Watermarking Video Clips with Workload Information for DVS

Yicheng Huang Samarjit Chakraborty Ye Wang
Department of Computer Science, National University of Singapore
E-mail: {huangyic, samarjit, wangye}@comp.nus.edu.sg

Abstract

We present a lightweight scheme for watermarking or annotating video clips with information describing the workload that would be incurred while decoding the clip. This information can be used at run time to scale the operating voltage/frequency of the processor on which the video clip is to be decoded. Our main contribution is a fast, low-cost bitstream analysis technique for estimating the decoding workload of a video clip. Using this technique the workload information can be inserted into a clip while it is being downloaded onto a battery-powered portable device from a desktop computer or a server, for later playback. In contrast to control-theoretic feedback techniques that have been traditionally used for predicting video decoding workload at runtime for dynamic voltage/frequency scaling, we show that our scheme performs better in terms of energy savings and has significantly lower buffer requirements.

1 Introduction

Energy efficiency is today one of the most critical issues in the design of battery-powered portable devices such as mobile phones, PDAs and audio/video players. The predominant workload running on most of these devices are now generated by multimedia processing applications (e.g. audio/video decoders). This has resulted in a considerable interest in power management schemes for portable devices running multimedia applications [2, 3, 4, 9, 10].

In this paper we propose a new approach for dynamic voltage scaling (DVS) in the context of multimedia applications. In what follows, we will only be concerned with MPEG-2 video decoding. However, the proposed scheme is very general and can be applied to both, other types of video, as well as audio processing applications. The scheme relies on a lightweight offline bitstream analysis of a video clip to predict the workload that will be generated while decoding the clip. Based on this analysis, the video clip is watermarked with the predicted workload information. Such watermarking can either consist of metadata information being inserted into the video clip or such information being saved as a separate file. At runtime, the decoder reads this metadata information and controls (or scales) the voltage and frequency of the processor. The metadata information will typically consist of the frequency at which the processor needs to be run at any point in time. However, the metadata might also consist of workload information (such as processor cycle demands), from which the required processor frequency is computed at runtime. The amount of metadata that needs to be inserted depends on the granularity, or how often the frequency of the processor needs to be changed. If the amount of metadata allowed is large, then potentially higher amounts of energy can be saved.

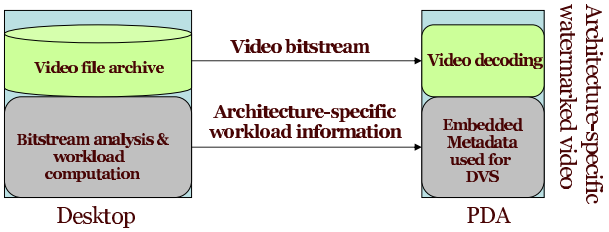

Figure 1. Overview of the proposed scheme.

Figure 1 illustrates the key idea behind our scheme. It shows a setup where a desktop computer or a multimedia server stores video clips. When such clips are being downloaded into a portable device, a lightweight bitstream analysis scheme runs on the desktop computer and watermarks the video clip with workload information. The watermarked video clips are then stored in the portable device. At runtime, the workload information is read out and used for dynamic voltage and frequency scaling. It may be noted here that downloading audio/video clips from a desktop into a portable device is currently the most common use scenario. The other possible scenario where such clips are directly downloaded into the portable device over the network is still fairly uncommon and many portable media players currently don't even support network interfaces; however that might change over the next few years.

