SAMPEG, a Scene Adaptive Parallel MPEG-2 Software Encoder

Dirk Farin[a], Niels Mache[b], Peter H.N. de With[c]

[a]Dept. Circuitry and Simulation, University Mannheim, Germany
[b]Dept. Image Understanding, IPVR, University Stuttgart, Germany
[c]EESI, Eindhoven University of Technology, The Netherlands

ABSTRACT

This paper presents a fully software-based MPEG-2 encoder architecture, which uses scene-change detection to optimize the Group-of-Picture (GOP) structure for the actual video sequence. This feature enables easy, lossless edit cuts at scene-change positions and it also improves overall picture quality by providing good reference frames for motion prediction. Another favourable aspect is the high coding speed obtained, because the encoder is based on a novel concept for parallel MPEG coding on SMP machines. This concept allows the use of advanced frame-based coding algorithms for motion estimation and adaptive quantization, thereby enabling high-quality software encoding in real-time. Our proposal can be combined with the conventional parallel computing approach on slice basis, to further improve parallelization efficiency. The concepts in the current SAMPEG implementation for MPEG-2 are directly applicable to MPEG-4 encoders.

Keywords: scene-change detection, parallel video coding, software compression

1. INTRODUCTION

Digital Video applications have become an established feature in various products, computers and consumer electronics. A dominant standard for video processing is the MPEG system, which is used for HDTV in the US, Digital Video Broadcast (DVB), Digital Versatile Disk (DVD) and related video applications in multimedia computers. Despite the high complexity, encoders for generation of the MPEG bit-streams are widely applied and are usually based on real-time hardware systems. However, applications do exist where real-time processing is not a prerequisite, such as DVD authoring. In this case, a software-based encoder is more powerful, since it enables sophisticated image content analysis prior to final compression coding of the input sequence. The analysis leads to the optimal handling of scene-changes and it reduces the quantization noise of the coding process.

The implementation of an advanced MPEG encoder in software* forces the programmer to exploit the possible parallelism both of the algorithm and of the platform onto which the encoder has to be mapped and executed. Concerning the parallelism of the MPEG encoding algorithm, a few potential problems do arise. The MPEG standard defines three different types of pictures (intra, predictive, and bidirectionally predictive), of which only the intra-pictures (I-pictures) can be coded independently from other pictures. Predictive-coded pictures (P-pictures) require data from a previously coded picture, and bidirectionally predictive-coded pictures (B-pictures) require the information of even two previously coded pictures. Furthermore, bit-rate control is a serial periodical process to monitor and control output buffer fullness. Despite the parallelism, this rate control should behave as if a regular implementation was chosen. The indicated data dependencies restrict the level of parallelism which can be exploited. The application of scene-change control complicates the situation by varying the GOP-structure, which consequently leads to varying data dependencies. The aforementioned brief problem statement already clarifies the major topics in the remainder of this article.

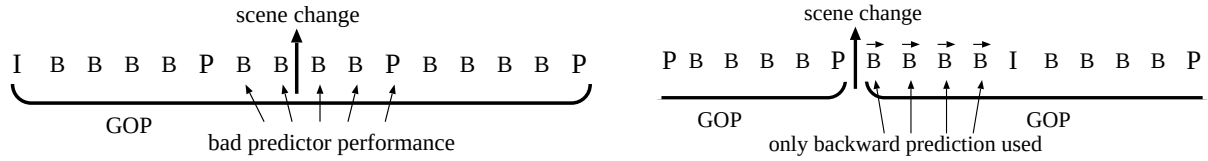
The next section concentrates on scene-change detection and the optimal handling of the encoding process (e.g. GOP structure) in case of scene-changes. Section 3 focusses on the parallelization of the MPEG algorithm and presents a graph for parallel encoding. Furthermore, scheduling of tasks is also addressed and presented with two different GOP structures.

*This work has been carried out at the Image Understanding Department of the Institute of Parallel and Distributed High-Performance Systems at the University Stuttgart, Germany, under the supervision of Prof. P. Levi.

2. SCENE-CHANGE DETECTION

2.1. Motivation

Adjusting the structure of Group-Of-Pictures (GOPs) to the image contents usually enhances the picture quality of MPEG coders, because good reference pictures for motion estimation are provided. Assume that a sudden scene-change occurs between two consecutive P-pictures (see Figure 1a). Since the image contents of the P-picture after the scene-change is completely different from the P-picture before, the motion estimator will not find good motion vectors for the second P-picture. Consequently, most macroblocks in the second P-picture will be intra coded. Moreover, all B-pictures between these two P-pictures can rely only on one reference picture, using either forward prediction (before the scene change) or backward prediction (after).



(a) Disadvantageous position of scene-change.

(b) Optimal GOP structure at scene-change.

Figure 1. GOP placement at scene-changes.

2.2. Optimal GOP arrangement in the presence of a scene-change

At first glance, the ideal GOP arrangement around a scene-change is to start with an I-picture directly after the scene-change. However, the Human Visual System (HVS) cannot perceive a picture at its full quality for about 100 ms after a sudden scene-change (temporal masking). Therefore, the optimal GOP arrangement is to code the first few pictures of the new scene as B-pictures with a low quality (see Figure 1b). In this way, bits can be saved for increasing the quality of the subsequent I-picture. As a consequence, the overall quality of the successive pictures is improved because the I-picture is the initial reference for all successive pictures.

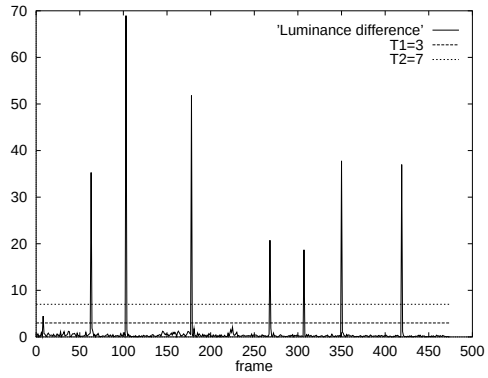
Note that it is advantageous to force the motion-estimator to not use forward prediction for the first B-pictures after the scene-change. First, forward prediction does not yield good predictions anyway, so that the corresponding time for motion search can be saved. Second, the restriction to only use backward prediction at the beginning of the GOP ensures that decoding becomes independent of the preceeding GOP, resulting in a *closed GOP*. This feature enables editing in the compressed domain, because the video sequence can be cut at the scene-change position without the need for recompressing the pictures around the edit cut.

