Timing constraints of MPEG-2 decoding for high quality video: misconceptions and realistic assumptions

Damir Isović, Gerhard Fohler
Department of Computer Engineering
Mälardalen University, Västerås, Sweden
{damir.isovic,gerhard.fohler}@mdh.se

Liesbeth Steffens
Information Processing Architectures
Philips Research The Netherlands, Eindhoven
liesbeth.steffens@philips.com

Abstract

Decoding MPEG-2 video streams imposes hard real-time constraints for consumer devices such as TV sets. The freedom of encoding choices provided by the MPEG-2 standard results in high variability inside streams, in particular with respect to frame structures and their sizes.

In this paper, we identify realistic timing constraints demanded by MPEG-2 video decoding. We present results from a study of realistic MPEG-2 video streams to analyze the validity of common assumptions for software decoding and identify a number of misconceptions. Furthermore, we identify constraints imposed by frame buffer handling and discuss their implications on decoding architecture and timing constraints.

1 Introduction

The Moving Picture Experts Group (MPEG) standard for coded representation of digital audio and video [1], is used in a wide range of applications. In particular MPEG-2 has become the coding standard for digital video streams in consumer content and devices, such as DVD movies and digital television set top boxes for Digital Video Broadcasting (DVB). MPEG encoding has to meet diverse demands, depending, e.g., on the medium of distribution, such as overall size in the case of DVD, maximum bitrate for DVB, or encoding speed for live broadcasts. In the case of DVD and DVB, sophisticated provisions to apply spatial and temporal compression are applied, while a very simple, but quickly coded stream will be used for the live broadcast. Consequently, video streams, and in particular their decoding demands will vary greatly between different media.

The encoded content has to be decoded and played out. Decoding can be performed in hardware or in software, or, as in most practical systems, in a mix of both. Both dedicated and programmable decoders can be based on average-case requirements if they provide means to gracefully handle overload situations. If not, both must support worst-case requirements. However, in a software implementation, it is possible to use the slack on the processor for other applications in average case. With dedicated hardware, there are no such possibilities. As a consequence, the behavior of a software decoder will be less regular than that of a dedicated hardware decoder. Coping with these irregularities is one of the objectives dealt with in this article.

While in the simplest case of sufficient resources, MPEG decoding is straight forward, i.e., simply a matter of transmitting and decoding to display frames with the required frequency, the considerable variations in the streams render such approaches too costly for many cases. If the processor cannot work fast enough to decode all the frames, the decoder has to speed up. There are two ways to do this: quality reduction, and frame skipping. With the quality reduction strategy, the decoder reduces the load by using a downgraded decoding algorithm, while frame skipping means that not all frames are decoded and displayed, i.e., some of the frames are skipped. In this paper, we focus on the frame skipping approach. Frame skipping can be used sparingly to compensate for sporadic high loads, or it can be used frequently if the load is structurally too high.

Many algorithms for software decoding of MPEG video streams use buffering and rate adjustment based on average-case assumptions. These provide acceptable quality for applications such as video transmissions over the Internet, when drops in quality, delays, uneven motion or changes in speed are tolerable. However, in high quality consumer terminals, such as home TVs, quality losses of such methods are not acceptable. In fact, producers of such devices have argued to mandate the use of hard real-time methods instead [4]. A server based algorithm for integrating multimedia and hard real-time tasks has been presented in [2]. It is based on average values for execution times and interarrival intervals. A method for real-time scheduling and admission control of MPEG-2 streams that fits the need for adaptive CPU scheduling has been presented in [7]. The method is not computationally overloaded, qualifies for continuous reprocessing and guarantees QoS. However, no consideration on making priorities on the B frame level has been done.

It is difficult to predict WCET for decoding parts. MPEG-2 can use different bitrates which can result in large differences in decoding times for different streams. This could lead to big overestimations of the WCETs. Work on predicting MPEG execution times has been presented in [3, 5]. Most standard real-time schedulers fail to satisfy the demands of MPEG-2 as they do not consider the specifics of this compression standard.

In this paper, we derive realistic timing constraints for MPEG-2 video decoding. We analyze realistic MPEG streams and match the results with common assumptions about MPEG, identifying a number of misconceptions. The correct assumptions are needed to identify realistic timing constraints for MPEG processing. Even frame skipping needs appropriate assumptions to be effective. Dropping the wrong frame at the wrong time can result in a noticeable disturbance in the played video stream. We discuss frame buffer handling and its impact on decoding design and temporal requirements. Based on correct assumptions, we provide guidelines for real-time MPEG processing, such as choosing buffer sizes and latency to derive the appropriate timing constraints. These constraints call for novel scheduling algorithms to appropriately meet the exact constraints without quality loss due to misconceptions about the stream characteristics.

2 Playing MPEG streams

In this section we present the main characteristics of MPEG-2 video stream and give an overview how the stream is processed, i.e., buffering, decoding and displaying.

2.1 MPEG-2 video stream characteristics

A complete description of the MPEG compression scheme is beyond the scope of this paper. For details on MPEG see e.g., [1, 14, 13]. The text presented in this subsection is summarized in figure 1.

Frame types - The MPEG-2 standard defines three types of frames, I, P and B. The I frames or *intra* frames are simply frames coded as still images. They contain absolute picture data and are self-contained, meaning that they require no additional information for decoding. I frames have only spatial redundancy providing the least compression among all frame types. Therefore they are not transmitted more frequently than necessary.