The key point to note here is that all the previously known techniques predict at *runtime* the processor frequency f with which a segment of the video clip s needs to be decoded *without* looking into s. In contrast to such techniques, we perform an offline analysis of the compressed bitstream corresponding to s and insert the metadata f *before* the start of s. The runtime system simply reads f and sets the processor frequency to this value. Also, note that the metadata information need not be equally spaced out within the video clip. If the computational workload of a clip is highly variable and irregular, then this might require

1063-9667/08 $25.00 © 2008 IEEE
DOI 10.1109/VLSI.2008.103

more metadata. Whereas certain portions of the clip might not exhibit any variation, in which case it might suffice to run the processor at a constant frequency (and hence only this constant frequency value needs to be inserted once). The inserted metadata information might consist of frequency as well as voltage values, depending on the type of the underlying processor. Again, in contrast to this, many current approaches attempt to scale the processor frequency at regular intervals (e.g. at all video frame boundaries). We show that our scheme leads to energy savings that are comparable and often better than those achieved by known DVS schemes based on runtime (control-theoretic) prediction techniques. More importantly, the buffer requirement in our scheme is significantly smaller.

Main Challenge: The main challenge in implementing the proposed scheme lies in the metadata computation process. Clearly, the exact values of the metadata inserted will depend on the architecture of the processor in the portable device (e.g. its instruction set architecture, voltage/frequency range and the steps in which they can be changed) and also on the decoder application running on this processor. The metadata computation task running on the desktop is aware of the architecture of the portable device. However, the generated metadata is transparent to the user; it only resides inside the portable device.

One possibility is to insert this metadata information directly during the encoding process. However, this would assume that the details of the decoder and also the processor on which the decoder would run are already known at the time of encoding. It would also amount to generating video clips which can only be played on certain devices or on devices manufactured by the same company, which are all based on the same or on similar processor architectures. Although this is a feasible option (e.g. the Windows Media format is only targeted towards Windows platforms), it is clearly very restrictive.

We therefore propose a scheme where the metadata information is directly inserted into a video clip based on the architecture of the portable device. Towards this, we assume the following scenario. To download a video file into such a device, it would be connected to a desktop computer on which an application program specialized for this device would run. This program would perform a bitstream analysis of the video file being downloaded, calculate the appropriate metadata information and insert this information into the file. Since the program is specialized for this device, the metadata computed is specific to its processor architecture and also to the decoder application running on the device. Each such device would therefore be shipped with an application program (that would run on the desktop computer) that is specific to the device. This scheme has two main advantages: (i) It is flexible, i.e. the portable device can play video files encoded in standard formats such as MPEG-2 and the metadata-inserted files are not visible to the external world; they only exist inside the portable device. (ii) The bitstream analysis process, which might be involved, can run on a desktop computer and not on the portable device, which would typically be resource constrained.

Metadata Computation: The only remaining question that needs to be answered is, given a video file, how is the metadata or the watermark information exactly computed? What follows in this paper will mostly be concerned with answering this question.

The most straightforward answer to this question is, simulate the decoding of the given video file on a software model of the processor's architecture. This would result in a trace of the file's processor cycle requirements, e.g. the number of processor cycles required to decode each macroblock of the video file. From this trace, the clock frequency with which the processor should be run while decoding any segment of the file can be computed. The computed frequencies will constitute the metadata information to be inserted into the video file. Towards this, it would be possible to use processor instruction set simulators like SimpleScalar [1] to compute the trace of processor cycle requirements of a video file. However, a cycle-accurate simulation of the execution of a processor is extremely expensive in terms of the simulation time involved. For example, simulating the decoding of a 30 seconds long MPEG-2 video clip requires more than half an hour using SimpleScalar. Hence, this scheme is not feasible if the metadata computation needs to be done while downloading a video file from a desktop computer into a portable device, without any perceptible delay.

We therefore propose an alternative lightweight scheme where we do not simulate the execution/decoding of the video clip. Instead we perform a bitstream analysis to *predict* the processor cycle requirements of each macroblock. The scheme has to be lightweight since it has to be fast enough and not delay the downloading process from the desktop computer to the portable device. We would again like to point out that in contrast to this, runtime prediction schemes predict the processor cycle requirement of a video segment *without* looking into the segment. Our scheme allows for the bitstream analysis because it is done offline (i.e. not at runtime) while the video file is being downloaded into the device. The prediction scheme we propose is based on classifying the video decoding tasks into two groups—those that are CPU-bound, such as motion compensation, and others which are memory-bound such as those responsible for dithering. The processor cycle requirements of memory-bound tasks are almost constant and are hence easy to predict. Hence, we shall mostly be concerned with predicting the processor cycle requirements of CPU-bound tasks. As already mentioned, in this paper we will use MPEG-2 for the sake of illustration.

