. Full sub-pixel motion estimation algorithm is used.

7. EXPERIMENTAL RESULTS

We implement our proposed scheme to encode AVS bit stream. The DCT, quantization, SAD and interpolation models are optimized by SIMD instructions. The server is DELL R900 with 4 Intel X7350 processors. There are total 16 cores with 8MB L2 cache.

7.1. The RD performance of the fast intra mode decision

Firstly, we test the RD (rate distortion) performance of the fast intra mode decision algorithm. The mode refined number for chroma is set to 2. We encode the 720p sequence setting all the frames to I frame. The QP value is set to 24, 28, 32 and 36. Rate control is disabled.

Fig. 9 to Fig. 14 show the RD performance of the fast intra chroma mode decision algorithm. Only two chroma modes with the least SAD cost gotten from the first stage are selected as the candidate modes. It can be seen that there is little loss in the performance. Then we test the fast intra luma decision algorithm. The luma mode number refined in the second stage is set to 2, 3, 4 and 5. The RD performance of the algorithm is shown in the following figures.

Fig. 15 and Fig. 16 show the RD performance of the fast intra luma mode decision algorithm. It can be seen that if the refine mode number is set to 3, the RD performance degradation is still acceptable.

7.2. The RD performance of the fast inter mode decision

In order to reduce the complexity, the motion estimation algorithm used in the first stage is set to three step motion search. The GOP structure is set to IBBP and the GOP length is set to 25. The intra chroma mode refine number is set to 2 and the intra luma mode refine number is set to 3. The inter refined mode number is set to 2, 3, 4, 5 and 6.

