Energy-Aware H.264 Decoding

Abstract. The increasing use of more resource-intensive multimedia applications in communication has made it essential to ensure better utilization of available computing resources. At the same time, energy consumption has turned out to be one of the most important resource constraints in modern systems. Digital videos are an important part of multimedia, and a large number of video standards are currently available. In this paper, we work on the most commonly used video standard named H.264. We propose a method to reduce the energy consumption involved in video decoding by selective degradation of video quality. Experiments on the LIVE video database show that our proposed method is quite effective in practice.

1 Introduction

The rise of both environmental concerns as well as limits placed on performance by energy dissipation has given rise to energy-aware computing. Moreover, embedded systems having limited sources of energy are increasingly being used for more and more complex tasks. Designers are, therefore, increasingly looking at ways of reducing the energy footprint of their applications [1].

In order to make them more user-friendly, computing systems are gradually turning towards communication through multimedia such as graphics, audio and videos. While this increasing use of multimedia has made computer systems more accessible to people, it has also made it essential to ensure that such systems have enough resources to deal with these demands. Since energy is one of the more important resources, building energy-efficient systems has turned out to be an important goal of designers.

With improvements in processor speeds according to Moore's Law [2], energy consumption by processors have gradually increased. In embedded systems, the amount of energy consumption is limited by the capacity of the power source. In desktop and server systems, energy consumption is limited by the amount of heat dissipation possible within the die area [3]. Thus, the constraint on energy consumption has placed a limit on the performance of embedded, desktop as well as server systems. An increasingly popular theme of research today is to limit the amount of energy consumption at the software level. This has been done at compiler-level, operating system level as well as at the application level. At the application level, energy-aware design of various commonly used software such as virtual machines, wireless sensors and video decoders are currently topics of active research [1].

A video decoder is an essential component in the playback of video files. Video decoders are, thus, commonly used on mobile devices having limited sources of energy. Thus, reducing energy consumption of video decoders is likely to enable users of mobile devices to decode more high resolution videos.

Among the many available video standards currently available, H.264 is the most commonly used standard. This is because H.264 offers very good video compression with little loss of quality[4]. For this reason, H.264 is widely used in both desktop and mobile platforms. It is also very commonly used for video streaming.

In this paper, we present a method to perform energy-aware decoding of H.264 videos. For simplicity, we have concentrated here only on intra-coded frames, i.e. frames in which information is derived solely from decoded pixels present in the same frame. We show that significant gains in energy consumption can be made by lowering quality of the video during the process of decoding. We perform experiments to show that significant energy gains can be made with little loss of video quality.

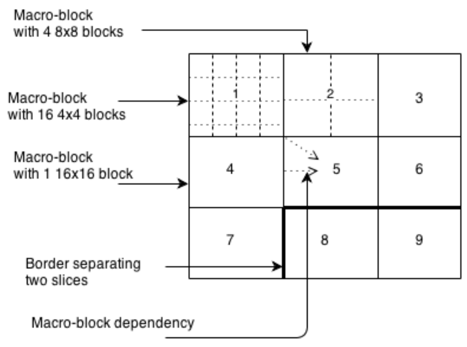
The rest of this paper is organized as follows. Section 2 briefly discusses the H.264 standard, existing methods of measuring energy consumption of a modern computing system and ways of measuring quality of videos. Section 3 describes our technique of reducing the energy footprint during the process of video decoding. The experiments performed and observations recorded are then discussed in Section 4. Section 5 lists some other work related to energy-aware video decoding, and the ways in which they differ from our contribution. Section 6 concludes this paper.

2 Background

2.1 H.264 standard

A H.264 video [5, 6] consists of a sequence of frames. A frame is an array of luma samples (related to luminance) and two corresponding arrays of chroma samples (related to red and blue chrominance). Each frame is further divided into spatial units called slices. A slice consists of blocks of 16 x 16 pixels, known as macro-blocks (MB). A macro-block contains type information describing the choice of methods used to code the macro-block, prediction information such as intra prediction mode, and coded residual data. Within a macro-block, luma samples may be coded as one of the three types of block sizes, namely 4x4, 8x8 or 16x16 pixels. Chroma samples are commonly coded as blocks of 8x8 pixels.