Paper Organization: The rest of this paper is organized as follows. In the next section we present our analysis scheme for MPEG-2. This consists of estimating the workload incurred by the different subtasks of an MPEG-2 decoder. In Section 3 we show how our approach compares with known DVS schemes that use control-theoretic techniques for online/runtime workload prediction. Finally, we conclude in Section 4 by outlining some directions for future work.

2 MPEG-2 Bitstream Analysis

An MPEG-2 video sequence is made up of a number of *frames*, where each frame contains several *slices*. Each slice in turn consists of a number of *macroblocks* (MBs). Decoding an MPEG-2 video can therefore be considered as decoding a sequence of MBs. This involves executing the following tasks for each MB: variable length decoding (VLD), inverse discrete cosine transformation (IDCT) and motion compensation (MC). Other tasks, such as inverse quantization (IQ) involves a negligible amount of computational workload and hence we ignore them for the purpose of our analysis. The analysis we present here can be used for voltage/frequency scaling at the MB granularity (clearly, the same analysis can be used at the slice or frame granularity as well). Given a sequence of MBs, in this section we describe how to predict the processor cycle requirements corresponding to the tasks VLD, IDCT and MC for each of these MBs. We compare our predicted results with those obtained from simulating the execution of these tasks using the SimpleScalar [1] instruction set simulator (with the Sim-Profile configuration), with the same sequence of MBs as input. Since we envisage the decoder to run on a general-purpose processor (such as those found on a PDA), we choose our processor to be a RISC processor (similar to a MIPS3000) without any MPEG-specific instructions. We use Test Model 5 (TM5) [7] as our MPEG-2 decoder application. Although not an optimized decoder, it is acceptable for our analysis since all MPEG-2 decoders have a similar code structure. We experimented with five different commonly used benchmark video clips, encoded with a 4M/s bitrate: (i) Flwr (has moderate motion), (ii) Tennis (still background with moving foreground), (iii) Susi (very low motion), (iv) V700 (still image) and (v) Football (very fast motion).

The Variable Length Decoding Task: The IDCT coefficients in MPEG-2 are encoded using variable length encoding, which involves Run-Length Coding followed by Huffman Coding. Some run-length codes are coded using longer Huffman codes compared to the others. The number of processor cycles required for the Huffman decoding depends on the length of the Huffman codes used. Therefore, the number of processor cycles required by the VLD task for any input MB is expected to depend on the number of non-zero IDCT coefficients in it. Our simulations confirm that this is indeed the case and the relationship is a linear one. Hence, we use the following function as an estimate of the number of processor cycles required by the VLD task for any MB: $n_{\text{vld}} = a \times n_{\text{coeff}} + b$. Here, n_{vld} is the estimated number of processor cycles, n_{coeff} is the number of non-zero coefficients in the MB and a and b are constants which depend on the processor architecture and the VLD code. From our experiments we determined the values of a and b to be 140 and 3000 respectively for our setup. For the Flwr video we noted that for around 36% of the MBs, the processor cycle requirements were predicted with an error of less than 2%. For *all* MBs, the prediction error was less than 10% (in the range of -1000 to $+2000$ processor cycles). Other clips also had similar error distributions. It should be noted that the values of a and b in this case capture the characteristics of one specific architecture. Recall that our watermarking is architecture-specific. For a different processor architecture and decoder implementation, these values will change and they are hardcoded in the metadata insertion task running on the desktop (see Figure 1).

The Motion Compensation Task: MBs constituting an MPEG-2 clip may be classified into three categories: those involving no motion compensation (I-type), those involving only forward motion compensation (P-type) and those involving both forward and backward motion compensation (B-type). Therefore, the MC task for P-type MBs incur about half the number of processor cycles compared to B-type MBs and I-type MBs do not incur any computational workload.

