UNIVERSITY OF OSLO
Department of Informatics

Multi-Rate VP8 Video Encoding

Master's Thesis

Dag Haavi Finstad

Multi-Rate VP8 Video Encoding

Dag Haavi Finstad

Contents

List of Figures

Abstract

Adaptive HTTP streaming is frequently used for both live and on demand video delivery over the Internet. Adaptiveness is often achieved by encoding the video stream in multiple qualities (and thus bitrates), and then transparently switching between the qualities according to the bandwidth fluctuations and the amount of resources available for decoding the video content on the end device. For this kind of video delivery over the Internet, H.264 is currently the most used codec, but VP8 is an emerging open-source codec expected to compete with H.264 in the streaming scenario. The challenge is that, when encoding video for adaptive video streaming, both VP8 and H.264 run once for each quality layer, i.e., consuming both time and resources, especially important in a live video delivery scenario.

In this thesis, we address the resource consumption issues by proposing a method for reusing redundant steps in a video encoder, emitting multiple outputs with varying bitrates and qualities. It shares and reuses the computational heavy analysis step, notably macro-block mode decision, intra prediction and inter prediction between the instances, and outputs video in several rates. The method has been implemented in the VP8 reference encoder, and experimental results show that we can encode the different quality layers at the same rates and qualities compared to the VP8 reference encoder, while reducing the encoding time significantly.

Acknowledgments

I would like to thank my supervisors Håkon Kvale Stensland and Pål Halvorsen, for their guidance, valuable feedback and for always being encouraging and positive.

Also a big thanks to all of my friends at the lab, for interesting discussions and laughs, and a motivating work environment.

Oslo, August 2011
Dag Haavi Finstad

Chapter 1

Introduction

1.1 Background

The number of video streaming services, both live and on-demand, is quickly increasing. For example, consider the emergent and rapid deployment of public available Internet video archives providing a wide range of content like newscasts, movies and scholarly videos. Furthermore, all major (sports) events like NFL Hockey, NBA basket ball, NFL football, European soccer leagues, etc. are streamed live with only a few seconds delay, e.g., bringing the 2010 Winter Olympics [3], 2010 FIFA World Cup [4] and NFL Super Bowl [4] to millions of concurrent users over the Internet supporting a wide range of devices ranging from mobile phones to HD displays. The number of videos streamed from such services is in the order of tens of billions per month [5], and leading industry movers conjecture that traffic on the mobile-phone networks will also soon be dominated by video content [6].

The currently de facto video delivery solution in these scenarios is adaptive streaming over HTTP [3, 4, 7–9]. In these systems, the bitrate (and thus video quality) can be changed dynamically to match an oscillating bandwidth, giving a large advantage over non-adaptive systems that are frequently interrupted due to buffer underruns or data

loss. The video is thus encoded in multiple bitrates matching different devices and different network conditions.

Today, H.264 is the most frequently used codec. However, an emerging alternative is the simpler VP8 which is very similar to H.264's baseline profile and supposed to be well suited for web-streaming with native support in major browsers, royalty free use and similar video quality as H.264 [10, 11]. For both codecs, the challenge in the multi-rate scenario is that each version of the video require a separate processing instance of the encoding software, and especially in the live scenario, where all the rates must be delivered in real-time. This process is both time and resource consuming.

To reduce the large video overheads in multi-rate scenarios, we investigate possibilities for reusing the output from different steps in the encoding pipeline as the same video elements are processed multiple times with only slightly different parameters. As a case study, we have analyzed and experimented with VP8's processing pipeline and implemented support for running multiple VP8 encoder instances in parallel. Inspired by several transcoding approaches trying to reuse motion vectors [12–14], our initial idea is to allow the encoder to share and reuse the computational heavy intermediate steps from analysis computations, notably macro-block mode decision, intra prediction and inter prediction between the instances. Furthermore, the proposed method has been implemented in the VP8 reference encoder, and we have performed a wide range of experiments using various rates, resolutions and content types. We show that we can encode the different videos at the approximately same rates and qualities compared to the VP8 reference encoder, while reducing the encoding time significantly.

1.2 Problem Statement

Encoding video into multiple bitrates for adaptive streaming over various networks to different end-devices is a resource expensive task. With the motivation of alleviating the resource consumption, we wish to investigate the effect of reusing intermediate computations when running multiple encoding instances in parallel.

Our work will be based on Google's VP8 encoder (libvpx). We will identify the most resource consuming parts of the encoder, and investigate the possibility of reusing intermediate results from these computations when targeting several bitrates. As a case study we will implement a multi-rate encoder based on libvpx, and evaluate its performance.

1.3 Main Contributions

Our main contribution is that we propose a way of reusing decisions from intra and inter prediction in the video encoder to avoid computational expensive steps that are redundant when encoding for multiple target bitrates of the same video object.

A proof of concept implementation has been developed, based on the VP8 reference encoder. Our encoder is capable of encoding videos of different quality layers at the same rates and approximately same qualities compared to the VP8 reference encoder, with significantly reduced complexity.

A paper describing the multi-rate encoder and evaluating its performance has been submitted and is currently pending review for the IEEE International Symposium on Multimedia (ISM2011) conference.

1.4 Outline

In chapter 2, we provide background on concepts in video coding and describe the VP8 video format. We also give an overview of the libvpx encoder and describe the encoding pipeline. Chapter 3 presents the design and implementation considerations. A description of the experiments performed and an evaluation of the performance of our implementation is found in chapter 4. Finally, in chapter 5 the work is concluded, along with a short discussion on further work.

Chapter 2

The VP8 Codec

In this chapter, we provide background on basic concepts in video representation and coding, and give an introduction to the VP8 video codec format.

The VP8 codec [15], developed by On2 Technologies as a successor to VP7, is a modern codec for storing progressive video. On2 was acquired by Google in 2010, which subsequently released VP8 as a royalty-free alternative to H.264 as part of the open source *webm* project. The webm format was later added as a supported format in the upcoming HTML5 standard, and all major browsers have implemented playback support for the format since webm is expected to be a major streaming format on the web in the coming years.

2.1 Digital representation of video

2.1.1 Temporal and spatial sampling

Representation of a visual scene digitally involves sampling the visual scene *temporally* and *spatially*. The *temporal* sampling relates to the sampling of the real scene at a pe-

riodic interval, providing a series of pictures. A higher temporal sampling rate gives smoother motion, but requires more samples to be stored.

The *spatial* sampling involves representing each picture as a collection of discrete pixel values. The pixels are arranged in a grid, each value representing a sample of the original picture at the corresponding position.

2.1.2 Color spaces

To represent color in a pixel value, at least three numbers are required. The *RGB* color model represents each color as a combination of red, green and blue. Any color can be represented by combining these in variying proportions.

A different way of encoding the RGB signal is the Y'CbCr format, which is a transformation of the RGB values. In Y'CbCr (also often referred to as YUV), the *luminance* is stored as a separate component (Y'), while the chroma is stored as the Cb and Cr components.

The human visual system is much more sensitive to variations in brightness than changes in chroma [16]. Storing the luminance as a separate component, allows for this to be taken advantage of. The bandwidth used for the color components can be reduced, without loss of perceived quality. This is done by *subsampling* of the color components, i.e. the color components are stored at a reduced resolution compared to the luma component.

VP8 works exclusively with an 8-bit YUV 4:2:0 image format. For YUV 4:2:0, the horizontal and vertical resolution of the U and V components are both half the resolution of the Y component. Figure 2.1 illustrates 4:2:0 chroma subsampling. Each pixel in the chroma planes (U and V) corresponds to a 2x2 block of pixels in the luma (Y) plane. Each of the color components contain a quarter of the number of pixels as the Y plane, so YUV 4:2:0 requires exactly half the bandwidth compared to RGB or YUV 4:4:4 (no subsampling).

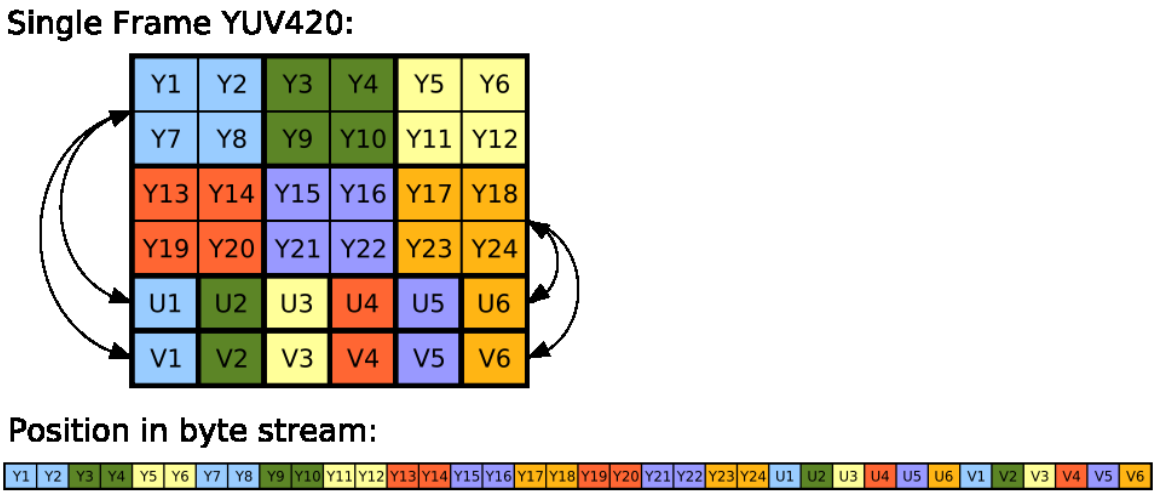


Figure 2.1: YUV 4:2:0 chroma subsampling. Taken from Wikipedia's article on YUV [1].