The second kind of frames are P or *predicted* frames. They are forward predicted from the most recently reconstructed I or P frame, i.e., they contain a set of instructions to convert the previous picture into the current one. P frames are not self-contained, meaning that if the previous

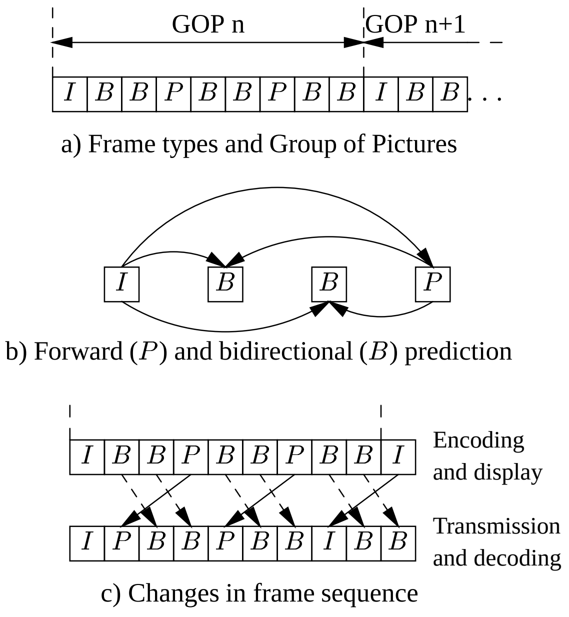

GOP n
GOP n+1
I B B P B B P B B I B B . . .
a) Frame types and Group of Pictures
I B B P
b) Forward (P) and bidirectional (B) prediction
I B B P B B P B B I
Encoding and display
I P B B P B B I B B
Transmission and decoding
c) Changes in frame sequence

Figure 1. MPEG-2 video stream

reference frame is lost, decoding is impossible. On average, P frames require roughly half the data of an I frame, but our analysis also showed that this is not the case for a significant number of cases.

The third type is B or *bi-directionally* predicted frames. They use both forward and backward prediction, i.e., a B frame can be decoded from a previous I or P frame, and from a *later* I or P frame. They contain vectors describing where in an earlier or later pictures data should be taken from. They also contain transformation coefficients that provide the correction. B frames are never predicted from each other, only from I or P frames. As a consequence, no other frames depend on B frames. B frames require resource-intensive compression techniques such as Motion Estimation but they also exhibit the highest compression ratio, on average typically requiring one quarter of the data of an I picture. Our analysis showed that this does not hold for a significant number of cases.

Group of Pictures - Predictive coding, i.e., the current frame is predicted from the previous one, cannot be used indefinitely, as it is prone to error propagation. A further problem is that it becomes impossible to decode the transmission if reception begins part-way through. In real video signals, cuts or edits can be present across which there is little redundancy. In the absence of redundancy over a cut, there is nothing to be done but to send from time to time a new reference picture information in absolute form, i.e., an I frame.

As I decoding needs no previous frame, decoding can begin at I coded information, for example, allowing the viewer to switch channels. An I frame, together with all of the frames before the next I frame, form a *Group of Pictures (GOP)*. The GOP length is flexible, but 12 or 15 frames is a common value. Furthermore, it is common industrial practice to have a fixed pattern (e.g., $I\ BB\ P\ BB\ P\ BB\ P\ BB$). However, more advanced encoders will attempt to optimize the placement of the three frame types according to local sequence characteristics in the context of more global characteristics. Note that the last B frame in a GOP requires the I frame in the next GOP for decoding and so the GOPs are not truly independent. Independence can be obtained by creating a *closed* GOP which may contain B frames but ends with a P frame.

Transmission order - As mentioned above, B frames are predicted from two I or P frames, one in the past and one in the future. Clearly, information in the future has yet to be transmitted and so is not normally available to the decoder. MPEG gets around the problem by sending frames in the "wrong" order. The frames are sent out of sequence and temporarily stored. Figure 1-c shows that although the original frame sequence is $I\ BB\ P\ ...$, this is transmitted as $I\ P\ BB\ ...$, so that the future frame is already in the decoder before bi-directional decoding begins. Picture reordering requires additional memory at the encoder and decoder and delay in both of them to put the order right again. The number of bi-directionally coded frames between I and P frames must be restricted to reduce cost and minimize delay, if delay is an issue.

2.2 MPEG-2 video processing

In its simplest form, playing out an MPEG video stream requires three activities: input, decoding, and display. These activities are performed by separate tasks, which are separated by input buffer and a set of frame buffers.

The input task directly responds to the incoming stream. It places en encoded video stream in the input buffer. In the simple case, this input activity is very regular, and only determined by the fixed bit rate. In a more general case, the input may be of a more bursty character due to an irregular source (e.g. the Internet), or it may have a varying input rate due to a varying multiplex in the transport stream. We assume that the video data is placed in the input buffer with a constant bitrate.

The decoding task decodes the input data and puts the decoded frames in the frame buffers. If sufficient buffer space is available, it may work asynchronously, spreading the load more evenly over time. Its deadline is determined by the requirements of the display task.

The display task is IO bound, and often performed by a dedicated co-processor. It is driven by the refresh rate of the screen. The display task, once started, must always find a frame to be displayed. In the simple case, the display rate equals the frame rate, but we will also consider situations where the display rate is higher than the frame rate.

3 Analysis of realistic MPEG streams

In this section we present an analysis of MPEG-2 video streams taken from original DVDs.

3.1 The analysis

We have analyzed 12 realistic MPEG streams and matched our results with the common MPEG assumptions Since some video contents are more sensitive for quality reduction than others [11], we have analyzed different types of movies; action movies, dramas, and cartoons. Due to space limitations we report only representative results for selected DVD movies. The complete results for all analyzed movies can be found in [9].

3.2 Simulation environment

The MPEG video streams have been extracted from original DVD movies. To extract the data out of an MPEG video stream, we have implemented a C-program. The decoding execution time measurements were performed on several PC computers, with different CPU speed (in the range 0.5-2.0 GHz). The time for measuring decoding execution times was equivalent to the length of the movies.