We used SimpleScalar simulations to obtain the processor cycle distribution for the MC task for each of our five MPEG-2 video clips. As expected, with the exception of the V700 clip (still image), the number of processor cycles for all of these clips were distributed into three distinct clusters. The first cluster (located around 0 processor cycles) correspond to the I-type MBs, the second (around 3000 - 7000 cycles) correspond to the P-type MBs, and finally the third cluster (around 9000 - 17000 cycles) correspond to the B-type MBs. In the V700 clip, almost all the MBs use the same type of motion compensation, thereby resulting in a single cluster.

Since the processor cycle distribution within each cluster is reasonably large, a prediction solely based on MB type will not be accurate enough. The variability within each cluster results from factors like whether the MC task is frame- or field-based and whether the motion vectors are half- or one-pixel accurate. We account for these as follows.

The code for the MC task may be considered to be composed of a number of subroutines, each of which is essentially the same function, but called with different parame-

ters. Let us denote this function by $\mathcal{F}$. The number of processor cycles required to execute $\mathcal{F}$ depends only on its input parameters. Depending on the input MB, these parameters include whether (i) Y[1] component's x-dimension is HALF-PIXEL, (ii) Y component's y-dimension is HALF-PIXEL, (iii) U or V component's x-dimension is HALF-PIXEL, (iv) U or V component's y-dimension is HALF-PIXEL, (v) forward or backward motion compensation is required, and (vi) the motion compensation window size is 16×8 or 16×16. Different MBs call $\mathcal{F}$ different number of times and with different values of the above boolean parameters. For example, a P-type non-progressively coded MB, which uses frame-based motion compensation, will call $\mathcal{F}$ twice. Both of these calls are with the same list of parameters $(0, 0, 0, 0, 1, 16 \times 8)$. Similarly, a B-type, progressively coded MB, which uses field-based motion compensation, will also call $\mathcal{F}$ twice, but with the parameters $(1, 1, 1, 1, 1, 16 \times 16)$ and $(1, 1, 1, 1, 0, 16 \times 16)$.

Based on this observation, we predict the processor cycle requirement of the MC task by first simulating the execution of $\mathcal{F}$ with all possible input parameter values. Since there are six boolean parameters, they result in a total of $2^6 = 64$ possible input values. The processor cycle requirement of $\mathcal{F}$ corresponding to each of these 64 possible inputs is stored in a table. Now, given a sequence of MBs, by parsing each MB, we determine the number of times $\mathcal{F}$ is called and with what input parameter values. Using these and our precomputed table of cycle requirements we are able to predict the cycle requirements for each of the MBs. For approximately 40% of the MBs the error incurred is less than 2%. Further, none of the MBs incur an error of more than 4%. Again, it should be noted that the contents of the precomputed table is architecture-specific.

The Inverse Discrete Cosine Transform Task: Usually each MB in MPEG-2 contains four Y blocks, one U block and one V block. Each of these blocks are of 8×8 pixels size. Hence, the input data size to the IDCT task is the same for all MBs, which results in the same computational workload being incurred for all MBs. However, an optimized implementation of the IDCT task takes into account that several IDCT coefficients might be zero and exploits this fact to save some computation. In spite of this, it is a reasonably good approximation to assume that the number of processor cycles incurred by the IDCT task for any MB is a constant, as is confirmed by our experimental results. We selected $2 \times 10^4 + 4000$ as the processor cycle requirement for any macroblock (where 4000 cycles is an architecture-specific "safety margin"). With this prediction, around 61% of the MBs incur an error of less than 2% and 91% of the MBs incur an error of less than 10%.

[1]Each frame in MPEG-2 is represented in the YUV color space. See the ISO MPEG-2 standard for details.

Total Cycle Requirements: The total number of processor cycles required to decode a MB may be predicted by summing up the predicted values for the VLD, MC and IDCT tasks and adding a safety margin of 500 cycles (this value may again be obtained from simulations and would depend on the processor architecture and the decoder code).