2.1.3 VP8 Frames and macroblocks

Like most other video codecs, in VP8 each picture is decomposed into smaller square subblocks of pixels. For the Y component, the picture is decomposed into blocks of 16x16 pixels. This is known as a *macroblock*. As noted in section 2.1.2, the chroma components have half the horizontal and vertical resolution of the Y component. A corresponding chroma macroblock hence has dimensions 8x8 pixels.

A macroblock is further decomposed into 4x4 subblocks. For each macroblock, there are 16 Y subblocks, 4 U subblocks and 4 V subblocks. Most of the steps in the encoding process is carried out at the level of macroblocks and subblocks.

In VP8, there are two types of compressed frames. *Intraframes* are represented without reference to any prior frames. They can therefore be decoded independently. In VP8, intraframes are also known as key frames. They can be used as starting points for playback.

Interframes are represented with references to previously encoded frames. Decoding an interframe directly depends on the correct decoding of up to three previous frames,

referred to as

- The *last* frame. This is the immediately previous frame.
- The *altref* frame. An alternative reference frame.
- The *golden* frame.

Every keyframe is automatically both golden and altref, and any interframe may optionally replace them. VP8 does not have bi-directional prediction. There is no concept of B-frames as in MPEG.

2.2 VP8 encoding process

Video compression is the act of compacting a digital video signal into a smaller number of bits. This is achieved through removing *redundancy* in the input video. The idea is to remove *subjective* redundancy, i.e. elements of the input video that can be removed without significantly affecting a viewer's percepted visual quality of the video.

Primarily, the reduction of redundancy happens in the *temporal* and *spatial* domain. The *spatial* domain refers to pixel values within the same picture. Reduction of redundancy in the spatial domain takes advantage of similarity between pixels within the same picture. Compression can be achieved by predicting a pixel from its neighboring values. This is known as *intra prediction*.

The temporal domain refers to pixel values between pictures. Reduction of redundancy in the temporal domain can be taken advantage of due to the fact that there is a high correlation between pixels in successive pictures. The pixels in a region can be predicted with a reference to an earlier encoded picture, along with an offset that specifies where in the reference picture the prediction should be copied from. This offset is known as a *motion vector*.

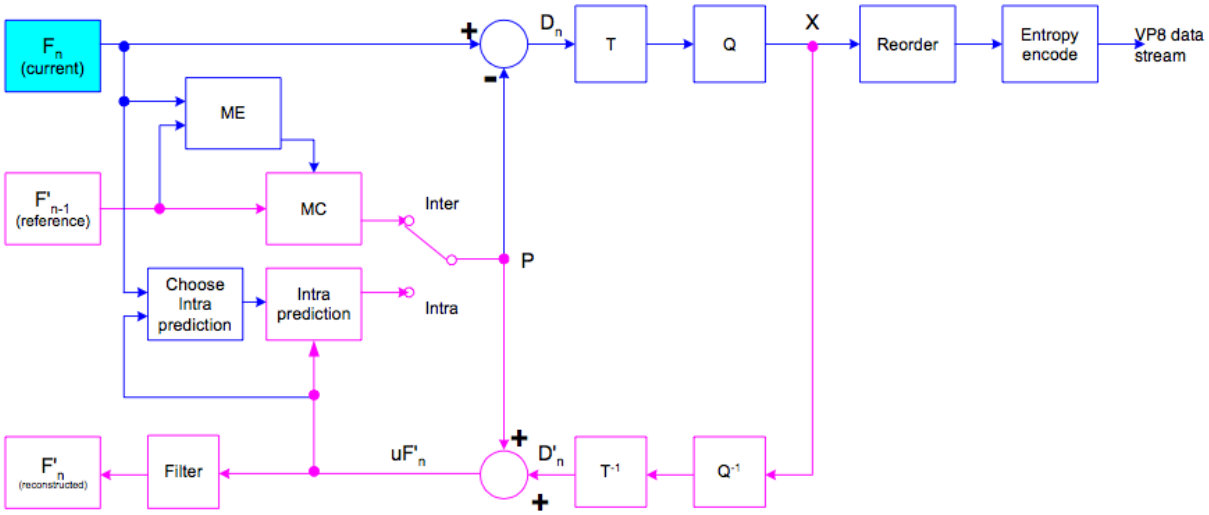


Figure 2.2: Encoder overview for VP8. Borrowed and modified from [2]

Figure 2.2 provides an overview for the VP8 encoding process. Encoding VP8 entails doing analysis (motion prediction), transform, quantization, entropy coding and in-loop filtering. Each of these steps will be described in more detail in the following sections.

2.2.1 Analysis

The analysis stage refers to mode decision and motion prediction. Each macroblock is analyzed to find a suitable mode and prediction.

Intra Prediction

In intra prediction the blocks are predicted from blocks within the same frame. In VP8, there are four macroblock modes for intra prediction.

- **DC_PRED** Predict using the average of above and left pixels.

Figure 2.3: Motion vectors. Inter-coded macroblocks are displayed as green, intra-coded as purple. The line extending from the center of each green block corresponds to the motion vector.

- **V_PRED** Vertical prediction. Predict rows using row above.
- **H_PRED** Horizontal prediction. Predict columns using column to the left.
- **TM_PRED** TrueMotion prediction. Prediction calculated from left, above and single top left pixel.

Prediction of 8x8 chroma macroblocks are restricted to these modes.

In addition, for luma macroblocks, there is **B_PRED**, which specifies that each subblock has its own prediction. There are ten different prediction modes for the 4x4 subblocks. Four of them correspond to the above 16x16 modes, and the last six utilize prediction in a diagonal direction.

Inter Prediction

When encoding an inter frame, all of the above block modes are available, in addition to the following:

- **NEARESTMV**
- **NEARMV**
- **ZEROMV**
- **NEWMV**
- **SPLITMV**

Modes NEARMV and NEARESTMV specify that the motion vector from a neighboring macroblock is used. ZEROMV specifies a motion vector that is zero, i.e. it references the exact same block in the reference picture. NEWMV specifies that a new motion vector should be coded.

SPLITMV specifies that each 4x4 subblock has its own prediction. Possible 4x4 inter modes are LEFT4x4, ZERO4x4, ABOVE4x4 and NEW4x4. LEFT4x4 and ABOVE4x4 specify that the left and above motion vector is to be used, respectively.

The motion vectors work at the quarter-pixel resolution. The fractional pixel values that lie "between" actual pixels are synthesised from applying a filter to the surrounding pixels.

2.2.2 Encoding

The selected best matching prediction from the analysis step (P) is subtracted from the input frame (F_n) to produce a residual frame D_n. Each residual macroblock is transformed into the frequency domain and quantized. The purpose of transforming it to

Figure 2.4: Residual (difference between prediction and input picture). Corresponds to "D_n" in figure 2.2

the frequency domain is to make the values more suited for entropy coding. The entropy is further reduced with quantization, by discarding high-frequency components. Both transform and quantization operates on 4x4 subblocks.

Transform

Each block is transformed to the frequency domain using a discrete cosine transform. The DCT concentrates the most significant coefficients in the top left of the matrix. The first coefficient is known as the DC coefficient, while the remaining 15 coefficients are known as AC coefficients.

For most prediction modes, the DC coefficient of each of the 16 Y subblocks is ex-

Figure 2.5: Frame constructed from prediction data before having residual added. Corresponds to "P" in figure 2.2

pressed via an additional block, referred to as the Y2 block. The purpose of this is to further increase entropy in the DCT blocks. The Y2 block is transformed using a Walsh-Hadamard transform (WHT).

Both the DCT and the WHT in VP8 are defined using exact integer operations, to ensure there is no mismatch between encoding and decoding.

Quantization

Quantization is the actual lossy step of the encoding. Quantization discards high-frequency data from the residuals. This is done by dividing the transformed macroblock by a quantization matrix. The quantization matrix is decided by a quantization

parameter, which applies to all macroblocks of a frame. The quantization parameter is chosen by the encoder, and is adjusted to match a desired bitrate and quality.

Frame Reconstruction

To avoid propagating errors introduced by the quantizer ("drift"), an encoder must use the same decoded reference frames for motion prediction as a decoder uses for motion compensation. It is therefore necessary for the encoder to reconstruct the encoded frame after quantization.