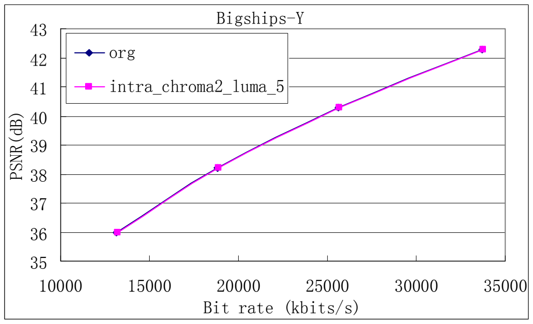


Fig. 9 The RD performance of the fast intra chroma mode decision

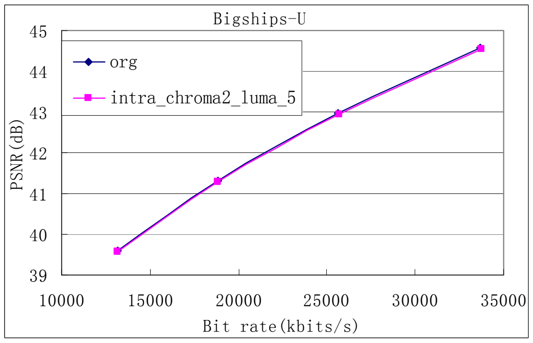


Fig. 10 The RD performance of the fast intra chroma mode decision

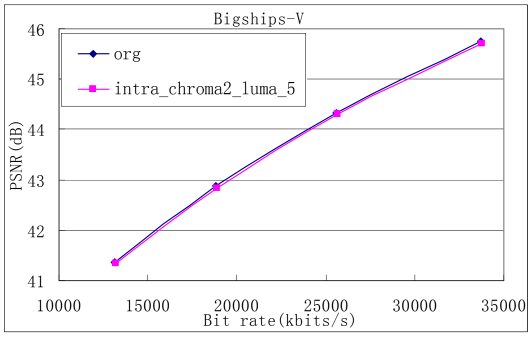


Fig. 11 The RD performance of the fast intra chroma mode decision

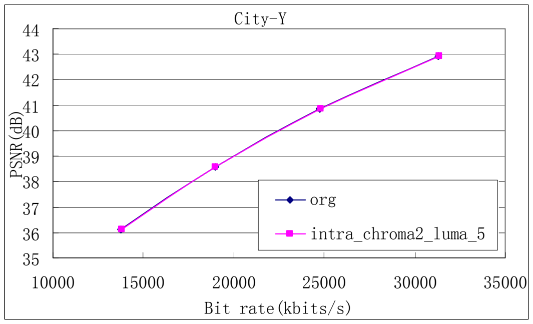


Fig. 12 The RD performance of the fast intra chroma mode decision

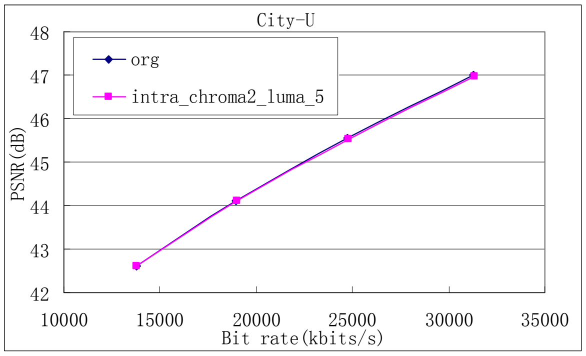


Fig. 13 The RD performance of the fast intra chroma mode decision

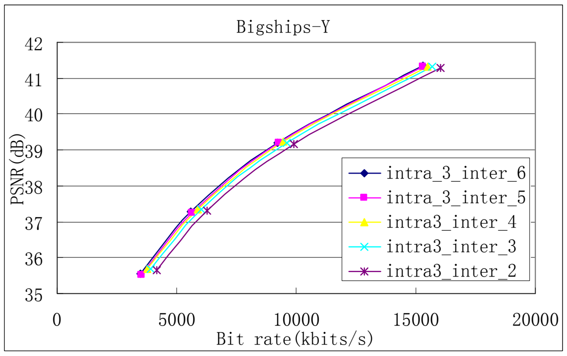


Fig. 17 The RD performance of the fast inter mode decision

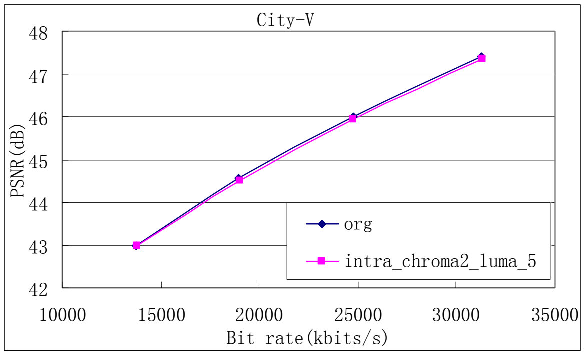


Fig. 14 The RD performance of the fast intra chroma mode decision

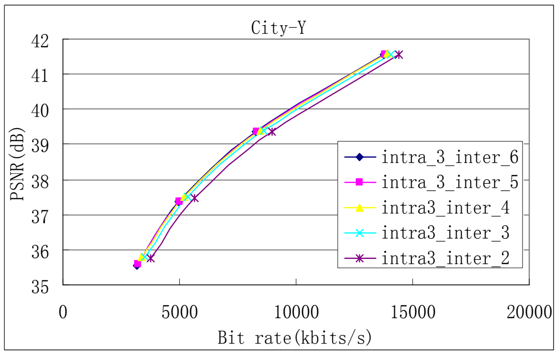


Fig. 18 The RD performance of the fast inter mode decision

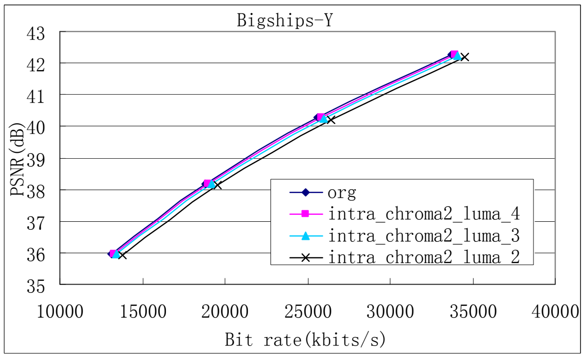


Fig. 15 The RD performance of the fast intra luma mode decision

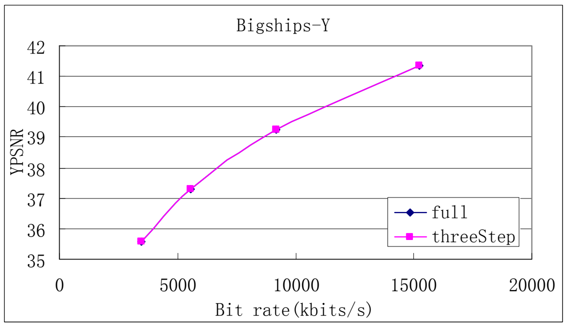


Fig. 19 The RD performance of the fast motion estimation

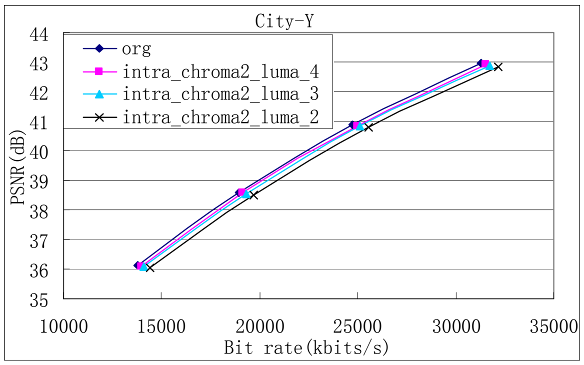


Fig. 16 The RD performance of the fast intra luma mode decision

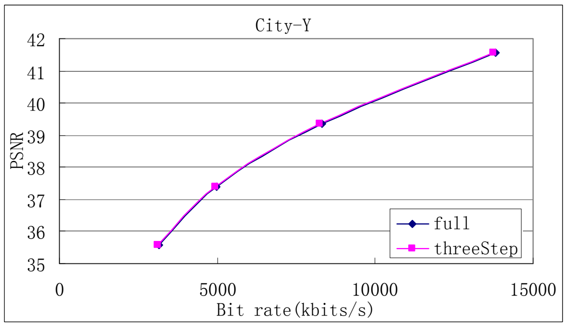


Fig. 20 The RD performance of the fast motion estimation

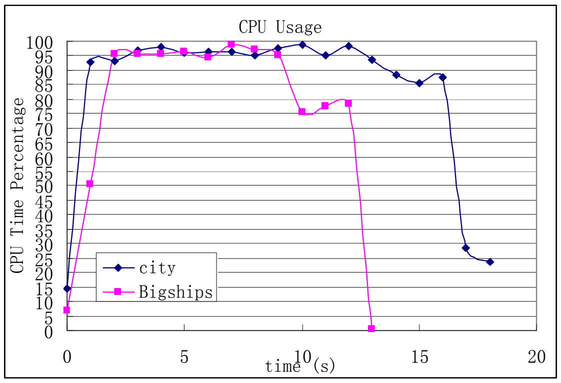


Fig. 21 The CPU usage of this scheme

Fig. 17 and Fig. 18 shows the RD performance of the fast inter mode decision algorithm. It can be seen that the inter mode refine number can be set to 3 with acceptable loss in the RD performance.

7.3. The RD performance of the fast motion estimation

The RD performance of the new three step motion estimation algorithm [12] based on this architecture is also tested. The full motion estimation algorithm is selected as the anchor algorithm. From Fig. 19 and Fig. 20, we find that loss in the performance can be neglected.

7.4. The speed of the fast mode decision algorithm

TABLE I
THE SPPED OF THE FAST MODE DECISION ALGORITHM

Sequence	QP	Org(fps)	Fast(fps)	Speedup
Bigships	24	38.6	52.0	34.72%
	28	40.0	53.1	32.75%
	32	40.8	53.6	31.37%
	36	41.0	53.0	29.27%
City	24	40.0	54.5	36.25%
	28	41.5	55.6	33.98%
	32	41.9	56.3	34.37%
	36	42.3	56.6	33.81%

Based on the RD performance of the fast mode decision algorithm, the intra luma mode refine number is set to 3. The intra chroma mode refine number is set to 2. The inter luma mode refine number is set to 3. The speed test result of the fast mode decision algorithm is shown in table I. The average speed up factor is about 35%.

7.5. The CPU usage of this scheme

