Reduced energy decoding of MPEG streams*

Malena Mesarina[1], Yoshio Turner[2]

[1] University of California Los Angeles, e-mail: malena@cs.ucla.edu

[2] Hewlett-Packard Laboratories, e-mail: yoshio_turner@hp.com

Received: March 31, 2002

Abstract Long battery life and high performance multimedia decoding are competing design goals for portable appliances. For a target level of QoS, the achievable battery life can be increased by dynamically adjusting the supply voltage throughout execution. In this paper, an efficient offline scheduling algorithm is proposed for preprocessing stored MPEG audio and video streams. It computes the order and voltage settings at which the appliance's CPU decodes the frames, reducing energy consumption without violating timing or buffering constraints. Our experimental results elucidate the tradeoff between QoS and energy consumption. They demonstrate that the scheduler reduces CPU energy consumption by 19%, without any sacrifice of quality, and by nearly 50%, with only slightly reduced quality. The results also evaluate how the QoS/energy tradeoff is affected by buffering and processor speed.

1 Introduction

Energy is a critical scarce resource for portable battery-powered appliances. Such devices typically consist of a variable voltage variable speed CPU, RAM, ROM, a radio interface, a micro-display, and glue logic. In a typical handheld computer, the CPU consumes as much as 30% of the system energy to decode and display multimedia streams [23]. This component is therefore an attractive target for energy minimization. In addition, the importance of reducing CPU energy consumption will increase as progress is made in reducing power consumption of other components such as displays and radios.

* Published in ACM/Springer Multimedia Systems Journal, 9(2), August 2003, pp. 202-213.

Emerging uses for portables include multimedia applications such as video telephony, movies, and video games. These applications impose strict quality of service requirements in the form of timing constraints. Ignoring energy consumption, operating the CPU at its highest speed is best for meeting timing constraints. However, high speed operation quickly drains the batteries. Thus there is a tradeoff between reduced energy consumption and increased quality of service.

For multimedia decoding applications, the processing speed and energy consumption required for a given quality of service depends on frame timing constraints and on task complexity. Timing constraints in turn depend on frame decoding order requirements, client display buffer availability, and stream synchronization limits. Throughout the playback of a stream, the complexity of frame decoding and the time remaining to meet the next deadline vary dynamically, introducing the potential for selectively reducing processing speed to reduce energy consumption when timing constraints can be met easily.

Voltage scaling technology has the potential to exploit such variability in the ease of meeting timing constraints. By adjusting the operating voltage of the processor, the energy consumption and speed can be controlled [3]. Power regulators and variable voltage processors with response times in the microseconds range are available [4]. Fast response time makes it practical to dynamically adjust the voltage at run time.

This paper evaluates the impact of dynamic voltage scaling (DVS) on the QoS/energy tradeoff. It proposes an efficient offline scheduling algorithm that assigns voltages to tasks such that timing constraints are met and energy is minimized in a uniprocessor platform with a known number of display buffers. The algorithm assigns a single voltage per task, and each task decodes without preemption a single media frame. The algorithm also determines the order in which the tasks are decoded, subject to precedence constraints. Namely, tasks within a stream are constrained to a fixed partial order of execution. The algorithm constructs an interleaved total order of execution that does not violate the partial order of any stream.

The algorithm could be employed by a media server delivering stored (i.e., not live) media to portable appliances. The insight is to leverage the relatively abundant computing and storage at media servers in order to manage more efficiently the scarce resources of portable clients.

To obtain the schedule, the server must pre-process the media. The information needed for pre-processing is the following: the display buffer capacity, and the execution time and energy consumption of each frame at each voltage level, for each client configuration. This has practical implications.

To obtain the time and energy information, it may be necessary to probe with measurement equipment a device identical to the portable client. Although that is feasible when the population is limited to a few device types, it may not be practical for unlimited client types. The applicability of our technique can be greatly increased if future clients have built-in power monitoring. Then, instead of probing clients manually, the server could use time and energy information reported by the actual clients. The server could use these reports to incrementally build a database used to compute schedules for subsequent clients.

At playback time, the server transmits both the media streams and the decoding schedule to the clients. The bandwidth overhead of transmitting the schedule is negligible. For example, four bits per frame, say, could select the voltage/frequency of execution, and eight bits per frame could represent its relative start time compared to the prior frame. For a frame size of 720x480 with 24 bits per pixel and a compression ratio of 25, the overhead is $\frac{(4+8)*25}{720*480*24} = 0.00004$ or 0.004%. The media and the schedule can be delivered to the client using the DSM-CC protocol [17]. Prior to playback, the server may present to the client a range of choices of playback QoS together with the corresponding levels of energy consumption. With DVS, the energy consumed at desirable resolutions may be lower than that consumed with a fixed voltage system. Note that although we focus on voltage assignment in this paper, the techniques we propose could be used to schedule other assignments that affect energy consumption, for example the number of processor pipelines that are powered up.

The paper is organized as follows. Section 2 summarizes related scheduling techniques for energy minimization. Section 3 formulates the energy optimization problem by deriving timing and precedence constraints from a model of the decoding hardware. Section 4 explains the scheduling algorithm. Section 5 describes support for user interaction. Section 6 reports the experimental results. Finally Section 7 presents conclusions.