The quantized frame is dequantized and inverse transformed (D'_n). D'_n is added to the prediction frame (P), forming an unfiltered reconstructed frame $u'F_n$.

To complete the reconstruction, $u'F_n$ is filtered to smooth out potential blockiness between edges of macroblocks. This is done not only to improve visual quality, but also to improve motion prediction.

The filtered reconstructed frame is in the encoder placed in a buffer to be used as a reference for subsequent frames.

2.2.3 Entropy Coding

Entropy coding is the final stage of the encoding. All of the information from the other steps are taken in and compressed losslessly to the output file.

2.3 The libvpx Encoder

Libvpx is the official VP8 encoder/decoder reference implementation, which was released as a part of the WebM project. The VP8 bitstream guide [15] states that the reference implementation also serves as the official specification of the format. Our

multi-rate encoder implementation is based on libvpx. This is explored in more detail in chapter 3.

2.3.1 Profiling analysis

In order to identify the most time consuming operations of the encoder, we have performed runtime profiling using the valgrind tool callgrind [17]. The profiling data has been visualized as a call graph using the tool KCachegrind [18]. The results of the profiling are presented in figure 2.6, along with a less detailed zoomed in view in 2.7.

The profiling was run with the same configuration as in our experiments (described in section 4.1), with the test sequence "pedestrian area".

The call graph presented in figures 2.6 and 2.7 displays the number of times each function is invoked, along with the execution time as a percentage of total execution time.

From the profiling analysis we observe that the analysis step is by far the most time consuming component. More than 80% of the execution time is spent in vp8_rd_pick_inter_mode.

vp8_rd_pick_inter_mode is called per macroblock. It iterates through every possible macroblock mode and picks the best prediction according to its calculated cost. If the result of this part can be reused for encoding operations for other bitrates, the resource consumption can be greatly reduced.

2.4 Summary

In this chapter we have introduced concepts in video coding with a description of the VP8 video format. We have given an overview of the implementation of the libvpx VP8 encoder, and through runtime profiling, identified macroblock mode decision and

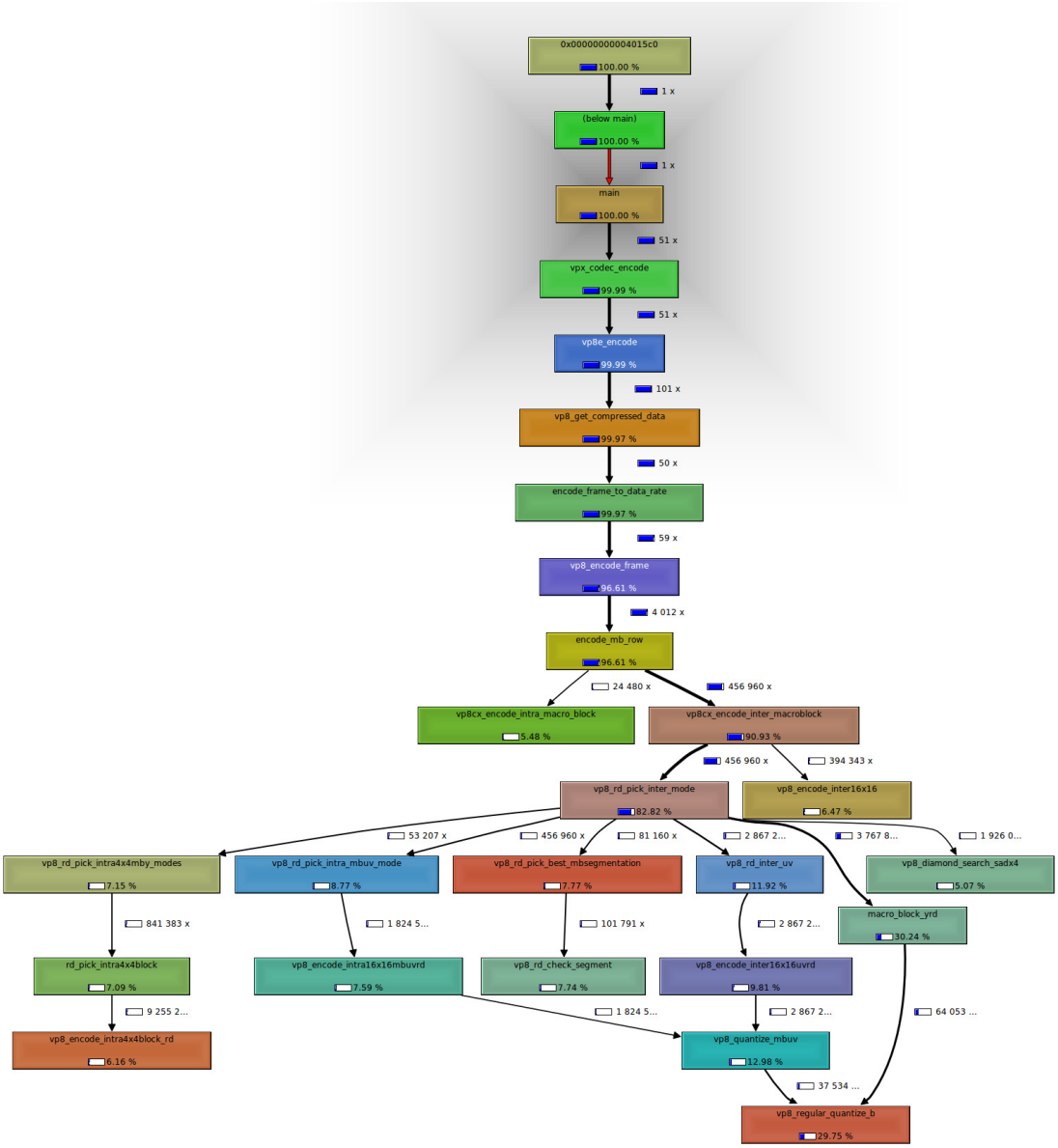


Figure 2.6: KCachegrind profiling data

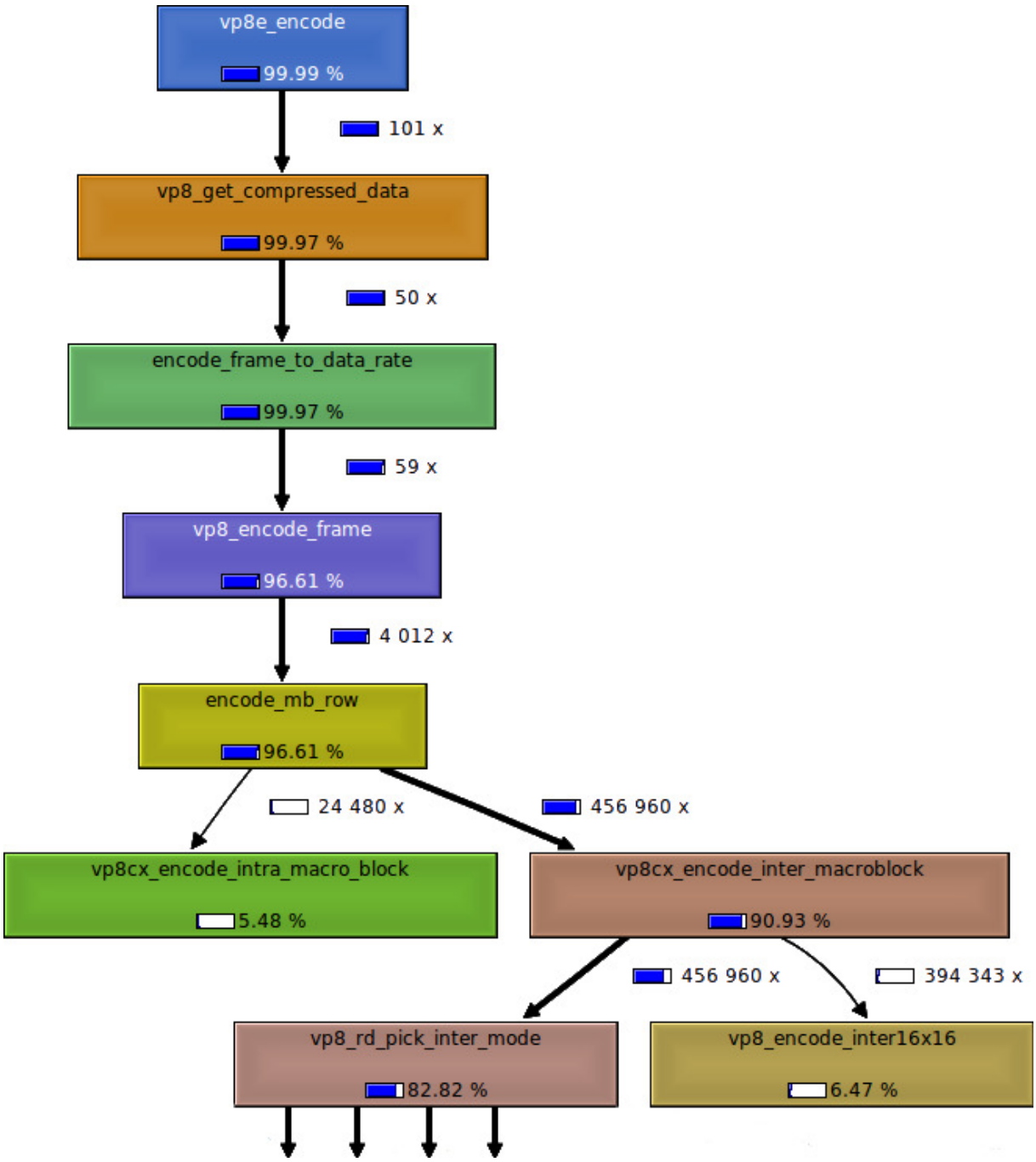


Figure 2.7: Excerpt from figure 2.6

motion prediction as the most time consuming part of the encoder pipeline. In particular, it was observed that over 80% of the execution time was spent in the function vp8_rd_pick_inter_mode, doing motion prediction. In the following chapter, we will look into the possibilities of reusing computations from motion prediction in a multi-rate scenario.

