Balanced Parallel Scheduling for Video Encoding with Adaptive GOP Structure

Hsu-Feng Hsiao, *Member*, *IEEE*, and Chen-Tsang Wu

Abstract—Due to the nature of a dynamic group of picture (GOP) structure, parallel scheduling for video encoding becomes challenging. To address this, the balanced frame-level parallel scheduling algorithms are developed. The proposed approaches first determine the frame priority and then the thread priority assignment for scheduling. The concept of the algorithms lies in the analysis of coding complexity, temporal influence, and the required temporal burden to finish coding. To complete the scheduling with the dynamic GOP structure, a block-based abrupt and gradual scene change detection algorithm is also proposed to determine the GOP structure adaptively. The experiments show that the scheduling performance is close to the optimal. In addition, the concept of batch processing is incorporated so that the required buffer can be reduced.

Index Terms—Adaptive GOP structure, parallel scheduling

1 Introduction

Attention has been paid intensively to video compression technologies due to emerging multimedia applications over computers, mobile devices, and entertainment systems that demand good video quality at increasing video resolution and acceptable bit rate. To further improve the compression ratio, many sophisticated predictive coding tools, transformation utilities, entropy coding tools, and adaptive filters are designed and implemented in the recent video compression standards such as the MPEG-4 Advanced Video Coding (AVC) standard and also in the new high-efficiency video coding that is currently under joint development by the ISO/IEC Moving Picture Experts Group and ITU-T Video Coding Experts Group. However, the price paid for the improved compression performance is heavy computation.

Each frame in a video may either represent a single event itself or belong to a part of sequential frames that constitute continuous related action, also known as a "scene." Coding performance is often influenced by the structure of group of picture (GOP) in video sequences. If predictive coding tools along temporal axis, such as motion prediction and compensation, are used on the frames across the boundary of two scenes, coding efficiency will usually suffer, and severe error drifting problem is likely to occur for video transmission over error-prone channels. In general, the GOP structure in a video shall be content dependent. For example, the GOP sizes for fast motion videos such as football game videos and slow motion videos such as surveillance videos shall be adjusted accordingly to achieve better coding efficiency. If scene change detection can be performed beforehand, a proper GOP structure can then be determined for better coding efficiency.

In addition, the increasing computation complexity for modern video compression technologies such as AVC should be addressed so that the required time for coding can be much reduced. Accelerating the coding can be achieved algorithm-wise and/or scheduling-wise. For solutions in the algorithm domain, popular methods mainly focus on reducing coding complexity at minimal loss of compression efficiency, such as fast motion estimation and fast coding mode selection [1]. In terms of scheduling, parallel processing with multicore processing units is quite feasible by distributing computation load among a number of parallel-executing processors. Several coarse-granularity parallelism methods have been proposed in the literature, such as GOP-level parallelism [2], frame-level parallelism [3], [4], slice-level parallelism [5], and MB-level parallelism [6], [7]. In this paper, a frame-level-based parallel scheduling is proposed for video encoding with adaptive GOP structure where the scheduling problem becomes more challenging.

This paper is organized as follows: In Section 2, we briefly discuss the related work of scene change detection and parallel processing for video encoding. In Section 3, the proposed scene change detection algorithm and balanced frame-level parallel scheduling algorithms for video encoding are presented in detail. In Section 4, the experimental results and the corresponding discussion are described, followed by the conclusion in Section 5.

2 Related Work

The main objective of parallel scheduling for video coding is to distribute computation load among multiple cores, while scene change detection can be beneficial to video coding. In this section, related work about these topics will be briefly introduced.

2.1 Scene Change Detection

Scene changes can be divided into two categories: abrupt scene changes and gradual scene changes. An abrupt scene change indicates that transition from one scene into another

• *The authors are with the Department of Computer Science, National Chiao Tung University, 1001 University Road, Hsinchu 30010, Taiwan.*

Manuscript received 27 Oct. 2011; revised 4 Oct. 2012; accepted 24 Dec. 2012; published online 10 Jan. 2013.
Recommended for acceptance by I. Ahmad.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2011-10-0797.
Digital Object Identifier no. 10.1109/TPDS.2013.7.

only spends a period of one frame. On the other hand, a gradual scene change takes a period of several frames to complete a scene transition.

In general, there are certain steps to detect scene changes. At first, selected features of video frames are extracted. Popular features for scene change detection include intensity statistics and color histogram information. Next, indication of scene change, based on the features, is determined. The usual indication is the difference of selected features of consecutive frames. A scene change or scene boundary can then be discovered by comparing the indication with a certain threshold.

For example, in the pixel-based method [8], the sum of mean absolute difference (SMAD) of pixel values at corresponding location between two sequential frames is calculated. If the SMAD is larger than a predefined threshold, it is considered for a scene boundary to occur. The disadvantage for the pixel-based method is that it is sensitive to object motion in the scene. In [17], a block-based similarity comparison based on first- and second-order statistics is proposed. A scene change is declared if the ratio of similar blocks between two frames is greater than some threshold. In general, the block-based approach can tolerate noises and slow motion. However, approaches by high-order statistics usually suffer from high computational complexity.

On the other hand, the method proposed in [9] utilizes the thresholding process on the difference of histograms between two sequential frames in order to determine the scene boundary. Without considering spatial information, it can be difficult to detect scene change where different scenes have similar histograms. In the optical flow-based approach [10], possible locations of objects are predicted according to the linear moving directions of the same objects in the previous three frames. If the prediction error is greater than a predefined threshold, it is believed that a scene change happens. The edge-based method in [11] detects edge pixels for every frame at first, followed by observing the variation of edge pixels between two sequential frames to determine the scene boundary. If compressed videos are available, scene change detection can take advantage of several video characteristics provided in the compressed domain. Such characteristics that are helpful for scene change detection include discrete cosine transform (DCT) coefficients [12], motion vectors [19], [20], and block modes/types [18]. The collection of DC coefficients in DCT forms a down-sampled version of original data, and the method in [12] computes the difference of DC image to determine scene boundary.

Scene change detection by using Markov Chain Monte Carlo (MCMC) algorithm [21] and clustering-based [22] approaches by applying K-means clustering algorithm also provide feasible solutions. The posterior probability calculation of the MCMC algorithm is computed based on the model priors and the data likelihood of the video and it requires much computation effort. Study [23] has shown that the complexity of clustering-based approaches is also higher than both pixel-based and block-based luminance difference approaches.

