Towards Class-Based Dynamic Voltage Scaling for Multimedia Applications

Richard Urunuela[1], Gilles Muller[1], Julia L. Lawall[2]

[1] Ecole des Mines de Nantes
44307 Nantes cedex 3
rurunuel@emn.fr, Giller.Muller@emn.fr
[2] DIKU, University of Copenhagen
2100 Copenhagen Ø, Denmark
julia@diku.dk

Keywords. Dynamic voltage scaling, multimedia applications, embedded systems

1 Introduction

As more and more of computing has become mobile, and thus reliant on battery or solar power, reducing energy consumption has become critical. One significant consumer of energy in any computer system is the CPU. To reduce the CPU energy consumption, many CPUs now allow dynamic scaling of the CPU voltage, known as DVS. As a linear reduction in CPU voltage leads to a quadratic reduction in its energy consumption [1], this approach is very attractive. Nevertheless, reducing the voltage also entails reducing the CPU frequency, and thus increasing the computation time, which is unacceptable for many applications.

One kind of application for which increasing the computation time can be acceptable is a video player. In practice, a video player only needs to meet its frame rate; any further increase in performance gives no benefit for the user. Furthermore, while decoding some frames may require the full computing power of the host platform, others are often much more simple to decode, providing intervals in which the CPU frequency can be reduced with no loss in performance. Video has thus been an attractive target for DVS algorithms [2,3,4,5]. Nevertheless, most of these approaches require either modifying the operating system process scheduler or the video codec, and thus they have not achieved wide use in practice.

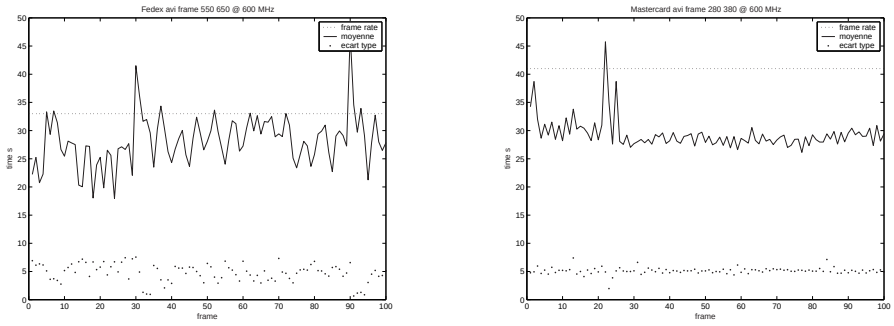
In previous work, we have proposed a DVS algorithm appropriate for use in the context of video kiosks, where we exploit the fact that the video is played over and over to identify the best CPU frequency for each frame based on its history of execution [6]. In this paper, we begin to consider how to extend this approach to the more common case, where a video is played only once, on heterogeneous platforms. For this, we propose to classify frames according to a model of their computation requirements and then to use this classification on an arbitrary platform to predict the treatment time of arbitrary frames at various CPU frequencies, based observation of the treatment time of a few frames in each class.

Dagstuhl Seminar Proceedings 07041
Power-aware Computing Systems
http://drops.dagstuhl.de/opus/volltexte/2007/1108

In the rest of this paper, we first consider the impact of using DVS in the context of video, then review our algorithm for video kiosks, and finally propose a class-based algorithm.

2 Video Applications and DVS

Video is periodic, in that frames must be displayed at a fixed rate, known as the *frame rate*. While all frames produce an image of the same size, sophisticated compression algorithms are used to reduce the size of video data, based on the observation that it is often the case that some frames are similar to nearby frames, and thus much information can be shared. Accordingly, the cost of decoding the various frames varies widely, depending on the amount of information to decode and the compression algorithms involved. For example, Figure 1 shows the treatment times of 100 frames of the Fedex and Mastercard divx videos from `www.divx.com` on a Via C3. Especially in the case of Fedex, the treatment times vary widely within the video. Furthermore, there is a significant difference in the treatment times between the two videos. Finally, in both cases, the treatment time is often well below the frame time (*i.e.*, the inverse of the frame rate), although in a few cases at the given CPU frequency it exceeds the frame time. For the frames where the treatment time is significantly below the frame time, it should be possible to reduce the CPU frequency, and thus achieve an energy savings.