Chapter 3

Design and Implementation of a Multi-rate Encoder

In this chapter, we present the design and implementation considerations for our multi-rate VP8 encoder. We present an overview of the solution and then go into a more detailed discussion of its implementation characteristics.

3.1 Background

The idea of running multiple VP8 encoder instances in parallel is inspired by transcoding approaches trying to reuse motion vectors [12–14]. In [12] the authors discuss transcoding with reuse of motion vectors in the context of spatial downscaling. The paper investigates the statistical characteristics of the macroblocks associated with the best matching motion vectors and define a likelihood score, which is used for picking the motion vectors.

Zhou et al [13] proposes an algorithm for reusing motion vectors in the context of spatial downscaling. Methods for synthesizing a new motion vector by reuse of the motion vectors from the higher resolution bitstream are discussed. A method for re-

fining the synthesized is also discussed. Senda et al [14] describes a realtime software transcoder with motion vector reuse. A method for reusing downscaled motion vectors is discussed, where the authors evaluate scaled motion vectors and their neighbors. A method for reducing the number of candidate motion vectors is proposed, and the best one is picked by finding the one with the lowest mean absolute error.

None of these papers reuse motion vectors for use with several encoder instances targeting different bitrates, they instead address the issue of reusing scaled motion vectors. We want to investigate the possibility for reusing data from parts of the encoding pipeline to be able to output multiple video streams targeting different bitrates.

3.2 Multi-rate Encoding

Our multi-rate encoder is based on the reference VP8 encoder, released as part of the webm project. Provided in figure 2.7 is a call graph of the VP8 reference encoder. In the call graph, we can see the flow of the program, how many times a function have been called, and how large percentage of the execution time is spent in different parts of the code. The basic flow of the entire encoder is illustrated in the upper part of figure 3.1.

The *analysis* part consists of macroblock mode decision and intra/inter prediction, this corresponds to `vp8_rd_pick_inter_mode` in figure 2.7. The *encode* part refers to transform, quantization, dequantization and inverse transform, corresponding to the functions `vp8_encode_inter*` and `vp8_encode_intra*` for the various block modes chosen. *Output* involves entropy coding and writing the output bitstream to file, this part of the encoder is not shown in the call graph. Profiling of the VP8 encoding[1] process shows that during encoding of the *foreman* test sequence, over 80 percent of the execution time is spent in the analysis part of the code, i.e., if this part can be reused for encoding operations for other rates, the resource consumption can be greatly reduced.

[1]Earlier analysis of the x264 processing pipeline found similar results [19].

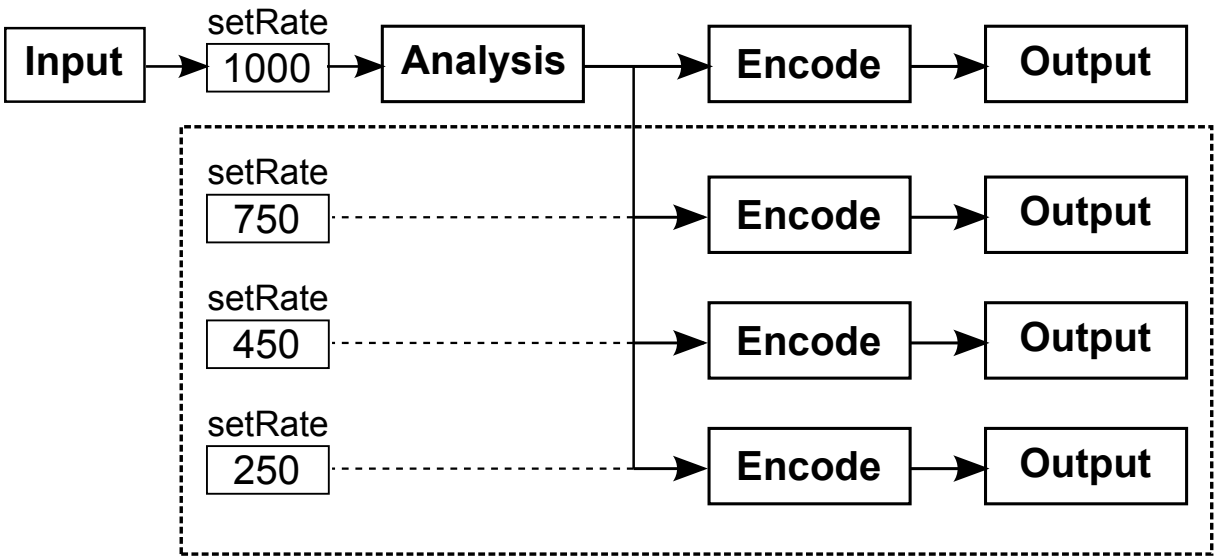


Figure 3.1: Basic flow for the ”multi-rate” VP8 encoder

Our modifications to the VP8 encoder only considers one-pass encoding, which is the predominant mode used for streaming live video. When starting the encoder, a *prediction bitrate* is specified which is used as input to the analysis step for finding intra- and inter-prediction parameters. The encoding instance for the prediction bitrate is considered the main encoder instance; this is the only encoder instance that will run the analysis computations. In the profile of the VP8 encoder seen in figure 2.7, this step is labeled as `vp8_rd_pick_inter_mode`. After the analysis step is completed, the main encoder instance provides the prediction data from the prediction bitrate to the other encoder instances, which will the encode the frame without doing any motion prediction.

Additionally, as seen in figure 3.1, the encoding instances select different target bitrates (giving different quantization parameters). The encoder starts one thread for each specified bitrate where each of these threads correspond to a separate encoding instance. The instances have identical encoding parameters such as keyframe interval, subpixel accuracy, etc., except for the target bitrate provided. Since the bitrate varies, each instance must maintain its own state and reconstruction buffers. The threads are synchronized on a frame by frame basis, where the main encoding instance analyses

the frame before the analysis computations are made available to the other threads. This involves macroblock mode decision, intra- and inter-prediction. The non-main encoding instances reuse these computations directly without doing the computationally intensive analysis steps. Most notably `vp8_rd_pick_inter_mode` (figure 2.7) is only performed by the main encoding instance.

3.3 Implementation details

3.3.1 Adapting libvpx for running multiple instances

The libvpx encoder is not written with the intention of running multiple encoding instances in parallel. The encoder went through significant changes in order to adapt it to run instances in parallel.

A new command line option `-multi-output-brs` was added. This command line takes a comma-separated list of bitrates as input, and enables multi-instance encoding. As described above, each instance will have the same initial configuration, except for the target bitrate. One of the instances will be the designated main instance, which is the instance that will perform all of the analysis steps, while the others will reuse its analysis computations.

Upon initialization of the encoder, a thread is created for each of the encoding instances. Instance-specific code in `main()` was extracted to its own function, for running in its own thread. Likewise, a struct was created for keeping track of various state needed for each encoder instance. There were several concerns regarding global state and race conditions. This was resolved by making variables thread-local and doing extra copying.

The code in in `main()` was adapted to have it simply read input frames, to be consumed by the encoder instances.

For synchronizing the threads, pthread barriers are used. The code is set up so that computations can be reused as described in pseudo code in listing 3.1.

Listing 3.1: Encoder instance synchronization using barriers

```
if(thread is main instance) {
  perform computation
  barrier_wait(b1)
}
else {
  barrier_wait(b1)
  reuse computation
}

barrier_wait(b2)
```

Ideally, the solution could instead be implemented so that the main instance is always kept one frame ahead of the other instances. This way, the other instances could process frame_{n-1} concurrently with the main instance processing frame_n. This would however require additional complexity with regards to buffering of computation results and a more sophisticated synchronization scheme. The idea was thus dropped due to time constraints.

