

Video Stream Annotations for Energy Trade-offs in Multimedia Applications*

Radu Cornea Alex Nicolau Nikil Dutt
Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{radu,nicolau,dutt}@ics.uci.edu

Abstract

Recent applications for distributed mobile devices, including multimedia video/audio streaming, typically process streams of incoming data in a regular, predictable way. The behavior of these applications during runtime can be accurately predicted most of the time by analyzing the data to be processed and annotating the stream with the information collected. We introduce an annotation-based approach to power-quality trade-offs and demonstrate its application on CPU frequency scaling during video decoding, for an improved user experience on portable devices. Our experiments show that up to 50% of the power consumed by the CPU during video decoding can be saved with this approach.

1. Introduction

Recent technological advances in processor and wireless technology have caused a shift in the computing industry towards mobile devices like handhelds, PDAs and laptops. At the same time, we find that increasingly these devices are being used in distributed multimedia streaming applications, common examples being movie streaming on demand, video conferencing and on-line gaming. But even with these advances in technology, battery life still remains a major limitation of portable devices. The main power consuming components of a handheld device are the CPU, display and network interface. Running multimedia applications further aggravates the situation, as these programs are known to be both CPU-intensive and to require higher network bandwidth.

Various research has been done over recent years toward minimizing power consumption, but there is a limited power gain that can be achieved statically. Variations in the input data stream present us with new ways to optimize dynamically, based on content.

Multimedia workloads are characterized by quasi regular patterns in their execution, mainly because media encoding/decoding is composed of processing filters, applied in a specific order. This regular behavior is confirmed by recent research [11]. The only changes are introduced by variations in the input data [9] and the algorithm itself. Knowledge of data characteristics can be exploited at all levels in a multimedia streaming application, and is especially important for portable devices where battery life and thus mobility are of utmost importance.

In this paper we propose a new approach for runtime power vs quality trade-offs in distributed multimedia applications based on data analysis and annotation. In this case annotations prove beneficial because they can be done “off-line” (statically) while saving the mobile device from the burden of analyzing the data at runtime.

The paper is organized as follows: we start by describing data annotations and presenting some related work. We then introduce our approach for using annotations in a CPU frequency scaling scheme for video decoding. Next we focus on the technique for data-aware dynamic frequency scaling, followed by the results of our experiments. The last section concludes the paper and discusses future work. For more information please refer to [6].

2. Data annotation

The idea of annotations is not new. Annotations have previously been applied to other domains. In compilers, annotations are sometimes used for retaining as much from the original semantics as possible, in an automated way. At the same time, a programmer can annotate the source code to pass information or hints to the compiler [8]. For example, register assignment for variable in C falls in this category. Other examples include the use of ‘pragma’ directives (C, C++, Ada).

Similarly, we see the process of annotating the data stream as either automated (performed statically, by an analysis step) or under user supervision, where the user can, for example, specify which parts or objects of the video stream

*This work was partially supported by NSF award ACI-0204028.

are more important and should be best preserved in a power-quality trade-off.

We define data annotation as the process of analyzing a stream of data and supplementing it with a summary of the information collected; this information will later be used for run-time data-aware optimizations. Annotations typically capture patterns or trends in the data that are difficult/impossible or too time-consuming to gather at run-time on the handheld device and that can be exploited later for either power or performance benefits.

The annotations can be performed either statically (for example off-line annotation/profiling in case of on-demand media serving, where the information is preprocessed and stored at the media servers) or dynamically (in case of live streams, the annotation can be done on-the-fly by an intermediary proxy node).

Annotations can be used at either the client side, for an increased user experience, or at a proxy node, which may perform additional on-the-fly operations on the data stream to adapt it to the capabilities of the client (transcoding, etc.). These annotations may prove useful in estimating the required bandwidth for communication, estimating computation or applying more aggressive QoS trade-offs based on image content.

Multimedia applications are typically studied at different abstraction layers: application, middleware/network, OS, hardware. Each of these layers can use, and benefit from, additional information on the data stream. In this paper we focus our attention at the OS/hardware level. Moreover, we assume that the annotation is performed off-line, by profiling an extensive data set, representative for the application domain. We focus on multimedia streaming in particular, and on annotations that are relevant to this domain.

