Principles for the Prediction of Video Decoding Times applied to MPEG-1/2 and MPEG-4 Part 2 Video

Michael Roitzsch Martin Pohlack

Technische Universität Dresden
Department of Computer Science
Operating Systems Group
{mroi,pohlack}@os.inf.tu-dresden.de

Abstract

In this paper, we present a method to predict per-frame decoding times of modern video decoder algorithms. By examining especially the MPEG-1, MPEG-2, and MPEG-4 pt. 2 algorithms, we developed a generic model for these decoders, which also applies to a wide range of other decoders. From this model, we derived a method to predict decoding times with an up-to-now unmatched accuracy while keeping the overhead low. We show the effectiveness of this method with an example implementation and compare the resulting predictions with the actual decoding times using video material from commercial DVDs.

1. Introduction

Scheduling complex real-time tasks is hard as their resource requirements may not be known. Currently for video processing one has to work with worst-case overprovisioning of resources or ungraceful quality degradation. The fundamental problem here is that a video codecs' resource usage mostly depends on the *data* to be processed, that is, the current video stream. The codec acts similarly to an interpreter executing an interpreted program.

To allow future systems to overcome these limitations, realtime and Quality-of-Service (QoS) considerations should be applied [23, 16] to allow close-to-average case resource allocation and graceful degradation of quality. In this paper we present a method for predicting the required resource usage based on easily obtainable metadata about video streams. Obviously, this method must provide the decoding time with significantly lower resource usage than the actual decoding.

Based on this inferred ahead-of-time knowledge, better CPU time scheduling, graceful degradation in overload situations, and energy aware adaptation become possible. Future systems employing such methods can provide better QoS than current ones or similar QoS with fever hardware resources by better utilizing available resources.

For example, a video player whose decoding component cannot keep up with the display speed and therefore has to skip decoding of frames should do so considering quality (i.e., introduced error) *and* resource savings (decoding time). Close-to-average case resource allocation and graceful degradation of quality are enabling mechanisms here.

Another possible application are video editing systems, where the user can stack multiple layers of video and apply effects to them. These systems need to decide, which parts of the final edit can be calculated on the fly (during playback) and which parts need to be pre-rendered. The pre-rendering has to include all critical parts to ensure sufficient playback quality, but at the same time, it should be reduced to the minimum to keep the application responsive.

As most of these application scenarios require ahead-of-time knowledge of the resource usage and because the primary task in the media applications considered here is video decoding, with CPU time being the key resource, this work discusses a way to pre-determine per-frame decoding times for recent video decoder technologies. It is beneficial, if the method needs to be calibrated only once with a set of sensible sample material and then works for a wide range of content. We target calculating the decoding times on the fly for each frame, because users cannot be expected to provide precalculated trace data or wait for its collection. This ensures that the viewers expectation of watching arbitrary videos without preprocessing delay can be fulfilled.

We chose the MPEG-1/2 [4, 5] and MPEG-4 pt. 2 [6] decoder algorithms as the key algorithms to analyze because of their wide acceptance in the video world and the availability of mature decoders and encoders in source code under open-source licenses: MPEG-2 is present on today's DVDs and in DVB [1]. Thoroughly tested open-source decoder software is available from the libmpeg2 project [7]. Sophisticated encoders can be found in both the commercial sector and the open-source community. MPEG-1 shows enough similarity to MPEG-2 to regard MPEG-1/2 as one. MPEG-4 pt. 2 is one video decoding algorithm specified in the MPEG-4 standard family. It is not as commonly used as MPEG-2 yet but is emerging. We use decoder and encoder code from the XviD project [10].

In this work we present a predictor, which is fed with only the compressed video data and produces estimates for the execution time of a decoder cycle. The decoder itself is not modified in any way. The predictor should be as independent of the actual decoder implementation as possible to allow future upgrades to newer decoder versions without any major modifications to the predictor.

In the next section we discuss related work and how our work relates to the state of the art. In Section 3 we describe our approach — breaking down the entire decoding of a frame into various sub-tasks. We achieve this by creating a generic decoder model, into which all examined decoders and hopefully a wide range of future decoders fit (see Subsection 3.1). We will then discuss the influence of each sub-task on the decoding time in Subsection 3.3. Additionally we point out characteristic properties of the stream that correlate positively with the decoding time and discuss the extraction of values from the compressed video stream that serve as metrics to quantify the discussed properties. The mathematics used to calculate the coefficients of the linear combination will be explained in Section 4, followed by comments on the implementation of the given principles in Section 5. An evaluation of the results and a conclusion together with an outlook conclude the paper.

2. Related work

Our work is founded on previous results in the research areas of decoder algorithms and mathematics. We used well known numerical techniques such as the QR decomposition as well as established algorithms like the Householder transformation [22]. This will be explained in Section 4. Our ideas have been first implemented on the Dresden Realtime Operating System DROPS [9] and its video player VERNER [20]. For results see [21]. Verner can utilize streams from a real-time filesystem [19] or real-time network connection [18], and displays content with real-time guarantees [13]. For this paper we focus solely on the decoding step. We use a Linux implementation as our experience shows that the numbers are not significantly different and measuring entire DVDs is more complex on DROPS due to memory constraints. We adapted open-source code from the libmpeg2 [7] and XviD [10] projects.

Decoding time prediction has been a research subject before, so previous results exist. Altenbernd, Burchard, and Stappert present an analysis of MPEG-2 execution times in [11]. The common idea behind their work and ours is to gain metrics from the video stream that correlate well with the decoding time. However, there are considerable differences in both approaches: Altenbernd et al. divide the decoder in two phases, using data extracted in the first phase to predict the decoding time of the second phase. We want to avoid such heavy modifications to existing decoder code and rather provide a solution based on preprocessing. We also model the decoder as a sequence of different phases, but our division will be more fine grained (see Figure 1). They also examine worst-case execution times, using source code analysis, to completely avoid underestimations. We do not want to rely on the specific source code, because the efforts of the analysis have to be repeated when the code changes due to optimizations. This simplification allows us to generalize our method to target more decoders than just MPEG-2. Because we cannot safely avoid underestimation with our approach, we have to settle with a soft-realtime solution, but the results are still important because the quality assuring scheduling algorithm QAS [14] developed at TU Dresden works on the basis of a probability distribution for execution times. It remains an area of future research to derive worst-case execution times with our method.