3.3.2 Reuse of motion prediction computations

The call graph in figure 2.7 describes the flow of the encoder when encoding in one-pass mode. Each frame is passed to `encode_frame_to_data_rate()`, which selects a quantization parameter based on the bits available for the targeted bitrate. The rate control scheme for one-pass encoding in libvpx is not overly sophisticated. It makes an initial guess at a quantization parameter, encodes the frame, and then if it significantly undershot or overshot the bits available, it simply recodes the frame with a refined quantizer guess.

Recoding a frame an arbitrary number of times doesn't work with the proposed syn-

chronization scheme (barriers). The priority was to get the prototype working, so the one pass rate control was further simplified. The rate control for the designated main instance is unchanged, but for the non-main instances the initial guess at quantizer is always used.

When the main encoding instance is finished encoding its frame and the prediction computations are ready, the other instance will run, utilizing the computations from the main instance.

`vp8_encode_frame()` sets up various data structures and iterates through the rows of the frame, invoking `encode_mb_row()` for each macroblock row.

In `encode_mb_row()`, each macroblock of the row is processed by calling `vp8cx_encode_intra_macro_block()` for the main instance, and `vp8cx_encode_intra_macro_block_nonmain()` for the other instances. The latter version of the function will completely skip motion prediction (`vp8_rd_pick_inter_mode`), and instead simply copy the contents of the corresponding `MODE_INFO` and `PARTITION_INFO` structs from the main encoding instance.

It is worth noting that only the prediction context is copied, i.e. the block mode decisions and motion vector information. Each encoder instance build their own predictors and calculate their own residuals as per usual to ensure no encoder-decoder mismatch.

3.4 Summary

In this chapter, the design and implementation considerations for our multi-rate encoder has been presented. In the following chapter we move on to evalute the performance of our solution.

Chapter 4

Experiments

In this chapter we perform experiments on the libvpx encoder with the modifications described in chapter 3. The purpose of these experiments is to evaluate how our implementation performs, both in terms of CPU time and visual quality of the resulting video encodes. The experiments are performed using several test sequences, and the results are compared to a reference version of the libvpx encoder.

4.1 Test environment

4.1.1 Input test sequences

In our testing, we have performed experiments targeting the scenarios of streaming to *mobile devices* and streaming to *HD* devices. The *mobile devices* scenario is characterised by low resolution video encoded at lower bitrates, typically suited for streaming to handheld devices connected through cellular networks. For the *HD* scenario we have chosen HD resolution test sequences encoded at higher bitrates. Since we reuse motion vectors for the encoding, we looked at different videos with different amount and kind of motion.

For the *mobile devices* scenario, we used the standard test sequences *foreman* and *akiyo* in CIF format (352x288, 29.97 FPS). We have chosen target bitrates of 250, 450, 750 and 1000 kbps for the different quality levels. Akamai [20] recommends that video should be encoded at 250 kbps for low quality and 450 kbps for high quality. Typical 3G networks can deliver bandwidths of 384 kbps (UMTS) to 7.2 Mbps (HSDPA).

The *foreman* test sequence is very high in motion. It is shot with a very unstable handheld camera and displays a man talking, with very lively and detailed facial expressions. Towards the end of the sequence, the camera pans to the side, and we are presented a more static view of a building.

The *akiyo* sequence shows a TV news presenter sitting in front of a static background. The camera is in a fixed position, and there is very little movement.

To test the other end of the scale, we have also performed experiments using *HD* resolution test sequences. The typical target audience for this scenario is home users with consumer internet connections and *HD* displays. Typical *ADSL* lines can deliver from about 750 kbps to 8 Mbps, and for this scenario we encode the test sequences with target bit rates of 1500, 2000, 2500 and 3000 kbps.

The chosen *HD* test sequences are the standard test sequences *pedestrian area* and *blue sky*. Both of these are in 1080p resolution, 25 frames per second. The *pedestrian area* test sequence shows a shot of a pedestrian area, with moving objects (people) passing by close to the camera with a mostly static background. The camera position is fixed. *blue sky* is a shot of the top of two trees against a blue sky, with the camera moving. The test sequence is high in contrast and detail, with small color differences in the background sky.

4.1.2 Test setup

Our work is based on version v0.9.5-173-geb8b4d9 of the libvpx encoder, and this is also the version we have used for testing against. Listing 4.1 shows the command line used for running the reference encoder. The multi-rate encoder used the exact same

configuration, with the exception of specifying several bitrates at once, using the new command line option `--multi-output-brs`.

Listing 4.1: Reference encoder command line

```
./vpxenc FILE -o output_vp8.webm \
 -p 1 -t 4 \
 --good --cpu-used=0 --target-bitrate=BR --end-usage=1 \
 --fps=FPS -v \
 --kf-min-dist=0 --kf-max-dist=500 \
 --token-parts=2 --static-thresh=0 \
 --min-q=0 --max-q=63
```

All experiments were performed on a test machine with a 4-core Intel Core i5 750 and 4 GB of memory. The test machine runs Ubuntu Linux, with kernel version 2.6.32-24.

The PSNR values used in the rate-distortion curves are measured by the libvpx encoder. The CPU time consumed is measured using *time*.

4.1.3 Metrics

Visual quality is inherently a subjective matter, and is therefore very difficult to measure objectively. However, in order to obtain reliable and repeatable results, objective measures are needed.

For evaluating the performance of the multi-rate encoder, we have plotted average PSNR vs bitrate, producing a *rate-distortion curve*.

PSNR is calculated by the libvpx encoder, by application of the following formula

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{4.1}$$

where *MSE* is the *mean squared error* between the encoded video and the input source. This is calculated from the sum of all squared value differences for each color plane Y, U and V.

Another widely used objective quality metric is structural similarity (SSIM) [21]. SSIM was designed to be more consistent with actual subjective perception. Calculation of the SSIM index is based on comparing the structural information in the image, along with contrast and luminance. This comparison is more in line with the properties and perceptions of the human visual system [21].

Even though SSIM in the general case is a better metric for quality comparison than PSNR, there are still cases where PSNR is highly relevant. The validity of PSNR as an objective video quality metric is investigated in [22]. It is shown that PSNR is a perfectly good metric for comparing the variation of video quality when comparing a codec to itself on individual clips.

To expand on this, it is found that that when comparing the quality of some video clip to the quality of a different video clip, PSNR is not a very reliable metric. Also, it is not very reliable when comparing the performance of a codec to the performance of another different codec. In our case, we only compare content encoded with the same codec, so we trust that PSNR is a valid metric.

For visual inspection, we also include sample pictures from the encoded videos.

4.2 Results

4.2.1 Encoding results

To evaluate our multi-rate encoder, we have first plotted the total CPU time used when encoding the *foreman* sequence in figure 4.1a for the four different output rates. To see if there is a difference for different chosen *prediction bitrates* when using the multi-rate encoder, we have included one test for each prediction bitrate. These results are

compared to the combined CPU time used when encoding the videos for the same rates using the reference encoder with both a single thread and multiple threads. The CPU time used in the multi-rate approach is more than 2.5 times faster than encoding the four sequences using the reference encoder. The multi-rate approach scales further if the number of encoded streams is increased. In addition, the time spent in kernel space is far less in the multi-rate approach compared to the reference encoder, and we believe this is a result of reading the source video from disk only once.

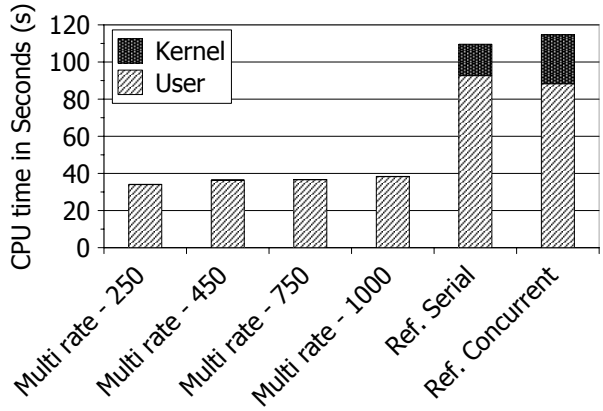
The *akiyo* results are plotted in figure 4.2a. We observe a performance gain comparable to that of the *foreman* sequence.

To see if there are differences between low and high resolution videos, we have also looked at *HD* sequences to validate our approach. Figure 4.3a shows the "pedestrian area" test clip with a *prediction bitrate* of 2000 kbps. We observe a 2.06 times reduction in CPU time for the multi-rate encoder as we saw for the *foreman* sequence.