2 Related Work

Previously proposed scheduling techniques for reducing CPU energy can be classified into two categories: best-effort, and hard real-time scheduling. Best-effort schedules lack deadline constraints, whereas hard real-time schedules enforce them. For example, a number of best-effort scheduling methods to reduce energy while preserving interactive response for general purpose computing have been proposed [24,5]. Other best-effort schedulers can handle general precedence constraints either

by formulating the problem in terms of DFGs [19] or computationally expensive linear programming [14,11].

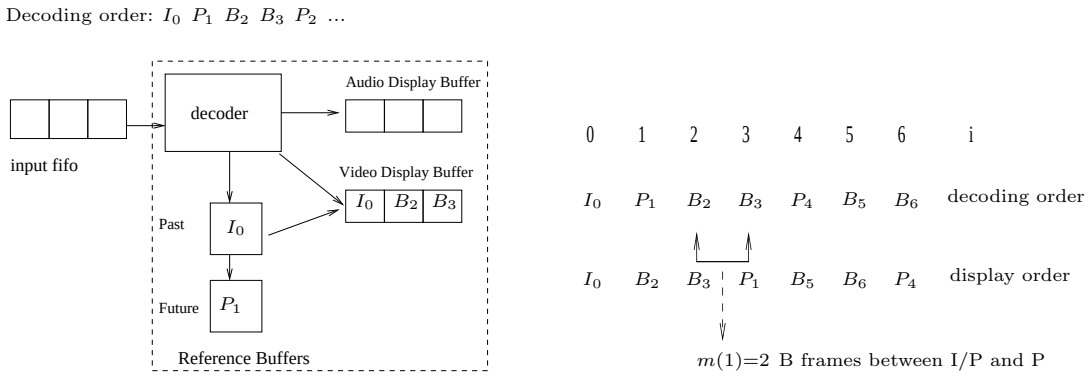
In this paper, we focus only on hard real-time schedules. For periodic tasks, an approach based on rate monotonic scheduling [15], with extensions for power reduction has been proposed [20]. Unlike our approach, that algorithm does not consider precedence constraints and assumes that the tasks are pre-emptable. A more general approach that handles arbitrary task arrival times and deadlines was presented by Yao et al [25]. That work, too, assumes pre-emptable tasks and does not include precedence constraints. Heuristics for scheduling non-preemptable tasks are proposed by Hong et al [7]. That work, however, also does not respect precedence constraints.

3 Optimization Constraints

The goal of the algorithm is to find a schedule for the portable client to decode and present MPEG movies with minimal CPU energy consumption while meeting all deadlines. In addition, the client's display buffers must not overflow. Our approach consists of two interdependent operations. One is to schedule the order of interleaving of the audio and video frame decoding tasks, subject to precedence constraints within each stream. The second operation is to assign for each frame the voltage and frequency at which it is processed.

An MPEG movie consists of a video stream and an audio stream. For quality playback, each stream must be displayed at its sampling rate (intra-stream), and the two streams must be synchronized (inter-stream). For instance, the sampling rates of video and audio can be 33 fps and 44K samples/sec [16]. The synchronization between corresponding video and audio frames must be within 80 ms to avoid perceptible degradation [22]. Flexibility in the synchronization increases the options for scheduling.

Decoding consists of three steps: input, decoding, and display. An example for video is shown in Figure 1(a) [6]. Encoded frames arrive to an *input buffer*. We assume that the input buffer masks any jitter on the input channel. Thus in our study we only schedule the hardware that is inside the dashed box in the figure. Next, the variable voltage CPU retrieves each frame from the input buffer, decodes it and places the result in either the audio or video *display buffer*. The decoded frames are removed from the display buffers by the *display hardware*, which displays audio and



(a) Decoding Hardware Organization (b) Decoding and Display Order

Fig. 1 Video Decoding

video frames simultaneously. For double buffering, each display buffer has minimum capacity of two frames. Deeper buffers increase scheduling flexibility.

The order of decoding and display can differ for video. This difference must be accounted for by the scheduling algorithm. The order differs when bidirectional predictive coded frames (B) are used. To decode a B frame, the previous (in display order) I or P frame and the next P frame are referenced. Therefore, two *reference buffers* are dedicated to store the corresponding I and P reference frames.

Each frame can potentially be decoded at a different voltage level. To determine the correct setting, the scheduling algorithm needs to know, for each frame, the energy consumption and execution time at each voltage setting. One way to gather that information in advance of scheduling is to probe with measurement equipment a device that is identical to the portable client.

The parameters used in the algorithm are listed in Table 1. Using that notation, we next derive the values of the display, deadline, and minimum start time parameters.