Another similar analysis has been conducted by Andy Bavier, Brady Montz, and Larry L. Peterson in [12], but is also limited to MPEG-1/2. They focus more on decoding times at the granularity level of network packets, and do not target transferability of the results between different videos. Yet they also follow the path of predicting decoding times by extracting metrics from the stream.

3. Decoder Analysis

This section deals with internals of the MPEG video coding schemes. Those unfamiliar with this subject are invited to refer to [17] for some introductory reading.

3.1. Decoder Model

By looking into the inner workings and functional parts of the decoder algorithms in consideration, we established a decoder model, generic enough to be applicable to a wide range of algorithms. The simple basic structure of the model can be seen in Figure 1: The decoder is modeled as a chain of function blocks that are executed iteratively in loops. Because we are concentrating on execution times, the edges of the graph represent logical control flow rather than data flow. Control starts at the bitstream parsing block and implicitly ends with post processing when the compressed data stream ends.

Every function block but the first can have multiple alternative and completely separated execution paths in the same decoder algorithm. Those execution paths would be chosen amongst by context data extracted from the compressed video stream, like a frame type. As a notable special case, the function blocks can have a "do-nothing" execution path. In the following we discuss each block and examine their input and output.

3.1.1. Bitstream Parsing The bitstream parsing receives the compressed video stream for every frame and extracts meta- and context-information from the stream that is required to control the following decoding steps.

3.1.2. Decoder Preparation When decoder algorithms exploit temporal redundancy, they often work with previous images. Those may need to be copied or modified without disturbing existing images.

3.1.3. Decompression This function block is the first that is executed inside a per-macroblock loop. A macroblock is a 16×16 pixel area of the target image whose compressed data is stored consecutively in the data stream and that is decoded in one iteration of the loop.

The data needed to further decode the macroblock is stored using a lossless compression technology like Huffman [15]. This compression is reversed in this step and the

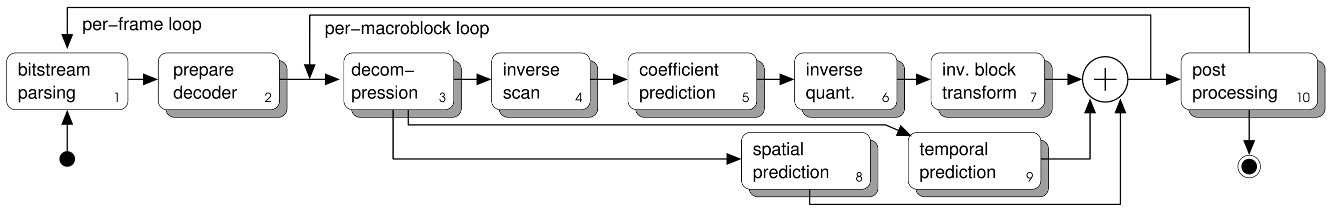

Figure 1. Chained block model for decoders

intermediate data is kept in a buffer. In addition to the macroblock data itself, each macroblock may come with metadata like a macroblock type or other information for following decoder steps. As a special case, a macroblock might consist entirely of such context data and no compressed image data at all.

3.1.4. Inverse Scan Because the resulting video image is two dimensional, the transforms used later in the compression are two dimensional as well. However, the decompressed macroblock we received from the previous stage is a one dimensional list of bytes. Those need to be rearranged as a 2D matrix. Because this would partly countereffect the preceding entropy compression step, this reordering is not done in a line-by-line fashion, but in a different, often diagonal pattern.

3.1.5. Coefficient Prediction Decoders can analyze the matrix to try to reconstruct image features that have been compressed away by extrapolating additional data from existing data on the coefficient level.

3.1.6. Inverse Quantization During encoding, the matrix of each macroblock has been multiplied with a quantization matrix, which, more than all other steps, makes the compression lossy. The quantization is reversed before decoding proceeds by multiplying with an inverse quantization matrix.

3.1.7. Inverse Block Transform The macroblock matrix we dealt with in the previous steps was not necessarily in the spatial domain. So the rows and columns of the matrix are not directly mapped to the image's x and y coordinates. The algorithms usually choose a different domain, in which operations like the quantization and coefficient prediction are fairly simple matrix operations and fit the visual perception model used by the algorithm in their effect on the final image. A common domain is the frequency domain, which has the nice property that quantization in the frequency domain will gradually smooth out details in the spatial domain.

This decoder step now transforms the macroblock matrix into the spatial domain, which involves complicated mathematics, sometimes even with floating point calculations. The resulting spatial matrix corresponds to a portion of the final image and has the same dimensions as the macroblock matrix.

3.1.8. Spatial Prediction The spatial and temporal prediction steps described now use previously decoded data of either the same frame (spatial prediction) or a different frame (temporal prediction) to reconstruct or merely guess the part of the image being covered by the currently decoded macroblock. This step can potentially be executed at the same time as the Steps 4 (inverse scan) to 7 (inverse block transform), but we will not pursue this parallelism, because the commonly available decoder implementations are single threaded. The outcome of the inverse block transform is then added to or otherwise merged with this prediction to reduce the residual error between the two.

3.1.9. Temporal Prediction Today's decoder algorithms benefit tremendously from temporal redundancy in the video. The most basic idea is to not always store entire video images, but to store the differences to the previous image. This has been extended over decoder generations to allow other reference images than just the previous one and to compensate for motion by storing translation vectors with the macroblocks. It even includes rotating and warping parts of the reference image, compressing the motion vectors by applying prediction to them or weighted interpolation between areas of multiple reference images in the most sophisticated algorithms.

The merging of the results of prediction and inverse transform ends the per-macroblock loop (denoted by $\oplus$). Execution will continue with the decompression of the next macroblock. It should be noted that most algorithms bundle consecutively stored macroblocks with common properties in a so called slice. This bundling is not yet relevant for considerations about execution time, but it will become more interesting when algorithms start to associate a semantic with slices, for example marking slices containing more important parts of the image like objects in the foreground.