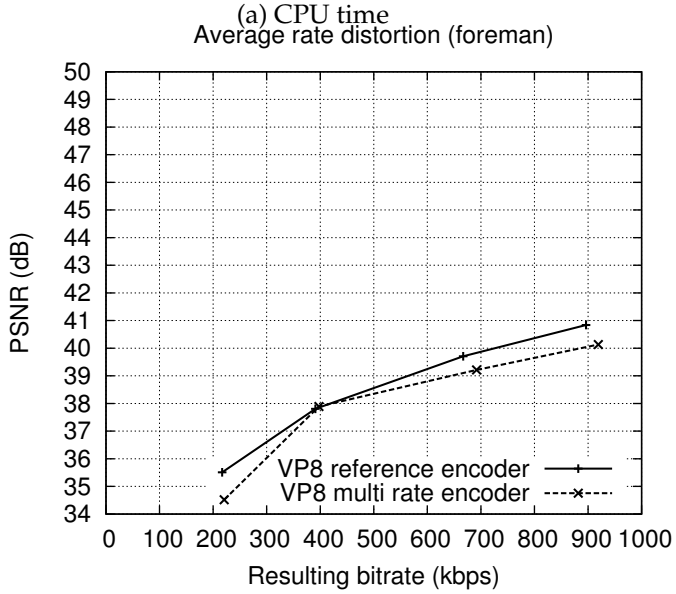
As noted in 4.1.1, we also looked at the test sequence "blue sky", to see how the multi-rate encoder performs on input which has different amount and kind of motion. The "blue sky" results are plotted in figure 4.4a with a performance gain of 2.47 times the performance of the reference encoder. Thus, for all our experiments using different rates, resolutions and content types, our multi-rate encoder reduce the total resource consumption.

4.2.2 Quality assessment

Using prediction parameters generated from a different bitrate than the target bitrate does have implications for the video quality. To investigate the trade off between reduced processing time versus degraded video quality, we have plotted a rate-distortion curve for the *foreman* sequence with a *prediction bitrate* of 450 kbps in figure 4.1b. We can see that reference encoder produces about 1 dB higher PSNR at the same bitrate than the multi-rate encoder. Depending on the intended usage, the significantly reduced CPU time might outweigh the small reduction in quality.

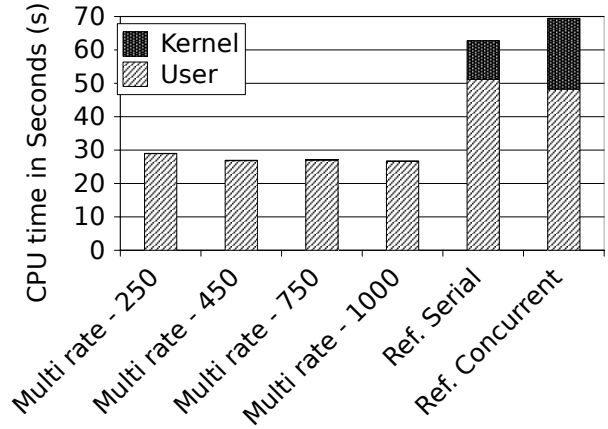


(a) CPU time

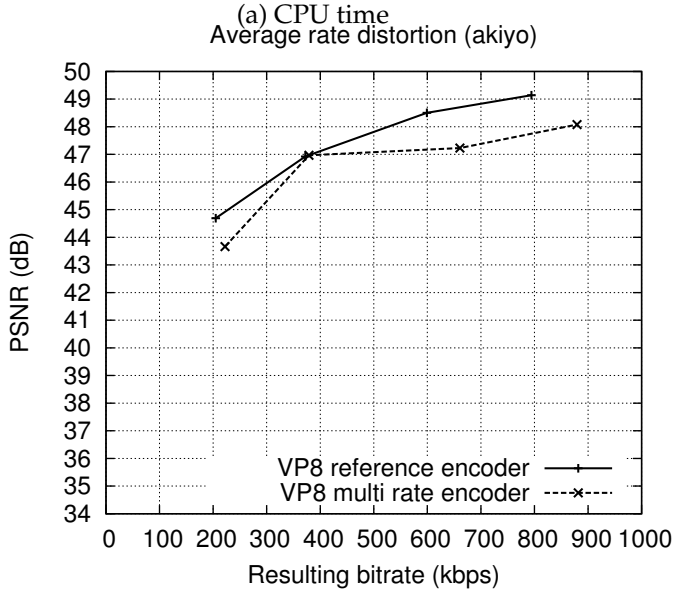


(b) Rate-distortion curve

Figure 4.1: CIF streaming scenario (“foreman”)

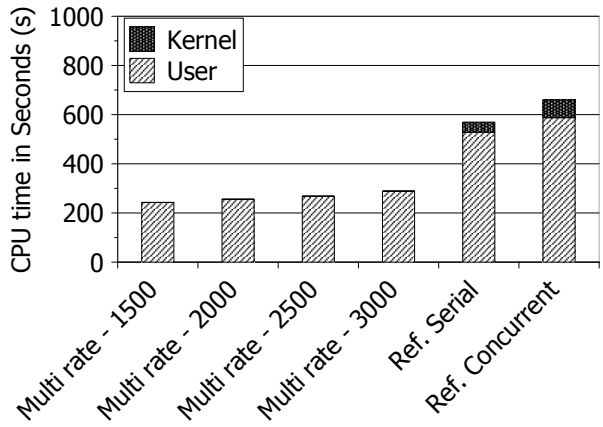


(a) CPU time

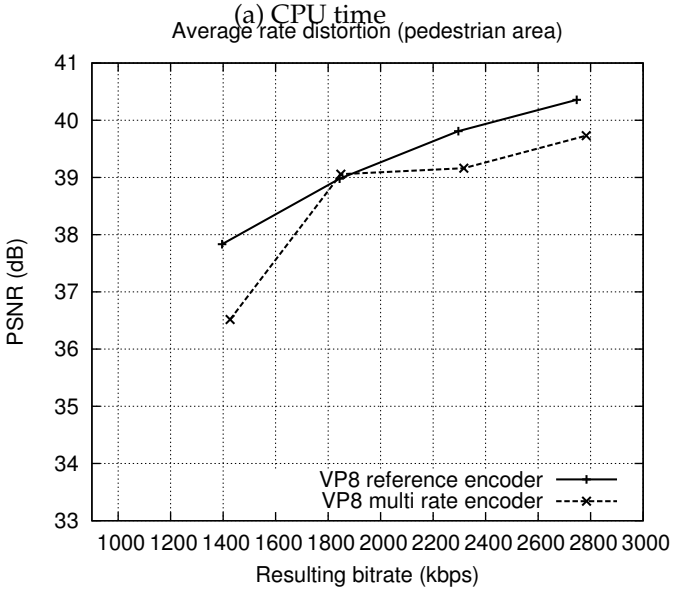


(b) Rate-distortion curve

Figure 4.2: CIF streaming scenario (“akiyo”)

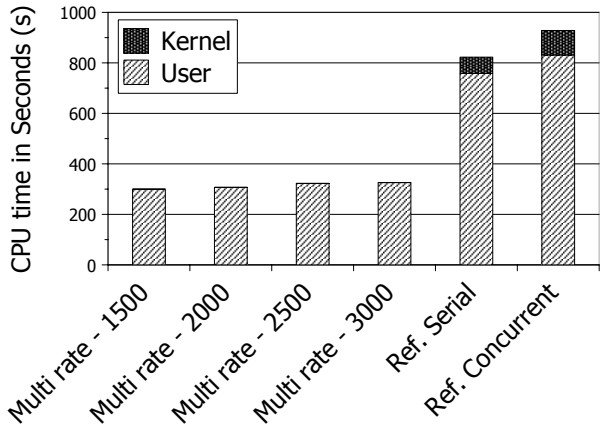


(a) CPU time

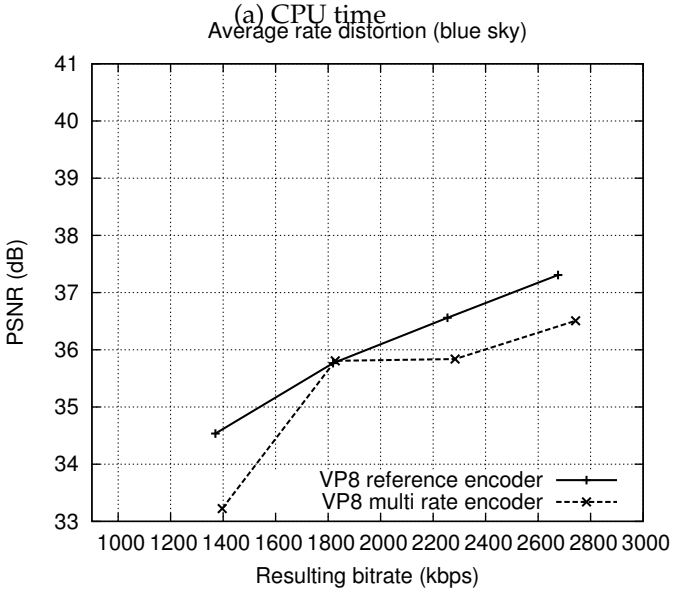


(b) Rate-distortion curve

Figure 4.3: HD streaming scenario (“pedestrian area”)



(a) CPU time



(b) Rate-distortion curve

Figure 4.4: HD streaming scenario (“blue sky”)

Similarly, when considering the distortion of the *HD* sequences, we have plotted rate-distortion curves in figure 4.3b and 4.4b for *pedestrian area* and *blue sky*, respectively. The reference encoder produces output that has 1.0 to 1.5 dB higher PSNR than the multi-rate encoder and distortion achieved for the two *HD* clips are very similar.