The advantage of annotating the data in advance (off-line) is two-fold. First, there is no overhead for doing all the work at runtime, by the client device. Second, because the information is available even before decoding the data, more optimizations are possible than would otherwise be possible at runtime. For example, annotations in the MPEG stream allow optimizations at the network level because the information exists in the MPEG header itself. Other optimizations (like frequency/voltage scaling) may be applied early, even before decoding is finished, because the annotated information is immediately available from the data header. Without annotations, the client application would first need to decode the data and analyze it or use some history-based prediction, because only a small window of data can be accessed at one time. This limited knowledge can potentially have serious consequences (severe quality degradations) if prediction proves wrong. It would also place a heavier load on the mobile device, thereby reducing its battery life.

We assume the distributed system model depicted in

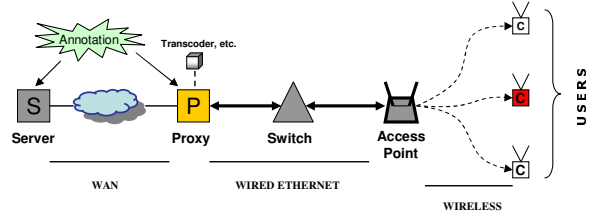


Figure 1. System model

Fig. 1. The system entities include a multimedia server, a proxy node that can perform various optimizations on the stream (e.g. transcoding), the users with low-power wireless devices and other network equipment along the way. The multimedia servers store media content and stream videos to clients upon requests issued by the users on their handheld devices. The communication between the handheld device and the servers can be routed through a proxy server – a high-end machine that has the ability to process the video stream in real-time. The proxy node can also perform in-line profiling/annotation for real-time video streaming (videoconferencing is an example of such an application).

The annotations can be generated and stored in the video stream (headers) at either the server side or the proxy node, therefore saving the client of any additional work. Another option is to send the annotations over a separate control channel.

3. Related work

In this section we are reviewing existing research work that focuses on analyzing the input data stream and deriving various techniques for improving either communication or computation in streaming applications.

The Aspire research group studies various data-shaping algorithms for mobile multimedia communication. They profile and annotate still images for improving transmission over a wireless channel usage (bandwidth, latency). In [7] the image data is compressed according to dynamic conditions and requirements. Content adaptation is classified depending on time (static, dynamic), content (to determine optimal compression) and goals of technique or metrics (constrained bandwidth, display size, response time).

Chandra performs an informed quality-aware transcoding in [3], based on image characteristics. He finds that a change in JPEG quality factor (compression metric controlled by quantization steps) directly corresponds to information quality lost. A prediction for computational overhead is applied, which approximates number of basic computation blocks based on image size, color depth and can predict output size for a particular transcoding.

In [4], the authors analyze the characteristics of images available on web sites (distribution of gif or jpg images, size, colors and quality). They classify images in: bullets, lines, icons, banners, true images based on heuristics and analyze various transcoding techniques for reducing image size (reducing spatial geometry or thumbnailing, reducing the number of unique colors, changing the image format or compression)

Tripathi and Claypool study different ways to reduce bandwidth in network transmission in [12], by either temporal scaling (dropping frames), quality scaling (reducing quality of frames) or spatial scaling (changing the size of frames). The quality degradation is evaluated through a user study.

[5] presents a DVS technique for MPEG decoding to reduce energy consumption while at the same time maintaining the quality of service. The approach is to separate decoding time into a frame-dependent part, which varies with the frame, and a frame independent part, constant regardless of the frame. The independent part is used to compensate the error that can appear during the depending frame part.

Bavier et al in [2] present a set of experiments to measure the CPU processing required for decoding a MPEG frame in software. The algorithm predicts the number of cycles required for a given frame by constructing a linear model between frame type, size and time. The accuracy is within 25% of the actual decode times.

In [10], the authors describe a feedback based controller for video decoding that applies DVS for individual frames. The complexity of frames is estimated using a simple correlation between frame length and decoding time.

In contrast, our technique performs an off-line profiling and annotation, which allows us to more accurately estimate the frame decoding times with a minimal overhead at run-time.