2.3. Detection and GOP construction

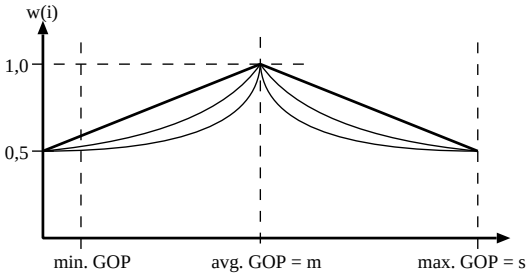
Existing methods for scene-change detection can be classified into three distinct concepts. The first class contains techniques operating on sequences in the compressed domain,[1] where image analysis data (e.g. motion information) already computed during encoding, is reused. A second class is formed by sophisticated algorithms for off-line scene segmentation with high accuracy. These algorithms are usually based on advanced motion analysis. An example application in this class is the automatic generation of an MPEG-7 scene description. The third class concentrates on real-time applications, where algorithms operating on image histograms are popular. Although not as accurate as those in the second class, the results are sufficient for applications, where a small number of detection errors can be tolerated.

Adjusting the GOP structure to the actual video sequence is an application, where sporadic detection errors do not have noticeable influence. For this reason, we adopted a computationally efficient way of detecting scene-changes, which is based solely on analyzing mean picture brightness. In addition, constraints are applied to favour equally-sized GOPs and to avoid very long or short GOPs.

For the scene-change detection, the maximum number of pictures that may be contained in a single GOP is read in and the respective mean luminances l_i for various pictures i are calculated. Beginning from the current position



(a) Mean-luminance differences of successive pictures. Also shown are the values for the thresholds T_1 and T_2.



(b) Weighting factor to favour the generation of equally sized GOPs.

Figure 2. Scene-change detection.

$i = 0$, the GOP end is placed at the position k, where

$$k = \underset{i}{\text{maxarg}} \left| l_i - l_{i+1} \right| w(i).$$

Let the weighting factor $w(i)$ be a constant, like $w(i) = 1$. In this case, the end of the GOP is placed at the position where the largest change in picture luminance occurs. However, if the scene changes gradually, the position of the maximum difference is almost random. This would result in an irregular and unpredictable GOP structure for smoothly changing scenes. In such a case, a constant GOP size is preferred, so that we define $w(i)$ to have a peak at the nominal GOP size, as shown in Figure 2b. The GOP size may be increased or decreased, but only if the luminance difference is strong enough. The stronger the luminance difference, the more can the GOP size be set near the limits.

The actual definition of $w(i)$ is not critical and we used

$$w(i) = \begin{cases} \frac{1}{2}(1 + (\frac{i}{m})^{\alpha}) & i \leq m \\ \frac{1}{2}(1 + (\frac{s-1-i}{s-1-m})^{\alpha}) & i > m \end{cases}$$

with m being the nominal GOP size and s being the maximum GOP size. The parameter α can be used to adjust the stringency with which the encoder tries to keep the mean GOP size near the nominal value. To further enhance the robustness of the detection, all detected "scene-changes" are classified into one of these three classes:

- **No scene-change**, $|l_i - l_{i+1}| < T_1$. The image luminance is nearly constant, no scene-change is assumed. In this case, the GOP is constructed with its nominal size m.
- **Potential scene-change**, $T_1 \leq |l_i - l_{i+1}| < T_2$. There is some noticeable luminance change in the sequence, but it is possibly not caused by a scene-change. Here, the GOP is rearranged to have size k, but no adjustment of the quantizer is done in case this is not a true scene-change. False detection of a scene-change, combined with coarser quantization, would result in a degraded picture quality, because the masking effect of sudden scene changes does not occur.
- **sure scene-change**, $T_2 \leq |l_i - l_{i+1}|$. The luminance change is large enough such that an abrupt scene-change can be assumed. In this case, we not only rearrange the GOP to have the size k, but also heavily reduce the quantizer resolution (depending on the strength of the luminance difference) for the first B-pictures following in the next GOP.

2.4. Results on scene-change detection

Although the proposed approach certainly fails at soft fades or wipe effects[2,3] between scenes, it is sufficient for video coding applications. The primary goal is to detect those moments, where the HVS shows comparable masking effects as occuring at abrupt scene changes, instead of finding semantically meaningful scene-changes such as in MPEG-7. An undetected scene-change has no severe consequences and detecting more scene-changes than there are is not critical as long as they are classified as a potential scene-change. However, visual impairments occurred when encoding a sequence containing many consecutive lightnings or flashlights in a predictable way. Due to the large luminance differences in such a sequence, the encoder strongly reduced the image quality, and combined with the predictable occurrence of the lightnings, this led to well visible coding artifacts.

Since the increase in quality obtained by using scene-change detection results from a shift in quantization noise between pictures, the improvement is only subjective and cannot be described in terms of SNR. Unfortunately, a clear improvement in SNR resulting from better reference frames for motion estimation, could not be observed. We measured experimentally how many bits are saved by the reduction of the quantizer resolution. Using streams with a scene-change frequency of about one change every two seconds, a bit-rate reduction of about 6-7% was observed without visible degradation (assuming playback is done in real-time speed). If this stream would be encoded in constant bit-rate, the saved bits would be used to enhance the quality of successive pictures.

3. PARALLELIZATION ARCHITECTURE

Improving the execution time of a software encoder by parallel computing can be implemented at three different levels of granularity. At coarse granular level, the total computations are distributed over a number of workstations, so coding is performed when the workstations are idle (e.g. at night). This kind of parallel processing is useful for offline processing but not for real-time applications. The computational burden is usually partitioned into large portions for efficiency reasons, leading to considerable coding delays. At medium level, the solution is multithreaded parallelism on multiprocessor workstations (Symmetric MultiProcessing, SMP). Workstations with multiple CPUs are becoming increasingly common, as their computing power is economically cheaper than single CPU systems. At fine granular level, parallelism is obtained by exploiting SIMD (Single Input Multiple Data) instructions of state-of-the-art processors. Some of these instructions are optimized for image processing and even MPEG processing, because MPEG coding is frequently used in computer applications. An alternative implementation is mapping on massiv parallel computers. In this paper, we concentrate on SMP parallelization; the use of SIMD instructions in the encoder is only partially covered.