As noted in section 4.1.3, the suitability of PSNR for video quality assessment is frequently discussed, and it is often unclear what the difference means in terms of the logarithmic scale. From the plot in figure 4.3b, we can see that the PSNR of the output from the reference encoder is up to 1.32 dB better than the multi-rate encoder outputs for the *pedestrian area* sequence, in the range of 1500 kbps to 3000 kbps. To see what this really means, a sample output of the "worst-case" scenario from figure 4.3b can be seen in figure 4.5. From this output, we can see that there is little visual difference between the reference encoder output and the multi-rate encoder. We also looked at the average structural similarity (SSIM) index number for the reference encoder and the multi-rate encoder. The SSIM numbers are 0.861 and 0.837, respectively, i.e., the difference is small. Thus, the quality reduction is small (we did not see notable difference viewing the resulting videos, but it might be different for other types of content).

Signs of discolored artifacts are sometimes observed in the encoded videos. In the particular picture (figure 4.5b), we observe this effect. Although the quality difference in terms of detail or structure is not very noticeable, a discoloration can be seen in the foreground sidewalk. These artifacts are introduced due to errors in the reuse of prediction data for the U and V color spaces.

For further visual comparison, we have also made sample videos available at `http://folk.uio.no/daghf/vp8/`.

4.2.3 Choosing the prediction bitrate

To evaluate which *prediction bitrate* gives the minimal distortion of the videos, we have plotted rate-distortion curves for *foreman* with various prediction rates in figure 4.6. We can see that the resulting bitrate is lower for the multi-rate encoder than the refer-

(a) Reference encoder

(b) Multi-rate encoder

Figure 4.5: Frame number 100 of the test sequence ”pedestrian area”, displaying the quality difference for the “worst-case” scenario in figure 4.3b of 1.32 dB PSNR of 1500 kbps

ence encoder, except for when the prediction bitrate exactly matches the target bitrate, resulting in a small spike in the plot.

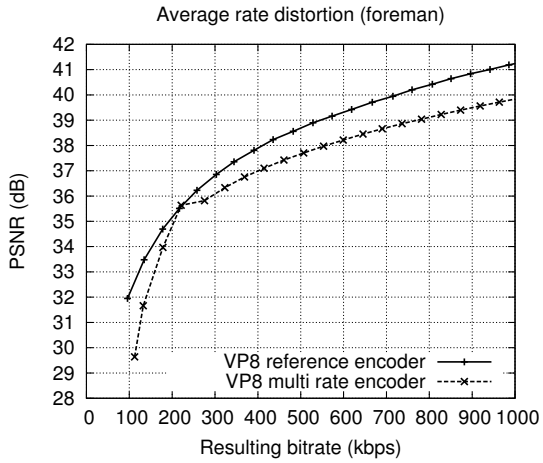
The lowest *prediction bitrate* (250 kbps) incurs the largest distortion difference of 2 dB for the 1000 kbps *resulting bitrate*. When using a 450 kbps *prediction bitrate*, the distortion difference is about 1 dB for bitrates between 250 kbps and 1000 kbps. By further increasing the *prediction bitrate*, we see that the distortion difference between the multi-rate and reference increases to 4 dB for the lowest output 250 kbps. Thus, the smallest distortion can be observed when using a *prediction bitrate* close to the average of the smallest and highest output bitrate, and we get a smaller penalty when the *prediction bitrate* is smaller than the output bitrate than vice versa.

Similar results can be observed when evaluating the *pedestrian* sequence, shown in figure 4.7. Lower *prediction bitrates* incur less distortion difference than higher *prediction bitrates* compared to the *target bitrate*. The distortion difference is further reduced by choosing a bitrate closer to the average of the extremes.

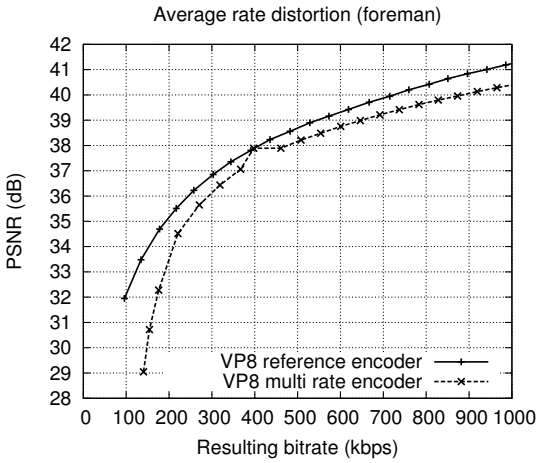
We have shown that choosing the correct *prediction bitrate* when doing multi-rate encoding has a highly significant effect on the quality of the output videos. Although CPU time was also affected as shown in figure 4.1a, the difference was much less considerable. Because of the distortion, having a too wide range of *target bitstreams* when doing multi-rate encoding is discouraged (see for example figure 4.7d), but for quality ranges typically used in segmented streaming as shown in our test sequences, the results prove that multi-rate encoding is useful.

4.3 Discussion and open issues

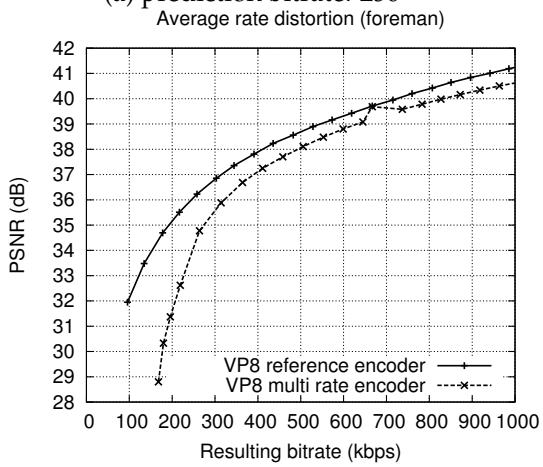
To support a wide range of devices and network conditions, most video service providers today use an adaptive, multi-rate HTTP streaming solution. In this respect, encoding the video into multiple qualities is an expensive operation. The idea investigated here is to reuse the results from the most expensive operations, share and reuse the compu-



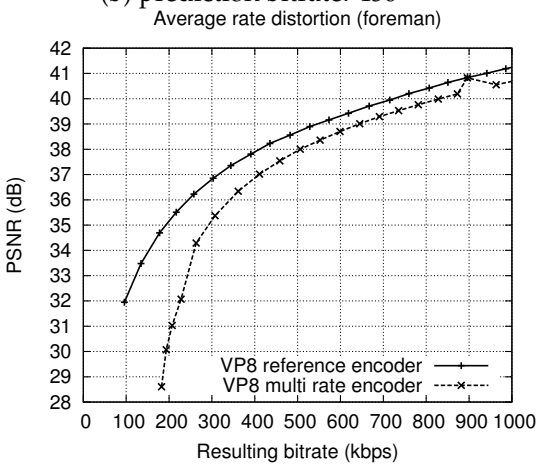
(a) prediction bitrate: 250



(b) prediction bitrate: 450

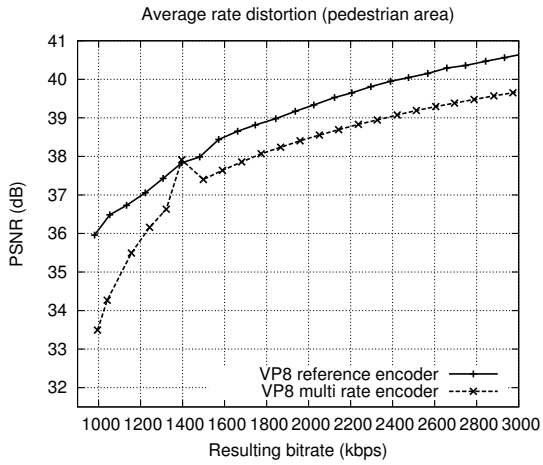


(c) prediction bitrate: 750

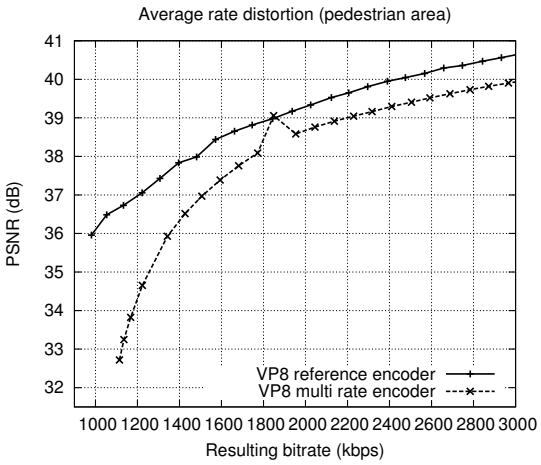


(d) prediction bitrate: 1000

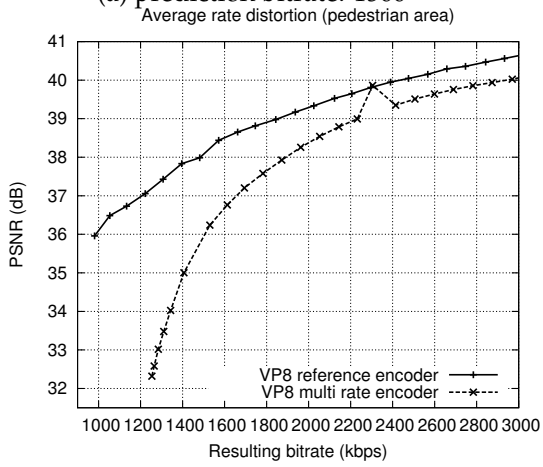
Figure 4.6: Rate-distortion curve for CIF test sequence ″foreman″ with different prediction bitrate (in kbps)