2.2 Parallel Processing for Video Encoding

Parallel processing with multicore processing units for video encoding is about distributing computation load among a number of parallel-executing processors. The parallel processing can be divided into instruction-level parallelism (ILP) and thread-level parallelism (TLP) [15]. One of the simplest methods used to accomplish ILP uses instruction pipelining. The steps to finish an instruction are instruction fetch, instruction decode, execute, memory access, and register write back. On the contrary, TLP is the parallelism inherent in an application, which runs multiple threads at once. The benefits of TLP are achieved by distributing the workload among a number of processor cores [16]. Several coarse-granularity methods based on TLP are often used, such as GOP-level [2], frame-level [3], [4], slice-level [5], and MB-level [6], [7] approaches.

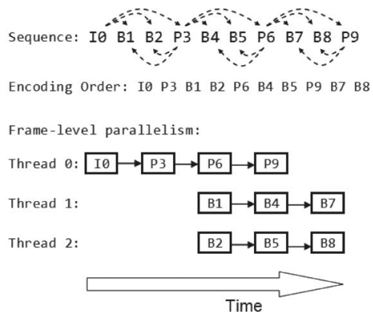

Fig. 1. An illustration of frame-level parallelism.

In general, GOPs are coded independently with some exceptions. For example, slices of a key picture can be either intracoded or intercoded according to the MPEG-4 AVC standard. If a key picture is intercoded, its reference frame can be from the previous GOP. Therefore, the two GOPs are no longer coded independently. However, due to error drifting problem over error-prone channels, key pictures are usually intracoded. In these cases, GOP-level parallelism [13] can be easily achieved with good load balance at the cost of long latency. For frame-level parallelism, the basic concept is that reference frames are given priority to be encoded than any other frames that rely on those reference frames. Fig. 1 shows an example of the frame-level parallelism, and the dashed lines indicate the prediction relations among frames. For example, *I0* is referred by *B1*, *B2*, and *P3*; *P3* is referred by *B1*, *B2*, *B4*, *B5*, and *P6*. Therefore, according to the frame dependence, the encoding order can be "*I0 P3 B1 B2 P6 B4 B5 P9 B7 B8*." Actually, after the encoding of *P3* is finished, *P6*, *B1*, and *B2* can be encoded simultaneously. Similarly, *P9*, *B4*, and *B5* can be processed in a parallel manner after *P6* is completed. One of the advantages of frame-level parallelism is that coding efficiency is not affected. However, the dependence among frames limits the thread scalability and it offers a challenge for scheduling.

In the MPEG-4 AVC standard, slices of the same frame are coded independently, and thus, slice-level parallelism is also feasible. The acceleration rate of slice-level parallelism is determined by the number of slices in a frame. Having more slices in a frame leads to better acceleration. However, pixel relevance in that frame cannot be exploited well and the consequence of having more slices includes higher bit rate [4]. For MB-level parallelism, due to the fact that motion vectors are coded predictively, the key point of parallel processing is that an MB can only be encoded

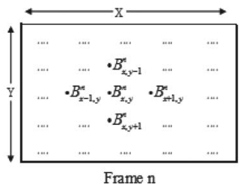

Fig. 2. The pixel with value $B^n_{x,y}$ and its four neighbors.

completely after the motion estimation of the referenced MBs (usually the left, top, and top-right neighboring ones) is finished.

3 The Proposed Parallel Processing Algorithms for Video Encoding with Adaptive GOP Structure

Parallel processing with regular coding structure, such as fixed GOP size, can be planned beforehand. However, the regular coding structure usually results in worse coding efficiency. In this section, a simple and efficient scene change detection scheme is first proposed, followed by the main contribution of this paper, the balanced frame-level parallel scheduling algorithms, which take varying GOP sizes and coding complexity into account.

3.1 Abrupt and Gradual Scene Change Detection

The fundamental idea of the proposed scene change detection is as follows: If there is a scene change between frame $N-1$ and frame N, the difference between these two frames shall be significant. However, the texture variation of a frame can complicate the judgment and shall be taken into consideration.

In this paper, the difference between two frames *n, m* is modeled as their interframe variation. It is defined as the sum of absolute temporal difference (SATD) between two frames shown as follows:

$$SATD(n,m) = \sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} \left|B^n_{x,y} - B^m_{x,y}\right|, \tag{1}$$

where $B^n_{x,y}$ is the pixel value at location *(x,y)* of frame n. X and Y are the width and height of a frame, respectively.

To measure the texture variation of a frame, the intraframe variation, defined as the sum of absolute spatial difference (SASD), is utilized. The absolute spatial difference of a pixel stands for the mean of absolute difference between the pixel and each of its neighboring pixels. Common definition of neighboring pixels includes four-connected pixels and eight-connected pixels. The SASD based on four-connected pixels is shown in (2) and Fig. 2. For the case of eight-connected pixels, the corresponding SASD is shown in (3). Since there is not much performance difference between four neighbors and eight neighbors as shown later, the SASD calculation with four neighbors is more appealing due to less computation complexity:

$$\begin{aligned} SASD(n) = \sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} \frac{1}{4}\big(&\left|B^n_{x,y} - B^n_{x-1,y}\right| + \left|B^n_{x,y} - B^n_{x+1,y}\right| \\ &+ \left|B^n_{x,y} - B^n_{x,y-1}\right| + \left|B^n_{x,y} - B^n_{x,y+1}\right|\big), \end{aligned} \tag{2}$$

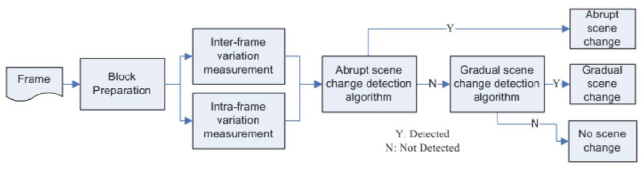
Frame
Block Preparation
Inter-frame variation measurement
Intra-frame variation measurement
Abrupt scene change detection algorithm
Gradual scene change detection algorithm
Abrupt scene change
Gradual scene change
No scene change
Y
N
Y
N
Y: Detected
N: Not Detected

Fig. 3. The flowchart of the proposed scene change detection scheme.

$$\begin{aligned} SASD(n) = \sum_{y=0}^{Y-1}\sum_{x=0}^{X-1} \frac{1}{8}\big(&\left|B^n_{x,y} - B^n_{x-1,y}\right| + \left|B^n_{x,y} - B^n_{x+1,y}\right| \\ &+ \left|B^n_{x,y} - B^n_{x,y-1}\right| + \left|B^n_{x,y} - B^n_{x,y+1}\right| + \big|B^n_{x,y} \\ &- B^n_{x-1,y-1}\big| + \left|B^n_{x,y} - B^n_{x-1,y+1}\right| + \left|B^n_{x,y} - B^n_{x+1,y-1}\right| \\ &+ \left|B^n_{x,y} - B^n_{x+1,y+1}\right|\big). \end{aligned} \tag{3}$$

The interframe variation in (1) can indicate the difference between two frames. Significant difference between two frames is usually a good indication of scene change. However, the interframe difference can result from both scene change and object/camera motion within a scene. If the level of intraframe variation by either (2) or (3) is higher, the interframe variation shall be stronger in order to avoid false alarm. In addition, since the pixel-based approach can be sensitive to texture distribution and addition noise in a scene, the block-based approach is taken to make the detection robust and to reduce required computation.