3 Experimental Results

To evaluate our scheme, we conducted several experiments using an MPEG-2 decoder application. For estimating the power consumption, we adopted the model from [5]: $P \propto \sum (v_{dd,i})^2 f_i$, where $v_{dd,i}$ and f_i are voltage and frequency values set by the scheduler at *i*th adaptation point. We assumed that $f \propto v_{dd,i}$. We also assumed that the processor can be scaled continuously. The experiments were repeated on five different video sequences that we listed at the beginning of Section 2. Further, we used three different adaptation intervals: every one macroblock of the bitstream, every ten macroblocks and every twenty macroblocks.

We compared our energy savings with those achieved by the DVS scheme published in Wu et al. [8]. Wu et al. uses a PID controller which tracks changes in the buffer fill level (at the input of the processor) and accordingly regulates the processor's speed and voltage. The reason we selected this scheme is because (i) it can handle the stream burstiness and the data-dependent variability in the decoder's execution demand; (ii) it is suitable for buffer-constrained architectures; and (iii) it also uses fixed adaptation intervals as we do in our scheme. Furthermore, to the best of our knowledge, the scheme of Wu et al. represents on of the most advanced DVS techniques recently published.

Comparative results: Here, we would like to point out the contrasts between our approach and that of Wu et al. We predict the workload incurred by the decoder in an offline fashion by analyzing the video clips. Wu et al. on the other hand uses control theoretic techniques to predict the future workload in an online fashion. As we show in this section, the energy savings obtained by both these schemes are similar, with our scheme performing marginally better for most clips. However, the main difference is in terms of the quality of the decoded video, especially in cases where the playout buffer is small. Using the PID controller to scale the processor frequency results in many decoded macroblocks to miss their deadlines (i.e. the output display device is occasionally unable to read a decoded macroblock from the playout buffer). This problem is especially acute when the playout buffer size is small (≤ 1.5 kB). On the other hand, our scheme consistently performs well even for very small buffer sizes and achieves better energy savings. Lastly, as already mentioned, because our scheme relies on an offline