Reconstruction is an important step in the decoding of an H.264 video frame. Reconstruction of a decoded macro-block involves obtaining the data from neighbouring macro-blocks based on which motion prediction had been made by the encoder. This cannot be done independently, but only after fetching data of neighbouring macro-blocks. In an intra-coded video frame, all dependencies are in the same frame of video. The H.264 standard specifies four neighbours for a macro-block in an intra-frame, namely, left, top-left, top and top-right. For example, in Figure 2.1, the macroblock labelled 5 has the macroblocks labelled 4, 1, 2 and 3 as its left, top-left, top and top-right neighbours respectively. In addition, a macroblock includes a variable amount of residual information that cannot be inferred from previous macroblocks. This residual information is converted into frequency domain using a modified form of Discrete Cosine Transform (DCT), and then stored within the encoded bitstream.



Macro-block with 4 8x8 blocks
Macro-block with 16 4x4 blocks
Macro-block with 1 16x16 block
Border separating two slices
Macro-block dependency
1
2
3
4
5
6
7
8
9

Fig. 1. A 3x3 H.264 frame

2.2 Measurement of Energy Consumption

In order to enable energy-aware computation, it is essential to monitor the amount of energy or power consumed while running a program. This can be done by reading the Model-Specific Registers (MSRs) which are available on almost all modern processors. Intel, in particular, allows the application user to read the amount of energy or power consumed by a code fragment by reading its MSR registers through its RAPL (Running Average Power Limit) driver.

Using RAPL to obtain energy consumption has two major disadvantages. First, it cannot measure energy consumption at very low granularity, i.e. the energy consumed is given in multiples of 125 mJ. Second, it does not quantify the energy consumption involved in different stages of instruction execution. Thus, it is not possible to suggest improvements in software design by simply using RAPL output without first studying the reasons which lead to high energy consumption [7].

In order to understand better the relation between program instructions and energy consumption, power models have been developed that can estimate the energy consumption involved depending on the number of instructions fetches, number of memory accesses at different levels of cache accesses and other factors. It has been shown that energy consumption by an architectural component is proportional to its activity ratio. The activity of an architectural component here refers to the number of operations performed by it per unit time. These power models allow us to recognize the components that consume more power, and thus point us towards techniques to reduce energy consumption by suggesting methods to reduce the number of operations performed by some component. In this paper, we have used one such power model to develop ways to reduce energy consumption of video decoders [8].

2.3 Measures of Video Quality

Measuring quality of video decoding is an active area of research. Our work focuses on measuring the quality of video obtained after the entire process of decoding is completed. This essentially implies that the decoded video obtained after degradation can be compared with another reference video decoded without degradation. This process of measuring video performance is known as Full-Reference video quality measurement. Wang et al. [9] provides an excellent survey of traditional methods of video quality along with their drawbacks.

The traditionally most common Full-Reference video quality measure is Peak Signal-to-Noise Ratio (PSNR). Peak Signal-to-Noise Ratio is obtained by:

$$PSNR = 10log_{10}\frac{MAX_I^2}{MSE}$$

where MSE or Mean Square Error is the square of the sum of differences in intensity of each color element, and MAX_I is the maximum possible intensity, which is obtained using the number of bits used to represent a particular color component. There are two different types of PSNR that can be used:

1. Average PSNR: In this case, the PSNR of each image is obtained individually, and their arithmetic mean is then calculated.
2. Global PSNR: In this case, the MSE of the entire video is first obtained, and then its ratio with the maximum possible intensity is obtained. The logarithmic operation is applied on this ratio. This method basically concatenates all the frames of the video to form a single image of very large dimension, and then calculates its PSNR [10].