3.3 Analysis results

GOP and frame size statistics of the selected movies are presented in table 1. We have also analyzed the relations between frame sizes on the individual GOP basis, see table 2. Furthermore, we have measured the decoding times for different frame types, see figure 2.

3.4 Common assumptions about MPEG

Here we present some common assumptions about MPEG and match them with our analysis results. We have looked into stream assumptions (1-4), frame size assumptions (5-8), and a decoding time assumption (9).

Assumption 1: - *The sequence structure of all GOPs in the same video stream is fixed to a specific I,P,B frame pattern*. This is not true. For example, in 18% of the GOPs in the action movie the GOP length was not 12 frames. Not all GOPs consist of the same fixed number of P and B frames following the I frame in a fixed pattern. That is because more advanced encoders will attempt to optimize the placement of the three picture types according to local sequence characteristics in the context of more global characteristics.

Genre	Avg I:P:B size ratio	Nr of frames	I frames			P frames			B frames		
			min	max	average	min	max	average	min	max	average
action	4:2:1	179412	11	247073	63263	2	152000	29352	4	96131	18525
drama	6:3:2	173054	17	183721	58985	4	126229	28893	4	79552	19054
cartoon	6:2:1	121406	7178	140152	84318	159	137167	31943	159	111405	14398

Table 1. Frame size statistics for selected analyzed MPEG streams (in bytes)

Genre	Open GOPs	Closed GOPs	Standard GOP length	Number of GOPs where					
				I largest	P largest	B largest	$P > I$	$B > I$	$B > P$
action	83%	17%	82%	90%	9%	1%	9%	5%	39%
drama	98%	2%	92%	94%	5%	1%	6%	3%	37%
cartoon	99%	1%	98%	92%	7%	1%	8%	1%	12%

Table 2. GOP statistics

Assumption 2: - *MPEG streams always contain B frames.* Not true. We have been able to identify MPEG streams that contain only I and P frames (IPP), or even only the I frames in some rare cases. I frame only is an older MPEG-2 technology that does not take advantage of MPEG-2 compression techniques. The IPP technology provides high quality digital video and storage, making it suitable for professional video editing. B frames provide the highest compression ratio, making the MPEG file smaller and hence more suitable for video streaming, but if the file size is not an issue, they can be excluded from the stream.

Assumption 3: - *All B frames are coded as bi-directional.* This is not true. There are B frames that do have bi-directional references, but in which the majority of the macroblocks are I blocks. If the encoder cannot find a sufficiently similar block in the reference frames, it simply creates an I block.

Assumption 4: - *All P frames contribute equally to the GOP reconstruction.* Not true. The closer the P frame is to the start of the GOP, the more other frames depend on it. For example, without the first P frame in the GOP, P_1, it would be impossible to decode the next P frame, P_2, as well as all the B frames that depends on both P_1 and P_2. In other words, P_2 depends on P_1, while the opposite is not the case.

Assumption 5: - *I frames are the largest and B frames are the smallest.* This assumption holds on average. In all the movies that we analyzed, the average sizes of the I frames were larger than the average sizes of the P frames, and P frames were larger than B frames on average. However, our analysis showed that this assumption is not valid for a significant number of cases. For example, in the action movie we have a case with 9% GOPs in which P have the largest size, and 1% of GOPs where a B frame is the largest one (see table 2), which corresponds roughly to 8 and 1 minutes respectively in a 90 minute film. Such deviations from average cannot be ignored.

Assumption 6: - *An I frame is always the largest one in a GOP.* This is not true. For example in the action movie the I frame was not the largest in 12% of the cases (in 9% of the cases some P frame was larger than the I frame, and in 3% of the GOPs, a B frame was larger than the I frame).

Assumption 7: - *B frames are always the smallest ones in a GOP.* Not true. For example, in the drama movie, a B frame was larger than the I frame in 3% of the cases, and larger than a P frame in 37% of the cases. As a consequence, even the assumption that P frames are always larger than B frames is also not valid.

Assumption 8: - *I,P and B frame sizes vary with minor deviations from the average value of I,P and B.* Not true. In the action movie, B frame sizes vary greatly around an average of 18525 bytes. The interval between 0.5 and 1.5 of average holds only some 60% of frames.

Assumption 9: - *Decoding time depends on the frame size and it is linear.* While some results on execution times for special kinds of frames have been presented, e.g., [5], a (linear) relationship between frame size and decoding time cannot be assumed in general. Our analysis shows, that the relation between frame size and decoding follows roughly a linear trend. The variations in decoding times for similar frame sizes, however, are significant for the majority of cases, e.g., in the order of 50-100% of the minimum value for B frames. As expected, the frame types exhibit varying decoding time behavior (see figure 2): I frames vary least, since the whole frame is decoded with few options only. On the other hand, B frames, utilizing most compression options, vary most.