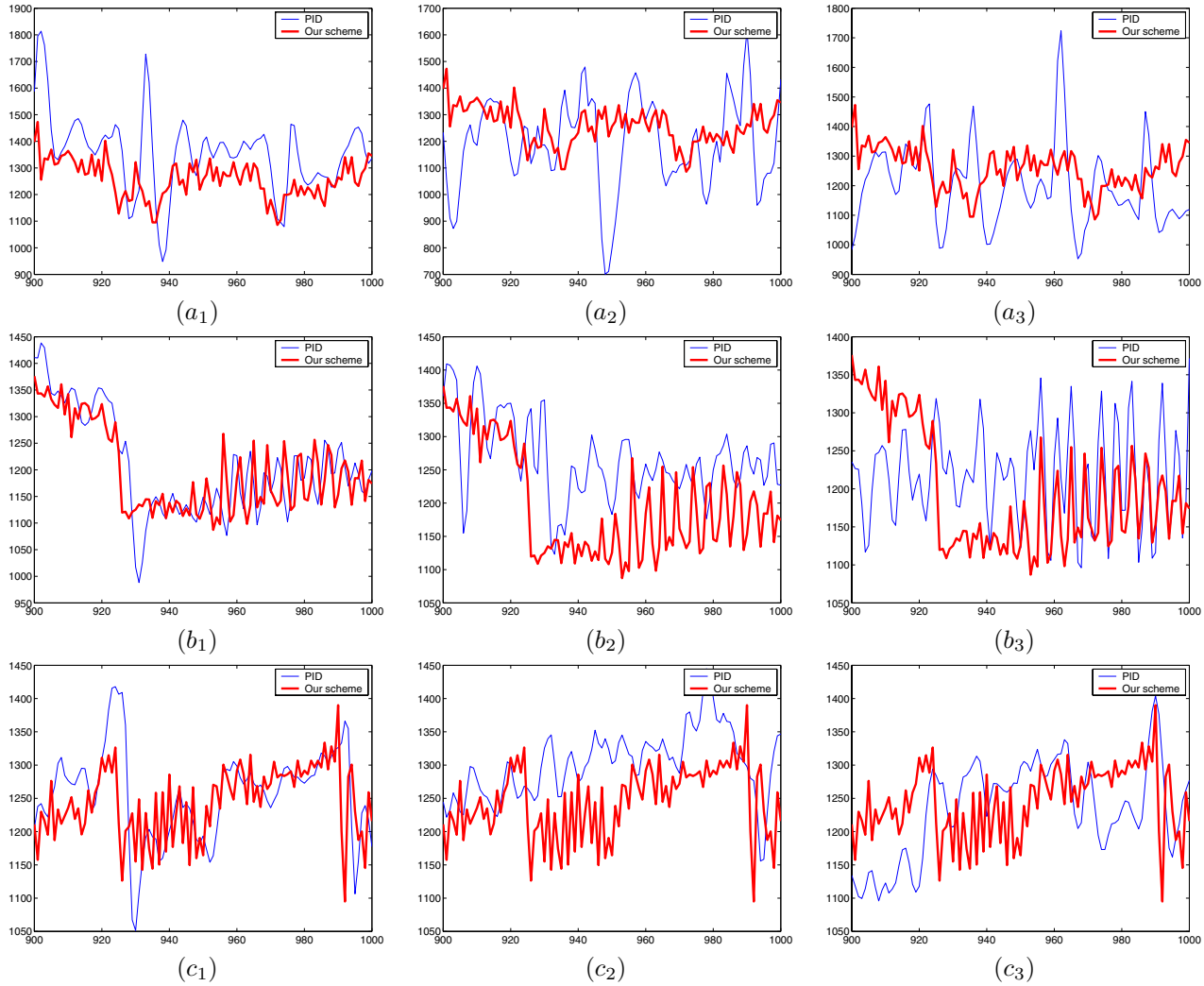

Figure 2. Excerpts from processor frequency schedules generated by our scheme and the PID controller. The horizontal axis represents the adaptation points (adaptation interval specified in terms of no. of macroblocks) and the vertical axis represents the processor frequency (in Hz). (a), (b) and (c) represent the video clips `Flwr`, `Susi` and `Football` respectively. Indices 1 - 3 represent the adaptation intervals of 1, 10 and 20 macroblocks respectively.

workload prediction, it incurs a smaller runtime overhead compared to online schemes.

Figure 2 shows how the processor's frequency is scaled using the PID controller and using our scheme. Each row in this figure represents a specific video clip. The first row shows the results obtained with the clip `Flwr`. Recall that this clip contains moderate amounts of motion. The second row shows the results for the clip `Susi`, which has very low motion. Finally, the third row shows the results for `Football` which has very high motion. All the clips have a frame resolution of 704×480 pixels and are displayed at the rate of 30 frames/second. Each column in this figure shows the results for a specific adaptation interval. Column 1 shows the results when the adaption interval was set to one macroblock, i.e. the processor's voltage and frequency are changed after the decoding of each macroblock. Column 2 shows the results for an adaptation interval of 10 macroblocks and finally Column 3 shows the results for an adaptation interval of 20 macroblocks. While an adaptation interval of one macroblock might incur too much of an overhead, we believe that adaptation intervals of 10 and 20 macroblocks are quite realistic.

The setup we used consists of a playout buffer which stores the decoded macroblocks. This buffer is read out by output device at a constant frame rate. For the PID controller, the expected buffer fill level was always set to half the size of this playout buffer. The baseline case—for comparing the energy savings obtained by both the schemes—was set as a constant processor frequency. This frequency was computed assuming that *all* macroblocks

Buffer size (kB)	Watermarking	PID
0.5	0.1987	42.8906
1.0	0.1852	26.8384
1.5	0.1768	10.3906
2.0	0.1700	2.5253
5.0	0.1077	0.0101

Table 1. Percentage of macroblocks missing their deadlines for different playout buffer sizes.

have the worst-case execution requirement and an output rate of 30 frames/sec will have to be sustained. With this base case, and a one-macroblock adaptation interval, both our scheme and the PID controller-based scheme achieved energy savings in the range of 89.1% and 96.3%. For all the five clips that we experimented with (each having a varying degree of motion), our scheme obtained larger energy savings except for the `V700` clip where our scheme obtained a savings of 94.6% and the PID controller-based scheme obtained a savings of 96.3%.