3.1. Parallelization on workstation clusters

Distributing the work load across several workstations[4,5] is a good choice for offline coding of source material. However, the disadvantage of this approach is the high network load because of the strong data dependencies. In this approach, the work units for a workstation can be made very large, for example the size of a GOP. This implies that difficulties can emerge, when the concatenated MPEG stream has to be VBV compliant or should be a constant bit-rate stream. Apart from the rate control problem, another complication is the encoding of the first B-pictures in a GOP. The encoder requires the reconstructed last P-picture of the preceeding GOP to be available, prior to coding the B-pictures. One solution could be to code all GOPs as closed GOPs. However, this is not desirable, as this would decrease compression efficiency. Therefore, when encoding these B-pictures, the synchronization of the encoders can impose a loss of efficiency.

3.2. SMP parallelization

3.2.1. Model for distributing tasks

The second level of granularity is multithreaded parallelization on multiprocessor SMP machines. Usually, this is achieved by partitioning each picture into segments which are coded independently.[6,7] This partitioning scheme is easy to implement, especially for a hardware implementation, but it has the drawback that not all steps in the coding process can be handled by this scheme. At least input acquisition and bit-stream generation have to be serialized. Other processing steps, such as rate control,[8] are difficult to implement or need to be modified, whereas advanced algorithms, like recursive block-matching[9] for motion estimation, give problems because data dependencies occuring within a single picture are introduced. In a software implementation, only one processor would be active for the case that these serialized tasks are computed. This limits the attainable efficiency in parallelism and execution speed.

Our encoder is based on a generalization of the way the work is partitioned into parallelizable work units. The coding of a picture is split into several functional tasks. In our implementation, each task always operates on a complete picture. The main tasks used are:

- **InitMBs, FillPicHdr and QScale** These tasks are minor initialization functions. They prepare internal data-structures and fill out the invariant fields in various headers.
- **MVFwd and MVBkw** Motion estimation, with separate tasks for forward and backward prediction. These tasks are not used for coding I-pictures, and in P-pictures only MVFwd is used.
- **Difference** The difference picture is constructed using the motion vectors calculated by the preceeding MVFwd and MVBkw tasks. In case of B-pictures, three difference pictures are calculated: one picture using the interpolation of both motion vectors and two pictures based on using the backward or the forward vector only. This information is used for the decision of the MB-coding type later on. After computing the difference pictures, both reference pictures can be removed from the main memory, provided that there are no other dependencies.
- **MBTypes** Macroblock coding type assignment. This decides what coding type is used for each macroblock, depending on the previously constructed difference pictures.
- **DCT/Quant** DCT-transformation, combined with adaptive quantization and rate-control. Although DCT and quantization can be considered as two separate tasks, there are no advantages resulting from splitting them into two tasks.
- **Decode** Decode the previously coded picture. This is required for I- and P-pictures to construct the reference frames for motion prediction.
- **MEstInit** Prepare image data for the motion estimation process. This includes the precalculation of half-pel interpolated pictures for half-pel resolution motion estimation.
- **Bitstream** VLC encoding to a memory buffer. The VLC encoding can be performed in parallel.
- **Write** Store previously VLC encoded picture data to disk. All write tasks are serialized in a way such that the pictures are stored in the correct bit-stream order, which differs from display order.
- **Delete** Remove all picture data from memory.

Naturally, inherent dependencies exist between the tasks. Almost all tasks need computation results of other tasks to perform their own computation. These data dependencies are represented in a *dependency graph*. Figure 3 shows a dependency graph for the coding of an artificial I-B-P sequence. For clarity, the dependencies in the graph are spread over two figures. Figure 3a shows the graph in display order and mainly contains dependencies, which originate from the natural coding process flow and reference picture dependencies. Figure 3b shows the graph in bit-stream order. The most important dependencies shown here are the serialization of the write tasks, taking care of concatenating the bit-stream in the correct picture order, and the dependency of the quantization on the VLC-encoder of the previous picture for rate control.

A task can only be started if all tasks supplying input data for that task have completed. The tasks and the corresponding dependencies are defined in a way that each of the tasks can be computed independently. No communication or synchronization is needed after the task has started. All image data and calculation results are collected in a central data pool, which can be accessed by all threads simultaneously, as the dependency graph based scheduler ensures that all input data is available.

3.2.2. Graph construction

Because the graph gets very complex in general, we additionally implemented a simplification of the graph. We combined the tightly coupled coding tasks into a single **CodeFrame**-task as shown in Figure 4. Although we lose some potential parallelism with this reduction, it was found that the practical loss in efficiency was negligible.

As the graph for coding a video sequence is connected, the encoding process of a continuous video sequence is described by a single, large graph. When scene-change detection is enabled, the GOP structure may vary and

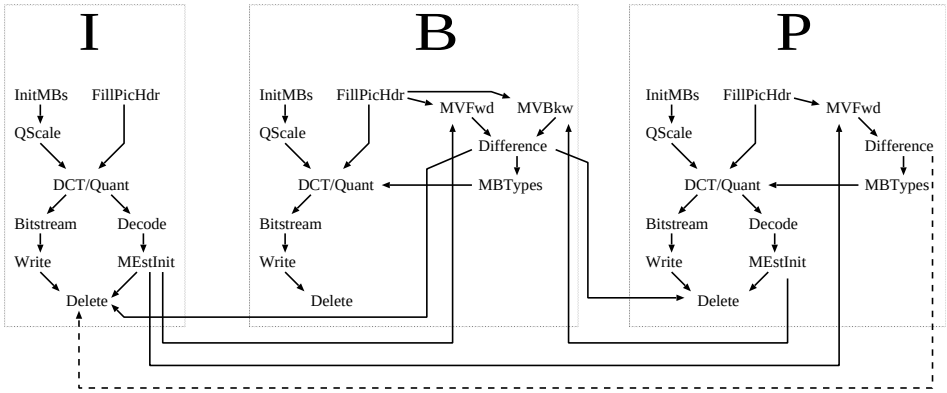
the graph structure is not even periodic any more. Consequently, the dependency graph has to be constructed dynamically during encoding of the video sequence. Only the part of the graph, that is currently in use, is actually present in the memory. The graph is built in two stages. First, a sequential chain of tasks for loading new pictures is created (Figure 5). After analyzing the picture contents, a task assigns picture coding types to each of the pictures. In a second stage, the graphs for actually coding the pictures can be created, as the coding type has been defined.

