Using Offline Bitstream Analysis for Power-Aware Video Decoding in Portable Devices

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

ABSTRACT

Dynamic voltage/frequency scheduling algorithms for multimedia applications have recently been a subject of intensive research. Many of these algorithms use control-theoretic feedback techniques to predict the future execution demand of an application based on the demand in the recent past. Such techniques suffer from two major disadvantages: (i) they are computationally expensive, and (ii) it is difficult to give performance or quality-of-service guarantees based on these techniques (since the predictions can occasionally turn out to be incorrect). To address these shortcomings, in this paper we propose a completely new approach for dynamic voltage and frequency scaling. Our technique is based on an offline bitstream analysis of multimedia files. Based on this analysis, we insert metadata information describing the computational demand that will be generated when decoding the file. Such bitstream analysis and metadata insertion can be done when the multimedia file is being downloaded into a portable device from a desktop computer. In this paper we illustrate this technique using the MPEG-2 decoder application. We show that the amount of metadata that needs to be inserted is a very small fraction of the total size of the video clip and it can lead to significant energy savings. The metadata inserted will typically consist of the frequency value at which the processor needs to be run at different points in time during the decoding process. Lastly, in contrast to *runtime prediction*-based techniques, our scheme can be used to provide performance and quality-of-service guarantees and at the same time avoids any runtime computation overhead.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and embedded systems*

General Terms

Algorithms, Performance, Design

Keywords

DVS, Video decoding, Metadata, Bitstream analysis

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 [3, 5, 6, 11, 12]. Towards this, two main directions have evolved over the last couple of years—dynamic voltage scaling (DVS) [4,7] and dynamic power management (DPM) [1]. DVS relies on changing the frequency and voltage of the processor at runtime to match the workload demand generated by an application. On the other hand, DPM-based techniques rely on switching off parts of a device (processor, hard disk, display, etc.) at runtime, based on their usage.

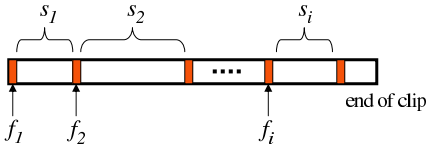

Figure 1: Video clip with metadata information.

Typically, multimedia applications exhibit a high degree of data-dependent variability in their execution requirements. For example, the ratio of the maximum and the average workload generated by an MPEG decoder application can be as high as a factor of 10. In addition, there might also be other types of variability such as the variability in the input-output rates of an application (e.g. the variable length decoding task in an MPEG decoder consumes a variable number of bits corresponding to each compressed macroblock that is generated). In the case of multimedia applications, many DVS-based techniques exploit this variability to scale the voltage and frequency of the processor at runtime to match the changing workload. Towards this, two broad classes of techniques have evolved. The first class relies on control-theoretic feedback techniques [9] to predict the future workload, based on the workload history in the recent past. The main disadvantages of this class of techniques are that (i) they are computationally expensive and are difficult to implement (especially considering that the voltage of the processor needs to be frequently changed at runtime), (ii) it is difficult to give performance and quality-of-service guarantees based on these techniques (because occasionally the predictions might be wrong). The second class of techniques attempt to statically characterize the bounds on the variability and use these bounds at runtime. Although this allows for quality-of-service guarantees, usually such static characterizations are overly pessimistic and do not lead to useful energy savings. There has been a significant amount of work on both these approaches, both, in the embedded systems, as well as in the multimedia communities and the disadvantages listed above are well recognized. However, the commercial availability of voltage/frequency-scalable processors (and the increasing concern about battery lifetime of portable devices) implies that research on DVS and DPM techniques will continue in the future.

Our Contribution: In this paper we propose a completely new approach for DVS in the context of multimedia applications. In what follows, we will only be concerned with video decoding. However,

MM'05, November 6–11, 2005, Singapore.