(a) (b)

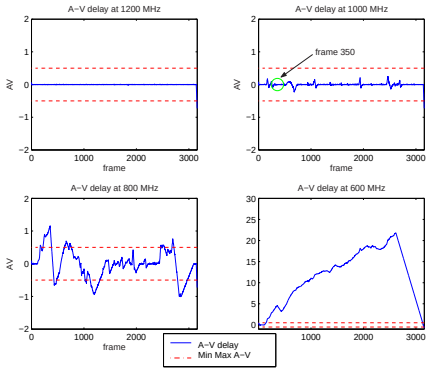
Fig. 1. Average treatment time, in milliseconds, for 100 frames of the (a) Fedex and (b) Mastercard insert codes videos on a Via C3 running at a CPU frequency of 800 MHz. The frame time is indicated by a dotted horizonal line in each graph.

The potential of DVS We now consider concretely the impact of CPU frequency scaling on video display with current codecs. For our tests we used a Pentium 4M Dell Inspiron laptop 510m with 6 CPU frequencies (1700 MHz, 1400 MHz, 1200

MHz, 1000 MHz, 800 MHz, and 600 MHz) and the video player MPlayer (www.mplayerhq.hu) running under Linux. This laptop has a battery with an average lifetime of 30 minutes. In this configuration, we play a 2 minute preview of the movie Jarhead (www.divx.com) continuously at various static CPU frequencies. For each CPU frequency, we start with a fully charged battery, and measure the battery lifetime. As shown in Figure 2(a), the battery lasts 30% longer at 1000 MHz than at the maximum CPU frequency of 1700 MHz. Thus, CPU frequency scaling is beneficial for this video.

Static CPU frequency (MHz)	Battery lifetime (min)
1700	18.2
1400	24.0
1200	25.0
1000	26.2

(a)



(b)

Fig. 2. a) Battery lifetime in minutes when playing the Jarhead video at various CPU frequencies. b) Per-frame execution time for the video Jarhead at various CPU frequencies.

Given this good result, one may wonder why we do not reduce the CPU frequency still lower, to 800 MHz or 600 MHz. Figure 2(b) measures the quality of service offered by MPlayer at various frequencies, including 800 MHz or 600 MHz, in terms of the "audio-video delay," which indicates the difference between the time of decoding the audio associated with a frame and the time of decoding its video. A value in the range from +0.5 to -0.5 is considered to be an acceptable level of quality. Our measurements show that the CPU frequencies 800 MHz or 600 MHz are too low to play some parts of the video in this range.

Even though both 800 MHz and 600 MHz are inadequate to play the entire video, Figure 2a suggests that it should be possible to use 800 MHz for some of the frames. Thus, it could be beneficial to use dynamic CPU frequency selection. In this case, the difficulty is to choose when to change the frequency to get the best result. Most existing DVS strategies can be categorized as interval-based [7,8] or task-based [9,10,11,12]. But the frequent variation in computation requirements across the different frames of a video implies that interval-based approaches tend to significantly under-approximate or over-approximate the CPU frequency required for a given frame [13,14], and task-based approaches are dif-

ficult to implement, as they often need specialized support from the operating system's process scheduler.

3 History-Based DVS

In previous work [6], we have proposed the History-based DVS (HbDVS) algorithm for minimizing the energy consumption of a video player based on execution history. This algorithm is oriented towards video kiosks, where a small set of videos are displayed repeatedly. It is based on constructing a *frequency plan*, giving the best CPU frequency for each frame based on observations of its computation requirements in previous iterations. On each iteration, we try to lower the CPU frequency used for each frame, as long as doing so does not unacceptably violate the video's frame rate.

Concretely, our algorithm works in two phases: an *adaptation phase*, which identifies the lowest possible CPU frequency for each frame, and a *post-adaptation phase*, which displays each successive frame at its CPU frequency, as indicated in the frequency plan, dynamically adjusting for any perturbation in the observed behavior. In the rest of this section, we describe these phases in more detail.

3.1 Adaptation phase

The adaptation phase is centered on the construction of the frequency plan, which concretely is an array mapping each frame to its identified ideal CPU frequency. Initially, the entries of this array are uninitialized. The adaptation phase also maintains a variable *F_Master*, which records the CPU frequency that should be used for frames for which the entry in the frequency plan is uninitialized. Initially, *F_Master* is the highest CPU frequency provided by the host platform.

The adaptation phase proceeds by iteratively displaying the video and filling in the frequency plan according to the observed behavior. On each iteration, for each frame, the player uses the CPU frequency indicated in the frequency plan, if that has been initialized, or *F_Master* otherwise. If *F_Master* is used, then after the treatment of the frame, the algorithm considers whether *F_Master* should become the permanent CPU frequency for this frame. We choose to do so if the treatment time plus any expected variance in this treatment time plus any overrun (*i.e.*, treatment time beyond the frame time) incurred for recent frames exceeds the frame time. In this situation, no lower CPU frequency is possible for this frame, and its entry in the frequency plan is set to *F_Master*. At the end of each iteration *F_Master* is lowered to the next lowest CPU frequency provided by the host platform. The adaptation phase ends after the iteration at the lowest possible CPU frequency or when all entries in the frequency plan have been initialized.