PSNR, though still widely used as a metric to measure video qualities, merely compares the differences between the reference video and the output video. This metric does not always agree with the human perception of video quality. Thus, researchers continued to look for better metrics of video quality.

A metric that is widely used to measure quality of images is the Structural Similarity Index Metric (SSIM). It uses the formula

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

where μ_x is the average value of pixels along the width of a frame, μ_y is the average value of pixels along the height of a frame, σ_x^2 is the variance of pixels along its width, σ_y^2 is the variance of pixels along its height, σ_{xy} is the covariance of x and y, c_1 and c_2 are constants that are typically used to stabilize the division with a weak denominator. SSIM is a linear metric that gives a number between 1 and -1, with 1 being returned when the reference and the test frame are identical to each other.

PSNR and SSIM, taken together, are the most common measures of video quality used nowadays by far. However, it has been observed that these two metrics do not take into account the interaction between different frames, or any information present in a video about the motion of objects [11].

One metric that takes into account the motion information and has been shown to provide a much better measure of video quality is the MOtion-based Video Integrity Evaluation (MOVIE) index, developed at the Laboratory for Image and Video Engineering at University of Texas, Austin. The MOVIE index consists of two distinct components — spatial and temporal MOVIE. The spatial index uses comparison techniques similar to other video quality assessment algorithms such as PSNR, but with much more detailed information. The temporal index captures any temporal distortions introduced into the video. Experiments have shown that the MOVIE index has a much higher correlation with human perception of video quality [12].

3 Methodology

In order to demonstrate the efficacy of our method, we need to first determine the amount of power or energy consumed by the decoder process. This requires calibrating the power model that we have selected with our processor. The next step involves profiling of the video decoder to determine the steps in the decode process that take up significant chunks of the total power consumption. Finally, we modify the video decoder in order to reduce the amount of energy consumed. These three steps are discussed in detail in the following subsections.

3.1 Calibration of Power Model

We have used a power model where the total power consumption can be decomposed into that of individual architectural components [8]. The motivation behind using such a power model is two-fold:

1. It allows us to determine the architectural components that consume most power, which in turn helps us identify the steps where reducing power consumption would be most beneficial, and
2. it allows us to study the relation between macroblock properties and the power consumption involved in decoding them. In general, RAPL does not give the required level of precision required to obtain the power consumption involved in decoding each macroblock.

In this power model, the power consumption of a component is assumed to be directly proportional to the activity ratio of each component. This power model considers a system to be made up of the following components:

1. Processor Frontend (FE),

2. Integer unit (INT),
3. Floating-point unit (FPU)
4. Single Instruction Multiple Data unit (SIMD)
5. Branch Prediction Unit (BPU),
6. L1 cache
7. L2 cache,
8. Memory and bus.

To obtain the weights associated with each architectural component, we have profiled the modified microbenchmark suite used in [8][1]. The microbenchmarks have been designed in such a way that it is easily possible to separate the power consumption of each individual component. This has helped us minimize the chances of regression error. The weights thus obtained (Table 1) have been validated by profiling the SPEC2006 benchmarks. The decoder that we worked with, does not use any floating-point and SIMD instructions, hence, we neglected the power consumption of these units in the above computation.

Profiling of the individual components has been done using Performance Application Programming Interface (PAPI) [7], which uses hardware counters to obtain various performance metrics. The total power consumed by the system has been measured using Intel Running Average Power Limit (RAPL) drivers. To ensure that power from other programs do not interfere with the total power, all other applications have been turned off as far as possible. Each microbenchmark has been run for 30 seconds each. This time limit has been imposed through asynchronous signals to ensure minimum interference in processor performance.