The proposed scene change detection algorithm that is able to distinguish abrupt scene change from gradual scene change is shown in Fig. 3. In the step of block preparation, every frame is decomposed into nonoverlapping blocks with block size determined empirically and the determination of the size will be discussed soon. After that, each block is assigned the average of the pixel values of that block as its *block value*. Those block values are used to calculate the interframe variation and the intraframe variation shown in (1), (2), and (3), where $B^n_{x,y}$ is now the block value at block location *(x, y)* of frame n. The abrupt scene change detection algorithm is then performed first, followed by the gradual scene change detection algorithm.

3.1.1 Abrupt Scene Change Detection Algorithm

An abrupt scene change indicates that a scene transition from one scene into another scene only spends one frame. The detection is according to the variation ratio $Ratio(n, n-1)$ of the interframe variation to the intraframe variation of two successive frames n and $n-1$. The variation ratio $\mathrm{Ratio}(i,j)$ of the interframe variation to the intraframe variation of two frames i and j is shown as follows:

$$Ratio(i,j) = \frac{SATD(i,j)}{SASD(i)}. \tag{4}$$

If $Ratio(n, n-1)$ is greater than the predefined threshold Th_{asc}, it is determined for an abrupt scene change to occur.

To observe the effect of the $Ratio(n, n-1)$ value upon the detection, a video mix with abrupt scene changes at CIF format is composed by several test sequences: *Akiyo*, *Bridge*, *Bus*, *Deadline*, *Foreman*, *Irene*, *Mobile*, *Tempete*, and *Waterfall*. The actual abrupt scene changes happen at frame 30, 89, 169, 209, 359, 429, 486, 520, 583, 621, 651, 691, and 721,

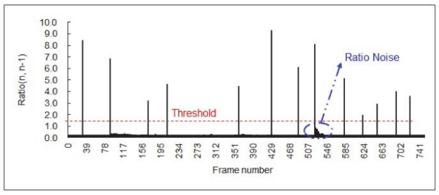

Fig. 4. The values of $Ratio(n, n-1)$ with four neighbors for the training video with abrupt scene changes.

respectively. Every frame is first divided into nonoverlapping blocks. The results of $Ratio(n, n-1)$ per frame with four neighbors and eight neighbors according to (4) are shown in Figs. 4 and 5, respectively. The value of $Ratio(n, n-1)$ is rather low for most of the time except for the 13 steep pulses whose locations exactly correspond to the frames where actual abrupt scene changes occur. The area enclosed by an ellipse in Fig. 4 indicates the "ratio noise," which means that the ratio of the interframe variation to the intraframe variation within the same scene can be close to 1. To identify the abrupt scene changes based on (4), the predefined threshold Th_{asc} can be chosen as 1.4 safely. From the results, the approach with four neighbors is better when taking ratio statistics and computation overhead into account.

Besides comparing $Ratio(n, n-1)$ with the threshold Th_{asc}, the peak can also be detected by taking the second derivative $f_R''(n, n-1)$ of the ratio as defined in (5). The local maxima of the ratios shall be at the positions where the values of $f_R''(n, n-1)$ are negative. According to the results in Fig. 6, it is determined to have an abrupt scene change when $f_R''(n, n-1)$ is smaller than -1.0. This threshold is actually not sensitive though:

$$f_R''(n, n-1) = Ratio(n+1, n) + Ratio(n-1, n-2) - 2 \cdot Ratio(n, n-1). \quad (5)$$

3.1.2 Gradual Scene Change Detection Algorithm

Because the transition of a gradual scene change takes a period of several frames, the ratio $Ratio(n, n-1)$ of the interframe variation to the intraframe variation of two successive frames n and $n-1$ can be insignificant.

The proposed gradual scene change detection is to monitor the values of $Ratio(n, m)$, where n stands for the current frame and m is the frame index where the last scene change occurs. In Fig. 7, an actual example with gradual scene change is shown. When a gradual scene change occurs, the scene transition can be divided into two phases: *the phase of gradual increment* and *the phase of convergence*.

For the phase of gradual increment, the interframe variation between the current frame and the frame where the last scene change occurs grows. A heuristic threshold "Threshold_L" $= 0.7$ for $Ratio(n, m)$ is used to detect the beginning of possible gradual increment. This threshold is actually not a sensitive parameter. Different values of "Threshold_L" between 0.3 and 1.0 do not affect the detection performance, according to the simulation results.

After the phase of gradual increment, $Ratio(n, m)$ will part from the increasing trend and enter the phase of convergence where $Ratio(n, m)$ shall be still greater than the threshold "Threshold_H." Since the gradual scene change is completed during the phase of convergence, the characteristic of $Ratio(n, m)$ of this phase shall be the same as the case for abrupt scene change. Therefore, the value of Threshold_H is equal to Th_{asc}. For better coding efficiency, the first frame of each new scene is encoded as an instantaneous decoding refresh (IDR) key frame. Since the location of a new scene is content dependent, the GOP structure will be dynamic and it is adaptively decided according to the detected scenes in a video. Therefore, the challenge for parallel scheduling becomes greater.

As mentioned earlier, the block-based approach for scene change detection is chosen to make the detection robust while reducing the computation effort. The video mix at CIF resolution used in Fig. 4 is taken to observe the variation of $Ratio(n, n-1)$ at different block sizes. To see the difference of $Ratio(n, n-1)$ between scene changes and nonscene changes, the $min\{10.0, Ratio(n, n-1)\}$ for each frame is recorded in this experiment.

In Fig. 8, we can notice that the block sizes at both "16×16" and "32×32" show better performance than the others by not only reducing the noise effectively but also enhancing the peak points where actual scene changes do occur. The smallest real peak of $Ratio(n, n-1)$ for 32×32 blocks is 1.6, while it is more than 2.0 for 16×16 blocks. The block size 16×16 is slightly better than 32×32; therefore, block size 16×16 is chosen for the proposed abrupt and gradual scene change detection scheme. If the video is at CIF resolution, there will be 22×18 nonoverlapped blocks in a frame. If the block number is too small, the results can

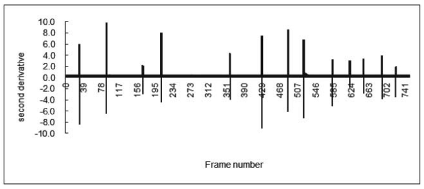

Fig. 6. The values of $f_R''(n, n-1)$ with four neighbors for the training video with abrupt scene changes.