3.1.10. Post Processing Post processing applies a set of filters to the resulting image so that compression artifacts are reduced and the perceived image quality is enhanced. However, with today's algorithms, post processing is often optional, so in realtime applications, a video decoder can skip the post processing stage if the scheduled CPU time is exceeded. Therefore, we will not analyze the execution time of the post processing stage and examine the mandatory parts of the decoding.

3.2. MPEG-4 part 2

We will now give a brief overview on how MPEG-4 pt. 2 fits into the function blocks of the developed decoder model using the same numeration as in Figure 1.

1. The bitstream parser handles the packetized MPEG-4 video elementary bitstream. Syntax elements are not

byte aligned. The stream can contain a complexity estimation header, which stores information we could use for our metrics extraction, like DCT block counts. Unfortunately this header is optional and the common encoder implementations do not use it, so existing videos would have to be reencoded with an extended encoder to benefit from this header.

2. The P-, B-, and S-frames use a special treatment of the edges of the reference frames for motion vectors beyond the frame boundaries.
3. The bitstream uses several tables for variable length coding of the coefficients and the motion vectors.
4. Varying inverse scan tables are available.
5. The coefficient prediction treats both the DC coefficient and the AC coefficients by copying rows from macroblocks above and columns from macroblocks left of the current position.
6. The inverse quantization uses a quantization matrix to scale the coefficients. Adaptive quantization by changing the matrix within one frame is possible.
7. The block transform is the IDCT.
8. There is no spatial prediction in MPEG-4 pt. 2.
9. The temporal prediction is a motion compensation using one forward and one backward reference frame with up to quarter pixel accuracy and macroblock subblocking. The S-frame uses one reference and shifts or warps the entire image using global motion vectors.
10. MPEG-4 pt. 2 contains an optional post processing step with different quality levels.

Other decoder algorithms can be similarly mapped to the steps of the model. For MPEG-1/2 we show this in [21].

3.3. Metrics for Decoding Time

Using the XviD [10] implementation of the MPEG-4 pt. 2 algorithm and a selection of example videos, we measured the per-frame execution time spent inside the various function blocks on a 400 MHz Pentium II machine. The example videos listed in Table 1 span a variety of stream parameters, which is necessary to evaluate their influence on the decoding time.

The profiling facility included in the XviD implementation has been a good starting point for a logical splitting of the decoder. It is interesting to note that over 50 % of the decoding time for every frame is spent on temporal prediction, so this step is most important (average has been taken over all sample videos from Table 1). We will now go through the various function blocks of the decoder model and show how their decoding time can be estimated using values derivable from the stream.

3.3.1. Bitstream Parsing (Step 1) Because most of the parsing takes place at the macroblock level, we can expect the execution time for this step to be closely related to the amount of pixels per frame, because the amount of macroblocks to parse correlates well with the amount of pixels. This works for I-frames as can be seen in Figure 2. (The samples appear in clusters because the pixel count is constant throughout each video.)

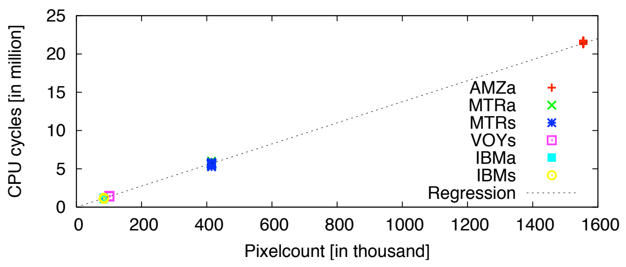

Figure 2. Estimating the bitstream parsing time for I-frames.

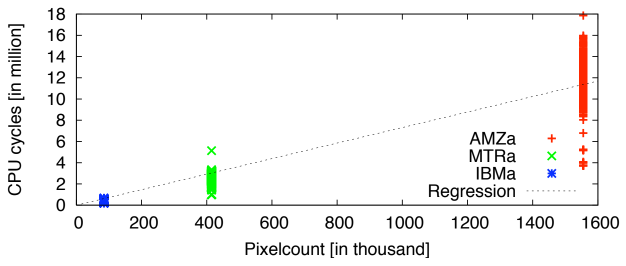

Figure 3. Estimating the bitstream parsing time for B-frames (1).

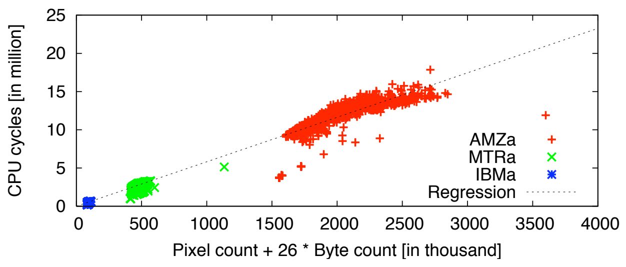

Figure 4. Estimating the bitstream parsing time for B-frames (2).

However, the P-, B- and S-frames show a different behavior. We chose the B-frames to demonstrate this in Figure 3, the P- and S-frames behave similarly. Because these frame types allow a greater variety of coding options for each macroblock than the I-frames, which naturally consist entirely of intracoded macroblocks, the amount of bytes per non-I-frame has an influence, too. As can be seen in Figure 4, the result improves when taking this into account by calculating a linear combination of pixel count and bytes per B-frame that matches the execution time best. This idea also works for P- and S-frames.

3.3.2. Decoder Preparation (Step 2) The MPEG-4 pt. 2 algorithm extends the edges of reference frames before they are used for temporal prediction. Because the length of the image edges correlates with the square root of the pixel count, we would expect a square root match between the pixel count and the execution time for this step. Figure 5 shows exemplary for P-frames that this assumption is correct. The B- and S-frames behave the same and for I-frames, there are no reference frames to be modified.

Diagr. sym.	Video name	Duration	Resolution	Size	Profile	Properties	Source
AMZa	Amazon documentary	1:37 min	1440×1080	97 MB	ASP	high quality HD content, detailed images	Download from [8]
MTRa	Movie trailer "The Sweetest Thing"	1:13 min	720×576	8.6 MB	ASP	dynamic, fast cuts	advertising DVD
MTRs				63 MB	SP		
VOYs	Voyager	1:20 min	352×288	13 MB	SP	low in movement, noisy image	TV recording
IBMa	IBM Linux commercial "Prodigy"	1:50 min	352×240	508 KB	ASP	low in movement, many monochrome shapes, ASP version highly compressed	IBM website (now at [3])
IBMs				2.8 MB	SP		

Table 1. The sample videos used in the decoding time analysis (SP=simple visual profile, ASP=advanced simple profile). The videos have been transcoded to MPEG-4 pt. 2 with varying encoder settings to investigate the effects of all *properties*.