We use the windows performance monitor to get the CPU usage information as shown in Fig. 21. The CPU usage is up to 95% or higher. When the sequence will be finished, the CPU usage drops down. This is because the first stage has finished its job and only the second stage is still working.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an efficient real time video encoder scheme to fully exploit the multi-core technology. The main advantage of this scheme is its effective and simplicity. No complexity modes, such as [7], are needed. No more slice is needed. A simple yet effective mode decision algorithm is developed. Based on this encoder scheme, other fast algorithms and preprocessing algorithm can also be integrated into this architecture. Thread pool technology can also be used to further release the OS scheduling burden.

9. ACKNOWLEDGE

This work was supported bythe grant of National Basic Research Program of China (973 Program) 2009CB320907, the grant of Key Projects in the National Science & Technology Pillar Program 2011BAH08B03, and the grant of Shenzhen Basic Research Program of JC201104210117A.

10. REFERENCR

[1] http://ark.intel.com/cpu.aspx?groupId=36947.

[2] http://www.tilera.com/products/processors.php.

[3] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra, “Overview of the h.264/avc video coding standard,” *Circuits and Systems for Video Technology*, IEEE Transactions on, vol. 13, no. 7, pp. 560-576, 2003.

[4] L. Yu, F. Yi, J. Dong, and C. Zhang, "Overview of AVS-video: tools, performance and complexity," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, S. Li, F. Pereira, H. Y. Shum, and A. G. Tescher, Eds., vol. 5960, July 2005, pp. 679-690.

[5] J.Fernández and M.Malumbres, "A parallel implementation of H.26L video encoder," 2002, pp. 195-208. [Online]. Available: http://dx.doi.org/10.1007/3-540-45706-2_117

[6] Y.-K. Chen, E. Q. Li, X. Zhou, and S. Ge, "Implementation of h.264 encoder and decoder on personal computers," *Journal of Visual Communication and Image Representation*, vol.17, no.2, pp. 509-532, April 2006.

[7] B. Jung and B. Jeon, "Adaptive slice-level parallelism for h.264/avc encoding using pre macroblock mode selection," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 558-572, December 2008.

[8] A. Rodriguez, A. Gonzalez, and M. P. Malumbres, "Hierarchical parallelization of an H.264/AVC video encoder," in *Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on*, 2006, pp. 363-368.

[9] Amit, G., Pinhas, A.: Real-Time H.264 Encoding by Thread-Level Parallelism: Gains and Pitfalls. *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, 2005.

[10] ISO/IEC IS 13818, General Coding of Moving Picture and Associated Audio Information, 1994.

[11] X. Ji, D. Zhao, F. Wu, Y. Lu, and W. Gao, “B-picture coding in AVS video compression standard,” *Signal Processing: Image Communication*, Vol.23, No.1, pp. 31-41, Jan. 2008.

[12] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 4, pp. 438-442, 1994.