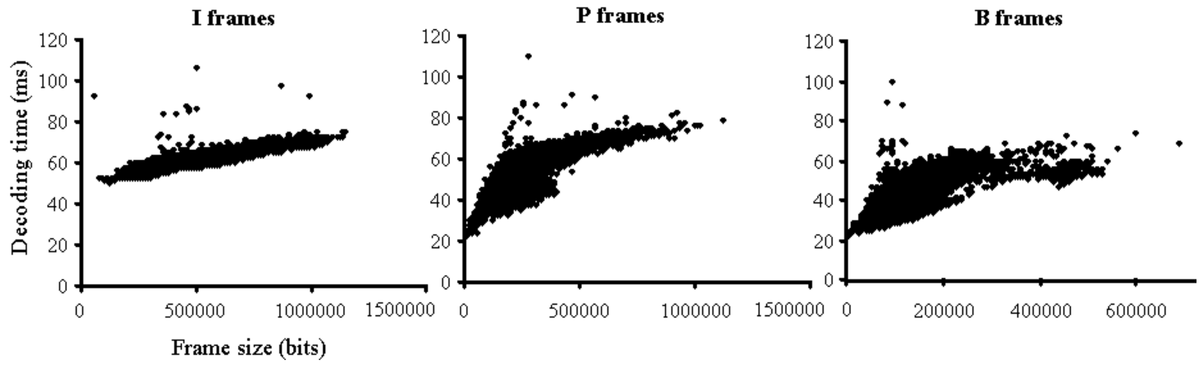


Figure 2. Decoding execution times as a function of frame bitsize

4 Latency and buffer requirements

The input, decoding and display tasks are separated by buffers: one input buffer used for storing the input video bit-stream data, and a frame buffer space that contains at least two frame buffers. In this section we describe system latency and buffer requirements.

4.1 Latency

Once we start to play out an MPEG stream, the *end-to-end latency* is fixed and it is measured from the arrival of the first bit at the input task to the display of the first pixel or line on the screen. If this latency is not fixed, the system cannot work correctly over time. The end-to-end latency is the sum of the *decoding latency*, and the *display latency*, which are not necessarily fixed. The initial decoding latency is measured from the arrival of the first bit at the input task to the reading of the first bit of the first frame, after the header, by the decoder. The initial display latency is measured from the reading of the first bit of the first frame, after the header, by the decoder, to the display of the first pixel or line on the screen. If the decoding task is strictly periodic, the decoding and display latencies are constant. If the decoder is asynchronous, i.e. if its activity is determined by the buffer fillings, the decoding and display latencies can vary.

4.2 Input buffer requirements

The input buffer serves several purposes. First, it has to compensate for the irregular data size. This irregularity is bounded, and the bounding is encoded in the stream, in the form of a parameter called *VBVbuffer size*, see MPEG video standard [1]. VBV stands for Video Buffering Verifier, a hypothetical decoder that starts when the first frame has completely arrived in its input buffer, and retrieves a complete encoded frame out of the input buffer at the start of a new frame period. The contents of the VBV input buffer never exceeds VBV buffer size. Figure 3 depicts the time lines and the buffer occupancy for a reference decoder that corresponds to the VBV. It shows minimum decoding latency,

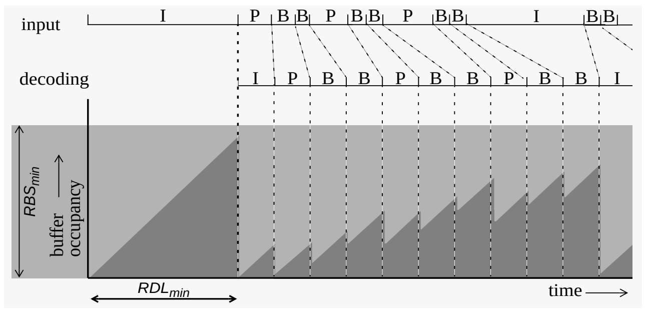


Figure 3. Minimum decoding latency

RDL_{min}, and minimum buffer size, RBS_{min} at the start of a new stream. The time lines represent the input and decoding tasks, respectively. Because of the fixed bit rate, BR, the duration of inputting one picture is directly proportional to the number of bits this picture takes up in the encoded stream. The buffer occupancy rises linearly during the decoding of each frame, and drops vertically at the start of a new frame, when the picture data are removed from the input buffer. The buffer occupancy is zero when the first picture has just been removed from the input buffer.

Second, the input buffer has to compensate for varying decoding times, which are not foreseen by the encoder. Therefore, this compensation cannot be bounded a priori.

Third, a realistic decoder retrieves the data from the input buffer according to its processing. The resulting non-zero retrieval time relaxes the buffer requirement, but can also not be bounded a priori. Therefore, the input buffer size is essentially a design choice, closely related to the initial decoding latency and the desired end-to-end latency. Once the size of the input buffer is chosen, the maximum decoding latency (RDL_{max}) is fixed: $RDL_{max} = IBS/BR$, where

IBS stands for the input buffer size.

4.3 Frame buffers requirements

The frame buffers serve a dual purpose. They serve as reference buffers for the decoder and as input buffers for the display task, or output buffer for the decoding task. It is possible that a certain frame buffer is used in both capacities at the same time. This makes frame buffer management somewhat more complicated than input buffer management. The display task cannot start until the first frame has been placed in the output buffer, and does not release the current output buffer until a second output buffer is available (double buffering scheme). In this way, the display task always has a frame to display. If the stream contains two or more B frames in sequence, the minimum number of frame buffers needed is 4: two for the reference frames, one for the B frame being displayed, one for the B frame being decoded.

The use of four frame buffers allows a certain irregularity in the delivery of output frames by the decoder. Figures 4 and 5 depict the behavior of a regular reference decoder, which takes exactly one frame period to decode a frame. In the first period in figure 4, a new I frame is being decoded in frame buffer $FB1$. This I frame is needed to decode the B frames B_7 and B_8 (that belong to the previos GOP but are being transmitted after the I frame which is their backward reference frame). In the next period, B_7 can be decoded, and in the third period, B_7 can be displayed, while B_8 is being decoded. If B_7 is the n-th frame to be displayed, it is the $(n+1)$-th frame to be decoded. Therefore, the minimum display latency equals *two* frame periods. If there are no B frames, there is no frame reordering, and the minimum display latency will be one frame period instead of two. In