3.2.3. Scheduling

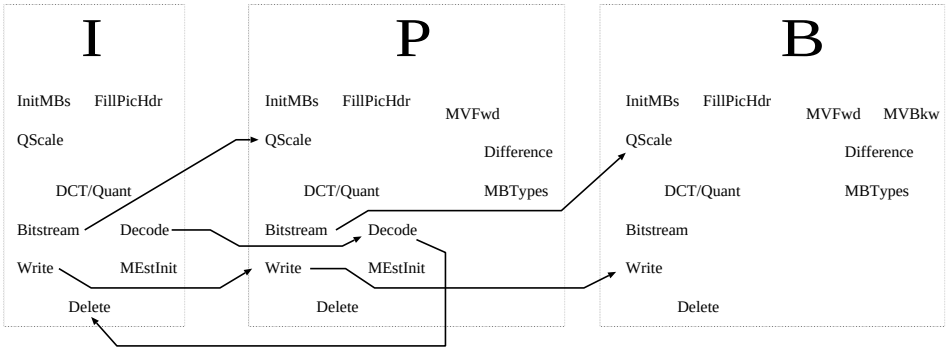
For each processor, a separate thread is started, calling the scheduler at each time when it is idle, in order to acquire a new task. If the scheduler cannot assign a new task, e.g. because it has to wait for the completion of a P-picture, the thread has to wait. All threads are equal and execute this main loop of task acquisition.

Usually, there are more tasks available at a time for computation. Task priorities are assigned to find the most efficient scheduling order of tasks. For example, the coding tasks of I- or P-pictures have higher priorities than those for coding B-pictures, because I- and P-pictures are reference pictures and there are no other tasks depending on the result from coding the B-pictures.

A special problem is the loading of new pictures to be coded into main memory. A high priority for loading would imply that the encoder would load all input frames prior to actual coding. The limited size of the main memory usually prevents this. On the other hand, we want these loading tasks to have a high priority, since loading the pictures cannot be done in parallel and having enough preloaded pictures in memory is a prerequisite for a high speedup. If new pictures would be loaded only when no other tasks are available, all threads would have to wait until



(a) Part one (display order).



(b) Part two (bit-stream order).

Figure 3. Dependency graph for a small I-B-P sequence.

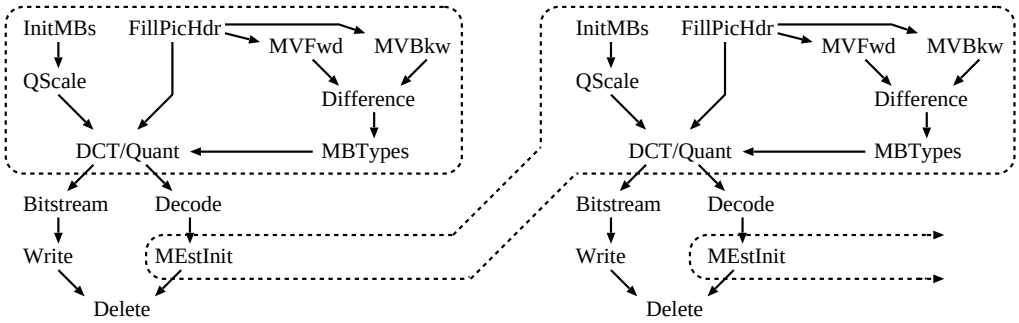


Figure 4. Reduction of tasks by using a combined coding-task.

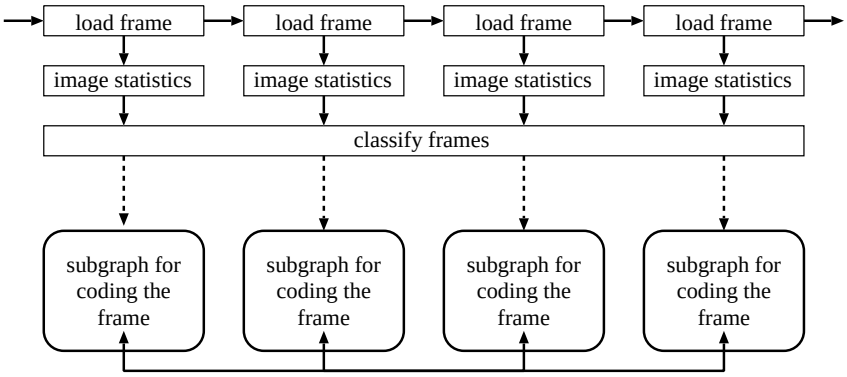


Figure 5. Dependency graph for picture loading and picture coding type assignment.

sufficient pictures would be available to execute the task assigning the coding types. A solution for this dilemma is to create new loading tasks only when the actual number of pictures contained in memory is below some predefined threshold, which is about twice the mean size of a GOP.

3.2.4. Visualization of the encoding process

An example of the encoding process flow is visualized in Figure 6, where the start of a real encoder is executed on 6 CPUs running in parallel. In the experiment, scene-change adaptation was disabled, and the sequence was encoded with a fixed GOP structure of size 20 and a P-distance of 5. Each row in the diagram represents one program thread with time increasing to the right. Each block stands for a coding task, labeled with a task-ID letter and the number of the frame that is processed in the task.

The small white tasks, labeled with **L**, stand for the loading of new images. **I**,**P**,**B** are the coding tasks of the simplified dependency graph for the three coding types. The boxes labeled **D** are the decoding tasks and the very thin dark grey bars with the partially visible **V**, mostly found behind the big coding tasks, represent the VLC encoding tasks. The very thin, light grey tasks located at the beginning and scattered over five processors, are referring to some statistical image analysis which is later used for scene-change detection and image coding; their **S**-label is partially visible. Finally, the tasks, that write the encoded data to disk, are too small to be visible. One can be found in Figure 8, fourth processor, just before B-48. However, they can be easily seen in Figure 8b and 9b as very small lines at the end of coding the GOP.