4. Approach

Multimedia applications place heavy demands on the CPU because they typically employ complex encoding/decoding kernels, applied on the video stream. If we observe the load on the CPU more closely, we notice that the amount of processing varies along the stream.

A typical MPEG video stream is composed of sequences of frames (I, P and B), some of which are entirely encoded in the stream (e.g. I - intracoded), some of which are predicted based on either previous (P - predicted) or a combination of previous and subsequent frames (B - bidirectionally predicted) (Fig. 2). Frames are further composed of macroblocks, which are also of type I, P, B: I-frames contain only I-macroblocks, P-frames contain I and P-macroblocks, while B-frames may contain any of the three types, though the B-type predominates.

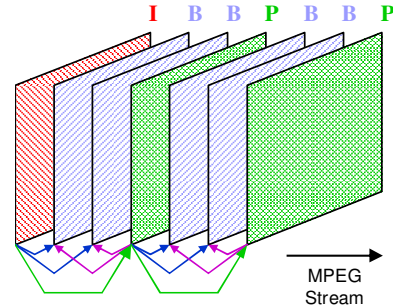


Figure 2. Frames in a MPEG stream

If we plot the distribution of macroblock types in a video clip we observe that the relative percentage of I, P, B macroblocks varies depending on the nature of the video stream and the amount of motion in it (e.g. Figure3). Therefore, for accurate prediction we need to separately profile video clips from different categories (clips with dynamic profile action vs mostly static, news type).

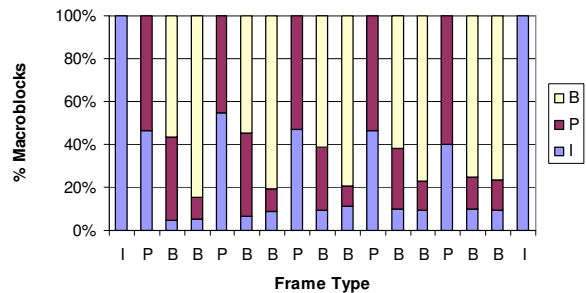


Figure 3. Macroblock Distribution(action clip)

Because I macroblocks are completely intracoded (all needed information is in one frame), they only require an IDCT (Inverse Discrete Cosine Transform) during decoding and are typically much larger than the other macroblocks. On the other hand, P and B macroblocks are predicted from other frames. Therefore they are smaller in size and require both IDCT and motion compensation. Normally, IDCT is more computationally intensive than motion compensation and depends on the size of the block to be processed. As a result, in MPEG, processing the I macroblocks have the highest processing requirements, followed by P and B macroblocks. The distribution of different macroblocks in a frame therefore allows us to roughly estimate the required time for decoding an entire frame (always containing a constant number of macroblocks, depending on its pixel size). As each frame has a different distribution of macroblock types, we store this information as an annotation at the frame level, together with the frame size.

While there are some variations in decoding time within each group type, there is a very marked difference in decoding times between the types of frames, as can be seen in Fig. 4. The figure plots the distribution of frame decoding times during the execution of a typical video clip. The relative variation of the points that we can observe on the plot follows the patterns of activity (motion) in the clips (more activity in the video generates more scattered points).

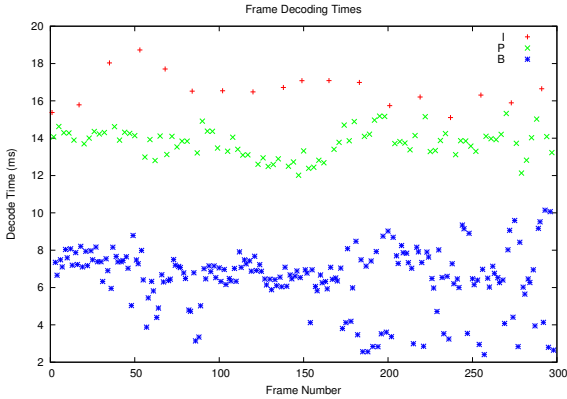


Figure 4. Frame decoding times (action clip)

In our approach, we use annotations to help estimate the computation needed for decoding a frame (decoding time). This estimation is then applied to a DVS power management scheme (slowing down the processor on a frame by frame basis, based on the above estimations).