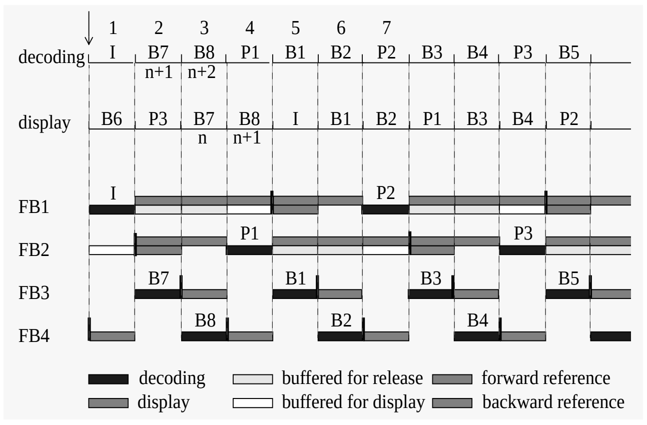


Figure 4. Minimum display latency

figure 4, the decoding cannot be done with less than four frame buffers, but these four frame buffers do allow a larger display latency. Figure 5 depicts a situation in which the display latency is maximised. The B frames are displayed not when they are completely decoded, but when the buffer is needed to decode the next frame. Now the n-th frame is being displayed while the $(n+3)$-th frame is being decoded, i.e. the display latency equals three frame periods. Thus the display latency is bounded between the minimum of two frame periods and a maximum of three frame periods.

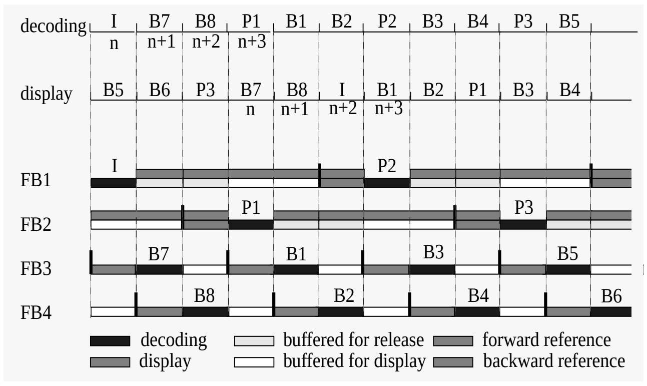


Figure 5. Maximum display latency

4.4 Buffer overflow and underflow

Since the decoder is asynchronous, there is a risk of buffer overflow and buffer underflow. Input underflow, and frame buffer overflow occur when the decoder is too fast, i.e when the decoding latency is too small and/or the display latency too large. They can easily be prevented by synchronization. The decoder is blocked until the input and/or output task catches up. Input overflow and output underflow occur when the decoder is too slow, i.e. when the decoding latency is too large and/or the display latency is too small. In case of output underflow, the display does not have a new frame to display, but this has been foreseen by retaining the previous frame for display until a new one arrives. Input overflow can be much more serious. In some cases, the input can be delayed, e.g. in case of a DVD player. In other cases, the input task cannot be blocked, especially in case of a broadcast input, where the input buffer must be made large enough to accommodate at least the variation that is allowed by the frame buffers. This will be discussed in more detail in the next section.

5 End-to-End flow control

The latency variation allowed is a design decision, based on the maximum allowed end-to-end latency, and the available buffer space. If the processor cannot work fast enough to meet the time constraints, the decoder has to speed up. There are two ways to do this: quality reduction, and frame skipping. Whichever strategy is chosen, we assume that the system organisation is such that the display task is never without data to display. This is not difficult to achieve. If a decoded frame does not arrive on time, and the display task has to redisplay the previous frame, this is a deadline miss for the decoder. With the given arrangement deadline misses have a penalty, in the form of a perceived quality reduction. Moreover, since the frame count has to remain consistent, the decoder must skip one frame.

5.1 Quality reduction

With the quality reduction strategy, the decoder reduces the load by using a downgraded decoding algorithm. Quality reduction for MPEG decoding and other video algorithms is discussed in [12], [17], [8], and [10]. This approach has two advantages over frame skipping. In general the decoding load is higher when there is more motion, but in that case, skipping frames may be more visible than reducing the quality of individual pictures. Moreover, quality reduction can be more subtle, whereas skipping frames is rather coarse grained. Control strategies for fine-grained control based on scalable algorithms are proposed in [15] and [16]. These control strategies use a mixture of preventive quality reduction and reactive frame skipping. The main disadvantage of the quality reduction approach is that it requires algorithms that can be downgraded, with sufficient quality levels to allow smooth degradation. Such algorithms are not yet widely available.

5.2 Frame skipping

Frame skips speed up the decoder, and increase the display latency, like a throttle. Unfortunately, the corrective step is rather coarse grained: the display latency is increased by a complete frame period. If the range of allowable display latencies is not large enough, this may lead to oscillation, in which frame skips and bounces on frame buffer overflow both are very frequent.

Frame skipping does not come for free. At the very least, the start of the new frame has to be found and the intermediate data have to be thrown away. There are two forms of frame skips, reactive and preventive.

A *reactive frame skip* is a frame skip at or after a deadline miss to restore the frame count consistency. In case of a deadline miss, there are two options, aborting the late frame, which is probably almost completely decoded, or completing the late frame, and skipping the decoding of a later frame. The effects of an abortion and of a reactive frame skip on the display latency are shown in figures 6 and 7. In the former case, the display latency stays low, and a next deadline miss is to be expected soon. In the latter case, the display latency is drastically reduced, because the decoder will be blocked due to output buffer overflow. An additional frame buffer would give more freedom, and a more stable system, at the cost of using additional memory.

In both cases, we have to make sure that the input buffer is large enough to allow the minimal display latency.