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 an offline bitstream analysis of a video clip to predict the workload that will be generated while decoding the clip. Based on this analysis, metadata information is inserted into the video clip or is 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 illustrates the key idea behind our scheme. It shows a video clip along with the inserted metadata. This metadata comprises of the frequency with which the processor needs to be run—frequency f_1 for the segment of the video clip s_1, frequency f_2 for the segment of the clip s_2, and so on. We assume that the decoder is specially designed to read the metadata information in addition to the original video data. It reads this metadata and changes the processor frequency at appropriate times. With the availability of *software-controlled* voltage/frequency-scalable processors, this approach is certainly feasible today.

The key point to note here is that all the previously known techniques predict at *runtime* the processor frequency f_i with which the segment s_i needs to be decoded *without* looking into s_i. In contrast to such techniques, we perform an offline analysis of the compressed bitstream corresponding to s_i and insert the metadata f_i. The runtime system simply reads f_i 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 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.

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 (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.

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 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 [2] 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.

We therefore propose an alternative 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. 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. In this paper we will use MPEG-2 for the sake of illustration.

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 com-

putational 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 [2] 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) [8] 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 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).

VLD 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 (see Figure 2). Figures 2(a) - (e) show the plot of the number of processor cycles required by the VLD task for different number of non-zero IDCT coefficients in a MB. Note that each of these plots consist of two distinct linear bands, where the upper band results from large MBs which involve extra I/O operations. We neglect these MBs, since they constitute less than 1% of the total number of MBs in a clip (this error may be reduced if necessary [10]). We fit a straight line on the lower band using least squares fitting.

The resulting function serves 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. The prediction error (in processor cycles) resulting from this function is shown in Figure 2(f) for the Flwr video clip. Other clips have similar error distributions. 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).

MC 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.

Figures 3(a) - (e) show the processor cycle distribution for the MC task for each of our five MPEG-2 video clips (obtained from SimpleScalar simulations). As expected, with the exception of the

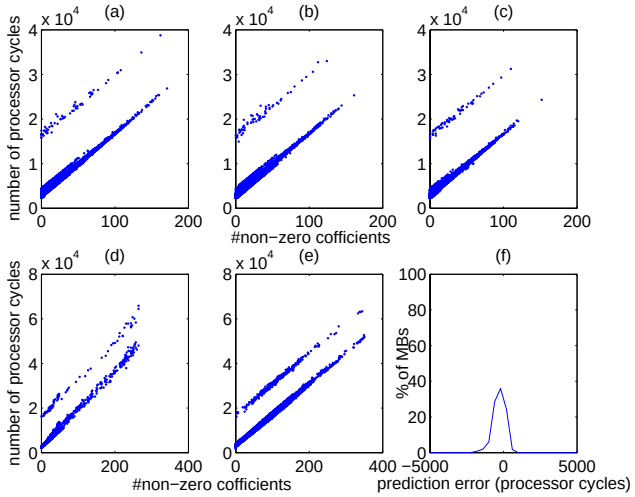

Figure 2: Workload generated by the VLD task.

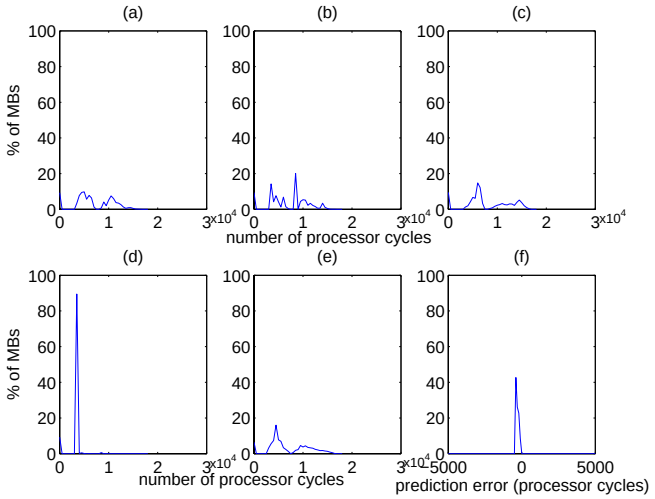

Figure 3: Workload generated by the MC task.