The above strategy assigns to each frame the CPU frequency at which the player first starts to fall behind the frame rate, rather than the last frequency at which it meets the frame rate, optimistically assuming that enough slack time

will be available in the treatment of later frames to absorb the overrun. Eventually, however, the overrun may accumulate to the point that the video player cannot provide the desired quality of service. In that case, the algorithm temporarily increases the CPU frequency used for the upcoming frames, until the desired quality of service is restored. At the same time, the algorithm permanently increases the stored CPU frequencies used for the same number of recent frames, so that the player will not fall so far behind on those frames in subsequent iterations. The temporary incrementing of the CPU frequency used with the upcoming frames implies that some frames might not get tested at *F_Master*. In that case, there is another iteration at the same value of *F_Master*, and thus the adaptation phase may comprise more iterations than the number of available frequencies. In practice, we have observed that at most 8 iterations at the same frequency are required [6].

3.2 Post adaptation phase

The post-adaptation phase normally just plays the video at the CPU frequency stored in the frequency plan. By construction, it is known to be possible to follow this plan and maintain the required quality of service. In practice, however, the treatment time for a given frame can vary across executions. Thus it is possible, although unlikely, that in the post-adaptation phase a sequence of frames will accumulate a delay that exceeds the quality threshold. As the adaptation phase has ensured that the video can normally be displayed according to the frequency plan with acceptable quality, the post-adaptation phase does not make further modifications to the frequency plan. Nevertheless, this phase detects such overruns and treats subsequent frames at higher frequencies within the current iteration until the delay has returned below the quality threshold.

3.3 Results

We have tested HbDVS on a variety of divx videos, on the Pentium 4M Dell Inspiron described in Section 2. In these experiments, we have observed increases in battery lifetime of up to 109% as compared to using the maximum CPU frequency of the machine, of up to 40% as compared to using the minimum CPU frequency that is adequate for the entire video, and of up to 40% as compared to using the widely available Linux utility `powernowd` (version 0.96). Further details about these results are available in our previous work [6].

4 Class-Based DVS

HbDVS gives a significant reduction in energy consumption, but because of the need to construct the frequency plan, it is limited to the case where the video is played repetitively on the same platform. We are currently considering how to extend the approach to the more common case where the video is played only once, on heterogeneous platforms that are not known in advance. The key

requirement of HbDVS is to know the treatment time of each frame at each possible CPU frequency on the host platform. In a setting where the video is only to be played once, it would incur too much overhead to test every frame. Thus, we propose a refined algorithm, Class-based DVS (CbDVS), that approximates the treatment time for various classes of frames based on the behavior observed for representatives drawn from each class.

4.1 Overview of the CbDVS algorithm

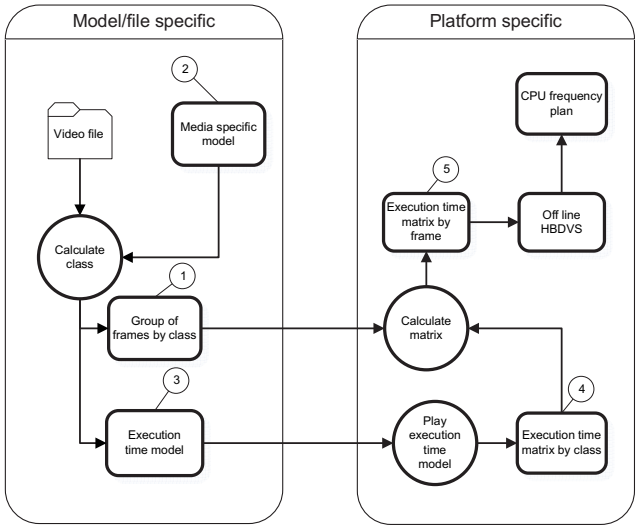


Fig. 3. The CbDVS algorithm

Figure 3 illustrates the architecture of our refined algorithm. This algorithm is divided into two phases, the first performed once, by the video provider, and the second performed on each host platform. In the first phase, the frames are categorized into a set of *classes* (1), based on a *media-specific model* (2). The model is designed so that frames within a given class have similar computational requirements. The first phase also selects representative frames from each class, as well as all of the frames on which these depend, to form an *execution-time model* (3). The categorization of frames into classes and the list of frames in the execution-time model are then distributed with the video for use in the second phase, which is carried out on each host platform. In the second phase, the video player first decodes the frames in the execution-time model at each possible CPU frequency to determine their treatment time (4). The observed treatment times are then used to construct an *execution-time matrix* for all of the frames in the video (5), indicating for each frame the anticipated treatment time at all

possible frequencies, as determined by its class. This matrix is then passed to the HbDVS algorithm, modified to obtain the treatment time for each frame from the execution-time matrix rather than from decoding the given frame. The resulting frequency plan is then used to play the video.