Table 1. Power Model for an Intel Core 2 Duo processor

Component	Power(mW)
Front-end, P_{FE},	789
Integer Unit, P_{INT}	261
Floating Point Unit, P_{FP}	502
Branch Processing Unit, P_{BPU}	1908
L1 cache, P_{L1}	856
L2 cache, P_{L2}	24437
Front-side Bus, P_{FSB}	8852
Static power (constant) P_{STATIC}	8701

3.2 Profiling of video decoder

We have used the video decoder of Joint Model reference software [13], which has been developed by the video standardization team for better understandability

[1] freely available at `http://rbertran.site.ac.upc.edu/tools/micro.tar.bz2`

of the standard. We have profiled separately the process of CABAC (entropy decoding), motion compensation and application of loop filter to conclude that the process of motion compensation consumes the largest amount of power. We have, therefore, concentrated on minimizing the power consumption during motion compensation.

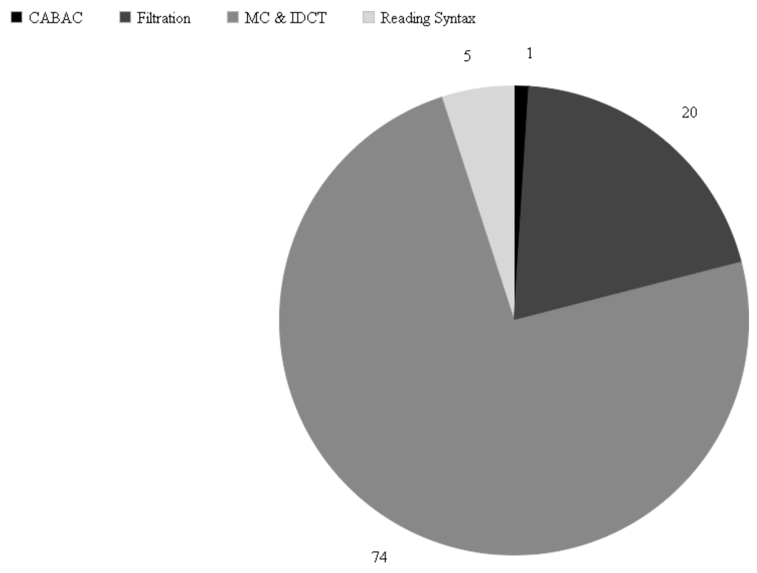


Fig. 2. Power Consumption at Different Stages of Decoding

The process of motion compensation essentially involves obtaining the pixel values of the predecessor macroblocks and summing them up with the residual data present in the encoded data. This fetching of residual data incurs significant power expense, since data has to be fetched from main memory, and then calculated for each pixel. By selectively ignoring some of the residual data, it is possible to significantly lower the amount of power consumption involved while minimizing the loss of video quality. This neglecting of residual data is done at the macroblock-level, where macroblocks satisfying some properties are selected as victims and their residual data is left unused.

3.3 Modification of Video Decoder

Using power analysis, we observed that a major amount of power consumption occurs due to the fetching of residual data. In order to reduce the amount of power consumption involved, we degrade the quality of some macroblocks by ignoring a portion of the residual data. The pixel data of such macroblocks is obtained solely through motion compensation. However, arbitrarily selecting macroblocks for such degradation would adversely affect the quality of video and render it unfit for use. We therefore, propose a heuristic that intelligently degrades video quality in order to minimize its effect on video quality.

Our strategy considers two major factors involved in selecting which macroblocks to ignore –the amount of power that decoding a macroblock requires, and the effect of degrading the macroblock on other neighbouring macroblocks. The quality of pixels in neighbouring macroblocks could degrade if we arbitrarily choose macroblocks for degradation, due to the presence of drifts. Since macroblock dependencies can exist in the form of long chains, an error introduced at some point in the macroblock could be transmitted to macroblocks much farther from the origin. To avoid such errors, we preferably choose macroblocks having fewer dependencies.

