2011 IEEE International Symposium on Multimedia

Improved Multi-Rate Video Encoding

Dag H. Finstad, Håkon K. Stensland, Håvard Espeland, Pål Halvorsen
Simula Research Laboratory, Norway
Department of Informatics, University of Oslo, Norway
Email: {daghf, haakonks, haavares, paalh}@ifi.uio.no

***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 paper, 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.

I. Introduction

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 [1], 2010 FIFA World Cup [2] and NFL Super Bowl [2] 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 [3], and leading industry movers conjecture that traffic on the mobile-phone networks will also soon be dominated by video content [4].

The currently de facto video delivery solution in these scenarios is adaptive streaming over HTTP [1], [2], [5]–[7]. 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 [8], [9]. 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 [10]–[12], 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.

The rest of this paper is organized as follows. Section II describes some related work. In section III, we briefly outline the VP8 processing pipeline before investigating how we can build an efficient multi-rate VP8 encoder in section IV. Section V presents our experimental results, and section VI gives a discussion of the solution and the results. Finally, we summarize, conclude and give direction for further research in section VII.

II. Related Work

The idea of running multiple VP8 encoder instances in parallel is inspired by transcoding approaches, trying to reuse motion vectors [10]–[13]. In [10], the authors discuss transcoding with the 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 then used for picking the motion vectors.

Zhou et al. [11] propose an algorithm for reusing motion vectors in the context of spatial downscaling. Methods for synthesizing a new motion vector by reuse of the original motion vectors from the higher resolution bitstream are discussed. A more advanced method for refining the synthesized vectors is also discussed. Senda et al. [12] describes a real time software transcoder with motion vector reuse. They discuss a method for reusing downscaled motion vectors, which evaluate the scaled motion vectors and their neighbors. A new 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.

Youn at al. [13] also investigates transcoding. They have observed that reusing the incoming motion vectors become non optimal, due to the reconstruction errors. They are proposing to use a fast-search adaptive motion vector refinement to improve the motion vectors. In [14], the authors also propose a new algorithm to do fast inter-mode decision and motion estimation for H.264. Both these approaches are different from our proposed solution. We are reusing data from the motion estimation, however, we are not using any new algorithms to refine the analysis data before reusing it for other bitrates.

None of these papers reuse the analysis step for use with several encoder instances. We want to investigate the possibility for reusing data from parts of the encoding pipeline to be able to output multiple video streams. Thus, we next present a brief overview of the VP8 codec where we show the basic processing pipeline and some profiling results to identify the most expensive operations.

III. The VP8 Codec

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.

The VP8 codec is heavily influenced by H.264. It has similar functionality as the H.264 Baseline Profile, but with the additional benefit of having an adaptive binary arithmetic coder instead of CAVLC. VP8 is not designed to be an all-purpose codec, but instead targets web and mobile application. Hence, VP8 has omitted features such as interlacing, scalable coding, slices and color spaces other than 4:2:0. This reduces encoder and decoder complexity while retaining video quality for the most common use case, i.e., making VP8 a good choice for lightweight devices with limited resources.

A VP8 frame are either of type *intra-frame* or *inter-frame*, corresponding to I- and P-frames in H.264, and it has no equivalent to B-frames. In addition, VP8 introduces the concept of tagging a frame as *altref* and *golden* frames, which are stored for reference in the decoder. When predicting, blocks may use regions from the immediate previous, from the last *golden* or from the last *altref* frame. Every key frame is both a *golden* and *altref* frame; other frames can optionally be marked as *golden* or *altref*. *Altref* frames are special and never shown to the user, instead they are only used for prediction.

Furthermore, the encoding loop of VP8 is similar to that of H.264 consisting of intra/inter prediction, DCT, quantization, dequantization, iDCT, followed by an in-loop deblocking filter. The result of the quantization step is entropy coded using a context adaptive boolean entropy coder and stored as the output bitstream. The output bitrate of the resulting video is thus dependent on prediction parameters in the bitstream and quantization parameter.