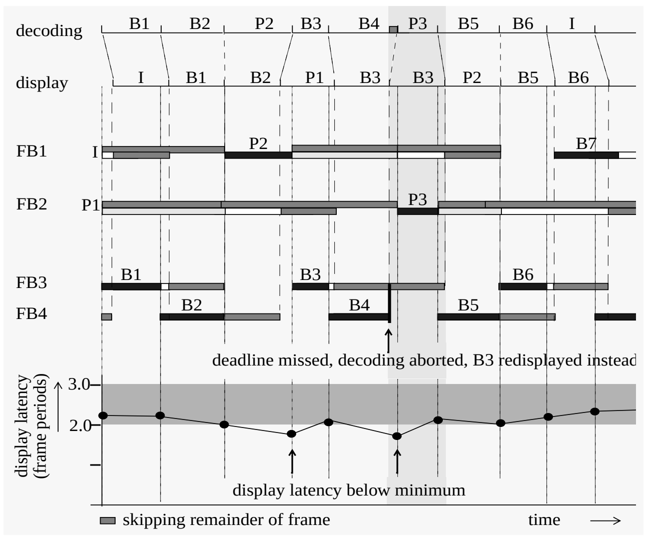

decoding
B1
B2
P2
B3
B4
P3
B5
B6
I
display
I
B1
B2
P1
B3
B3
P2
B5
B6
FB1
I
P2
B7
FB2
P1
P3
FB3
B1
B3
B6
FB4
B2
B4
B5
deadline missed, decoding aborted, B3 redisplayed instead
display latency (frame periods)
3.0
2.0
display latency below minimum
skipping remainder of frame
time

Figure 6. Deadline missed - frame aborted

A *preventive frame skip* preventively increases the display latency. The effect of a preventive frame skip on the display latency is depicted in figure 8. The decision to skip preventively is taken at the start of a new frame, and is based on an measurement of the lateness of the decoder.

5.3 Criteria for preventive frame skipping

Not all the frames are equally important for the overall video quality. Dropping some of them will result in more degradation than others. Here we identify some criteria to decide the relative importance of frames.

Criterion 1: - *Frame type.* According to this criterion, the I frame is the most important one in a GOP since all other frames depend on it. If we lose an I frame, then the decoding of all consecutive frames in the GOP will not be possible. B frames are the least important ones because they are not reference frames. If we would apply this criterion only, then we would pull out all B frames first, then P frames and finally the I frame.

Criterion 2: - *Frame position in the GOP.* This is applied to P frames. Not all P frames are equally important. Skipping a P frame will cause the loss of *all* its subsequent frames, and the two preceding B frames within the GOP. For instance, skipping the first P frame (P_1) would make it impossible to reconstruct the next P frame (P_2), as well as all B frames that depends on both P_1 and P_2. And if we skip P_2 then we cannot decode P_3 and so on.

Criterion 3: - *Frame size.* Applies to B frames. According to the previously presented analysis results, there is

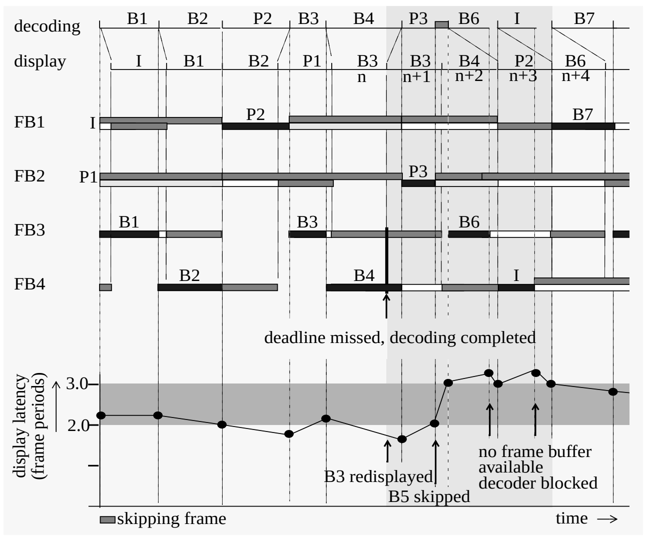
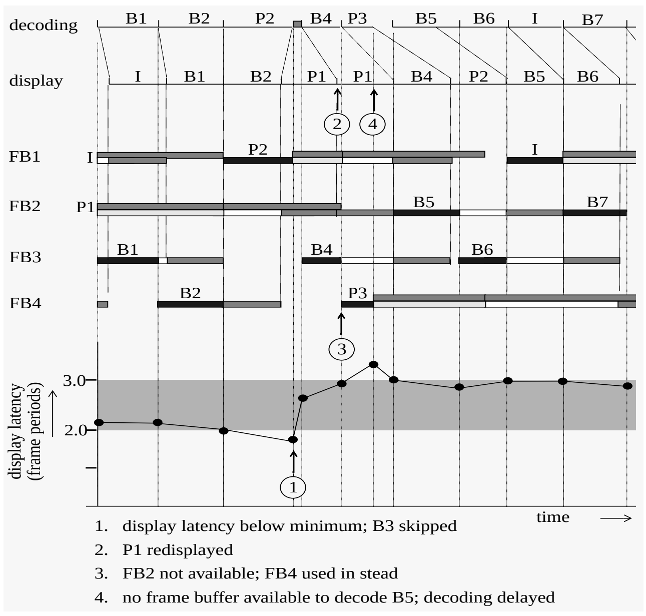

decoding
display
FB1
FB2
FB3
FB4
deadline missed, decoding completed
display latency (frame periods)
B3 redisplayed
B5 skipped
no frame buffer available decoder blocked
skipping frame
time

Figure 7. Deadline missed - subsequent frame skipped



1. display latency below minimum; B3 skipped
2. P1 redisplayed
3. FB2 not available; FB4 used in stead
4. no frame buffer available to decode B5; decoding delayed

Figure 8. Preventive frame skipping

a relation between frame size and decoding time, and thus between size and gain in display latency. The purpose of skipping is to increase display latency. So, the bigger the size of the frame we skip, the larger display latency obtained.