For video, the mapping $d(i)$ from decode order $(\tau_0, \tau_1, \tau_2, \dots)$ to display order $(\tau_{d(0)}, \tau_{d(1)}, \tau_{d(2)}, \dots)$ is as follows:

$$d(i) = \begin{cases} i-1 & \text{If } \tau_i \text{ is a B-frame} \\ \\ i+m(i) & \text{If } \tau_i \text{ is a P-frame or I-frame} \end{cases} \quad (1)$$

where $m(i)$ is the number of consecutive B frames immediately after τ_i in decode order. An example of the difference between decode and display order is shown in Figure 1(b).

The display time D_i of video task τ_i is $t_0 + T_s \cdot d(i)$. Similarly, the display time D'_j of audio task τ'_j is $t_0 + T'_s \cdot j + K$. Note that the video stream begins no earlier than the audio stream because video ahead of audio is tolerated better than the reverse [22].

Each frame must be decoded before its display time. In addition, a frame used as a forward reference frame (i.e. P frames and some I frames) must be decoded before the display time of the B frame that follows it immediately in decode order. Therefore, the decoding deadline R_i for task τ_i is the following:

$$R_i = \begin{cases} & \text{If } \tau_i \text{ is a B-frame, or} \\ D_i & (\tau_i \text{ is an I-frame and} \\ & \tau_{i+1} \text{ is an I- or P-frame}) \\ & \\ & \text{If } \tau_i \text{ is P frame, or} \\ D_{i+1} & (\tau_i \text{ is an I-frame and} \\ & \tau_{i+1} \text{ is a B-frame}) \end{cases} \tag{2}$$

The minimum start time M_i for the decoding of video frame τ_i is determined by the fixed decoding order within a stream and by the video display buffer capacity. For those P and I frames that are decoded into the reference buffers instead of the display buffers, the minimum start times are determined only by the fixed decode order. Thus for those frames, $M_i = M_{i-1}$. Otherwise, for all other frames that do not satisfy this condition, the minimum start time is the maximum of M_{i-1} and the time when decoding gets as far ahead of the display process as possible. That limit is determined by the size of the display buffer. Therefore M_i equals the maximum of M_{i-1} and the display time of

Table 1 Algorithm Parameters

- b, b' number of extra video and audio display buffers (example: $b = 1$ for double buffering for video).
- D_i, D'_j display time for video frame τ_i and audio frame τ'_j.
- E total energy consumption.
- E_{idle} the energy consumed in one time unit in idle mode.
- $E_{i,l}$ the energy spent by video task τ_i at voltage level l.
- $E'_{j,l}$ the energy spent by audio task τ'_j at voltage level l.
- K synchronization skew between the end of display of a video and audio frame ($0 \leq K \leq K_{max}$).
- M_i, M'_j minimum start times for video frame τ_i and audio frame τ'_j.
- N, N' highest numbered video and audio frames.
- R_i, R'_j decoding deadline for video frame τ_i and audio frame τ'_j.
- T_s, T'_s sample time (normalized to 1 ms units of time) for video and audio frames.
- $T_{i,l}$ the execution time of video task τ_i at voltage level l.
- $T'_{j,l}$ the execution time of audio task τ'_j at voltage level l.
- t_0 is the time of display of the first video frame
- τ_i frame i of the video stream, $i = 0, 1, \ldots, N-1$.
- τ'_j frame j of the audio stream, $j = 0, 1, \ldots, N'-1$.
- v_l the supply voltage for $l = 0, \ldots, l_{max}$ number of discrete voltages.

the frame which is b ahead of τ_i in display order. That frame is $\tau_{d^{-1}(d(i)-b)}$. For audio task τ'_j, the minimum start time M'_j depends only on the display buffer occupancy. Thus:

$$M'_j = D_{j-b'}$$

$$M_i = \begin{cases} & \text{If } (\tau_i \text{ is I/P } \& \ \tau_{i-1} \text{ is B}) \\ M_{i-1} & \text{or } (\tau_i \text{ is P } \& \ \tau_{i-1} \text{ is I}) \\ & \text{or } (\tau_i \text{ is I } \& \ \tau_{i-1} \text{ is P}) \\ & \\ & \text{If } (\tau_i \text{ is I } \& \ \tau_{i-1} \text{ is I}) \\ max(M_{i-1}, D_{d^{-1}(d(i)-b)}) & \text{or } (\tau_i \text{ is P } \& \ \tau_{i-1} \text{ is P}) \\ & \text{or } (\tau_i \text{ is B}) \\ & \\ 0 & \text{If i} = 0 \end{cases} \tag{3}$$

The scheduling problem is as follows:

> Find a voltage setting (V_i or V'_j) for each task (τ_i or τ'_j) and a non-preemptive execution schedule such that the total energy consumption
>
> $$E = \sum_{i=0}^{N-1} E_{i,V_i} + \sum_{j=0}^{N'-1} E'_{j,V'_j} \tag{4}$$
>
> is minimized subject to ordering and timing constraints. Frames in a stream must be processed in decode order, and their processing must obey the minimum start times and deadline constraints.