IV. Multi-Rate Encoding

Our multi-rate encoder is based on the reference VP8 encoder, released as part of the webm project. Provided in figure 1 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 2.

The *analysis* part consists of macroblock mode decision and intra/inter prediction, this corresponds to `vp8_rd_pick_inter_mode` in figure 1. The *encode* part refers to transform, quantization, dequantization and inverse transform, corresponding to the functions `vp8_encode_inter*` and `vp8_encode_intra*` for

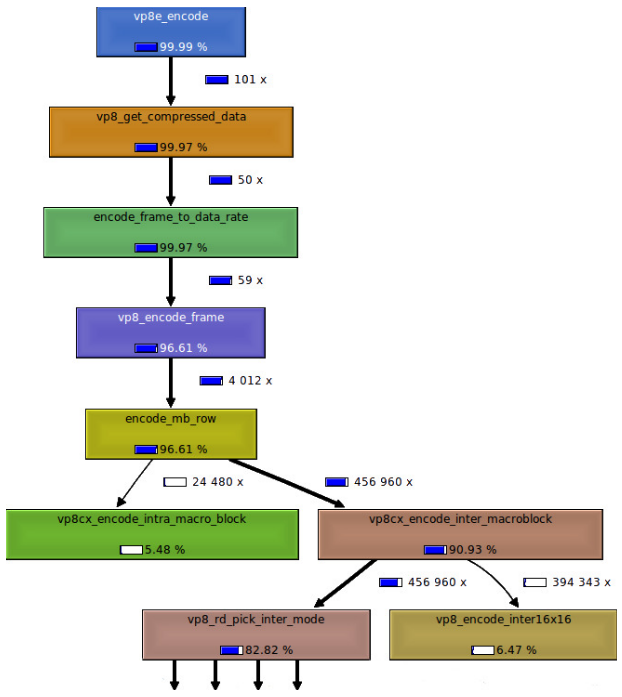


Figure 1. Profile of the main parts of the reference VP8 encoder

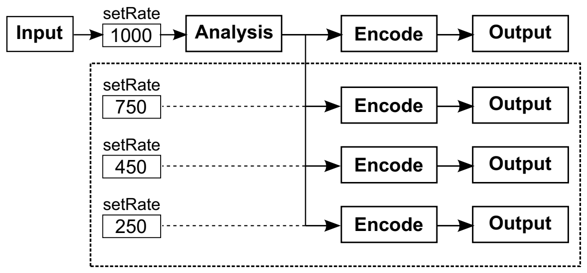


Figure 2. Basic flow for the "multi-rate" VP8 encoder

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.

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 1, this step is labeled as `vp8_rd_pick_inter_mode`. After the analysis step is completed, the main encoder instance feeds images and the prediction data from the target bitrate to the other encoder instances, which will only run the `vp8cx_encode_intra_macro_block` and `vp8_encode_inter16x16` part of the code.

Additionally, as seen in figure 2, 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 1) is only performed by the main encoding instance.

[1]We have also earlier analyzed the x264 processing pipeline and found similar results [16].