5. Annotation-based DVS

As we have shown previously, the decoding time of a macroblock depends on its type and is affected by the size of the macroblock. Moreover, if we plot decoding time for a frame as a function of the frame size, we observe a strong correlation between the two in practically all video clips. For example for the same video clip as in Fig. 4 the plot is shown in Fig. 5. This observation suggests that a relative good approximation of the decoding time is feasible.

In order to estimate decoding time based on frame size we apply regression fitting algorithms and find a correlation function between frame sizes and decoding times, which minimizes errors. Next, we devise an estimation heuristic for frame decoding time and use it to control a DVS scheme.

It is well known that the dynamic power consumed by a CMOS circuit can be approximated through the formula $P = C f V_{dd}^2$, where C is the switching capacitance of the circuit, f is the clock frequency and V_{dd} is the supply voltage level.

Since the platform simulated in our experiments (the iPq PDA) does not currently support voltage scaling, we

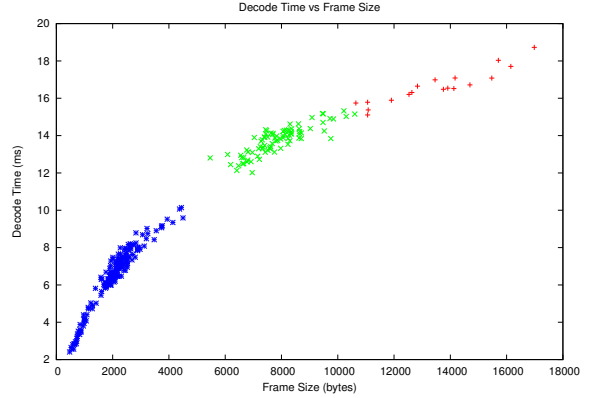


Figure 5. Decode time vs frame size

limit our experiments to frequency scaling with the mention that voltage scaling (which may be present in future devices) and finer grain voltage/frequency levels would even further improve the results. Therefore in the remainder of the paper the term DVS will refer to “frequency scaling” only.

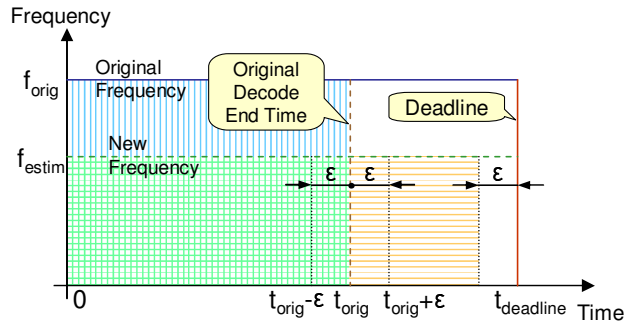


Figure 6. Frequency scaling for a frame

Our technique slows down the processor (saving power) during each frame decode to the lowest frequency that still allows it to finish before the frame deadline.

Fig. 6 shows the frequency and time involved in the decoding of a single MPEG frame. Originally, the processor runs at full frequency (f_{orig}) and the frame is decoded in t_{orig} time. The deadline for decoding a frame is given by $t_{deadline}$ and depends on the frame rate at which the video is encoded.

We use an estimation function F_{estim} (generated in the analysis stage) and compute the estimated decode time from the frame size s (stored in our annotations, together with the distribution of different macroblocks types in the frame).

$$t_{estim} = F_{estim}(s)$$

Estimation errors cause the computed decode time to vary slightly from the actual decode time by a ϵ error value. The smaller ϵ , the better our estimation is:

$$t_{estim} - \epsilon < t_{orig} < t_{estim} + \epsilon$$

We compute the new (lower) frequency to run the processor from the equation:

$$(t_{estim} + \epsilon)f_{orig} = t_{deadline}f_{estim}$$

In case of discrete CPU frequencies, we choose the closest frequency just under the computed value. Based on the new frequency, we compute the new estimation-based power P_{estim} as

$$P_{estim} = Cf_{estim}V_{dd}^2$$

To evaluate the power savings, we compare our approach to the original case, where no DVS technique is in effect and the processor runs at full power all the time. We also compare our technique against a simple heuristic, which we call “simple WCET DVS”. This heuristic assumes a constant processor frequency for the duration of the entire clip, chosen so that all frames can be decoded before the deadline (slows down the processor such that the worst case decoding time is still before the deadline). Power consumption in this case is:

$P_{wcet} = Cf_{wcet}V_{dd}^2$, where the frequency f_{wcet} is determined from:

$t_{max}f_{orig} = t_{deadline}f_{wcet}$, with t_{max} bounding the largest (WCET) decode time during the entire clip (or a large window in the execution).