4 Scheduling Algorithm

To be efficient, the scheduling algorithm must implicitly rule out a large number of orderings without explicitly examining them. The key observation that enables enough orderings to be pruned is that many schedules share identical dependences at particular intermediate points in their executions. Specifically, suppose that a number of feasible schedules all begin by executing (in various orders and voltage settings) exactly i video frame tasks and j audio frame tasks. Suppose each such schedule finishes processing the i video and j audio frame tasks at exactly the same time T_{split}. After time

T_{split}, all the schedules have the same remaining work and same time to meet future deadlines. Therefore, the scheduling of tasks after T_{split} is independent of the differences in the schedules prior to time T_{split}.

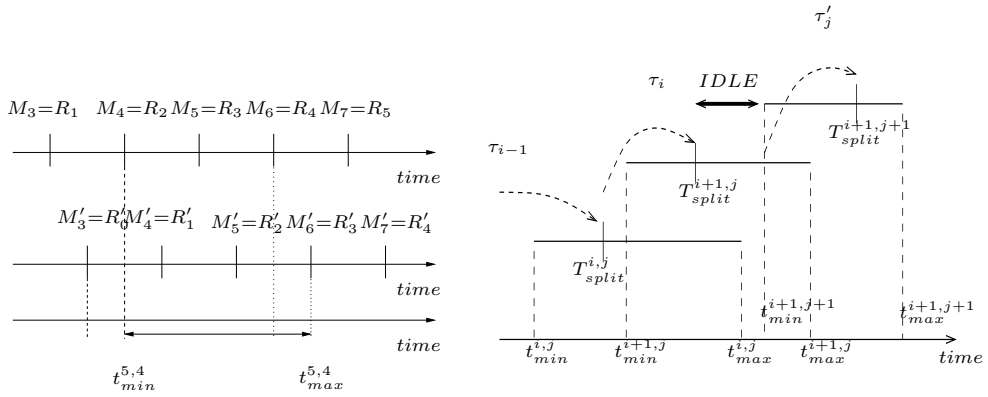
Conceptually, we can split each schedule above into two independent "subschedules": the *initial subschedule* prior to T_{split}, and the *subsequent subschedule* after T_{split}. A complete energy optimal schedule can be constructed by concatenating any minimum energy initial subschedule to any minimum energy subsequent subschedule.

An early development in the theory of real-time task scheduling that used a similar concept was a dynamic programming problem formulated by Lawler and Moore [13]. Their algorithm finds a non-preemptive schedule that minimizes an arbitrary non-decreasing cost function under task deadline constraints. Our optimization problem can be partially mapped to that approach, with two differences. A difference that requires only straightforward modifications is that our tasks have minimum start time constraints. The more significant difference is that we support multiple synchronized streams of tasks, which requires a search of the feasible interleaved orderings of tasks of multiple streams. One way to support multiple streams is to add dimensions to the dynamic programming formulation. However, that would increase the computational complexity by a factor of n for each new stream, where n is the number of tasks in a stream. For long streams or for many streams, that cost is unacceptable. We show below how to avoid it by exploiting knowledge about the system's memory resources. With this approach, the display buffer size b bounds the number of task orderings to consider. It also constrains the number of possible task completion times to be within a small *time window*.

We define the time windows $w_{i,j}$ in which i and j are the number of tasks in each stream that have executed in a subschedule. The range of times $[t^{i,j}_{min}, t^{i,j}_{max}]$ within window $w_{i,j}$ includes the set of all permissible *completion* times of the last task executed (either τ_{i-1} or τ'_{j-1}). Let t be an offset into the time window $w_{i,j}$ (i.e. $0 \leq t \leq t^{i,j}_{max} - t^{i,j}_{min}$). The lower bound $t^{i,j}_{min}$ for $w_{i,j}$ is the earliest time when *both* τ_{i-1} and τ'_{j-1} are complete. To assure that both are complete after $t^{i,j}_{min}$, its value is the maximum of the minimum start times of both tasks. Both tasks are guaranteed to be complete by time $t^{i,j}_{max}$, which is the latest deadline of both tasks. Thus

$$t^{i,j}_{min} = max(M_{i-1}, M'_{j-1}) \tag{5}$$

$$t^{i,j}_{max} = max(R_{i-1}, R'_{j-1}) \tag{6}$$



(a) Example: time window bounds for $w_{5,4}$. Example minimum start times and deadlines are shown for each stream. Assume for simplicity that all video frames are I or B frames, thus display time equals decoding deadline, just as for audio. Buffer sizes are $b = 2$ and $b' = 3$.

(b) Example: windows of adjacent vertices. Windows $w_{i,j}$, $w_{i+1,j}$, $w_{i+1,j+1}$ are shown. Note that task execution can be interrupted by idle periods.

Fig. 2 Time Windows and Task Execution

As an example, Figure 2(a) shows a time window $w_{5,4}$. In the example, $t^{5,4}_{min} = M_4$ because $M_4 > M'_3$. Also, $t^{5,4}_{max} = R'_3$ because $R'_3 > R_4$. It can be shown that an upper bound on the length $(t^{i,j}_{max} - t^{i,j}_{min})$ of any time window is the product of the sampling time and the number of display buffers for one stream.

The range of values or i and j is given by the following condition:

$$i, j \text{ such that } t^{i,j}_{min} < min(R_i, R'_j) \tag{7}$$

If i and j violate this condition, then the time window starts too late to complete one or both τ_i or τ'_j, and the time window is not considered by the algorithm.