V. Experiments

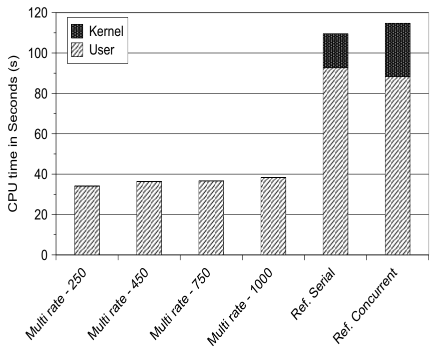
We have performed experiments for several scenarios. One example is streaming to *mobile devices* over 3G networks. Here, Akamai [17] recommends that video should be encoded at 250 kbps for low quality and 450 kbps for high quality. Typical 3G networks can deliver bandwidths of 384 kbps (UMTS) to 7.2 Mbps (HSDPA), and we have here measured the resource consumption for coding the standard *foreman* test sequence in *CIF* resolution using bitrates of 250, 450, 750 and 1000 kbps. Furthermore, to also test the other end of the scale, we have evaluated *HD* resolution videos. Typical ADSL lines can deliver from about 750 kbps to 8 Mbps, and for this scenario, we have performed experiments using the 1080p resolution standard test sequences *pedestrian* and *blue sky* encoded at 1500, 2000, 2500 and 3000 kbps. To measure the performance, we have used *time* to measure the consumed CPU time. Additionally, to evaluate the resulting video quality, the PSNR values are measured by the VP8 video encoder. Bitrates shown in the plots are the *resulting bitrates* achieved in output bitstreams, and not the specified *target bitrates*. Resulting bitrates are generally a bit lower than target bitrates because of a conservative rate estimator. All experiments were run on a 4-core Intel Core i5 750 processor.

A. Encoding results

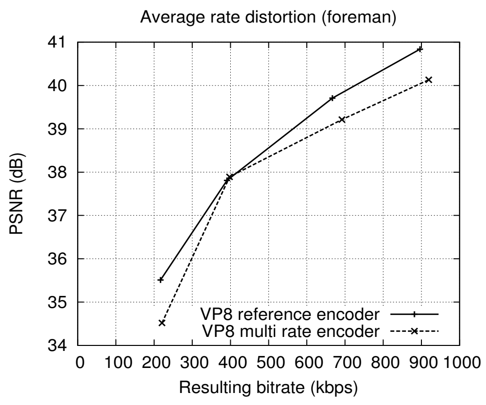
To evaluate our multi-rate encoder, we have first plotted the total CPU time used when encoding the *foreman* sequence in figure 3(a) for the four different output rates. To see if there is a difference for different chosen *prediction bitrates* when using the multi-rate encoder, we have included one 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.

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(a) shows the "pedestrian" 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.

Finally, since we reuse motion vectors for the encoding, we looked at different videos with different amount and kind of motion. The "pedestrian" has a fixed camera with objects (people) moving. The "blue sky" video has more or less fixed objects, but with a moving camera. The "blue sky" results are plotted in figure 5(a) with a performance gain of 2.47 the performance of the reference encoder. Thus, for all our experiments using different rates, resolutions and

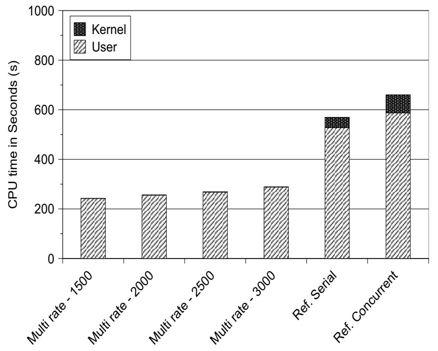


(a) CPU time

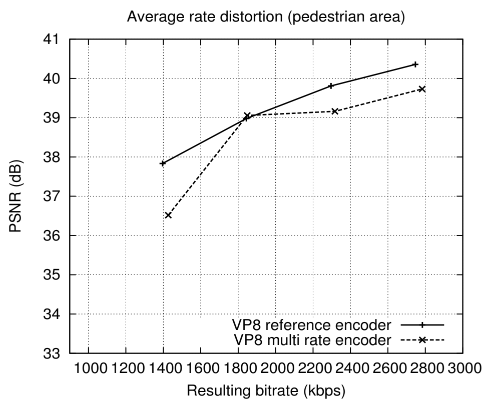


(b) Rate-distortion curve

Figure 3. CIF streaming scenario (“foreman”)