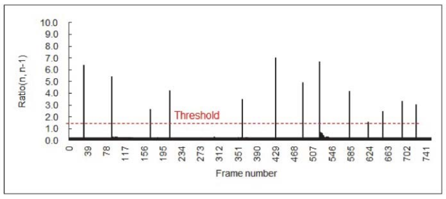

Fig. 5. The values of $Ratio(n, n-1)$ with eight neighbors for the training video with abrupt scene changes.

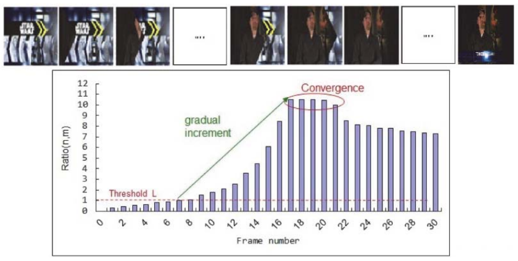

Fig. 7. The variations of $Ratio(n, m)$ in the example with gradual scene change.

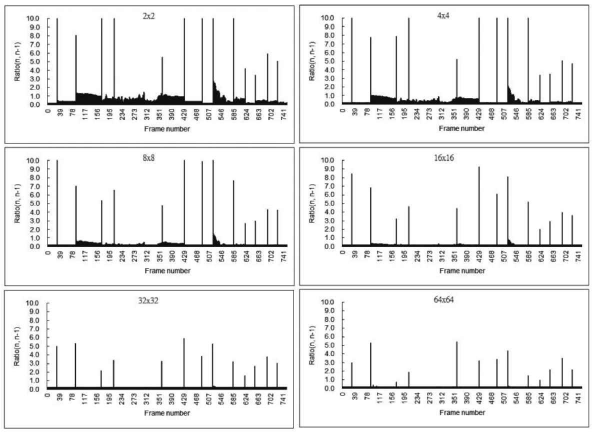

Fig. 8. The variations of $Ratio(n, n-1)$ at different block sizes.

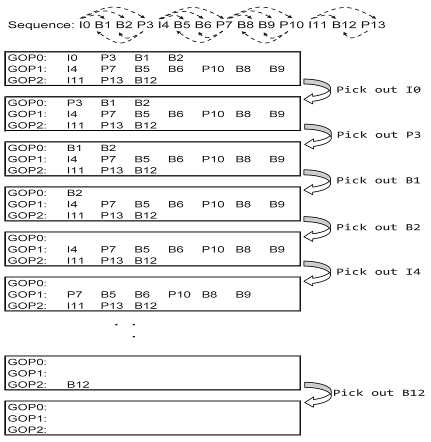

Fig. 9. Priority selection by using prediction structure.

be inaccurate due to insufficient statistics information. More simulation results at other video resolutions are available in Section 4.

3.2 Balanced Frame-Level Parallel Scheduling

For frame-level parallel scheduling, coding dependence among frames limits the thread scalability. Certain threads can be idle during parallel processing, and the whole encoding time is still determined by the thread which finishes its job last.

After frames are cut into several GOPs according to the scene change detection scheme described in the last section, the proposed scheduling algorithm will determine which thread takes care of which frames and in what order.

The frame-level parallel scheduling procedure is divided into two stages: frame priority selection and thread priority assignment. At the stage of frame priority selection, the frame with priority is picked for the thread choosing stage where a certain thread will be assigned to implement the frame coding by balancing and shortening the operation time.

3.2.1 Frame Priority Selection for Scheduling

At the stage of frame priority selection, the frame with priority will be recognized for the processing during thread priority assignment.

Priority can be given according to the ordering of interframe prediction. In other words, reference frames shall have priority over other frames in a GOP. In video encoding, if a frame refers to another frame by motion compensation, the second frame is the reference frame of the first frame and that reference frame shall have priority during scheduling. For example, given a video sequence of three GOPs with coding index "*I0 B1 B2 P3 I4 B5 B6 P7 B8 B9 P10 I11 B12 P13*," the English letter of a coding index stands for the frame type and the following number is the display order of that frame. As shown in Fig. 9, the arrows indicate the reference structure. After the priority assignment according to the reference structure, the order of frames sent to the stage of thread priority assignment is "*I0 P3 B1 B2 I4 P7 B5 B6 P10 B8 B9 I11 P13 B12*" shown step by step in Fig. 9. For convenience' sake, the method of frame priority selection described above is called *priority selection by using prediction structure*.

However, coding dependence is not the only factor that can influence the results of parallel scheduling. The coding complexity of a video frame will determine when the other frames which will use that video frame as their reference frame can start to encode. In addition, for parallel scheduling with the dynamic GOP structure, the length of a GOP cannot be presumed. The complexity of GOP should be taken into account, since the GOP which takes more time to complete coding is usually the bottleneck of parallel scheduling.

For the reasons above, *priority selection by using coding complexity and temporal influence* is proposed. The flowchart of the priority selection is shown in Fig. 10. A frame is assigned priority by completing the frame arrangement and the temporal complexity analysis. The details of these steps are as follows.

In this proposed approach, frames of each GOP are rearranged first according to frame dependence and temporal influence. Frame dependence is determined by the reference structure. Temporal influence of a frame is an indication of the number of frames that are affected by it.

If the influence of each frame is the same, frames of each GOP are rearranged according to frame dependence and display order.

After the rearrangement of each GOP, the GOP with the longest temporal complexity will be chosen to select a frame with priority. The priority order of frames in a GOP is determined by the rearranged order mentioned earlier, and the temporal complexity of a GOP stands for the summation of estimated time complexity of the remaining frames in the GOP.

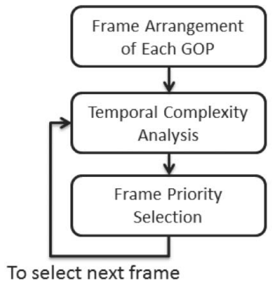

Fig. 10. The flowchart of the priority selection by using coding complexity and temporal influence.

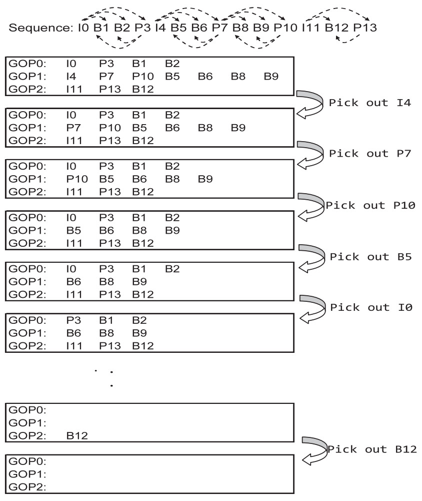

Fig. 11. Priority selection by using coding complexity and temporal influence.