Algorithm 1 SelectVictimBlock

```
1: $S[MB] \leftarrow ReadMBSyntax$
2: $D[MB] \leftarrow GetDependencies(S[MB])$
3: $A[MB] \leftarrow meanQuantization(S[MB])$
4: if $mode = 1$ then
5:     $R[MB] \leftarrow ResidualData$
6:     $DecodeMB(S, D, R)$
7: else if $mode = 2$ then
8:     if $Quantization(S[MB]) < A[MB]$ then
9:         $O[MB] \leftarrow DependencyCount(S[MB])$
10:        if $O[MB]! = 0$ or $NeighboursDegraded(S[MB]) = FALSE$ then
11:            $R[MB] \leftarrow ResidualData$
12:        end if
13:    end if
14: else if $mode = 3$ then
15:    if $Quantization(S[MB]) < A[MB]$ then
16:        if $O[MB] > 1$ or $NeighboursDegraded(S[MB]) = FALSE$ then
17:            $R[MB] \leftarrow ResidualData$
18:        end if
19:    end if
20: end if
21: $DecodeMB(S, D, R)$
```

We have investigated degradation strategies for different classes of videos and come up with three different options (or modes, as used in the algorithm above) in which our algorithm works. These options are as below:

– Mode 1: No degradation,
– Mode 2: Degradation less than α, and
– Mode 3: Degradation more than α,

where α is a context-dependent parameter to be provided by the user. The actual value of α may vary for different classes of videos. In our experiments, we have chosen α as 12%.

Algorithm 1 shows our overall strategy. The algorithm selects macroblocks whose residual data will be ignored. The macroblocks are selected based on the mode in which the user wants the decoder to run, the number of dependencies of the macroblock, and also depending on whether any adjacent macroblocks have been degraded. In the algorithm, variable S refers to the syntax elements of a macroblock, D refers to the list of dependencies, A is a temporary variable that stores the mean quantization value of macroblocks and R stores the list of blocks within a macroblock in which residual data is present. The variable O refers to the list of outgoing edges from a block. The function *ReadMBSyntax* parses the syntax elements of the bitstream and populates the data structure of the macroblock. The function *GetDependencies* then uses the data dependency information stored in the data structure to obtain the data dependency information required for motion compensation. The function *meanQuantization* returns the mean of the quantization step sizes among all macroblocks within the frame.

As shown in Algorithm 1, while running in no degradation mode, the video is decoded without any modifications. For all other modes where degradation is necessary, macroblocks with lower quantization step than average are first selected. Within these macroblocks, those blocks on which no other blocks outside depend, are selected for degradation in mode 2. The residual data of these blocks are ignored, and the pixel data for these are obtained solely using motion compensation. For mode 3, blocks which have one or fewer dependencies are selected for degradation. However, if its neighbouring block has already been degraded, then it is left untouched in order to ensure that the changes are not reflected over too large an area of the frame.

4 Experiment and Results

One major problem faced by researchers while working on video quality is the lack of standard benchmark videos present that is widely accepted in academia or industry. In order to mitigate this problem, the LIVE database for videos was developed [14–16]. The LIVE Video Quality Database contains ten uncompressed high-quality videos. These videos are – bs (blue sky), mc (mobile and calendar), pa (pedestrian area), rb (river bed), rh (rush hour), sf (sunflower), sh (shield), st (station), tr (tractor) and pr (park run). The videos are given in the form of planar 4:2:0 yuv files at a resolution of 768 × 432. They have been extensively used for subjective and objective video quality assessment. Seven of the videos have a frame rate of 25 frames per second, while the rest have a frame rate of 50 frames per second. The videos that are used by the Video Quality Experts Group (VQEG) for standardization of measures of video quality which have been made available are also a part of this database. We have used these videos for this work.

We have used the Joint Model reference software [13] version 18.3 to perform our experiments. The decoder is executed to decode the encoded video, and the energy consumption as well as PSNR of the decoded video is then recorded. Now, the decoder is modified as discussed in section 3, and then this modified decoder is used to decode the same reference video. The measures of quality of the two versions of the decoded video so obtained are then calculated.