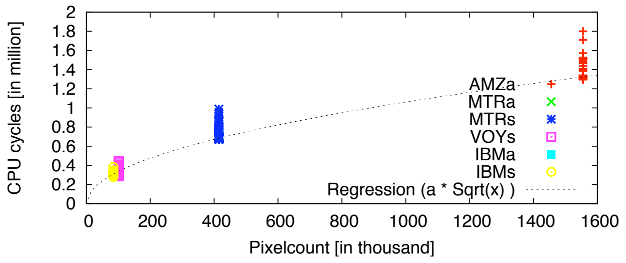

Figure 5. Estimating the decoder preparation time for P-frames.

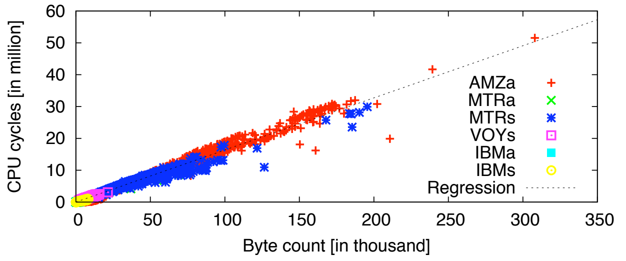

Figure 6. Estimating the decompression time for I-, P-, B- and S-frames.

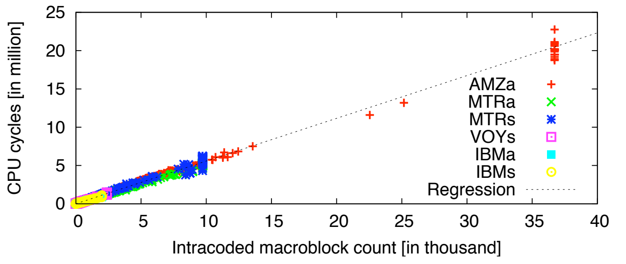

Figure 7. Estimating the coefficient prediction time for I-, P-, B- and S-frames.

3.3.3. Decompression and Inverse Scan (Steps 3 and 4)

The decompression should be the dominant task here, because the inverse scan is as easy as selecting a scan table and then using one table lookup for every decompressed coefficient. The execution time therefore correlates well across all frame types with the per-frame length of the bitstream. Figure 6 shows the match.

3.3.4. Coefficient Prediction (Step 5) The coefficient prediction is only done for intracoded macroblocks, so the execution time of this step correlates with their number. Intracoded macroblocks can occur within every frame type, but the estimation works equally well for all types as shown in Figure 7.

3.3.5. Inverse Quantization (Step 6) The inverse quantization is done once for every macroblock, so this correlates well with the total macroblock count. The macroblock count should not be mistaken for a scaled pixel count. Because of not coded macroblocks, the pixel count only yields an upper bound for the macroblock count. The diagram for all frame types can be seen in Figure 8.

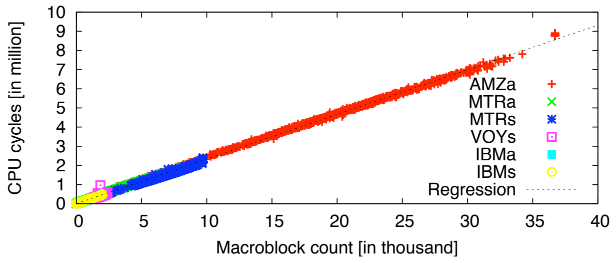

Figure 8. Estimating the inverse quantization time for I-, P-, B- and S-frames.

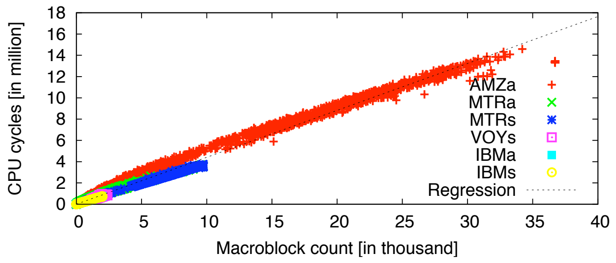

Figure 9. Estimating the inverse block transform time for I-, P-, B- and S-frames.

3.3.6. Inverse Block Transform (Step 7) As with the inverse quantization, the total macroblock count gives a good estimate, as Figure 9 shows. There is still deviation from the linear match, which might stem from either cache effects or different amounts of zero values in the macroblocks. Depending on the implementation, algorithms can be faster when more coefficients are zero. However, looking into that has the disadvantage of increased prediction overhead, so we decided against it, as the potential gain in accuracy is small.

3.3.7. Temporal Prediction (Step 8) This step is the most time consuming. Unfortunately, its execution time is also the hardest to predict. Because motion compensation is only done for intercoded macroblocks, one might be tempted to derive the execution time from the count of intercoded macroblocks. Figure 10 shows that this fails.

The reason is that there are various different methods of motion compensation due to macroblock subblocking and different storage types for motion vectors. These options are independent of the macroblock type. Distinguishing be-

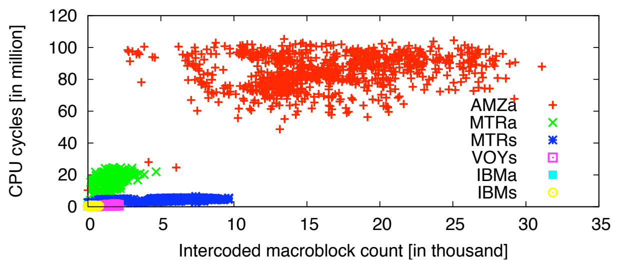

Figure 10. Estimating the temporal prediction time for P-frames (1).