The time complexity of a frame is estimated by using the previously encoded frame of the same frame type. For the initial values of time complexity, nine video sequences (*"Akiyo," "Bridge," "Bus," "Irene," "Foreman," "Deadline," "Mobile," "Tempete,"* and *"Waterfall"*) are encoded and the averaged time complexity will be used. The results of the average time to encode those videos are 120 ms for I frame, 290 ms for P frame, and 360 ms for B frame. Only the ratios of time complexity between different frame types (or slice types) have effect on the decision of scheduling. As a result, the initial values shall be good enough even if a different processor is used to encode videos.

The procedure of the priority selection by using coding complexity and temporal influence is summarized by the following example, as shown in Fig. 11.

All the frames of each GOP are rearranged according to the frame dependence and temporal influence. For example, the encoding order for the frames in *GOP1* is *"I4 P7 P10 B5 B6 B8 B9."*

In order to select a frame with priority, temporal complexity of each GOP is calculated as follows, where t_x stands for the time complexity for frame x. According to the temporal complexity calculation shown below, *GOP1* is the bottleneck and its first frame, *I4*, is selected to have priority to be assigned a thread at the next stage:

$$\begin{aligned}&\text{GOP0}:\\&t_{I0}+t_{P3}+t_{B1}+t_{B2}=120+290+360+360=1{,}130,\\&\text{GOP1}:\\&t_{I4}+t_{P7}+t_{P10}+t_{B5}+t_{B6}+t_{B8}+t_{B9}=120+290+290\\&\quad+360+360+360+360=2{,}140,\\&\text{GOP2}:\\&t_{I11}+t_{P13}+t_{B12}=120+290+360=770.\end{aligned}$$

After that, temporal complexity of *GOP1* is calculated again. The GOP with most temporal complexity is chosen to find another frame with priority. As shown below, *GOP1* is still the bottleneck and *P7* is chosen to have priority. The steps above are repeated until all the frames are processed:

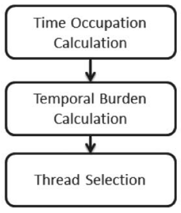

Fig. 12. The flowchart of the thread priority assignment.

$$\begin{aligned}&\text{GOP0}:\\&t_{I0}+t_{P3}+t_{B1}+t_{B2}=120+290+360+360=1{,}130,\\&\text{GOP1}:\\&t_{P7}+t_{P10}+t_{B5}+t_{B6}+t_{B8}+t_{B9}=290+290+360\\&\quad+360+360+360=2{,}020,\\&\text{GOP2}:\\&t_{I11}+t_{P13}+t_{B12}=120+290+360=770.\end{aligned}$$

3.2.2 Thread Priority Assignment for Scheduling

At the previous stage of frame priority selection, the frame with priority will be recognized. In this section, thread priority assignment for scheduling is proposed to assign a thread to implement the coding. The flowchart of the thread priority assignment is shown in Fig. 12. Time occupation of each thread is determined first, and then, a thread with minimal temporal burden is chosen. The details of these steps are as follows.

The proposed approach relies on the time occupation of a thread, which is defined as the indication of the end time $t_{l,end}$ of the last frame l coded by this thread shown as follows:

$$T_{n,end}=t_{l,elapsed}+t_{l,spent}. \tag{6}$$

In (6), $T_{n,end}$ is the end time of thread n, i.e., the time occupation of thread n. $t_{l,elasped}$ is the elapsed time before the last frame l starts to encode and $t_{l,spent}$ is the temporal duration to encode the last frame l. The start time of encoding the last frame depends on the second last frame and the reference frame. In other words, the last frame can start its encoding after the encoding of the second last frame and the reference frame is completed. Therefore, $t_{l,elasped}$ can be expressed as follows:

$$t_{l,elapse}=Max(t_{l-1,end},t_{r,end}). \tag{7}$$

In (7), $t_{l-1,end}$ is the end time of the second last frame, and $t_{r,end}$ is the end time of the reference frame.

We continue to use the GOP1 in Section 3.2.1 to explain the concept of time occupation of a thread. The coding index of the GOP1 is *I4 B5 B6 P7 B8 B9 P10*, and the encoding order of these frames in each thread is assumed as the ordering in Fig. 13. The time occupation of thread 0 and 1 can be calculated as follows:

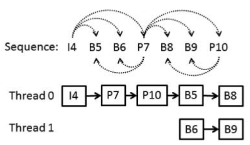

Fig. 13. An example to calculate the time occupation of a thread.

Fig. 14. The screen shots of the 20 videos used in the first test sequence in Section 4.

$$\begin{aligned}T_{0,end} &= t_{B8,elapsed} + t_{B8,spent} = Max(t_{B5,end}, t_{P7,end}, t_{P10,end})\\ &\quad + t_{B8,spent}\\ &= t_{B5,end} + t_{B8,spent} = Max(t_{P10,end}, t_{I4,end}, t_{P7,end})\\ &\quad + t_{B5,spent} + t_{B8,spent}\\ &= \cdots = t_{I4,spent} + t_{P7,spent} + t_{P10,spent} + t_{B5,spent} + t_{B8spent}\\ &= 120 + 290 + 290 + 360 + 360 = 1{,}420,\end{aligned} \tag{8}$$

$$\begin{aligned}T_{1,end} &= t_{B9,elapsed} + t_{B9,spent} = Max(t_{B6,end}, t_{P7,end}, t_{P10,end})\\ &\quad + t_{B9,spent}\\ &= t_{B6,end} + t_{B9,spent} = Max(t_{I4,end}, t_{P7,end})\\ &\quad + t_{B6,spent} + t_{B9,spent}\\ &= \cdots = t_{I4,spent} + t_{P7,spent} + t_{P10,spent} + t_{B6,spent} + t_{B9spent}\\ &= 120 + 290 + 290 + 360 + 360 = 1{,}420.\end{aligned} \tag{9}$$

The thread priority assignment for scheduling is based on the time occupation of a thread defined above. The key concept is to determine which thread shall experience the least temporal burden if the selected frame from the frame priority selection step would have been assigned to that thread.

As also shown in (10), the temporal burden T^i_{burden} for thread i is defined as the time occupation if the selected frame is assigned to the thread and the new time occupation to complete the new assignment is greater than the current time occupation. If the new time occupation is equal to the current time occupation, the temporal burden is defined as zero. The thread with the minimal temporal burden will be chosen to implement the coding of the selected frame:

$$T^i_{burden} = \begin{cases} 0, & \text{if } T'_{i,end} \le T_{i,end},\\ T'_{i,end}, & otherwise. \end{cases} \tag{10}$$

In (10), $T'_{i,end}$ is the new time occupation of thread i, i.e., the new time occupation if the selected frame is implemented in thread i.