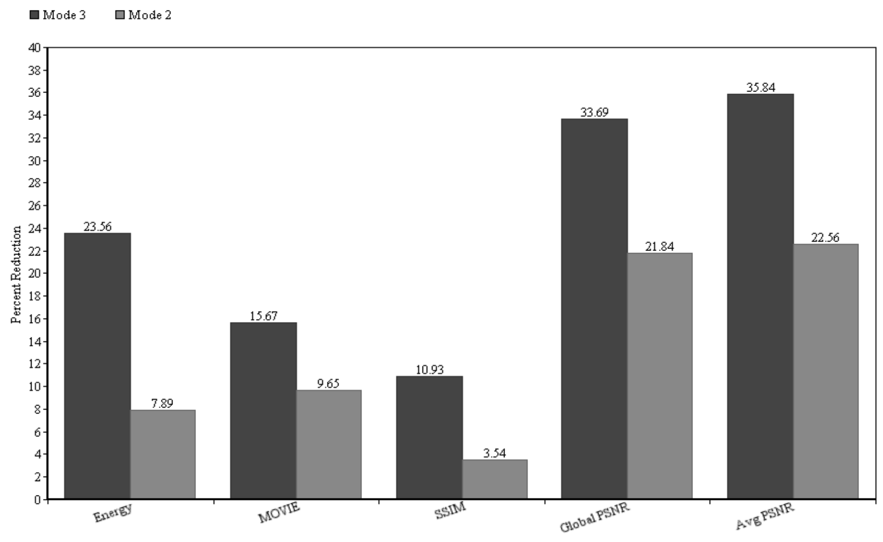


Fig. 3. Reduction in Video Quality and Energy Consumption on LIVE videos

As discussed in Section 2, there is no single widely accepted measure of video quality. We have, therefore, provided results on four different metrics to better understand the amount of video degradation resulting from our strategy. The metrics used are a) average PSNR b) global PSNR c) average SSIM and d) average of the MOVIE indices. Both average and global PSNR as well as SSIM of the videos have been calculated using the open-source software *libyuv*[10]. MOVIE index has been calculated using the software provided by the original proposers of the method. The activation ratio of each unit has been calculated using the open-source Performance Application Programming Interface (PAPI) version 5.2.0 [17]. This has been multiplied by the weights given in Table 1 and then the static power is added. The value obtained is multiplied by the time required to perform the decoding in order to determine the total energy consumed.

Fig. 3 shows the arithmetic mean of video quality degradation, measured in average and global PSNR, SSIM and MOVIE, and the corresponding energy gain, taken over each of the ten videos. We note that, as expected, more energy gains are made in mode 3 than in mode 2. Similarly, greater loss of video quality is seen in mode 3 as compared to mode 2. This can be explained by observing that the quality of video decreases with an increase in the number of victim macroblocks. We select more macroblocks in mode 3 than in mode 2, and, therefore, the quality of videos in mode 3 reduces further. We also note that, in each case, the loss in quality as measured using PSNR was greater than the other video metrics. This can be explained by observing that PSNR does not take into account human perception, unlike SSIM and MOVIE index, and so the value of PSNR reduces even if introducing an error does not reduce video quality according to human perception.

5 Related Work

Minimization of energy for software and hardware systems has become an important area of active research. In [1], Ahmad and Ranka provide a good survey of methods, both at hardware and software level, of reducing energy computation.

Since H.264 is widely used in resource-constrained systems, improving its performance has long been a source of active research. In [18], Chang et al. work on reducing power consumption of an H.264 encoder. In [19], Nam et al. discuss modifying the video decoder so that the amount of time needed to decode a video is reduced. Park et al. suggest adding an additional re-quantization step to the decoding process, and combining it with motion compensation step to reduce energy consumption in [20]. Huang et al. discuss in [21] methods for reducing power consumption of a decoder using Dynamic voltage and frequency scaling (DVFS). Xu and Choy proposed techniques to reduce power consumption of a hardware H.264 decoder in [22].