Figures 8b and 9b portray the same data in a different way. Here the tasks are printed with picture number versus time. The tasks can be identified by their greyscale color. This presentation has the advantage that the relation between GOP structure and parallelization can be noticed clearly.

Let us now analyze Figure 8. A sequence is coded with a fixed GOP structure of size 20 and P-distance 5 running on 8 processors. After coding the initial I-picture and the first P-picture, the encoder starts to code intermediate B-pictures and the next P-picture in parallel. Afterwards, sufficient input pictures are loaded, so that it can start the coding of the next GOP. Note that the coder starts the next GOP even though it has not yet completed the current GOP. After coding the third and last P-picture of the first GOP, 8 B-pictures are available, which can be

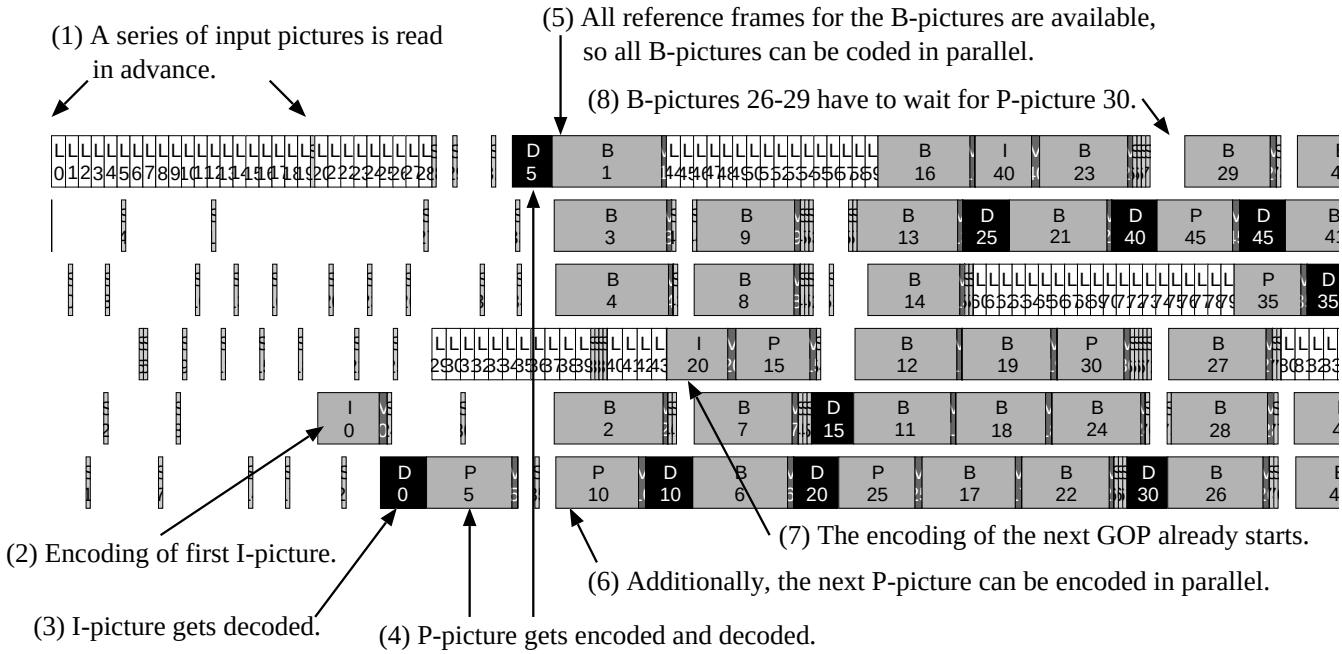


Figure 6. Scheduling on 6 processors. See text for description.

coded simultaneously. Encoding proceeds according to this principle, but due to differing encoding times, the process deviates increasingly from this strategy. The encoder is not dedicated to a fixed scheduling scheme, but rather tries to do anything possible in a greedy manner. However, generally, the main strategy is to code the chain of dependent I- and P-pictures as soon as possible and use the remaining time to code the B-pictures.

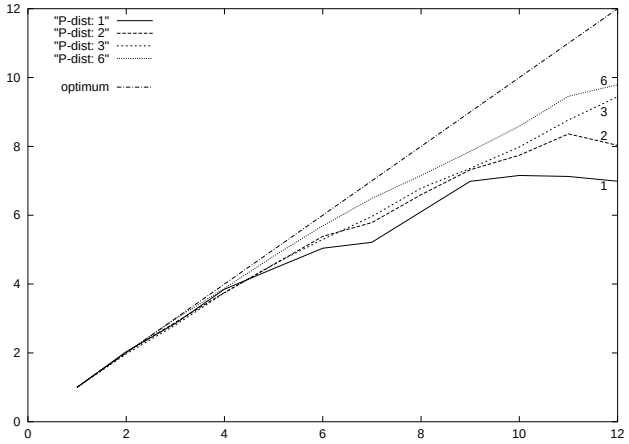
The worst case situation for the proposed picture-based parallelization architecture is the coding of GOPs without B-pictures. In Figure 9, the same video stream was encoded using a GOP structure of one I-picture, followed by 19 P-pictures. Since all P-pictures have to be coded sequentially, the encoder attempts to initiate coding as many GOPs in parallel as possible. Figure 9b portrays that bit-stream generation and the output is delayed until all preceeding GOPs have been coded.

3.2.5. Results on SMP parallelization

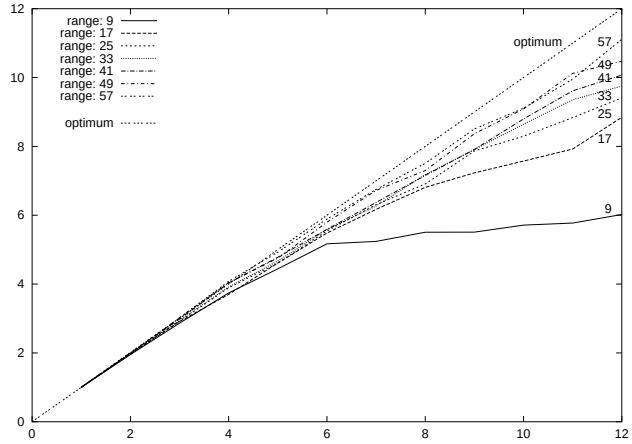
The speed improvement resulting from SMP parallelism was evaluated on a 12-processor UltraSparc computer using varying encoding parameters. For all experiments, a fixed GOP size of 18 pictures and a 2DFS (2-D Full Search) motion-estimation algorithm with a square search range was selected. Despite the use of 2DFS, the encoding times for pictures differ up to a factor of about 2 (see Figure 9a), because our modified 2DFS algorithm does not search areas with prediction errors obviously larger than the optimum.