4 Experimental Results

The first test sequence of 1,268 CIF frames is composed of *"Bowing," "Container," "Flower," "Hall," "Google Map," "Highway," "Ice," "Mother-daughter," "News," "Silent," "Soccer," "Students," "Coastguard," "Star Wars," "Crew," "Football," "Husky," "Harbour," "Nissin Noodles,"* and *"City."* Their screen shots are shown in Fig. 14. There are 30 abrupt scene changes and 4 gradual scene changes in this sequence. The video sequence in the following simulations is encoded by the *x264* software [14], which can generate bit streams conforming to the H.264/MPEG-4 AVC standard.

TABLE 1
The Accuracy Comparison between the Proposed Method and the Pixel-Based Method

	Abrupt scene change			Gradual scene change			Accuracy
	Correct	False	Lost	Correct	False	Lost	
Proposed method	30	0	0	3	0	1	97%
Pixel-based method	29	0	1	0	0	4	85%

4.1 Experimental Results of Adaptive GOP Structure

The determination of adaptive GOP structure is achieved by the proposed scene change detection algorithm described in Section 3.1, and the first frame of each new scene is encoded as an IDR key frame.

First, the performance of the proposed abrupt and gradual scene change detection algorithm will be examined and compared with the pixel-based approach [8]. Besides numbers of correct detection and false alarm, a performance metric, *accuracy*, is defined as follows:

$$\text{Accuracy} = \frac{\text{Correct Detection}}{\text{Correct Detection} + \text{Lost Detection} + \text{False Detection}}. \tag{11}$$

In (11), false detection denotes the number of the false scene changes decided by an algorithm, and lost detection is the number of the actual scene changes that are not detected correctly. As shown in Table 1, the accuracy measurement of the proposed abrupt and gradual scene change detection algorithm is 97 percent, while that of the pixel-based approach is 85 percent. The performance of the proposed algorithm is quite well for both abrupt and gradual scene changes. The pixel-based approach also shows good ability in abrupt scene change detection; however, it cannot detect the gradual scene changes and its computation effort is greater than the block-based approach of the proposed scheme.

In terms of coding efficiency, we consider the GOP structure of the same size, the adaptive GOP structure with pixel-based scene change detection algorithm [8], and the adaptive GOP structure with the proposed scene change detection scheme. For the GOP structure of the same size, several sizes (4, 8, 16, 32, and 37) are simulated. The average GOP size of the proposed algorithm is about 37 frames. According to the rate-quality results shown in Fig. 15, the coding efficiency of the approaches with adaptive GOP structure is better than that of the approaches with fixed GOP sizes. It can also be noticed that the coding efficiency is not affected by the gradual scene changes much. The reason is that even though gradual scene changes are important to several video processing technologies, such as video indexing and retrieval, the transition of gradual scene changes makes little difference if a key frame is used or not.

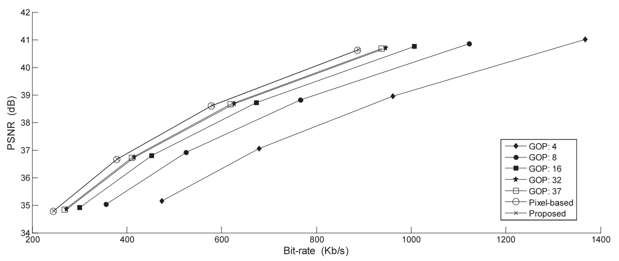

Fig. 15. Rate-quality performance comparison between the adaptive GOP structure and the fixed GOP structure.

Fig. 16. The screen shots of "*MerryGoAround*," "*SpinningWheel*," and "*Aurora*."

TABLE 2
The Detection Performance of the Proposed Scheme

	Abrupt scene change			Gradual scene change			
	Correct	False	Lost	Correct	False	Lost	Accuracy
CIF	64	0	0	5	0	1	98.6%
4CIF	10	0	0	1	1	1	84.6%

To see the performance of the proposed scene change scheme, two more test sequences are composed at CIF and 4CIF sizes. The test sequence at CIF format is composed by "*Bridge*," "*Coastguard*," "*Container*," "*Flower*," "*Foreman*," "*Hall*," "*Highway*," "*News*," "*Paris*," "*Silent*," "*Stefan*," "*Tempete*," "*Waterfall*," "*MerryGoAround*," "*SpinningWheel*," and "*Aurora*." Each of the videos appears in the composed video multiple times with random orders. There are 70 scene changes in total and 64 of them are abrupt scene changes. "*MerryGoAround*," "*SpinningWheel*," and "*Aurora*" are sequences with rotating objects and luminance variation. Their screen shots are shown in Fig. 16. For the test sequence at 4CIF resolution, there are 10 abrupt scene changes and two gradual scene changes. It is composed of "*City*," "*Crew*," "*Harbor*," "*Ice*," "*Soccer*," "*MerryGoAround*," "*SpinningWheel*," and "*Aurora*."

The results are shown in Table 2. By using the same threshold parameters determined according to the training sequence in Section 3, the detection performance is good for videos at CIF and 4CIF. The object rotation in sequences "*MerryGoAround*" and "*SpinningWheel*" does not affect the detection results of abrupt scene changes, nor does the luminance condition in "*Aurora*."

4.2 Simulation Results of Frame-Level Parallel Scheduling

In this section, four parallel scheduling algorithms are simulated for performance comparison.

The first one is the proposed balanced parallel scheduling algorithm according to the ordering of inter-frame prediction, referred as the "BPSA-OIFP" for easier discussion below. The second one "BPSA-CCTI" is another proposed balanced parallel scheduling algorithm according to the coding complexity and temporal influence. The third approach is the "Random" which assigns frames to different threads randomly, and the fourth one "In turn" assigns frames to different threads in turn.

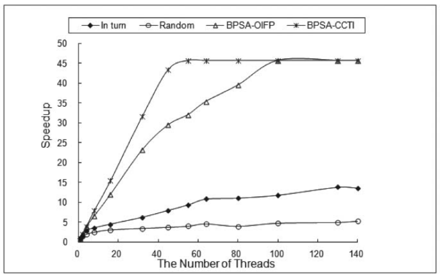

Fig. 17. Speedup factors for different scheduling methods.

TABLE 3
The Encoding Time Comparison with Respective to the Optimal Solution by the Exhaustive Search

		Sequence	
		IIIPIPIPPP	IIIBPIBPBP
Method	In Turn	1.69 sec	1.90 sec
	Random	1.69 sec	1.54 sec
	BPSA-OIFP	1.52 sec	1.54 sec
	BPSA-CCTI	1.06 sec	1.25 sec
	Exhaustive Search	1.06 sec	1.25 sec

4.2.1 Speedup Analysis

The performance of the different parallel approaches is evaluated according to the speedup factor defined as follows:

$$\text{Speedup}_n = \frac{\text{Encoding time for a single thread}}{\text{Encoding time for } n \text{ threads}}. \qquad (12)$$

The speedup performance of the four methods is shown in Fig. 17. The proposed "BPSA-CCTI" shows rather good load balance early on, as we can see that the slope before 45 threads is almost one. If the slope is equal to one, it means that the multiple threads are utilized completely. The speedup factor of the proposed "BPSA-CCTI" saturates at around 46. The reason for that is the frame dependence, which limits the thread usage when the number of threads is large enough.