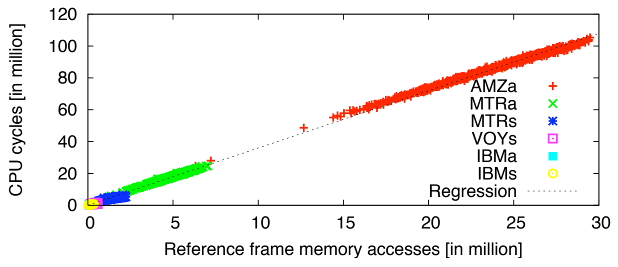

Figure 11. Estimating the temporal prediction time for P-frames (2).

tween all these methods and accounting for them individually is problematic, because there are too many combinations and covering them all with sample videos is difficult.

But when looking at the bigger picture, we realized that motion compensation is basically just copying pixels from a reference image into the current frame. Because of half pixel or quarter pixel accuracy and the necessary interpolation and filtering, one pixel copy operation can range from 1 to 20 memory accesses. Therefore, the number of memory accesses into the reference frame should result in a far better prediction. Of course this number cannot easily be measured directly, but looking at the code for motion compensation, we can count the memory accesses. Note that all decoder implementations must access the same amount of data, because of the required interpolation. We assume that no decoder implementation will make many superfluous memory accesses, so their count should be similar across implementations.

Depending on the lower bits of the motion vectors, which differentiate between full, half or quarter pixel references, we created a formula to calculate the number of memory accesses. For that, the motion vectors need to be decoded completely, which takes time and increases prediction overhead. However, because the temporal prediction accounts for a big portion of the overall decoding time, we think this step is necessary. We will show the effect on the overhead in Subsection 6.2.1. The promising results for P-frames can be seen in Figure 11. This works equally well for B-frames.

It may be a bit surprising that memory accesses alone estimate the execution time so well, given that different interpolation and filtering is done for full, half and quarter pixel accesses. We assume that with today's processors, these additional operations are covered up by the memory accesses

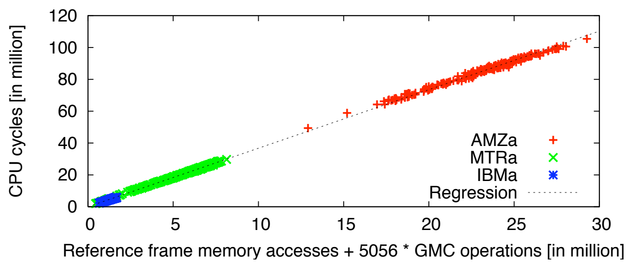

Figure 12. Estimating the temporal prediction time for S-frames.

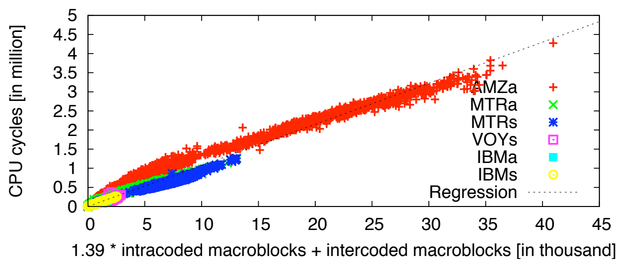

Figure 13. Estimating the merging time for P-, B- and S-frames.

because of parallel execution. To verify this assumption on a different hardware architecture, this estimation has been run on a PowerPC (MPC7447a) machine, which showed similar results for P- and B-frames. The match was not as linear as in Figure 11, but still correlated well (correlation coefficient 0.998).

S-frames, however, encompass global motion compensation (GMC), so a different approach is required here. Because we did not want to look into the more complicated (hard to predict) warping algorithm, we just counted the number of macroblocks using global motion compensation and calculated a linear combination of the memory accesses from non-GMC macroblocks and the amount of GMC macroblocks to match the execution time. The good results shown in Figure 12 indicate that the execution time per GMC macroblock is fairly constant.

3.3.8. Merging (Step $\oplus$) The merging combines the results of the temporal prediction with the decoded macroblocks, so a linear match of intra- and intercoded macroblock counts should estimate the execution time well enough. Because this step has only a small influence on the total decoding time, we can tolerate the deviations seen in Figure 13

3.3.9. Summary We have now established which metrics are useful to get reasonable estimations for the execution time of the various stages. It is evident that the separation of the decoding process into the suggested steps simplifies finding useful metrics, because the decoding time behavior of the individual function blocks is much easier to overlook. The time required for the entire decoding process is the sum of the separate execution times, so these metrics will also allow predictions for the frame decoding times. The metrics are:

- Pixel count,

- Square root of pixel count,
- Byte count,
- Intracoded macroblock count,
- Intercoded macroblock count,
- Motion compensation reference frame memory accesses, and
- Global motion compensation macroblock count.

With similar considerations as presented for MPEG-4 pt. 2, we have selected metrics for the prediction of MPEG-1/2 [21]. In Section 5 we will discuss how to actually obtain these metrics.

4. Numerical Background

We have now extracted a set of q metric values for each frame of the video. In a learning stage, on which we will present details in Section 5, we will receive a metric vector $\underline{m}_i$ and the measured frame decoding time t_i for each of a total p frames ($i = 1 \ldots p$). Accumulating all the metric vectors as rows of a metric matrix M and collecting the frame decoding times in a column vector $\underline{t}$, we now want to derive a column vector of coefficients $\underline{x}$, which will, given any metric row vector $\underline{m}$, yield a predicted frame decoding time $\underline{m}\,\underline{x}$. Because the prediction coefficients $\underline{x}$ must be derived from M and $\underline{t}$ alone, we model the situation as a linear least square problem (LLSP):

$$\|M\underline{x} - \underline{t}\|_e^2 \to \min_{\underline{x}}$$

That means the accumulated error between the prediction $M\underline{x}$ and the measured frame decoding times $\underline{t}$ is minimized. The error is expressed by the square of the Euclidean norm of the difference-vector. Because of its insensitivity against badly conditioned matrices M, we chose QR decomposition with Householder's transformation as the method to solve the LLSP. For a more detailed explanation of the involved mathematics, please refer to literature such as [22, 21].

4.1. Metric Selection and Refinement

For the general problem of metric finding we see two approaches: **(a)** First, a domain expert has to model the problem using smaller sub-steps. Then, by looking at the work done in the sub-steps, he has to guess interesting metrics which can be easily obtained from the data to be processed and which correlate with the work done in the sub-steps. These selected metrics are then verified with obtained resource usage statistics of the original problem. **(b)** Second, for more simple problems, one could get useful results without splitting up the original problem into smaller pieces and without a domain expert selecting metrics. One could just use *all* easily available metrics and try to find the relevant metrics by validating it against measured data. In both cases only those metrics are relevant for our approach which can be obtained with much less resource usage than solving the original problem.