In the first experiment, we varied the P-distance (Figure 7a), while keeping all other parameters constant. It can clearly be seen that the speedup increases with an increasing number of B-pictures. This is explained by the shorter P-picture chains, so that more B-pictures become available for coding after each finished P-picture. However, even in the case of coding a pure IPPPP-sequence without B-pictures, the encoder parallelized nicely up to about 9 processors for a GOP-size of 18 pictures. The speedup is obtained by reading a larger amount of pictures ahead, thereby enabling the coding of several GOPs at once (see Figure 9).

In a second experiment, we kept the P-distance constant at 6 and varied the motion-estimation search range. The result is shown in Figure 7b. Generally, it can be noticed that the speedup increases with larger search ranges. This behaviour is not obvious, because there is no communication cost between tasks running in parallel so that an increase in motion-estimation time should only increase overall encoding time by a constant factor and not affect speedup. The breakdown in speedup for the small search range of 9 results from slower reading of pictures from disk than encoding proceeds on 6 parallel processes. Since all input pictures were MJPEG compressed, loading times play a significant role.



(a) Speedup depending on P-distance. Motion estimation search range was fixed to 33.



(b) Speedup depending on time spent in motion estimator. The parameter is the motion estimator search range. P-distance is fixed to 6.

Figure 7. Speedup measurements.

This non-obvious speed improvement in Figure 7b can be explained by the following estimation. Let f_B and f_P be the maximum number of B- and P-pictures, respectively, that can be computed per second if there are no locking situations. Since all P-pictures have to be computed sequentially, we can consider one processor exclusively coding P-pictures. It follows that

$$f_P = \frac{1}{t_P} \quad \text{and} \quad f_B = \frac{p-1}{t_B}$$

with t_P and t_B being the coding times and p the number of processors available. Consider now the coding of a GOP structure with $m \geq 1$ B-pictures between successive P-pictures. After coding a single P-picture, which is the reference picture for m B-pictures, these m B-pictures can be coded. We state a condition to be fulfilled such that no locking situation occurs:

$$\text{no locking} \iff \underbrace{f_B}_{\text{computationally possible}} \leq \underbrace{f_P \cdot m}_{\text{B-pictures available for coding}}$$

or equivalently

$$\frac{p-1}{m} \leq \frac{t_B}{t_P}.$$

Hence, as motion estimation on B-pictures approximately takes twice the time compared to compute P-pictures, the more time is spent in the motion estimation, the more $\frac{t_B}{t_P}$ approaches 2. However, in practice, the parallelism is higher, because the size of a GOP is finite and many computationally intensive tasks of pictures belonging to independent GOPs, can be coded in parallel. Only rate-control and bit-stream creation need to be serialized again.

3.3. SIMD parallelization

The last level of parallelization is the utilization of SIMD instructions, which have become common in recent microprocessors under the name of multimedia extensions. For example, the `psadbw` instruction of the MMX2 (Multimedia-Extension) instruction set of IA32 architecture processors computes the sum of absolute differences of 8 pixels in parallel. SAMPEG exploits extensively MMX instructions in the case of IA32 processors and VIS (Visual Instruction Set) instructions in case of UltraSparc processors at computationally expensive parts like motion estimation. We achieved a speedup of the motion estimation algorithms by a factor of 30 compared to scalar C code. Further parts where SIMD instructions are well applicable are DCT, motion compensation and colorspace conversion.

4. CONCLUSIONS

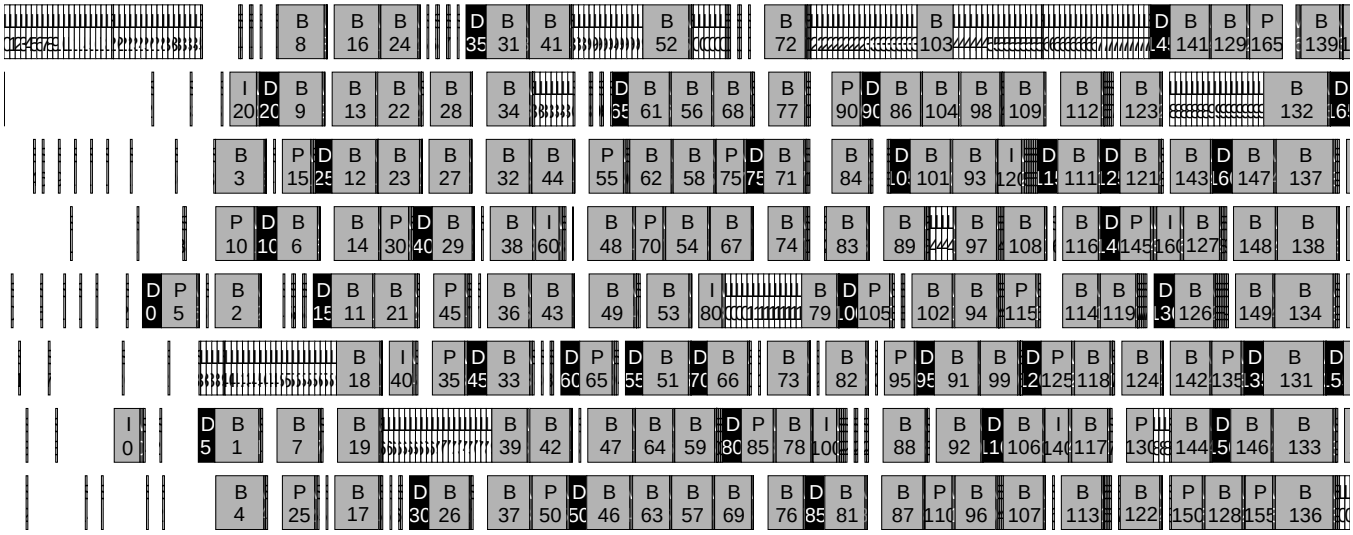
A novel technique for parallel software-based MPEG encoding has been presented. Scene-change detection has been used to adapt the GOP structures to the video contents. This feature improved the image quality due to low-quality coding of the first B-pictures after the scene change so that bits can be saved for the subsequent reference pictures. We introduced an easy, yet robust algorithm for identifying and classifying scene-cuts on the basis of mean luminance values. By selective use of backward and forward prediction and modified quantization near the scene-change, post editing has become easier and the overall image quality is optimized.