To have a better understanding of the load-balancing capacity of the four approaches, exhaustive search is performed on two shorter video sequences so that the search can be finished within reasonable time.

The encoding time with respect to different sequences and scheduling methods is shown in Table 3. The thread number is two. The estimation of the encoding time of I-frame, P-frame, and B-frame is 120, 290, and 360 ms, respectively, as also described in Section 3.2. The encoding time of the proposed "BPSA-CCTI" is the same as the encoding time according to the scheduling done by the exhaustive search method. To reach the scheduling decision of 10 frames only, the time required for the exhaustive search method is already 846 times more than the time for the "BPSA-CCTI" algorithm. The computation complexity to complete the developed parallel scheduling can be considered as a negligible overhead, since only simple addition and comparison operations are required.

4.2.2 Encoding End Time of Each Frame

In the case of eight threads, the encoding end time of each frame is arranged according to the display order of frames

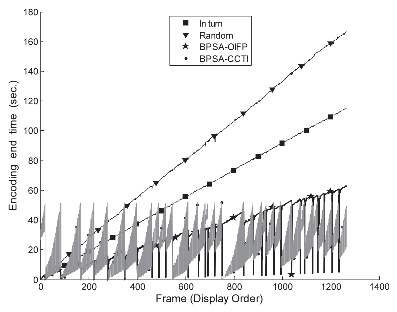

Fig. 18. Encoding end time of each frame with eight threads.

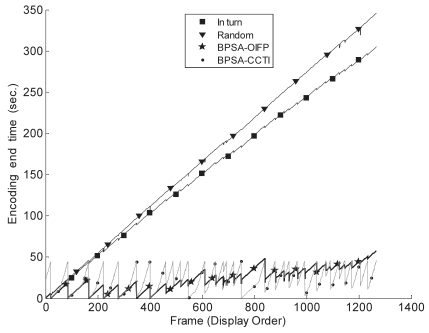

Fig. 19. Encoding end time of each frame with eight threads. Frame type: I and P only.

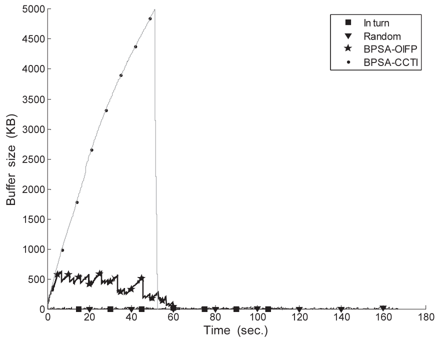

Fig. 20. Dynamic buffer size to encode the test sequence with respect to the four scheduling approaches.

in Fig. 18. The end time curves of the "In turn" and the "Random" methods are linear with respective to display order. Therefore, the efficiency of parallel processing suffers. For the curve of the "BPSA-CCTI," the curve shows zigzag pattern and the time to complete the coding of every frame is much shorter.

If the frame type during video encoding is restricted to either I frame or P frame, the encoding end time will increase considerably for the methods "In turn" and "Random." However, the end time is not affected much in the case of the methods "BPSA-CCTI" and "BPSA-OIFP," as shown in Fig. 19.

4.2.3 Dynamic Buffer Size

"Dynamic buffer size" means the buffer capacity required to store the encoded frames before they are pushed out of the encoder.

From the results in Fig. 20, the "In turn" and "Random" methods only need a small buffer, while the "BPSA-CCTI" method requires a larger buffer size for some period of the time. Though good parallelism usually pays the price of

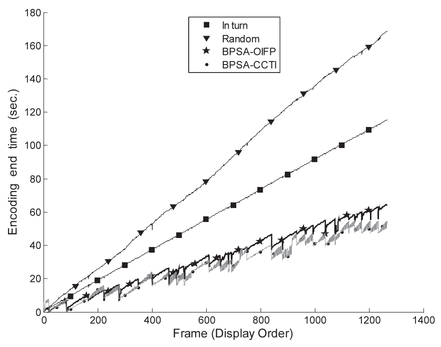

Fig. 21. Encoding end time of each frame with eight threads. The batch processing is applied.

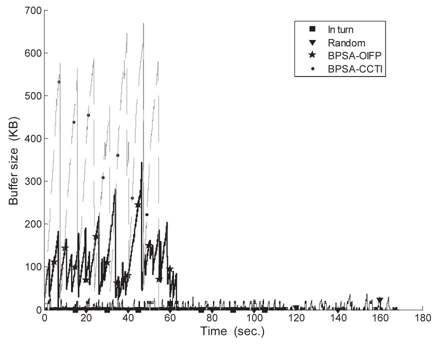

Fig. 22. Dynamic buffer size to encode the test sequence with respect to the four scheduling approaches. The batch processing is applied.

higher resource consumption, the buffer requirement of the "BPSA-OIFP" method is much less.

4.2.4 Recommended Methods and Batch Processing

From the descriptions and simulations above, the parallel scheduling approach of the "BPSA-CCTI" shows the best speedup performance, but a larger buffer is required to complete the coding. If the bottleneck of video coding is coding speed, the "BPSA-CCTI" method is the recommended approach. However, if the buffer size and coding speed are equally important, the "BPSA-OIFP" will be the better candidate.

Nevertheless, another trade-off can be made by encoding the frames in batches. An experiment is conducted in a processor with eight threads. Every time a batch of 200 frames is scheduled. From the results shown in Figs. 21 and 22, the buffer size is reduced, when compared with the nonbatch processing. Therefore, if the video sequence is very long and the capability of hardware is not sufficient, the proposed algorithm with batch processing can present a good solution as well.

5 Conclusions

In the paper, the parallel processing for video encoding with adaptive GOP structure is proposed. There are two contributions. The major contribution of this paper is the balanced frame-level parallel scheduling algorithms that first determine frame priority, followed by the thread priority assignment. The other contribution is the block-based abrupt and gradual scene change detection algorithm, which can detect both types of scene changes at less computation effort. The adaptive GOP structure is formed according to the scene structure. There are two variations to determine the frame priority: "priority selection by using coding complexity and temporal influence" and "priority selection by using prediction structure." The first one can take advantage of multiple threads so well that its performance is the same as the speedup performance of the exhaustive search scheduling, according to the experimental results. The second approach requires much less buffer. In addition, the batch processing can also help reduce the required buffer overhead if the buffer size shall be taken into account.