For this paper we took the first approach, as the domain is highly complex and a lot of different metrics are available. For both approaches an automatic method for metric validation is required, which we describe in the following.

In general, it should be possible to feed the LLSP solver with sensible metrics and it should figure out which ones to use and which ones to drop by itself. Of course, the best result for the linear least square problem is always achieved by using as many metrics as possible, but one of the design goals is to make the results transferable to other videos, which might not always work when using metrics too greedily. Using too many metrics can lead to overfitting to the training material, leading to bad predictions for videos not included in the training set. The main cause for this are similarities of columns with linear combinations of other columns. The special case of this situation is an actual linear dependency, resulting in a rank-deficient matrix. This leads to instabilites in the resulting coefficients, such that we can increase certain coefficients and compensate by decreasing others with little or no influence on the prediction results. The barebone LLSP solver will always search for the optimal fit, which might be too specific to predict other video's decoding times with the resulting coefficients. To overcome this problem, we drop metrics before solving the LLSP, deliberately making the fit less good for the training set, but more transferable to other videos outside the training set.

In the resulting R matrix of a QR decomposition, the remaining error, called residual sum of squares, for an n-column matrix is the square of the value in the n'th column of the n'th row. This value indicates the quality of the prediction: The smaller, the better. If we have to drop columns for transferability, we want to do so without too much degradation on the quality of the result. Therefore, we iteratively drop columns and then choose the one that best fits our goals, but results in the smallest increase of this error indicator. A linear dependency or a situation close to it can also be detected with this indicator: If we drop a column and there is only a minor increase in the residual sum of squares, the dropped column had little to no influence on the result, so the column can be sufficiently approximated as a linear combination of others. We propose an algorithm to eliminate such situations in [21].

5. Metrics Extraction and LLSP Solver

We used an open-source decoder for the extraction with additional instructions to collect the desired metrics. We removed all actual decoding code, so that only the required bitstream parsing remained, creating our metrics collector. For MPEG-1/2, the libmpeg2 decoder library [7] in version 0.4.0 and for MPEG-4 pt. 2, the XviD library [10] in version 1.0.3 have been used. Our stripped-down parsers have about 20 % LoC (each ca. 4,000) of the original libraries and run completely independently of the actual decoding steps.

The LLSP (linear least square problem) solver and the collector support two phases of operation:

- Learning mode, in which the collector accumulates metrics and a timed unmodified decoding step delivers real frame decoding times,
- Prediction mode, in which previously obtained LLSP coefficients are multiplied with online-collected metrics to predict frame decoding times.

During learning mode, the solver collects metric values in a matrix. If the data accumulation is finished, the coefficient vector $\underline{x}$ is calculated with the enhanced QR decomposition presented above. This step has a complexity of $O(p * q^4)$, where q is typically fixed and small, compared to p being unbound. Therefore, the video length has linear impact which is what you would expect. The resulting coefficients are then stored for use in prediction mode, typically on videos other than those in the learning set.

In both learning and prediction mode, the potential for decoder frame reordering has to be taken into account. Details on this can be found in [21].

6. Results

We will now present decoding time predictions on real-life video material taken from commercial DVDs (complete movies), covering a huge feature variety. We also discuss the prediction overhead, how to choose a sensible training set, and where to apply the prediction. Because MPEG-4 pt. 2 is one of our main contributions we discuss MPEG-1/2 results only very briefly.

6.1. Choosing Videos for Learning Mode

The videos used in learning mode have to span the entire feature set of the decoder and each of the metrics needs to vary at least once. For MPEG-4 pt. 2, the video clips need to cover both the simple and the advanced simple visual profiles to represent the feature combinations of the algorithm sufficiently well. To calculate the influence of the pixel count the training videos must differ in their resolution. The training videos and the video material used for verification in the following are *completely disjunct*. One could probably gain even more prediction accuracy by tuning the training set towards certain use cases, but we did not yet explore this possibility.

6.2. Prediction Accuracy and Overhead

Frame decoding time prediction could be done anywhere from the actual encoding step until directly before decoding. We see three useful methods: First, directly on encoding, by embedding relevant metric into the stream, such that the last prediction step on the target machine will be extremely cheap. However, the decoder has to be adapted to use this information. Second, by using a stripped down stream parser as described in Section 5 and running it directly before the decoding component. In this case, additional stream parsing overhead occurs, but no change is required in the decoder. And third, directly in the decoder, such that metadata extraction has to be done only once. Here, the actual stream parsing would have to be split into two phases as suggested in [11]. For the last two cases one requires some frames being buffered to actually utilize the scheduling information.

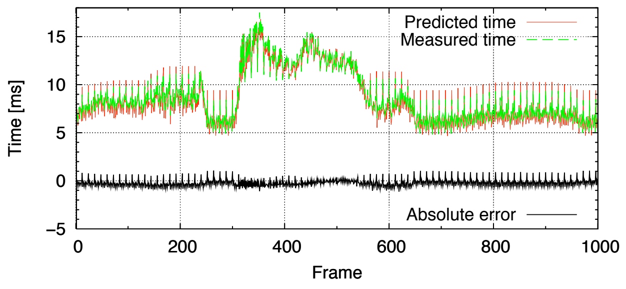

Figure 14. Predicted and measured frame decoding times and the absolute error for "The Sixth Sense".

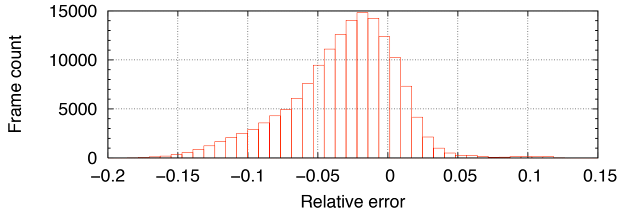

Figure 15. Histogram for the relative error of "The Sixth Sense" decoding time prediction.

In the following we will discuss the second method as it is the least invasive, there are no changes required for the video streams or the decoder. If one strives for a solution with even less overhead, our prediction approach does work with all three of them.