To understand how the condition $t^{i,j}_{min} < min(R_i, R'_j)$ limits the algorithm's complexity by limiting the combinations of i and j values, consider for simplicity the case of equal sampling times for the two streams: $T_s = T'_s$. Also suppose the video stream does not contain reference frames. Then, some algebra reveals that the condition is satisfied by $j = 1, 2, \ldots, N'$ and $i \in [d^{-1}(j - b' + K/T_s), d^{-1}(j + b + K/T_s)]$. The intuition is as follows. First, suppose the skew K equals zero. In that case, at any time the video and audio frames being displayed have the same frame number in display order. When one display buffer is full (say, video) and the other buffer (say, audio) is

empty, then the next (video) frame to be decoded has to wait. Thus a video frame with decode order number i cannot begin decoding until the video frame that is b ahead of it in display order enters display, which happens at the same time for the audio frame with the same display order number (which is $j = d(i) - b$, hence $i = d^{-1}(j + b)$). Thus video frames numbered $d^{-1}(j + b)$ and higher cannot complete prior to decoding audio frame j. Similarly, the audio frame with decode order number j cannot begin decoding until the audio frame that is b' ahead of it in display order enters display, which happens at the same time for the video frame with the same display order number (which is $d(i) = j - b'$, hence $i = d^{-1}(j - b')$). Thus audio frames j and above cannot execute prior to completing video frame $d^{-1}(j - b')$. As the skew K increases, the deadlines and minimum start times of the audio tasks are delayed relative to their corresponding video tasks. That decreases the task number of the next audio frame that can execute at each point in time without affecting the task number of the next video frame that can execute. Therefore the allowed value of i is increased by K/T_s relative to j, which explains the shift by K/T_s in the range for i. The condition $t_{min}^{i,j} < min(R_i, R'_j)$ considerably reduces the number of (i, j) subschedules that need to be checked. For the simple case above with zero skew and adding the assumption that $b = b'$, the maximum number of (i, j) sub-schedules that need to be considered is equal to the number of frames in one stream times the size of the buffer, i.e., $N * b$.

We now describe the iterative steps of the scheduling algorithm, which is listed in pseudocode in Figure 3. The scheduling process can be visualized as the traversal of a graph. Each vertex $V_{i,j}$ represents the set of energy optimal initial subschedules that consist of exactly i video and j audio frame tasks. Vertex $V_{i,j}$ is associated with time window $w_{i,j}$, the range of feasible completion times T_{split} of initial subschedules. An edge from vertex $V_{i,j}$ to vertex $V_{i+1,j}$ represents the execution of video frame task τ_i immediately after an initial subschedule. Execution of τ'_j is similarly represented by an edge from $V_{i,j}$ to $V_{i,j+1}$. Figure 2(b) shows a possible flow of execution of tasks τ_{i-1}, τ_i and τ'_j. Note the idle time between the completion of τ_i and the start of τ'_j. τ'_j is delayed until its minimum start time ($M'_j = t_{min}^{i+1,j+1}$).

For initialization, the display time t_0 of video frame τ_0 is set to the time when all the display buffers first become full as a result of executing tasks at lowest voltage prior to any display. The algorithm next creates (line 14) and visits vertices one "row" at a time, in each row covering all the values of i for a fixed value of j. A vertex is created if its subscripts satisfy the constraint in Equation 7: $t_{min}^{i,j} < min(R_i, R'_j)$. At vertex $V_{i,j}$, the algorithm iterates through the time window

1: Suppose t_0 is the display time of τ_0. Then,
2:
3: $t^{i,j}_{max} = max(R_{i-1}, R'_{j-1})$
4: $t^{i,j}_{min} = max(M_{i-1}, M'_{j-1})$
5: $t_0 = \sum_{j=0}^{b'} T'_{j,0} + \sum_{i=0}^{b+1} T_{i,0}$ (Assumes reference buffers can be filled)
6:
7: Procedure SCHEDULE
8: **for** $K = 0$ to K_{max} **do**
9: $i = 1$, $j = 0$: create vertex $V_{1,0}$ and vertex $V_{0,1}$
10: record execution of τ_0 in $V_{1,0}$
11: record execution of τ'_0 in $V_{0,1}$
12: **repeat**
13: **repeat**
14: Conditionally generate vertices $V_{i+1,j}$ and $V_{i,j+1}$
15: **for** $t = 0$ to $(t^{i,j}_{max} - t^{i,j}_{min})$ **do**
16: **if** $V_{i+1,j}$ exists and an initial subschedule has been recorded for time window offset t **then**
17: Consider execution of τ_i (all voltages) after the initial subschedule, such that τ_i meets timing constraints
18: Record new subschedule in $V_{i+1,j}$ if it has lower energy than found so far at the same offset of $V_{i+1,j}$
19: **end if**
20: repeat steps 16-18 for $V_{i,j+1}$ and τ'_j
21: **end for**
22: $i++$
23: **until** $i > N$ or vertex $V_{i+1,j}$ does not exist
24: $j++$ /* next row */
25: i = lowest numbered *col* such that $V_{col,j}$ exists
26: **until** $j > N'$
27: **if** a new optimal schedule found **then**
28: keep it
29: **end if**
30: delete the graph
31: **end for**
32: report the optimal schedule

Fig. 3 Scheduling Algorithm

(lines 15-21). At each T_{split}, it considers what would happen if task τ_i or task τ'_j were to execute next at each voltage level. Execution of a task at a voltage that causes it to miss its deadline is discarded. For each point in the time window, each proposed next task execution is appended to the best initial subschedule. If the resulting longer subschedule has lower energy than that recorded in the next vertex, then the record in that vertex is overwritten (line 18).