We have presented a new concept for parallelization, based on a dynamic scheduling scheme. In contrast with the conventional parallelization techniques based on picture partitioning, we have employed a functional partitioning of the encoding process. Task distribution to the idle processors has been implemented with a fixed priority-driven dynamic scheduler. The advantage of this concept is, that it adapts itself to a wide range of number of processors, GOP structures and differing coding times per picture. The proposed system identifies data dependencies among the tasks and thus relieves all tasks from considering synchronization to other tasks. This also simplifies the development time for integrating new algorithms operating on complete pictures, as parallelization issues can be ignored within the tasks itself.

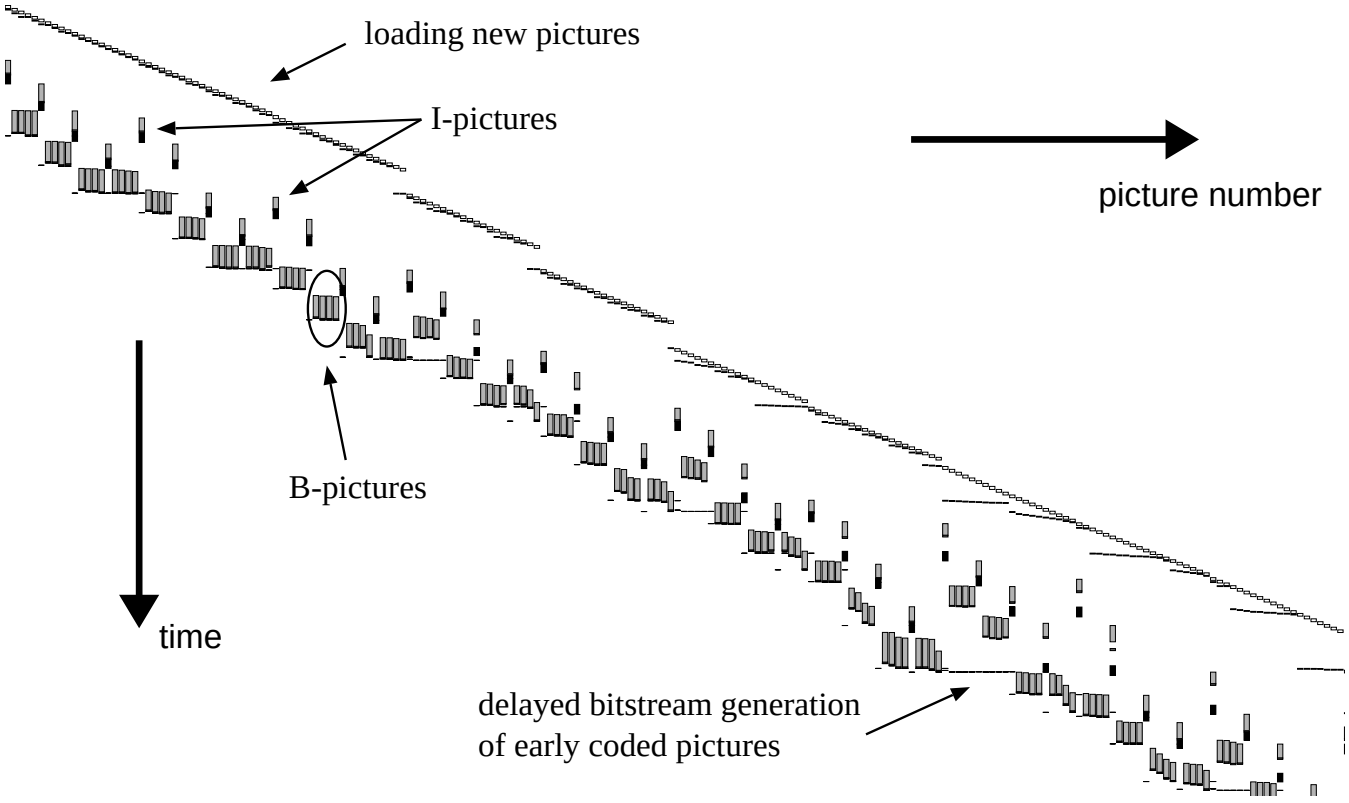
Since the proposed task partitioning is independent from the slice-based partitioning, both can be combined to achieve a further increase of efficiency. The inherent general nature of our architecture provides a framework in which both schemes can be handled uniformly.

REFERENCES

1. M. Sugano, Y. Nakajima, H. Yanagihara, and A. Yoneyama, "A fast scene change detection on MPEG coding parameter domain," in *IEEE International Conference on Image Processing*, vol. 1, pp. 888–892, 1998.
2. M. Wu, W. Wolf, and B. Liu, "An algorithm for wipe detection," in *IEEE International Conference on Image Processing*, vol. 1, pp. 893–897, 1998.
3. S. J. Golin, "New metric to detect wipes and other gradual transitions in video," in *SPIE Visual Communications and Image processing*, vol. 3653, pp. 1464–1474, 1999.
4. K. L. Gong and L. A. Rowe, "Parallel MPEG-1 video encoding," *Picture Coding Symposium, Sacramento, CA* , Sept. 1994.
5. S. Bozóki and R. L. Lagendijk, "Scene adaptive rate control in a distributed parallel MPEG video encoder," in *IEEE International Conference on Image Processing*, vol. 2, pp. 780–783, 1997.
6. K. Suguri, T. Yoshitome, and M. Ikeda, "A scalable architecture of real-time MP@HL MPEG-2 video encoder for multi-resolution video," *SPIE Visual Communications and Image Processing* **3653**, pp. 895–904, Jan. 1999.
7. A. Bilas, J. Fritts, and J. P. Singh, "Real-time parallel MPEG-2 decoding in software," *Proceedings of the 11th International Parallel Processing Symposium (IPPS-97)* , pp. 197–203, 1997.
8. S. Nogaki, "A study on rate control method for MP@HL encoder with parallel encoder architecture using picture partitioning," in *IEEE International Conference on Image Processing*, vol. 4, pp. 261–265, 1999.
9. P. H. N. de With, "A simple recursive motion estimation technique for compression of HDTV signals," in *4th IEE International Conference on Image Processing & its Applications*, pp. 417–420, 1992.