V700 clip (still image), the number of processor cycles for all of these clips are distributed into three distinct clusters. The first (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 (Figure 3(d)).

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 parameters. 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

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

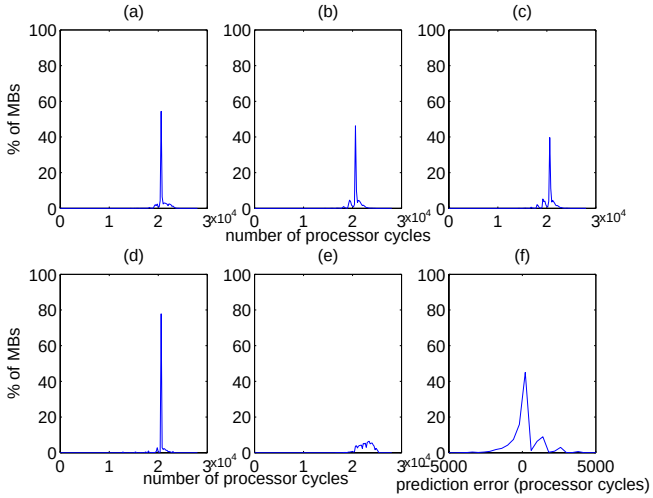

Figure 4: Workload generated by the IDCT task.

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. The error distribution resulting from this prediction scheme is shown in Figure 3(e). 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%.

IDCT Task: Normally, 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. This is once again confirmed by our experimental results shown in Figures 4(a) - (e). Note that the variation around the processor cycle requirement of 2×10^4 cycles results from the optimized IDCT implementation. From these results we select $2 \times 10^4 + 4000$ as the processor cycle requirement for any macroblock (where 4000 cycles is the "safety margin"). The error distribution resulting from this prediction is shown in Figure 4(f). Around 61% of the MBs incur an error of less than 2% and 91% of the MBs incur an error of less than 10%.

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. CONCLUDING REMARKS

To evaluate the effectiveness of our scheme for dynamic frequency scaling, we conducted three sets of experiments: (i) without processor frequency scaling, (ii) using frequency scaling based on our proposed scheme, and (iii) frequency scaling based on a moving history-based workload prediction. We assumed that the clock frequency of our processor may be scaled in the range of 200 - 500MHz, in steps of 50MHz (which corresponds to processors found in high-end PDAs). At any point in time, the processor has to run at a frequency such that it can sustain the output frame rate of 25fps. In the absence of dynamic frequency scaling, the processor's frequency is determined by this output frame rate and the *maximum* number of processor cycles required to process any MB. Compared to this baseline case, our scheme achieves more than 75% energy savings. Further, this scheme incurs at most 2.7% workload prediction error, in comparison to around 12% in the case of moving history-based prediction schemes where the cycle requirement of a MB is predicted from the requirements of the immediately preceding MBs (at the cost of significant runtime overheads). Moreover, this error substantially increases if the prediction is done for a group of MBs, rather than for *every* MB. Lastly, the amount metadata inserted in our scheme is less than 0.01% of the clip size when frequency scaling is done at a half-frame interval. We skip further details due to space constraints. The work presented here is a part of ongoing research; in future we plan to conduct more detailed experiments to evaluate our scheme and also compare it with other known dynamic frequency scaling techniques.

4. REFERENCES

[1] A. Acquaviva, L. Benini, and B. Riccó. An adaptive algorithm for low-power streaming multimedia processing. In *Design, Automation and Test in Europe (DATE)*, 2001.

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

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

[4] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *ICCAD*, 2002.

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

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

[7] C.J. Huges, J. Srinivasan, and S.V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *IEEE MICRO*, 2001.

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

[9] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *RTAS*, 2005.

[10] P. Soderquist and M. Leeser. Optimizing the data cache performance of a software mpeg-2 video decoder. In *ACM Multimedia (MM)*, 1997.

[11] 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.

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