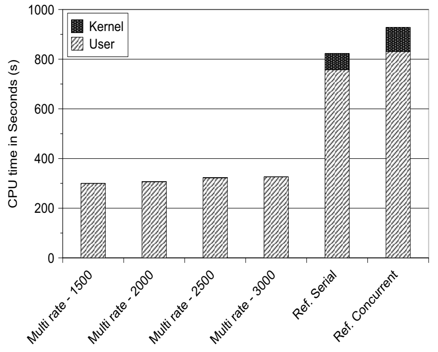


(a) CPU time

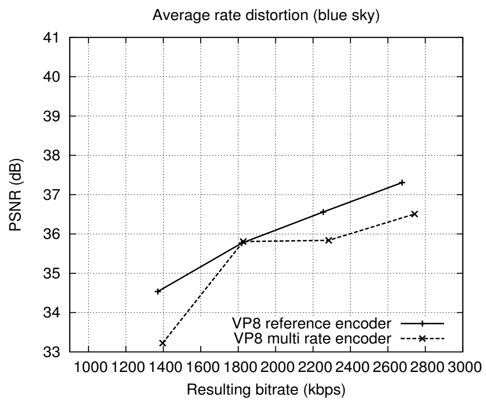


(b) Rate-distortion curve

Figure 4. HD streaming scenario (“pedestrian”)



(a) CPU time



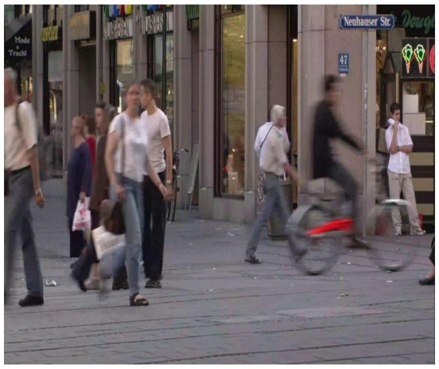
(b) Rate-distortion curve

Figure 5. HD streaming scenario (“blue sky”)

content types, our multi-rate encoder reduce the total resource consumption.

B. 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 3(b). We can see that reference encoder produces about 1 dB higher peak signal-to-noise ratio (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


(a) Reference encoder

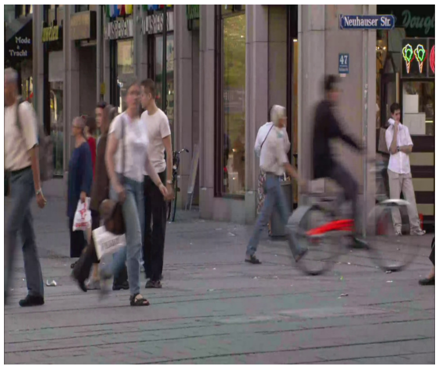
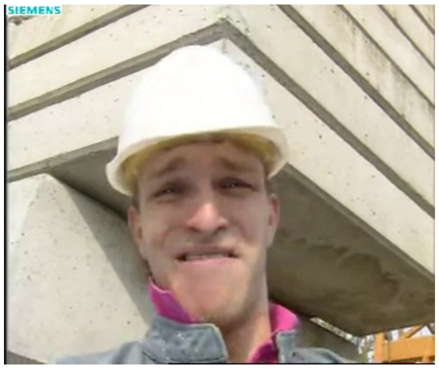

(b) Multi-rate encoder

Figure 6. Quality difference for the “worst-case” scenario in figure 4(b) of 1.32 dB PSNR of 1500 kbps


(a) Reference encoder

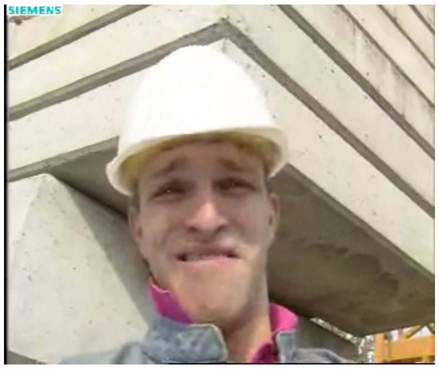

(b) Multi-rate encoder

Figure 7. Quality difference for the “worst-case” scenario in figure 3(b) of 0.99 dB PSNR of 250 kbps

quality.