The “simple WCET heuristic” is similar to what a current DVS-capable device would actually perform and does not take advantage of the time difference between decoding different frames. In contrast, our estimation based approach tries to compute (within a reasonable error) the decode time for each individual frame based on the annotations stored in its header. An illustration of these three cases (no DVS, “simple WCET DVS” and estimation-based DVS) is depicted in Fig. 7.

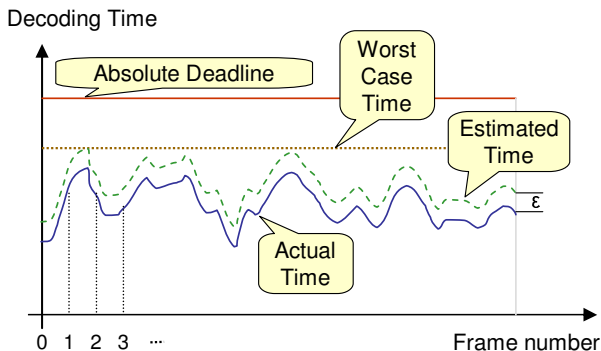


Figure 7. Different DVS heuristics

The energy savings in our technique come from the difference between the original CPU power (or WCET power) and our estimation-based power, which is always lower or equal to the original. We lower the CPU operating frequency by taking advantage of the slack time in the decoding phase, which we more accurately estimate through annotations and profiling.

We estimate the CPU power savings of our annotation-based technique over both the original power consumption, with no DVS scheme applied, and the power consumption in the “simple WCET DVS” case. The results are computed for both offline annotation (streaming of stored material) and real-time video streaming (e.g. videoconferencing).

Annotations can be stored to the stream in different ways: inside empty packets in the MPEG stream or in a dedicated control channel. In our experiment we assume that only the parameters of the fitting curve are transmitted for each clip, which requires just a number of bytes per clip (the evaluation of the function is performed at runtime, being a very simple quadratic function of the frame size). Another possibility would be to precompute the estimated decode time and store it with each frame (saving even more processing time at runtime, but requiring an additional number of bytes in the stream, though a minimal overhead compared to the frame size overhead compared to the frame sizes).

6. Experimental results

We use Sim-Panalyzer[1], a power architecture simulator, based on SimpleScalar. Sim-Panalyzer models an ARM processor architecture and performs cycle accurate simulation. At the end of execution the simulator reports a number of statistics, including power consumption by the internal units of the processor.

The simulated architecture is a StrongArm processor (200MHz), found in typical iPaq PDAs. The overhead for switching frequency is in the order of microseconds for a StrongArm processor, as opposed to tens of milliseconds for a frame decoding time and therefore is assumed negligible in our experiments. For measuring decoding times, we run the MPEG decoder, which is part of Berkeley MPEG Tools. The time estimator is written in Matlab.

As data input, we use video clips of 10 second each (300 frames) with a framerate of 30fps. Eight of the clips are encoded in a CIF (352x288) pixel format, and are taken from the multimedia community (akiyo, coastguard, container, foreman, hall, mobile, news, silent); they cover the entire range from very still to very dynamic. The other two clips are encoded in a 320x240 format (action, news).

Our experiments assume a frame-based DVS. Other approaches are possible too: for example DVS applied to a group of frames. In this latter case, buffers are required between the decoder and the display, to smoothen the variation between decoding times and avoid frame delays. Also, the power savings would be slightly lower, but the technique could be applied to even more devices (those with less DVS steps, more overhead).

We start by profiling the video clips in our simulator and generating the information required for data annotation.