6 Conclusion

In this paper, we present a method to reduce the power consumption involved while decoding a video encoded in H.264. Our experiments showed significant gains with relatively small amounts of video degradation. This method could be used by video decoders in embedded and streaming systems to prolong the lives of their power sources, and lead to lower energy consumption in desktop systems.

References

1. I. Ahmad and S. Ranka, “Handbook of Energy-Aware and Green Computing-Two Volume Set,” 2012.
2. G. E. Moore *et al.*, “Cramming more components onto integrated circuits,” 1965.
3. J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach.* Elsevier, 2012.
4. H. Schwarz and T. Wiegand, “The emerging JVT/H.264 video coding standard,” *Proc. of IBC 2002*, 2002.
5. I. E. Richardson, *The H. 264 advanced video compression standard.* Wiley, 2011.

6. T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.
7. V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore, "Measuring energy and power with PAPI," in *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pp. 262–268, IEEE, 2012.
8. R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*, pp. 147–158, ACM, 2010.
9. Z. Wang and A. C. Bovik, "Mean squared error: love it or leave it? a new look at signal fidelity measures," *Signal Processing Magazine, IEEE*, vol. 26, no. 1, pp. 98–117, 2009.
10. "libyuv - yuv scaling and conversion functionality." http://code.google.com/p/libyuv/, 2013. Accessed: August 18, 2013.
11. 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*, vol. 13, no. 4, pp. 600–612, 2004.
12. K. Seshadrinathan and A. C. Bovik, "Motion tuned spatio-temporal quality assessment of natural videos," *Image Processing, IEEE Transactions on*, vol. 19, no. 2, pp. 335–350, 2010.
13. K. Shring, "H.264-avc joint model software." ttp://ipome.hhi.de/suehring/tml/, 2013. Accessed: July 8, 2013.
14. K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack, "Study of subjective and objective quality assessment of video," *Image Processing, IEEE transactions on*, vol. 19, no. 6, pp. 1427–1441, 2010.
15. K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack, "A subjective study to evaluate video quality assessment algorithms," in *IS&T/SPIE Electronic Imaging*, pp. 75270H–75270H, International Society for Optics and Photonics, 2010.
16. "Live video database." http://live.ece.utexas.edu/research/quality/live_video.html, 2013. Accessed: August 18, 2013.
17. "Papi." http://icl.cs.utk.edu/papi/software/index.html, 2013. Accessed: September 4, 2013.
18. H.-C. Chang, J.-W. Chen, B.-T. Wu, C.-L. Su, J.-S. Wang, and J.-I. Guo, "A dynamic quality-adjustable H. 264 video encoder for power-aware video applications," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 19, no. 12, pp. 1739–1754, 2009.
19. H.-M. Nam, J.-Y. Jeong, K.-Y. Byun, J.-O. Kim, and S.-J. Ko, "A complexity scalable h. 264 decoder with downsizing capability for mobile devices," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 2, pp. 1025–1033, 2010.
20. S. Park, Y. Lee, J. Lee, and H. Shin, "Quality-adaptive requantization for low-energy MPEG-4 video decoding in mobile devices," *Consumer Electronics, IEEE Transactions on*, vol. 51, no. 3, pp. 999–1005, 2005.
21. Y. Huang, S. Chakraborty, and Y. Wang, "Using offline bitstream analysis for power-aware video decoding in portable devices," in *Proceedings of the 13th annual ACM international conference on Multimedia*, pp. 299–302, ACM, 2005.
22. K. Xu and C.-S. Choy, "Low-power bitstream-residual decoder for H. 264/AVC baseline profile decoding," *EURASIP Journal on Embedded Systems*, vol. 2009, p. 9, 2009.