The degradation in video quality is due to the instances' reuse of analysis computations. As described in section IV, the analysis part of the encoder pipeline is only carried out by the encoder instance targeting the *prediction bitrate*, and is hence only subject to the constraints of this instance. The mode decisions and motion vectors for the other instances will differ from how they would have been had they been chosen by separate analysis, leading to a degradation in video quality.

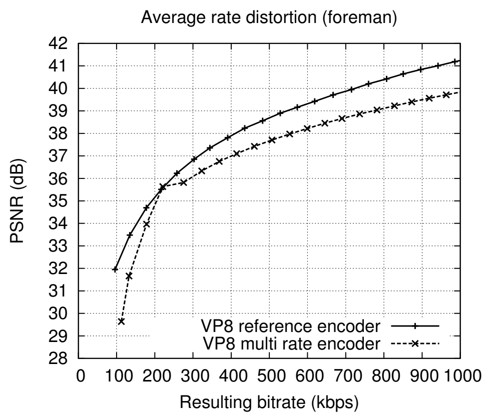
Similarly, when considering the distortion of the *HD* sequences, we have plotted rate-distortion curves in figure 4(b) and 5(b) for *pedestrian* and *blue sky*, respectively. The reference encoder produces output that has 1.0 to 1.5 dB higher PSNR than the multi-rate encoder and distortion achieved for the two *HD* clips are very similar.

The suitability of PSNR for video quality assessment is frequently discussed, and it is often unclear what the difference mean in terms of the logarithmic scale. From the plot in figure 4(b), we can see that the PSNR of the output from the reference encoder is up to 1.32 dB better than the multi-stream encoder outputs for the *pedestrian* 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(b) can be seen in figure 6. From this output, we can see that there is little visual difference between the reference encoder output and the multi-stream encoder. We also looked at the average structural similarity (SSIM) index number for the reference encoder and the multi-stream 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 a difference viewing the resulting videos, but it might be different for other types of content). In figure 7, the “worst-case” scenario from figure 3(b) can be seen. In this sequence, the PSNR for the reference encoder is of 0.99 dB better than the multi-rate encoder. The SSIM index numbers for this scenario are 0.873 for the reference encoder and 0.856 for the multi-stream encoder.

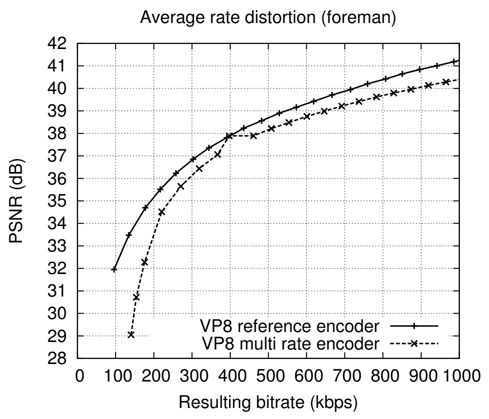
C. Choosing the prediction bitrate

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 8. We can see that the resulting bitrate is lower for the multi stream encoder than the reference encoder, except for when the prediction bitrate exactly matches the target bitrate, resulting in a small spike in the plot.

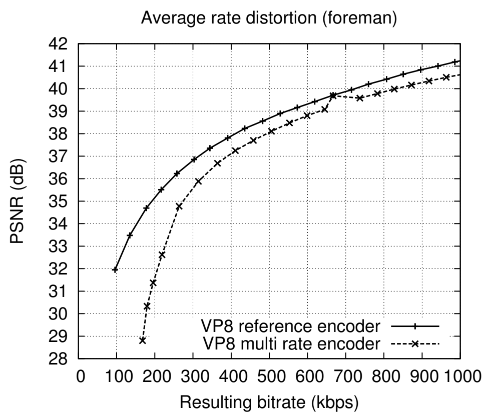
The lowest *prediction bitrate* (250 kbps) incurs the largest distortion difference of 2 dB for the 1000 kbps *resulting bi-*