When the adaptation interval was set to 10 macroblocks, the energy savings for both the schemes were in the range of 83% and 96.1%. Again, for all the clips, our scheme performed better than the PID controller except in the case of the `V700` clip. Finally, with an adaptation interval of 20 macroblocks, the energy savings were again in the range of 83% and 96.3% with our scheme performing worse than the PID controller for the `V700` clip.

Our results show that both the schemes are fairly robust in terms of the adaptation interval and are hence practical to implement. However, the buffer requirements for the two schemes are quite different. For the one macroblock adaptation interval, the buffer requirement of our scheme was between 19% and 50% of that of the PID controller-based scheme (when the buffer size was not constrained). In particular, for the `V700` clip, the buffer requirement of our scheme was only 19% of what was required by the PID controller scheme. For adaptation intervals of 10 and 20 macroblocks, the buffer requirements of our scheme varied between 22% and 88% of that required by the PID controller.

When the buffer size was restricted, the PID controller-based scheme suffered from severe deterioration in the output video quality because of several macroblocks missing their deadlines. Table 1 lists the percentage of macroblocks missing their deadlines for different playout buffer sizes. Note that for relatively small buffer sizes, more than 20% of the macroblocks can miss their deadlines when the PID controller-based scheme is used. Our proposed scheme, on the other hand, consistently performs well even for small buffers. However, with relative large buffers, almost all the macroblocks meet their deadlines with the PID controller.

In summary, the energy savings achieved by both the schemes are comparable. However, the buffer requirements for the PID controller scheme, on an average, was almost double of what was required by our scheme. Hence, our scheme is particularly interesting given the current interest in DVS schemes for buffer-constrained architectures [6]. Further, as mentioned before, our scheme does not involve any runtime overhead, whereas online workload prediction schemes such as the PID controller often incurs a considerable runtime overhead and also requires constant monitoring of the buffer fill level.

4 Concluding Remarks

In this paper we presented a novel scheme for watermarking video clips with workload information, which can be extracted at runtime for dynamic voltage/frequency scaling. The inserted metadata is not visible to the external world, in contrast to previous proposals for the video *encoder* to generate the workload information, which we believe is less flexible because it restricts the platforms on which the resulting video clips can be decoded. Our main contribution is a fast bitstream analysis technique to predict the computational workload involved in decoding a video clip. We believe that the basic scheme that we presented in this paper can be further refined to more accurately estimate the decoding workload. In particular, the cache/memory organization of the target device (e.g. PDA) may be taken into account in conjunction with the bitstream analysis technique that we presented here.

References

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[2] M. Buss, T. Givargis, and N. Dutt. Exploring efficient operating points for voltage scaled embedded processor cores. In *Real-Time Systems Symposium (RTSS)*, 2003.

[3] H. V. A. et al. Energy-aware system design for wireless multimedia. In *DATE*, 2004.

[4] S. M. et al. Integrated power management for video streaming to mobile handheld devices. In *ACM Multimedia (MM)*, 2003.

[5] Y.-H. Lu, L. Benini, and G. D. Micheli. Dynamic frequency scaling with buffer insertation for mixed workloads. *IEEE Trans. on CAD of Integrated Circuits andd System*, 2002.

[6] A. Maxiaguine, S. Chakraborty, and L. Thiele:. DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In *CODES+ISSS*, 2005.

[7] http://www.tns.lcs.mit.edu/manuals/mpeg2/.

[8] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. *ASPLOS*, 2004.

[9] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[10] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. In *ACM Multimedia (MM)*, 2004.