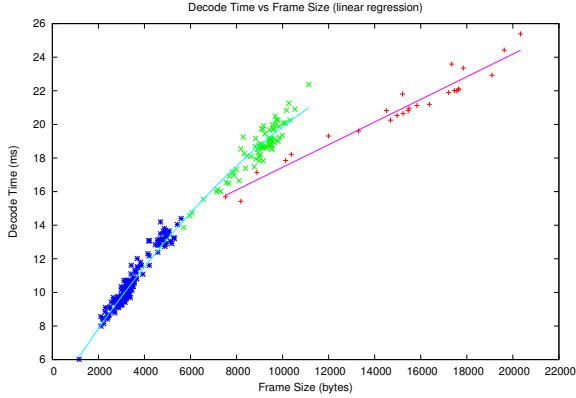


Figure 8. Curve fitting for *foreman clip*

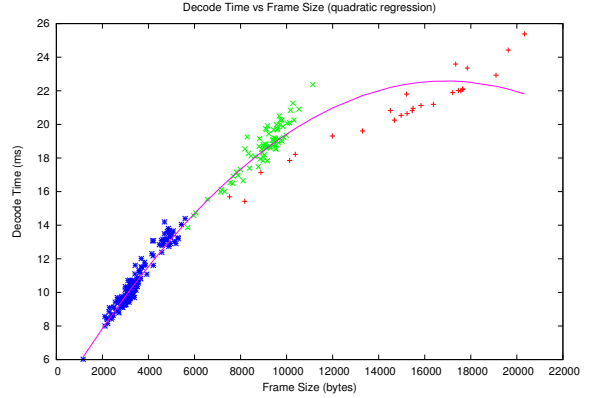


Figure 9. Curve fitting for *foreman clip*

The next step is to derive the approximation (estimation) function (decode time vs frame size).

We found that the best fit is given by a combination of linear ($f(x) = a * x + b$) and quadratic ($f(x) = a * x^2 + b * x + c$) functions.

The coefficients in these functions are determined using the “least square error” method. To evaluate the quality of the results, we compute the coefficient of determination (commonly known as the R^2 value) for the fit, which indicates the percent of the variation to be expected in the data. The closer that R^2 is to 1, the better the model “fits” the data.

We also compute the maximum residual value, which bounds the maximum variation from the predicted value we can expect (ϵ in our discussion above). This bound limit is used for assuring that almost no deadlines are missed, even in the presence of variations, thereby maximizing quality.

In our experiments we computed this threshold such that virtually no deadlines were missed. If the user can tolerate some quality degradation (frame loss), we could lower the threshold even further, significantly improving power savings. This is the trade-off between the quality of service delivered and the maximum power savings possible.

By analyzing a large number of video clips from different domains we concluded that the best fit (best R^2 value) is achieved when using the following two fitting curves: a linear function for I-frames, and a quadratic function for P and B-frames (example in Fig. 8). A single quadratic function would not capture the curve variations present in some of the clips between the different types of frames (see Fig. 9). The variation is introduced by the extra motion compensation step, which is present only in P and B-frames).

To simulate both an off-line scenario (stored media, annotations are performed off-line) and a real-time streaming (videoconferencing, annotations are performed in-line, by the proxy node) we experimented with two different setups:

- (a) **Individual estimation:** the case of stored material

Because the profiling is performed off-line in this case, we applied the profiling and estimation steps individually on each clip (i.e. each clip is first profiled, annotated, then its own estimator is used for DVS).

- (b) **Global estimation:** simulates a real-time scenario. For an in-line annotation, we first profile globally all clips from the same domain and with similar encoding parameters and derive a single global estimator. Next, we run this estimator on each clip separately. This is a more realistic approach for live video streams, where we may not want or be able to separately profile each clip, instead use a set of similar clips from the same video domain for the off-line profiling step.

The power consumption results for both scenarios are presented in Fig. 10, against the original (no DVS) and the “simple WCET DVS” case. The numbers are normalized to the original power consumption (i.e. 100% means no DVS).

We observe that the power savings for the processor are up to 50% over no DVS and up to 40% over “simple WCET DVS” (which already performs much better than the original case), for the individual estimation. Even for the global estimation case where the savings are slightly lower, the power gains are still high when compared to the other approaches. Our savings translate to around 15% for the entire device, with virtually no quality loss. More savings are possible if a lower quality of service is allowed.

7. Conclusion

This paper presents a new approach toward optimization for distributed multimedia streaming, using data annotation. We have shown how annotation can be used to accurately estimate runtime behavior. We have described our experiments where we used annotations for data-aware DVS.