In this paper, the GOP structure is determined without the consideration of parallel scheduling. Therefore, a simple and efficient approach is developed so that it will not be the computational bottleneck. A potential future work is to take advantage of multiple cores to speed up scene change detection such that more sophisticated algorithms with better performance of scene change detection can be considered at the same time of encoding videos.

REFERENCES

[1] P. Yin, H.Y. Cheong, A. Tourapis, and J. Boyce, "Fast Mode Decision and Motion Estimation for JVT/H.264," *Proc. IEEE Int'l Conf. Image Processing,* vol. 2, pp. 853-856, 2003.

[2] K. Shen, L.A. Rowe, and E.J. Delp, "Parallel Implementation of An MPEG-1 Encoder: Faster Than Real Time," *Proc. SPIE,* vol. 2419, pp. 407-418, Feb. 1995.

[3] Y. Chen, E. Li, X. Zhou, and S. Ge, "Implementation of H.264 Encoder and Decoder on Personal Computers," *J. Visual Comm. and Image Representation,* vol. 17, pp. 509-532, Apr. 2006.

[4] S. Ge, X. Tian, and Y.-K. Chen, "Efficient Multithreading Implementation of H.264 Encoder on Intel Hyper-Threading Architectures," *Proc. IEEE Pacific-Rim Conf. Multimedia,* vol. 1, pp. 469-473, 2003.

[5] M. Roitzsch, "Slice-Balancing H.264 Video Encoding for Improved Scalability of Multicore Decoding," *Proc. Seventh ACM and IEEE Int'l Conf. Embedded Software,* pp. 269-278, Sept. 2006.

[6] S.M. Akramullah, I. Ahmad, and M.L. Liou, "A Data-Parallel Approach for Real-Time MPEG-2 Video Encoding," *J. Parallel and Distributed Computing,* vol. 30, no. 2, pp. 129-146, 1995.

[7] T.R. Jacobs, V.A. Chouliaras, and D.J. Mulvaney, "Thread-Parallel MPEG-2, MPEG-4 and H.264 Video Encoder for SoC Multi-Processor Architectures," *IEEE Trans. Consumer Electronics,* vol. 52, no. 1, pp. 269-275, Feb. 2006.

[8] H.J. Zhang, A. Kankanhalli, and S.W. Smoliar, "Automatic Partitioning of Full-Motion Video," *Multimedia System,* vol. 1, pp. 10-28, 1993.

[9] L. Wu, X. Huang, J. Niu, Y. Xia, Z. Feng, and Y. Zhou, "FDU at TREC2002: Filtering, Q&A, Web and Video Tasks," *Proc. 11th Text Retrieval Conf.,* 2002.

[10] O. Fatemi, S. Zhang, and S. Panchanathan, "Optical Flow Based Model for Scene Cut Detection," *Proc. Canadian Conf. Electrical and Computer Eng.,* vol. 1, pp. 470-473, 1996.

[11] R. Zabih, J. Miller, and K. Mai, "A Feature-Based Algorithm for Detecting and Classifying Scene Breaks," *Proc. Third ACM Int'l Conf. Multimedia (Multimedia '95),* pp. 189-200, 1993.

[12] B. Yeo and B. Liu, "Rapid Scene Analysis on Compressed Video," *IEEE Trans. Circuits and Systems for Video Technology,* vol. 5, no. 6, pp. 533-544, Dec. 1995.

[13] J.C. Fernández and M.P. Malumbres, "A Parallel Implementation of H.26L Video Encoder," *Proc. Eighth Int'l Euro-Par Conf. Parallel Processing (Euro-Par '02),* vol. 2400 pp. 830-833, 2002.

[14] x264 Software, http://www.videolan.org/developers/x264.html, 2013.

[15] I. Ahmad, Y. He, and M.L. Liou, "Video Compression with Parallel Processing," *Parallel Computing,* vol. 28, pp. 1039-1078, 2002.

[16] J.F. Martinez and J. Torrellas, "Speculative Synchronization: Applying Thread-Level Speculation to Explicitly Parallel Application," *ACM SIGOPS Operating Systems Rev.,* vol. 36, pp. 18-29, Dec. 2002.

[17] R. Kasturi and R. Jain, "Dynamic Vision," *Computer Vision: Principles,* pp. 469-480, IEEE CS Press, 1991.

[18] S.-C. Pei and Y.-Z. Chou, "Novel Error Concealment Method with Adaptive Prediction to the Abrupt and Gradual Scene Changes," *IEEE Trans. Multimedia,* vol. 6, no. 1, pp 158-173, Feb. 2004.

[19] I. Koprinska and S. Carrato, "Detecting and Classifying Video Shot Boundaries in MPEG Compressed Sequences," *Proc. IX European Signal Processing Conf. (EUSIPCO),* pp. 1729-1732, 1998.

[20] H.C. Liu and G.L. Zick, "Automatic Determination of Scene Changes in MPEG Compressed Video," *Proc. IEEE Int. Symp. Circuits and Systems,* pp. 764-767, 1995.

[21] Y. Zhai and M. Shah, "Video Scene Segmentation Using Markov Chain Monte Carlo," *IEEE Trans. Multimedia,* vol. 8, no. 4, pp. 686-697, Aug. 2006.

[22] B. Gunsel, A. Ferman, and A. Tekalp, "Temporal Video Segmentation Using Unsupervised Clustering and Semantic Object Tracking," *J. Electronic Imaging,* vol. 7, pp. 592-604, 1998.

[23] S. Lefèvre, J. Holler, and N. Vincent, "A Review of Real-Time Segmentation of Uncompressed Video Sequences for Content-Based Search and Retrieval," *Real-Time Imaging,* vol. 9, pp. 73-98, 2003.

Hsu-Feng Hsiao (M'05) received the BS degree in electrical engineering from the National Taiwan University, Taipei, Taiwan, in 1995, the MS degree in electrical engineering from the National Chiao Tung University, Hsinchu, Taiwan, in 1997, and the PhD degree in electrical engineering from the University of Washington, Seattle, in 2005. He was an engineering officer in the Communication Research Laboratory of the Ministry of National Defense, Taiwan, from 1997 to 1999. From 2000 to 2001, he was a software engineer at HomeMeeting, Redmond, Washington. He had been then a research assistant in the Department of Electrical Engineering, University of Washington till 2005. He has been an assistant professor in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, since August 2005. His research interests include multimedia signal processing and wired/wireless communications. He is a member of the IEEE.

Chen-Tsang Wu received the BS degree in electrical engineering from Tamkang University, New Taipei City, Taiwan, in 2001, and the MS degree in computer science from National Chiao Tung University, Hsinchu, Taiwan, in 2010. His research interests include video processing and parallel computation.