Criterion 4: - *Skipping distribution.* With the same number of skipped B frames, a GOP with *evenly* skipped B frames will be smoother than a GOP with uneven skipped B frames, since the picture information loss will be more spread [11].

Criterion 5: - *Buffer size.* There is no point in having a nice skipping algorithm without having sufficient space to store input data and decoded frames.

Criterion 6: - *Latency.* An algorithm that takes entire GOP into account requires a large end-to-end latency, and corresponding buffer size.

When deciding the relative importance of frames for the entire GOP, we could assign values to them according to all criteria collectively applied, rather than applying a single criterion. Since the criterion 1 is the strongest one, the I frame will always get the highest priority, as well as the reference frames in the beginning of the GOP, while in some cases we would prefer to skip a P frame towards the end of the GOP than a big B frame close to the GOP start.

6 Timing Constraints

Timing constraints for an MPEG video decoder stem from roughly three sources:

First, *the MPEG stream*, in particular frame ordering and their dependencies, poses mostly *relative* constraints.

Second, *the display rate*, related to the refresh rate of the screen, defines mostly *absolute* constraints. It depends on hardware characteristics, which in turn define when a picture should be ready to be displayed. Consumer TV sets typically have refresh rates of 50, 60, or 100Hz, computer screens may have more diverse values.

Third, the frame buffers incur *resource and synchronization constraints*. The number and handling of frame buffers depends on hardware and architecture design, i.e., the constraints will be implementation dependent. Therefore we do not include specific constraints, which would change with design decisions.

6.1 Start time constraints

The earliest time at which decoding a frame can begin is the earliest point in time at which all of the following start time conditions, *STC*, hold.

STC1: *Frame header parsed and analyzed.*

STC2: *For B and P frames: the decoding completion time of the forward / backward reference frame.*

STC3: *Frame data available in input buffer.* The cumulative input time of frame i is calculated as:

$$CIT(i) = \sum_{j=1}^{i} \frac{fs(j)}{BR(j)}$$

with $fs(j)$ the frame size of frame j, and $BR(j)$ the bitrate of frame j. BR(j) and fs(j) are available from the frame header

STC4: *Free frame buffer available.* This is always naturally true for reference frames: they require at least two buffers, see section 4. When a new reference frame is being decoded, at most one of them is needed for reference. As a consequence, for reference frames, STC4 becomes true one frame period earlier than it would for B frames.

The last two constraints are necessary for unblocked video stream processing.

6.2 Completion time constraints

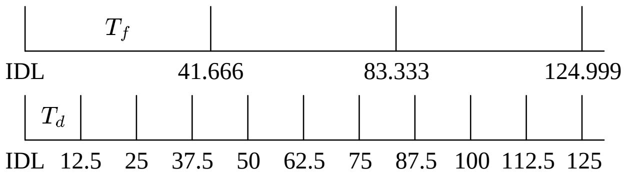
The latest time at which decoding a frame has to be completed is the earliest point in time at which any of the following latest time conditions, *LTC*, holds:

LTC1: *Display time of the frame*. If we have a TV set displaying a broadcast stream (DTV), the input frame rate is equal to the display frame rate: 50 - 60 Hz, depending on the region. Other input streams may have different frame rates, and other displays may have different display rates.

If the display rate is an integer multiple of the input rate, the solution is simple, re-display the frame several times.

If this is not the case, things are more complicated. Here is an example: assume that we have an input frame rate of 24 Hz (original film material), and a display rate of 80 Hz (computer display). In this case, the frame period, T_f, is 1/24 = 41.666 ms, whereas the display period, T_d, is 1/80 = 12.5 ms.

Let IDL denote *initial display latency*, i.e., the display time of the first frame, as described in section 4.1. The first frame starts latest at time IDL, the second one at IDL+41.666, and so on, as illustrated bellow:



Since the decoder task is not in phase with the display task, the decoding deadline for each frame will occur between two display deadlines, e.g., the decoding deadline for the second frame, IDL+41.666, will be in between the display deadline IDL+37.5 and the display deadline IDL+50 (se figure above).

Let L denote the closest display instance from left (in this example, the one with the deadline IDL+37.5), and R the closes one from right (IDL+50 in the example). There are two ways to display frames.

Approach 1 - Always postpone, i.e., use the display instance R to display the decoded frame. In this case, the required display time, RDT, of a frame f_i^j, where i is the decoding number, and j the display number of the frame is given by:

$$RDT(f_i^j) = IDL + \left\lceil \frac{(j-1)T_{fr}}{T_{dis}} \right\rceil T_{dis}$$

For the example above, this approach would lead to the following display times, frame intervals, and repetition rates:

f	i	j	RDT	$frame\ int$	$rep.\ rate$
I	1	1	IDL + 0	50	4
B	3	2	IDL + 50	37.5	3
B	4	3	IDL + 87.5	37.5	3
P	2	4	IDL + 125	50	4
...	...	...	...	...	...

Note that, as outlined in section 2.1, the decoding order will differ from the display order, i.e., $i \neq j$, if the stream contains B frames. For B frames $j = i-1$, for I and P frames, the display number depends on the MPEG stream and has to be determined via look-ahead.

Approach 2 - Use the closest instance of the display task to show the frame, i.e., either R or L, whichever is closest. For example, the instance of the display task that is closest to the decoding deadline of the second frame (IDL+41.666) is the one with deadline IDL+37.5, not the one that occurs later with the deadline IDL+50. We have shown above how to calculate the required display time for R, and the same is valid even for L, except that we use the floor function to get the instance index:

$$RDT(f_i^j) = IDL + \left\lfloor \frac{(j-1)T_{fr}}{T_{dis}} \right\rfloor T_{dis}$$

For the example above, this approach gives:

f	i	j	RDT	$frame\ int$	$rep.\ rate$
I	1	1	IDL + 0	37.5	3
B	3	2	IDL + 37.5	50	4
B	4	3	IDL + 87.5	37.5	3
P	2	4	IDL + 125	37.5	3
...	...	...	...	...	...