With this approach, the overhead on the host platform for testing frames at various frequencies is determined by the number of frames in the execution-time model, not by the length of the video. Thus, if the size of the execution-time model is sufficiently small, the approach is applicable to videos that are played only once. Its effectiveness, however, is determined by the appropriateness and stability of the class selection strategy.

4.2 Current work

The media-specific model must be platform independent, so that the analysis required to classify the frames can be performed offline by the video provider. It must also be robust enough to adequately predict the treatment times of all frames at all frequencies based on results on the platform for only a few frames. We are currently analyzing divx and H.264 video to find such a model. Previous work on MPEG-2 video in a video conference setting has found that it in this context it is adequate to consider the frame type (I, P, or B) and size. MPEG-2, however, is simpler than modern codecs such as divx and H.264, and typical movies involve more variety than a video conference. These factors imply that the treatment times may vary widely for a given frame type and size. We are thus considering what other information can be taken into account, without excessively increasing the number of frames that have to be tested at run-time before playing the video. Possible information includes the set of encoding algorithms used for each frame [15] and measures of each frame's computational requirements via performance counters [12].

References

1. Genossar, D., Shamir, N.: Intel Pentium M processor power estimation, budgeting, optimization and validation. Intel Technology Journal **7** (2003) 44–49
2. Burchard, L.O., Altenbernd, P.: Estimating decoding times of MPEG-2 video streams. In: Proceedings of International Conference on Image Processing (ICIP 00), Vancouver, Canada (2000)
3. Im, C., Ha, S.: Dynamic voltage scaling for real-time multi-task scheduling using buffers. In: Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, Washington, DC, USA (2004) 88–94
4. Maxiaguine, A., Chakraborty, S., Thiele, L.: DVS for buffer-constrained architectures with predictable QoS-energy tradeoffs. In: CODES+ISSS '05: Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, New York, NY, USA, ACM Press (2005) 111–116
5. Pouwelse, J.: Power Management for Portable Devices. PhD thesis, Delft University of Technology (2003)

6. Urunuela, R., Lawall, J., Muller, G.: Energy adaptation for multimedia information kiosks. In Min, S., Yi, W., eds.: International Conference on Embedded Software, EMSOFT'06, Seoul, South Korea, ACM (2006)
7. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'94), Berkeley, CA, USA, USENIX Association (1994) 13–24
8. Govil, K., Chan, E., Wasserman, H.: Comparing algorithms for dynamic speed-setting of a low-power CPU. In: Proceedings of the First Annual International Conference on Mobile Computing and Networking (MOBICOM '95), Berkeley, CA (1995) 13–25
9. Flautner, K., Mudge, T.N.: Vertigo: Automatic performance-setting for Linux. In: Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA (2002) 105–116
10. Lorch, J.R., Smith, A.J.: PACE: A new approach to dynamic voltage scaling. IEEE Trans. Computers **53** (2004) 856–869
11. Yuan, W., Nahrstedt, K.: Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, Bolton Landing (Lake George), New York (2003) 149–163
12. Weissel, A., Bellosa, F.: Process cruise control: Event-driven clock scaling for dynamic power management. In: Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems CASES'02, Grenoble, France (2002) 238–246
13. Grunwald, D., Levis, P., Farkas, K.I., Morrey III, C.B., Neufeld, M.: Policies for dynamic clock scheduling. In: 4th Symposium on Operating System Design and Implementation (OSDI 2000), San Diego, CA (2000) 73–86
14. Pering, T., Broderson, R.: The simulation and evaluation of dynamic voltage scaling algorithms. In: Proceedings of the 1998 International Symposium on Low Power Electronics and Design, 1998, Monterey, CA (1998) 76–81
15. Mattavelli, M., Brunetton, S.: Implementing real-time video decoding on multimedia processors by complexity prediction techniques. IEEE Transactions on Consumer Electronics **44** (1998) 760–767