Once the algorithm reaches vertex (N, N'), it scans all the entries in the time window of (N, N') to find the schedule that uses the least energy. To extract the best schedule, the algorithm traces backward through the graph, building a stack of task numbers, start times, and voltage settings.

The algorithm's outer repeat loop executes for all possible settings of the skew between streams (K). K ranges from 0 to K_{max}.

To derive the computational complexity of the algorithm, we consider the major steps it must complete for two streams. At each vertex, it performs an O(1) operation for each of the O($T_s * b$) values in the time window. For the O(K_{max}) values of K, the algorithm visits O($N * b$) vertices. Therefore, the algorithm has complexity O($K_{max} * T_s * N * b^2$).

5 Support for User Interaction

Interactive user operations such as forwarding and rewinding can be supported by augmenting the schedule information with buffer state information. The goal is to allow the user to jump to any point in the presentation while continuing to follow the appropriate reduced energy schedule. To do this, the state of the display buffer after forwarding or rewinding to the desired point in the media should correspond to the state of the buffer had the media been presented continously from the beginning. Three sources of information are precomputed to support resuming playback from an arbitrary frame numbered n. First the display buffer at the client should be loaded with those frames that would have been already enqueued in the display buffer at the time of display of frame n. This information enables the client to update the buffers to a state that follows the schedule at the new point of presentation. Second, the relative difference between the display time D_n of frame n and the time decoding begins for the next frame should be computed prior to playback. This tells the client when to start decoding a new frame after playback resumes. Third, the time skew between the display of the video frame and audio frame at the new point of presentation is computed. This allows the video and audio streams to be re-synchronized.

To provide the first source of information, the first step is to create two helper tables, $T1$ and $T2$, each of size N (number of video frames). Equivalent sets of tables must be created for audio and video. Here we focus on the case of video. Tables $T1$ and $T2$ are indexed by frame decode number. Each table entry $T1_i$ records the number of frames decoded (or partially decoded) by the time frame i enters display. Each table entry $T2_i$ records the number of frames displayed by the time frame i enters display. Tables $T1$ and $T2$ can be filled by a straightforward processing of the DVS schedule. Using tables $T1$ and $T2$, we then construct a table S, indexed by frame decode number, in which

each entry S_i contains the list of frames that should be in the display buffer when frame i enters display.

Table S is computed from tables $T1$ and $T2$ as follows. Each entry S_i corresponds to the set difference between two sets: the set of frames that have been decoded, and the set of frames that have been displayed, by the time frame i enters display. We construct that set difference by generating and comparing the sets of frames that correspond to the values recorded in $T1_i$ and $T2_i$. To do that efficiently, we introduce another table, $G(d(i))$, which takes the display number of a frame as input and produces the highest frame decode number such that frames with equal or lower decode number have display numbers less than the input display number. Thus only frames with decode numbers in the range $G(T2_i), \ldots, T1_i - 1$ need to be considered for possible membership in the set difference. The number of such frames is O(b) (the number of frames to consider would be O(N) if we did not use $G()$). The table $G(d(i))$ can be calculated in a onetime O(N) operation. Thus the complexity of generating the table S is O($N * b$).

The second source of information is the relative difference between the display time D_n of frame n and the decode time of the next frame to be enqueued. The decode number $next$ and start time ST_{next} of the next frame are readily obtained from the schedule. The relative start time provided to the client is $ST_{next} - D_n$.

The third source of information, the skew between the video and audio frames, can be obtained by taking the difference between the display times of the frame n and the next frame of the audio stream to be displayed.

When the user wishes to forward or rewind to a particular point in time of the presentation, the client sends a request packet to the server specifying the time point in the presentation to which it would like to skip. Since a GOP forms an independent group of pictures, the server locates the GOP that falls within this time, and identifies the display number of the first I frame of this GOP. With this information, it can then access the S table to find the list of frames to send to the client to populate the display buffer. It also computes the time information mentioned above. Finally, it sends the frame and time information to the client, which resumes playback using those parameters.

The scheduling algorithm in Section 4 has higher time complexity than the operations described here and hence performance is dominated by that previous algorithm. Note that the techniques described here, which support jumping to different points in the same schedule, can also be used to

jump to an arbitrary point in a different schedule. That ability enables users to select at any time a new desirable operating point in the tradeoff between energy and quality.