Approach 1 is a little more relaxed in terms of precise latencies, and thus deadlines. Apparently, the choice between approach 1 and 2 does not really matter with respect to relative frame jitter. In both cases, we get a cycle of three frame intervals: 50, 37.5, 37.5. However, the relative frame jitter is important for perception. In high quality video where the jitter is not accepted, this problem has been solved by using interpolation, i.e., making new frames. This feature is called *natural motion* [6].

In case the completion time constraint is missed, consistency between content and display and between display and audio can be disturbed. Either the display rate is compromised by waiting for the completion of the frame, which will display it after a too long time interval after the previous, and the next one at a too short one. Or the frame sequence is compromised by discarding the late frame and redisplaying the current one. In the first case, it may be necessary to skip the next frame, so as not to propagate the late display and slow down the video. Frame skipping is a delicate issue involving many design and engineering decisions, including stream semantic and decoder capabilities. The treatment of the temporal implications of frame skipping is beyond the scope of this paper.

LTC2: *Imminent overflow of input buffer*. By a judicious choice of input buffer size, as outlined in section 4, LTC2 will always be met. Should the completion constraint be missed, though, data loss at the input buffer will occur, with the risk of having to recapture the stream, which will take at least the complete GOP or until the next sequence header.

7 Conclusion

In this paper, we presented a study of realistic MPEG-2 video streams and showed a number of misconceptions for software decoding, in particular about relation of frame structures and sizes. Furthermore, we identified constraints imposed by frame buffer handling and discussed their implications on timing constraints.

Using the analysis, we determined realistic flexible timing constraints for MPEG decoding that call for novel scheduling algorithms, as standard ones that assume average values and limited variations, will fail to provide for good video quality.

Our current work includes extending the study to the sub frame level, e.g., relationship between framesize and execution time, motion vectors, and sub frame decoding. Furthermore, we are formulating a quality based frame selection algorithm to be used in a real-time scheduling framework.

Acknowledgements

The authors wish to thank the reviewers for their fruitful comments which helped to improve the quality of the paper. Further thanks go to Clemens Wüst and Martijn J. Rutten from Philips Research Laboratories, Eindhoven, for their careful reviewing and useful comments on this paper.

References

[1] Iso/iec 13818-2: Information technology - generic coding of moving pictures and associated audio information, part2: Video. 1996.

[2] L. Abeni and G. C. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, Madrid, Spain, 1998.

[3] A. Bavier, A. Montz, and L. Peterson. Predicting mpeg execution times. In *Proceedings of ACM International Conference on Surement and Modeling of Computer Systems (SIGMETRICS 98)*, Madison, Wisconsin, USA, June 1998.

[4] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. Qos for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, Florence, Italy, May 2001.

[5] L. O. Burchard and P. Altenbernd. Estimating decoding times of mpeg-2 video streams. In *Proceedings of International Conference on Image Processing (ICIP 00)*, Vancouver, Canada, September 2000.

[6] G. de Haan. IC for motion compensated deinterlacing, noise reduction and picture rate conversion. *IEEE Transactions on Consumer Electronics*, August 1999.

[7] M. Ditze and P. Altenbernd. Method for real-time scheduling and admission control of mpeg-2 streams. In *The 7th Australasian Conference on Parallel and Real-Time Systems (PART2000)*, Sydney, Australia, November 2000.

[8] C. Hentschel, R. Braspenning, and M. Gabrani. Scalable algorithms for media processing. In *Proceedings of the IEEE International Conference on Image Processing (ICIP)*, Thessaloniki, Greece, pp. 342-345, October 2001.

[9] D. Isovic and G. Fohler. Analysis of mpeg-2 streams. Technical Report at Malardalen Real-Time Research Centre,Vasteras, Sweden, March 2002.

[10] Y. C. John Tse-Hua Lan and Z. Zhong. Mpeg2 decoding complexity regulation for a media processor. In *Proceedings of the 4th IEEE Workshop on Multimedia Signal Processing (MMSP)*, Cannes, France, pp. 193 - 198, October 2001.

[11] J. K. Ng, K. R. Leung, W. Wong, V. C. Lee, and C. K. Hui. Quality of service for mpeg video in human perspective. In *Proceedings of the 8th Conference on Real-Time Computing Systems and Applications (RTCSA 2002)*, Tokyo, Japan, March 2002.

[12] S. Peng. Complexity scalable video decoding via idct data pruning. In *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, pp. 74-75, June 2001.

[13] L. Teixera and M. Martins. Video compression: The mpeg standards. In *Proceedings of the 1st European Conference on Multimedia Applications Services and Techniques (ECMAST 1996)*, Louvian-la-Neuve, Belgium, May 1996.

[14] J. Watkinson. *The MPEG handbook*. ISBN 0 240 51656 7, Focal Press, 2001.

[15] C. Wüst. Quality level control for scalable media processing applications having fixed CPU budgets. In *Proceedings Philips Workshop on Scheduling and Resource Management (SCHARM01)*, 2001.

[16] C. Wüst and W. Verhaegh. Dynamic control of scalable meadia applications. In *Algorithms in Ambient Intelligence*. editors: E.H.L. Aarts, J.M. Korst, and W.F.J. Verhaegh, Kluwer Academic Publishers, 2003.

[17] Z. Zhong and Y. Chen. Scaling in mpeg-2 decoding loop with mixed processing. In *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, pp. 76-77, June 2001.