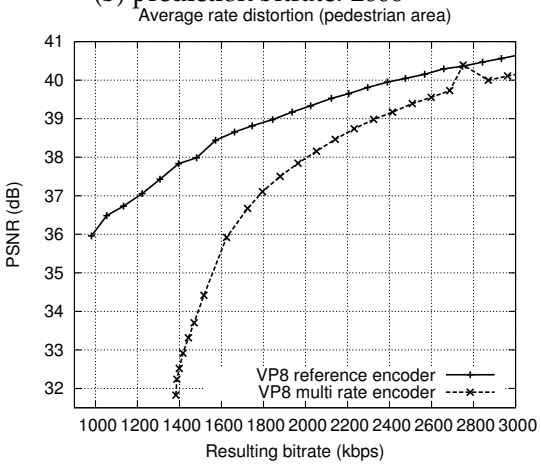
(a) prediction bitrate: 1500



(b) prediction bitrate: 2000



(c) prediction bitrate: 2500



(d) prediction bitrate: 3000

Figure 4.7: Rate-distortion curve for HD test sequence ”pedestrian area” with different prediction bitrate (in kbps)

tational heavy intermediate steps from analysis computations, in order to reduce the processing requirement.

To prove the idea, we have implemented a prototype trying to reuse the most expensive operations based on profiling of the encoding pipeline. In particular, our multi-rate encoder reuses the analysis part consisting of macroblock mode decision and intra/inter prediction. The experimental results indicate that we can encode the different videos at the same rates with approximately the same qualities compared to the VP8 reference encoder, while reducing the encoding time significantly. However, our prototype is a small proof-of-concept, and there are numerous open issues.

In the prototype, we used VP8 as a case study since it is an emerging open-source codec. However, VP8 is for example very similar to the baseline profile in H.264, and in general, most video codecs use similar ideas for compression. Thus, our ideas are not implementation specific to VP8, but also applicable for other codecs like MPEG-1/2/4, H.263/4, VC-1/2/../8, Theora, etc., which compress the video data in a similar way.

One open issue is looking into solutions for improving the quality for the other bitrates, aside from correctly choosing the prediction bitrate. By virtue of our method of reusing analysis computations directly, the quality will suffer when the target bitrate is not equal to the prediction bitrate. One potential quality improvement could be to do *predictor refinement*, inspired by the approach taken in [13]. This would however lead to increased complexity in the encoder. Section 4.2.3 demonstrates how reuse of the analysis computations impacts the quality/complexity trade off for encoding the same input at different rates. It would also be interesting to look at this using a more systematic approach, and see how it affects specific prediction modes. A limitation with our multi-rate encoder is that all the bitstreams encoded must use the same number of reference frames, or in the case of VP8, the same golden frame for the method to be viable. Another potential for further work is to investigate if there are other parts of the VP8 encoder where the processing can be fanned out like in the analysis step.

Chapter 5

Conclusion

5.1 Summary and contributions

Encoding video into multiple bitrates for adaptive streaming over various networks to different end-devices is a resource expensive task. We have investigated the effect of running multiple encoding instances in parallel, where the different instances reuse intermediate results. This way, several encoding steps are avoided for the subsequent encoding operations. In particular, we have analyzed and performed experiments with Google's VP8 encoder, encoding different types of video to multiple rates for various scenarios.

Our main contribution is that we propose a way of reusing decisions from intra and inter prediction in the video encoder to avoid computational expensive steps that are redundant when encoding for multiple target bitrates of the same video object. The method can be used in any video codec comprising an analysis and encoding step with similar structure as H.264 and VP8. Furthermore, The method has been implemented in the VP8 reference encoder as a case study, and the experimental results show that the computational demands are significantly reduced at the same rates and approximately the same qualities compared to the VP8 reference implementation, i.e., for a negligible quality loss in terms of PSNR, the processing costs can be greatly reduced.

However, the quality loss is dependent on the distance from the initial bitrate, i.e., if the gap between the output bitrates is too large, the quality loss become larger. In such scenarios, we still need multiple instances of the whole operation.

5.2 Further work

The work described in this thesis is to be considered a proof of concept implementation, and has potential for improvement. Several open issues have also been addressed in section 4.3.

5.2.1 Visual quality

One change that could be made to our implementation would be to make sure U and V channels get a correct prediction. As noted in section 4.2.2, this has a visible impact on the quality of the resulting encode. We anticipate that doing explicit motion prediction for U and V channels would not have a significant impact on the total CPU time used.

5.2.2 Performance

- Increase parallelity in the multi-rate encoder. Discussed in section 3.3
- Consider more reuse (yuv420 decoding?)

Bibliography

[1] Wikipedia. Yuv — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/YUV`, 2011. [Online; accessed 22-July-2011].

[2] I.E.G. Richardson. *H. 264 and MPEG-4 video compression*. Wiley Online Library, 2003.

[3] Alex Zambelli. Smooth streaming technical overview. http://learn.iis.net/page.aspx/626/smooth-streaming-technical-overview/, 2009.

[4] Move Networks. Internet television: Challenges and opportunities. Technical report, Move Networks, Inc., November 2008.

[5] Stephanie Lyn Flosi. comScore releases April 2010 U.S. online video rankings. Press release, comScore, Inc., June 2010.

[6] Cisco Systems, Inc. Visual networking index. http://www.cisco.com/en/US/-netsol/ns827/networking_solutions_sub_solution.html, May 2010.

[7] Roger Pantos, Jim Batson, David Biderman, Bill May, and Alan Tseng. HTTP live streaming. http://tools.ietf.org/html/draft-pantos-http-live-streaming-04, 2010.

[8] Adobe. HTTP dynamic streaming on the Adobe Flash platform. http://www.adobe.com/products/httpdynamicstreaming/-pdfs/httpdynamicstreaming_wp_ue.pdf, 2010.

[9] Thomas Stockhammer. Dynamic adaptive streaming over HTTP - standards and design principles. In *Proc. of ACM MMSys*, pages 133–144, 2011.

[10] Patrick Seeling, Frank H. P. Fitzek, Gergö Ertli, Akshay Pulipaka, and Martin Reisslein. Video network traffic and quality comparison of vp8 and h.264 svc. In *Proc. of MoViD*, pages 33–38, 2010.

[11] Jan Ozer. First look: H.264 and vp8 compared, May 2010. http://www.streamingmedia.com/articles/editorial/featured-articles/first-look-h.264-and-vp8-compared-67266.aspx.

[12] C.-C.J. Kuo and N. Jayant. An adaptive non-linear motion vector resampling algorithm for down-scaling video transcoding. In *Proc. of ICME*, pages 229–232, July 2003.

[13] Hong Zhou, Jingli Zhou, and Xiaojian Xia. The motion vector reuse algorithm to improve dual-stream video encoder. In *Proc. of ICSP*, pages 2359–2362, October 2008.

[14] Y. Senda and H. Harasaki. A realtime software mpeg transcoder using a novel motion vector reuse and a simd optimization techniques. In *Proc. of ICASSP*, pages 2359–2362, March 1999.

[15] James Bankoski, Paul Wilkins, and Yaowu Xu. VP8 data format and decoding guide. IETF Internet-Draft, March 2011.

[16] C.J.B. Lambrecht. *Vision models and applications to image and video processing*. Kluwer Academic Publishers, 2001.

[17] Callgrind. `http://valgrind.org/docs/manual/cl-manual.html`. [Online; accessed 22-July-2011].

[18] Kcachegrind. `http://kcachegrind.sourceforge.net/html/Home.html`. [Online; accessed 22-July-2011].

[19] HÃ¥vard Espeland. Investigation of parallel programming on heterogeneous multiprocessors. Master's thesis, University of Oslo, Norway, July 2008.

[20] Akamai. Akamai HD for iPhone encoding best practices. http://www.akamai.com/dl/whitepapers/-Akamai_HDNetwork_Encoding_BP_iPhone_iPad.pdf, 2010.

[21] Z. Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.

[22] Q. Huynh-Thu and M. Ghanbari. Scope of validity of PSNR in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.