6 Performance Evaluation

Our initial goal for evaluation is to quantify the tradeoff between quality and energy savings. Our hope is to improve the tradeoff through the use of dynamic voltage scaling (DVS), which exploits variability in the execution times of frames. Our approach aims to provide insight into the design space by studying the impact on quality and energy of two design parameters for the client hardware: processor frequency, and display buffer capacity.

6.1 Experimental setup

We measured decoding times on two machines, each having a fixed processor frequency and voltage: a Pentium III at frequency $F_{hi} = 500$ MHz and voltage $V_{hi} = 1.9$V, and a Pentium II at frequency $F_{hi} = 300$ MHz and voltage $V_{hi} = 1.7$. Execution time per frame was measured for a 1000-frame segment of the movie *Batman Forever* in MPEG2 format. We obtained the execution time ($T_{i,hi}$) for frame i by instrumenting a software decoder to measure elapsed time per frame. In the case of video we used the livid MPEG2 software decoder, which uses MMX operations [1]. For audio we used the livid AC3 software decoder [1].

We wish to model client platforms, each having two voltage (V_{lo}, V_{hi}) and frequency (F_{lo}, F_{hi}) settings. We extrapolated the frame execution time measurements from the fixed voltage machines in order to obtain the task energy-time tables for the DVS scheduling algorithm. We made three assumptions for the extrapolation. First, frequency is inversely proportional to gate delay [7]. Second, the number of cycles per frame remains constant at any processor frequency. Here we assume that stalls due to the memory hierarchy structure are negligible [8]. Third, for a given voltage setting, power dissipation is assumed constant. Thus energy is proportional to execution time. This is a reasonable assumption since studies have shown that the power per instruction remains fairly constant in the absence of non-ideal effects such as pipeline stalls [21].

The data sheets for the Pentium II and Pentium III give the range of core voltages at which these processors can operate [9,10]. We derived the frequency at which the processor would operate at the lowest voltage. Using assumption one, frequency at some reference voltage is $F_{ref} = 1/tp_{ref} * k$.

Propagation delay is $tp_{ref} = \gamma * V_{ref}/(V_{ref} - V_t)^2$, where γ is a constant that depends on technology and total capacitance and V_t is the threshold voltage [18]. Taking the ratio, F_{lo}/F_{hi}, and solving for F_{lo},

$$F_{lo} = F_{hi} * \frac{V_{hi}/(V_{hi} - V_t)^2}{V_{lo}/(V_{lo} - V_t)^2} \tag{8}$$

Using assumption two, $T_{i,lo} = cycles/F_{lo} = F_{hi} * T_{i,hi}/F_{lo}$.

Using assumption three, energy per frame at the high voltage is $E_{i,hi} = P_{hi} * T_{i,hi}$, where P_{hi} and $T_{i,hi}$ are respectively the dynamic power and execution time of frame i at V_{hi}. The dynamic power is given by $P = \alpha * C_l * V_{dd}^2 * f$, where $\alpha * C_l$, is the effective switching capacitance of the processor, V_{dd} is the supply voltage and f is the processor's frequency [18]. We normalize power P_{hi} to 1 when the processor operates at V_{hi}. To extrapolate to operation at a lower voltage V_{lo}, we derive power P_{lo} as a function of the previous parameters. Taking the ratio, P_{hi}/P_{lo}, and solving for P_{lo}, we get,

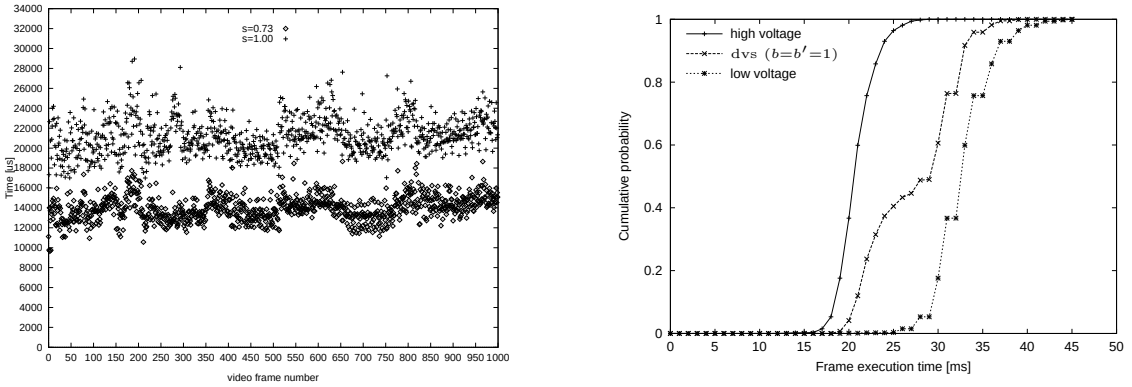
$$P_{lo} = P_{hi} * (F_{lo}/F_{hi}) * (V_{lo}/V_{hi})^2 \tag{9}$$

Thus $E_{i,lo} = P_{lo} * T_{i,lo}$.