Our results show that good CPU power savings can be obtained with annotation-based estimations performed on video clips, for both off-line and on-line scenarios.

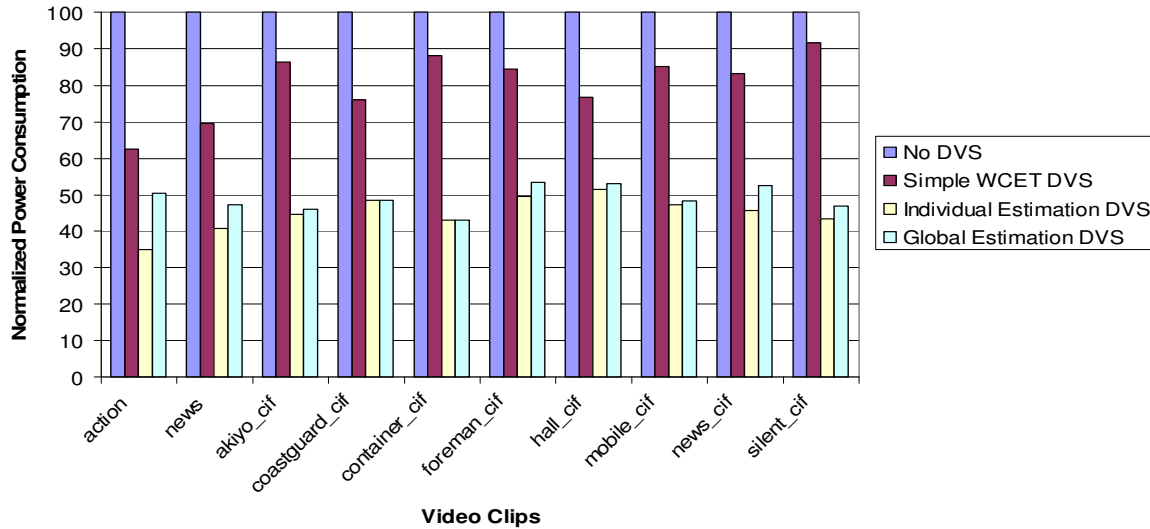


Figure 10. Normalized CPU power consumption results

As future work, we plan to extend our algorithm to include dynamic estimation correction when the prediction error becomes too high in real-time video streaming. We also plan to look into mobility issues and the optimizations possible when multiple clients are streaming the same video.

References

- [1] Sim-analyzer 2.0 reference manual. Technical report, University of Michigan, University of Colorado.
- [2] A. Bavier, B. Montz, and L. L. Peterson. Predicting MPEG execution times. Technical Report 97-15, University of Arizona, 1997.
- [3] S. Chandra and C. S. Ellis. JPEG compression metric as a quality-aware image transcoding. In *USENIX Symposium on Internet Technologies and Systems*, 1999.
- [4] S. Chandra, A. Gehani, C. S. Ellis, and A. Vahdat. Transcoding characteristics of web images. In *Multimedia Computing and Networking (MMCN'01)*, 2001.
- [5] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, pages 732–737. ACM Press, 2002.
- [6] R. Cornea, A. Nicolau, and N. Dutt. Stream annotations for energy trade-offs in a video decoder for multimedia applications. Technical Report 06-09, University of California, Irvine, 2006.
- [7] D.G.Lee, D.Panigrahi, and S.Dey. Network-aware image data shaping for low-latency and energy-efficient data services over the Palm wireless network. In *World Wireless Congress (3G Wireless)*, 2003.
- [8] S. Z. Guyer and C. Lin. An annotation language for optimizing software libraries. In *Domain-Specific Languages*, pages 39–52, 1999.
- [9] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *International Conference on Computer Architecture*, pages 254–265, 2001.
- [10] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips. Power-aware video decoding. In *22nd Picture Coding Symposium*, April 2001.
- [11] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *International Symposium on Computer Architecture*, 2003.
- [12] A. Tripathi and M. Claypool. Improving multimedia streaming with content-aware video scaling. Technical Report WPI-CS-TR-01-02, Worcester Polytechnic Institute, 2001.