The codecs in use were optimized for the target CPU and used the available SIMD extensions (MMX, SSE, Altivec), except for the XviD codec, where no working Altivec-optimization was available.

6.2.1. MPEG-4 pt. 2 We used the two IBM commercials and the two movie trailers from Table 1 on page 5 to train the coefficients. We played these four clips in learning mode on a 1.5 GHz AMD Sempron 2200+ machine to calculate the coefficients. In prediction mode, we used the derived coefficients to predict the frame decoding times for several transcoded commercial DVDs, which were *not part of the training set*. The promising results can be seen in Table 2.

Detailed results for the DVD "The Sixth Sense",

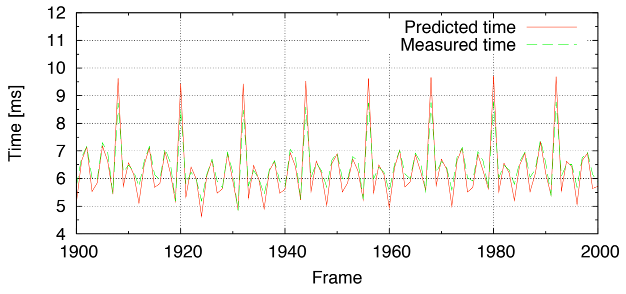

Figure 16. Prediction of stochastical load fluctuations for "The Sixth Sense" video.

transcoded to MPEG-4 pt. 2, are presented in Figures 14–16. The relative error has an average of $-2.8\,\%$ at a standard deviation of $3.9\,\%$ (error ranges from $-42.6\,\%$ to $16.1\,\%$ with an $95\,\%$ quantile of 0.92 ms, meaning that 0.92 ms overprovisioning yields $95\,\%$ overpredicted frames). The absolute error has an average of -0.27 ms at a standard deviation of 0.37 ms (error ranges from -5.97 ms to 2.88 ms). Negative errors here mean that the prediction underestimated, positive errors imply overestimation. The correlation coefficient of predicted against measured values is 0.99, so the prediction is quite accurate. In particular, we can predict not only the long-term behavior of the decoding time, but also short-term fluctuations as can be seen enlarged in Figure 16. [23] uses the terms "structural load fluctuation" and "stochastical load fluctuation" for these long-term and short-term variations respectively. We believe, now that both can be predicted, the term "stochastical" should be reconsidered.

We also tested the prediction quality under completely different and extreme conditions: a PowerPC system (PowerBook G4 1.33 GHz running Mac OS X 10.4), training the coefficients with the same set of short learning clips and then predicting the decoding times for the Amazon advanced simple profile *high definition* video. Here the relative error has an average of $3.2\,\%$ at a standard deviation of $4.5\,\%$ (error ranges from $-43.1\,\%$ to $16.5\,\%$). The absolute error shows an average of 3.31 ms at a standard deviation of 4.66 ms (error ranges from -25.13 ms to 17.53 ms). Given that the coefficients have been derived from standard definition content (i. e., the training set does not contain HD content) and are now applied to a high definition video with about four times the frame size, we think these results are outstanding and demonstrate the robustness of our approach.

6.2.2. Overhead The average overhead introduced by the online metrics extraction for the above Amazon HDTV video is $5.6\,\%$. The main source for overhead is the bitstream parsing and macroblock decompression. Directly accessing single macroblocks in the compressed stream is not possible, as there is no index facility. The only means of finding the position of a macroblock is to decompress the complete preceding bitstream in the slice. A possible approach is to split the decoder in two and parse and decompress the bitstream in the first part. The metrics can then be extracted from the preprocessed data and the second decoder part would do the rest of the decoding, using the already decompressed macroblocks as an input. We wanted to avoid such constructions because they usually require heavy modifications to decoder code, so new decoder implementations would be difficult to deploy. Altenbernd, Burchard, and Stappert have taken this approach in [11], and they also ended up with overheads of 4–9 %, depending on the video, so there may not be much benefit in pursuing this.

Another way to reduce the overhead would be to not decode the motion vectors that we use to predict the temporal prediction step as discussed in Subsection 3.3.7. This would lower the overhead from $5.6\,\%$ to $4.1\,\%$, but would also reduce the quality and transferability of the prediction coefficients.

6.2.3. MPEG-1/2 We used two videos with different resolutions that cover both MPEG-1 and MPEG-2 to train the predictor and did a thorough evaluation of our method with some commercial DVD material. The good results in Table 2 have again been measured on an AMD Sempron 2200+ machine (1.5 GHz).

6.2.4. Summary Although small parts still could use improvement, we think the overall goal of accurately predicting decoding times for MPEG-1/2 and MPEG-4 pt. 2 in both the simple and advanced simple profiles has been accomplished. The prediction coefficients derived from one set of learning videos can be applied to a wide range of content *not included in the learning set*, which is critical to future applications' usability, because it reduces the amount of learning necessary for good results.

7. Conclusion

We presented the design and implementation of a system to predict decoding times with an up-to-now unmatched accuracy (avg. errors down to $-0.0\,\%$) and acceptable overhead ($5.6\,\%$). The prediction relies on preprocessing and statistical evaluation of training runs rather than requiring heavy source code analysis or decoder modifications. This ensures that the presented results will not be obsoleted by further decoder development such as code optimizations, because the method is largely independent of the specific decoder code. For the prediction we only require *one calibration* to the target machine with a sensible set of short training videos, then the prediction works for other videos. We also verified our approach with material from popular commercial DVDs and achieved very accurate predictions (i. e., all relative errors of predictions were below $5\,\%$ and all absolute error were below 0.4 ms).

8. Outlook

To deal with current or future multimedia requirements overprovisioning of resources, as it is common today, is uneconomic. Here, the prediction of decoding times itself is already helpful, but the results should be regarded in a larger context. Schedulers of future operating systems will benefit from knowing resource usage beforehand when supporting QoS applications. Our work can provide this knowledge and should be complemented by research on perception models, which could assigns benefit values to video frames. This would allow frames to compete for CPU time on the basis of a true price-performance ratio, resulting in optimal video presentation even in high load situations to really improve the user's experience.