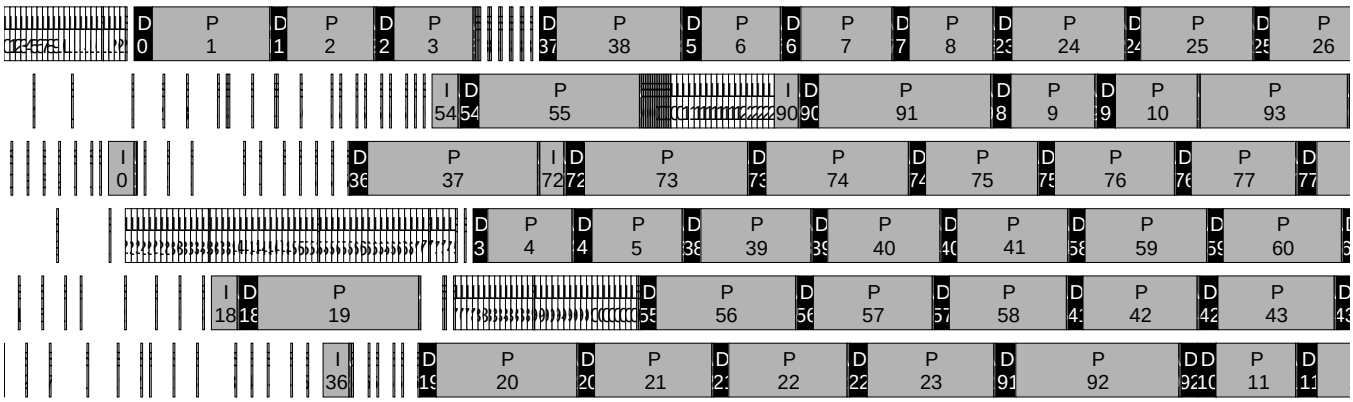


(a)

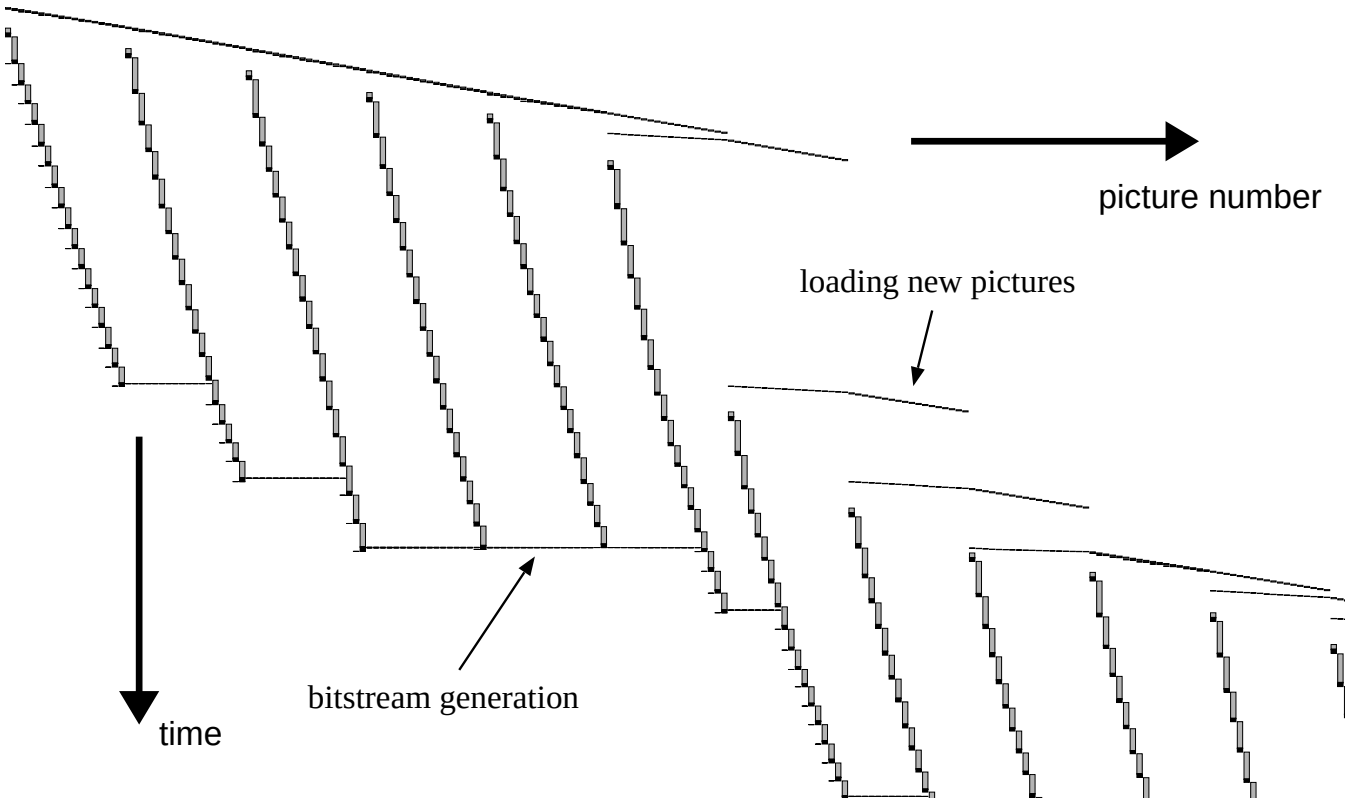


(b)

Figure 8. Scheduling of an IBBBBPBBBBP... sequence on 8 processors.



(a)



(b)

Figure 9. Scheduling of an IPPPPP-stream on 6 processors.