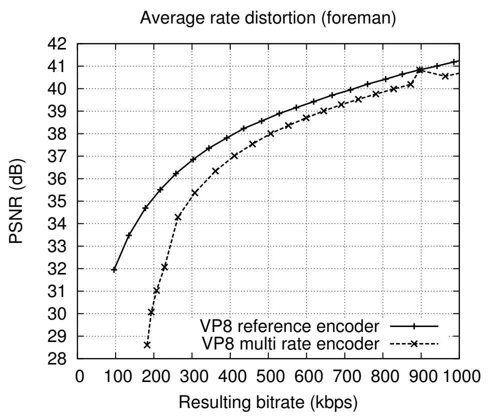
(a) prediction bitrate: 250



(b) prediction bitrate: 450



(c) prediction bitrate: 750



(d) prediction bitrate: 1000

Figure 8. Rate-distortion curve for CIF test sequence ”foreman” with different prediction bitrate (in kbps)

trate. 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 stream 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.

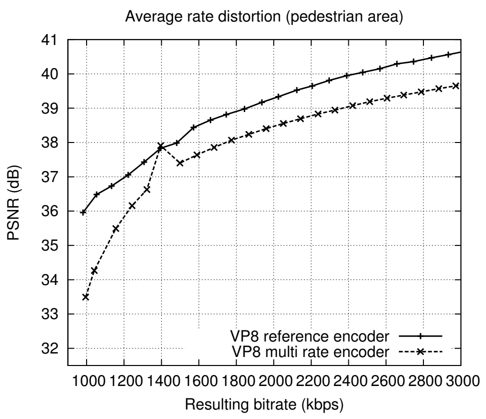
Similar results can be observed when evaluating the *pedestrian* sequence, shown in figure 9. Lower *prediction bitrates* incur less distortion difference than higher *prediction bitrates* compared to the *target bitrate*. The distortion difference is further reduced by choosing a bitrate closer to the average of the extremes.

We have shown that choosing the correct *prediction bitrate* when doing multi-rate encoding has a profound effect on the quality of the output videos. Although CPU time was also affected as shown in figure 3(a), 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 9(d)), but for quality ranges typically used in segmented streaming as shown in our test sequences, the results prove that multi-rate encoding are useful.

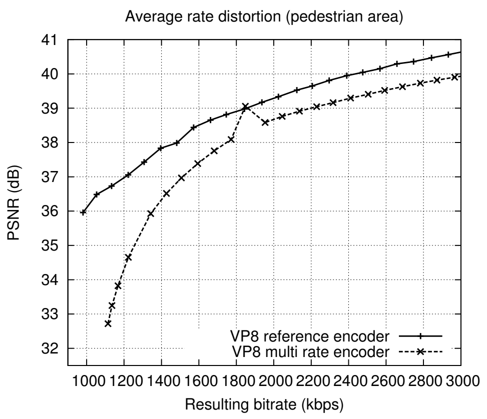
VI. Discussion and open issues

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

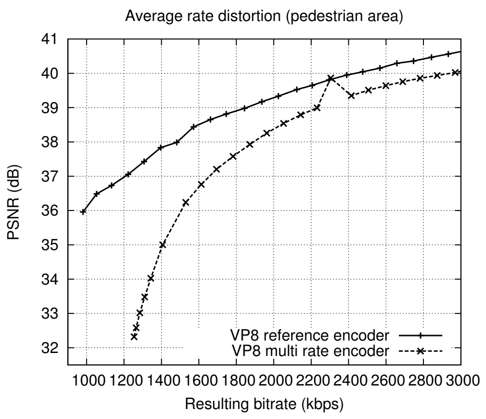
To prove the idea, we have implemented a prototype trying to reuse the most expensive operations based on profiling of the encoding pipeline. In particular, our multi-rate encoder reuses the analysis part consisting of macroblock mode decision and intra/inter prediction. The experimental results indicate 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.