To compensate for the complexity of future algorithms, the bitstream parsing overhead should be reduced significantly. The most promising approach is to pre-determine the required metrics already during encoding and embed

DVD	Properties	Algorithm	rel. error (%)	abs. error (ms)	values w/in ±0.1 rel. error	values w/in ±0.5 ms abs. error	95% quantile*
Chicken Run	animated, claymation	MPEG-2	-0.5 (5.6)	-0.02 (0.28)	96.8 %	97.5 %	0.32 ms
		MPEG-4 pt. 2	-4.0 (4.8)	-0.32 (0.41)	87.7 %	71.0 %	1.05 ms
The Fifth Element	colorful, fast action	MPEG-2	-4.5 (5.3)	-0.18 (0.20)	86.3 %	95.1 %	0.49 ms
		MPEG-4 pt. 2	-2.0 (3.4)	-0.18 (0.32)	97.0 %	82.4 %	0.77 ms
King Kong (1933)	still camera, black and white	MPEG-2	-4.4 (4.4)	-0.23 (0.22)	90.1 %	88.4 %	0.60 ms
		MPEG-4 pt. 2	-2.3 (3.3)	-0.23 (0.34)	97.8 %	78.4 %	0.83 ms
Lola rennt	many fast steadicam shots	MPEG-2	-0.0 (6.2)	-0.01 (0.31)	94.0 %	96.3 %	0.35 ms
		MPEG-4 pt. 2	-2.2 (4.3)	-0.18 (0.39)	93.8 %	78.9 %	0.82 ms
The Sixth Sense	slow motion, dark atmosphere	MPEG-2	2.8 (7.5)	0.08 (0.26)	82.4 %	88.5 %	0.25 ms
		MPEG-4 pt. 2	-2.8 (3.9)	-0.27 (0.37)	94.3 %	72.9 %	0.92 ms

* Increasing the predictions by this value results in 95 % overestimation.

Table 2. Prediction results for some German Region 2 DVDs without copy protection. The table lists the average errors with the respective standard deviations in brackets. The MPEG-2 streams have been taken directly from the DVDs, the MPEG-4 pt. 2 streams have been created from them with ffmpeg [2] (coding options: -f m4v -vcodec mpeg4 -b 2000 -qpel -mv4 -gmc -bf2). With both algorithms, the entire movies have been measured.

them at prominent positions inside the bitstream. The MPEG-4 pt. 2 bitstream already contains this concept in a complexity estimation header, which stores information like macroblock and DCT coefficient counts. Unfortunately this header is optional and the common encoder implementations do not make use of it. Future video bitstreams and container formats should make this header mandatory. We estimate a compressed bitrate of 2 KBit / s for this information, which seems quite affordable compared to about 4 MBit / s for current Video DVDs (ca. 0.05 %).

Another challenge is to further automate the implementation of new algorithms like it has already been done with column dropping. One could think of automatic derivation of metrics from profiling runs and function call frequency. This might even lead to results for codecs that are only available in binary form. On the other hand, the prediction could be made more precise and it would be an interesting research subject to include source code analysis into our method to completely avoid underestimations in the predicted decoding times.

References

[1] *ETSI TR 101 154: Digital Video Broadcasting (DVB); Implementation guidelines for the use of MPEG-2 systems, video and audio in satellite, cable and terrestrial broadcasting applications*.

[2] FFmpeg project. http://ffmpeg.sourceforge.net/.

[3] IBM Linux commercial. http://rxns-rbn-sea02.rbn.com/ibmpdc/pdc/open/qtdemand/aug03/prodigy90_med.mpg.

[4] *ISO/IEC 11172-2: Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 2: Video*.

[5] *ISO/IEC 13818-2: Generic coding of moving pictures and associated audio information, Part 2: Video*.

[6] *ISO/IEC 14496-2: Coding of audio-visual objects, Part 2: Visual*.

[7] libmpeg2 project. http://libmpeg2.sourceforge.net/.

[8] Microsoft WMV HD Content Showcase. http://www.microsoft.com/windows/windowsmedia/content_provider/film/ContentShowcase.aspx.

[9] The Dresden Real-Time Operating Systems Project. http://os.inf.tu-dresden.de/drops/overview.html.

[10] XviD project. http://www.xvid.org/.

[11] P. Altenbernd, L.-O. Burchard, and F. Stappert. Worst-Case Execution Times Analysis of MPEG-2 Decoding. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS)*.

[12] A. Bavier, B. Montz, and L. L. Peterson. Predicting MPEG Execution Times. In *Proceedings of the joint international conference on measurement and modeling of computer systems*, 1998.

[13] N. Feske and H. Härtig. Demonstration of DOpE — a Window Server for Real-Time and Embedded Systems. In *24th IEEE Real-Time Systems Symposium (RTSS)*, pages 74–77, Cancun, Mexico, Dec. 2003.

[14] C.-J. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig. Quality Assuring Scheduling - Deploying Stochastic Behavior to Improve Resource Utilization. In *22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, Dec. 2001.

[15] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, 1952.

[16] D. Isović and G. Fohler. Quality aware MPEG-2 Stream Adaptation in Resource Constrained Systems. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*.

[17] D. Isovic, G. Fohler, and L. Steffens. Timing constraints of mpeg-2 decoding for high quality video: misconceptions and realistic assumptions. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS 03)*, Porto, Portugal, July 2003. IEEE.

[18] J. Loeser and H. Härtig. Low-latency Hard Real-Time Communication over Switched Ethernet. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 13–22, Catania, Italy, June 2004.

[19] L. Reuther. *Disk Storage and File Systems with Quality-of-Service Guarantees*. PhD thesis, TU Dresden, Fakultät Informatik, Nov. 2005.

[20] C. Rietzschel. VERNER – ein Video EnkodeR uNd playER für DROPS, 2003. Master's thesis.

[21] M. Roitzsch. Principles for the Prediction of Video Decoding Times applied to MPEG-1/2 and MPEG-4 Part 2 Video, 2005. Großer Beleg (Undergraduate thesis).

[22] J. Stör and R. Bulirsch. *Introduction to Numerical Analysis*. Springer-Verlag, 1980.

[23] C. C. Wüst, L. Steffens, R. J. Bril, and W. F. Verhaegh. QoS Control Strategies for High-Quality Video Processing. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*.