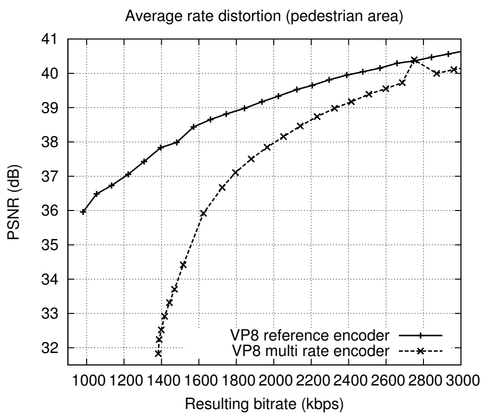
(a) prediction bitrate: 1500



(b) prediction bitrate: 2000



(c) prediction bitrate: 2500



(d) prediction bitrate: 3000

Figure 9. Rate-distortion curve for HD test sequence ”pedestrian” with different prediction bitrate (in kbps)

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 [11]. This would however lead to increased complexity in the encoder. Section V-C 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.

VII. Conclusion

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 sub-sequent 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.

Since the VP8 codec has a very similar encoding pipeline as H.264, our approach should be suitable for H.264 (and similar codecs) as well, but showing the same experimental results is left as further work. Additionally, our aim has been to point at an operation that potentially can be optimized, and we suggested one possible solution. However, as indicated in section VI, there are several open issues where potentially other steps that can be improved with respect to the resource consumption – all promising research topics to pursue.

ACKNOWLEDGEMENTS

This work has been performed in the context of the *iAD* centre for Research-based Innovation (project number 174867) funded by the Norwegian Research Council.

REFERENCES

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

[2] Move Networks, "Internet television: Challenges and opportunities," Move Networks, Inc., Tech. Rep., November 2008.

[3] S. L. Flosi, "comScore releases April 2010 U.S. online video rankings," comScore, Inc., Press Release, Jun. 2010.

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

[5] R. Pantos, J. Batson, D. Biderman, B. May, and A. Tseng, "HTTP live streaming," http://tools.ietf.org/html/draft-pantos-http-live-streaming-04, 2010. [Online]. Available: http://tools.ietf.org/html/draft-pantos-http-live-streaming-04

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

[7] T. Stockhammer, "Dynamic adaptive streaming over HTTP - standards and design principles," in *Proc. of ACM MMSys*, 2011, pp. 133–144.

[8] P. Seeling, F. H. P. Fitzek, G. Ertli, A. Pulipaka, and M. Reisslein, "Video network traffic and quality comparison of vp8 and h.264 svc," in *Proc. of MoViD*, 2010, pp. 33–38.

[9] J. 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.

[10] C.-C. Kuo and N. Jayant, "An adaptive non-linear motion vector resampling algorithm for down-scaling video transcoding," in *Proc. of ICME*, Jul. 2003, pp. 229–232.

[11] H. Zhou, J. Zhou, and X. Xia, "The motion vector reuse algorithm to improve dual-stream video encoder," in *Proc. of ICSP*, Oct. 2008, pp. 2359–2362.

[12] 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*, Mar. 1999, pp. 2359–2362.

[13] J. Youn, M.-T. Sun, and C.-W. Lin, "Motion vector refinement for high-performance transcoding," *Multimedia, IEEE Transactions on*, vol. 1, no. 1, pp. 30 –40, mar 1999.

[14] Z. Zhou and M.-T. Sun, "Fast macroblock inter mode decision and motion estimation for h.264/mpeg-4 avc," in *Image Processing, 2004. ICIP '04. 2004 International Conference on*, vol. 2, oct. 2004, pp. 789 – 792 Vol.2.

[15] J. Bankoski, P. Wilkins, and Y. Xu, "Vp8 data format and decoding guide," IETF Internet-Draft, March 2011.

[16] H. Espeland, "Investigation of parallel programming on heterogeneous multiprocessors," Master's thesis, Aug. 2008.

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