There are many choices for metric of quality. For our experiments, we chose to use the scale factor $s = \frac{resolution\ of\ frame}{max\ frame\ resolution}$ as the metric of quality, where we define *resolution* as the product of the X and Y dimensions of the frame (keeping the aspect ratio approximately constant). Despite our use of scale factor as a convenient way to represent different resolutions, we do *not* mean to imply that there is a linear relationship between frame resolution and quality. A scale factor is assumed to have better quality than any lower one, but otherwise it is left to the user to assess the relative desirability of different scale factors (resolutions). We expect that most users would experience high quality by operating close to scale factor $s = 1$. The maximum frame resolution of the movie *Batman* is 720x480 = 354,600. To obtain lower resolution qualities of the movie, we used the FlaskMPEG encoder [2] to recode the movie to lower resolutions such that the scale factor varies between 0 and 1. To maintain the aspect ratio of the original picture (720/480 = 1.5), we only recoded to frame resolutions that kept this ratio constant.

6.2 Frame execution times

Dynamic voltage scaling has the potential to reduce energy consumption by exploiting variability in the workload. We measured the variability in frame execution time for audio and video. For audio,



(a) Decoding time vs frame number

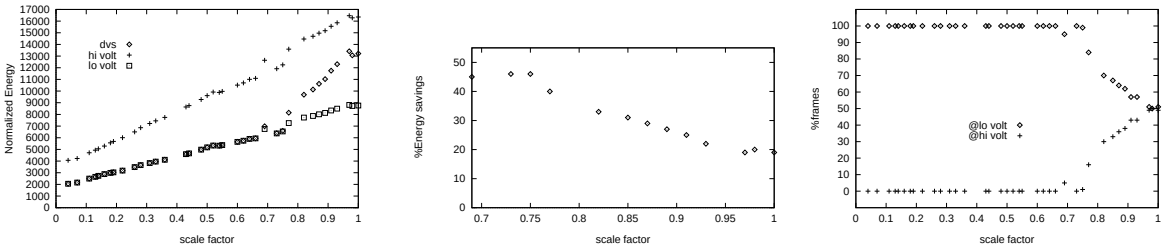
(b) Decoding time distributions

Fig. 4 Variation in video frame decoding time

little variability was found; all frames took approximately 3 ms to decode. For video, more variability is expected because I, P, and B frames require different types of processing. Figure 4(a) shows the measured video frame execution times for scale factors 0.73 and 1. Execution time varies significantly for different frames. The ratio of the maximum to the minimum execution time is 1.33, a result that agrees with results reported recently by Hughes et al [8]. Figure 4(b) shows the cumulative probability distributions of video frame decoding for scale factor 1 and three cases: decoding all frames at fixed high voltage, decoding all frames at fixed low voltage, and decoding using DVS. Decoding with DVS takes samples from both the low and the high voltage distributions and thus exhibits more variation than either of the other two.

6.3 Energy savings vs picture quality

Our goal is to explore the relationship between levels of picture quality (QoS) and energy consumption. We expect the energy consumption of DVS to increase with higher QoS, since DVS would have to speed up (using the higher voltage setting) the decoding of more frames in order to meet the display deadlines. We show how much energy can be saved if voltage-frequency per frame are scheduled by the DVS algorithm as opposed to decoding all frames at the fixed highest voltage. Our experiments start with the following client hardware configuration: the Pentium III processor, two core voltage settings, $V_{hi} = 1.9V@500MHz$ and $V_{lo} = 1.4V@316MHz$ and one video ($b = 1$) and



(a) Total energy

(b) Percentage energy savings vs scale factor

(c) Percentage of frames at high and low voltage vs scale factor

Fig. 5 Pentium III: $V_{hi} = 1.9$V, $V_{lo} = 1.4$V, $b = 1$, $b' = 1$

one audio buffer ($b' = 1$). To reveal the energy savings delivered by the DVS algorithm, we plot normalized energy vs. scale factor (QoS) in Figure 5(a).

The *dvs* curve shows energy consumption incurred by the DVS algorithm. The *hi volt* curve shows energy consumption when all frames are decoded at the highest voltage (highest speed). And the *lo volt* curve shows energy consumption when all frames are decoded at the lowest voltage (lowest speed). Of the three curves, *dvs* and *hi volt* guarantee deadlines, but *lo volt* does not (at points where *dvs* uses more energy). From Figure 5(a), we draw several conclusions. The Pentium III processor can decode most of the low quality streams (< 0.69) entirely at the lowest voltage, and thus DVS has no impact in that range. At scale factor 0.69, not all frames can be decoded at the lowest voltage and meet the deadlines. Above 0.69, there is a sudden increase in energy used by DVS. Despite this increase, the DVS algorithm decodes streams at lower energy than at the fixed higher voltage setting.

Figure 5(b) shows the percent energy savings achieved by DVS versus decoding all frames at the highest voltage, at the same scale factors. Even at the highest quality (scale = 1), DVS delivers 19% savings in energy. Note that savings between 40% and 50% are achieved with only modest decrease in quality. The percent savings decreases with higher quality because more frames must be decoded at the higher voltage. This is shown in Figure 5(c), where we show the percentage of frames decoded with DVS at the high and low voltage vs scale factor.

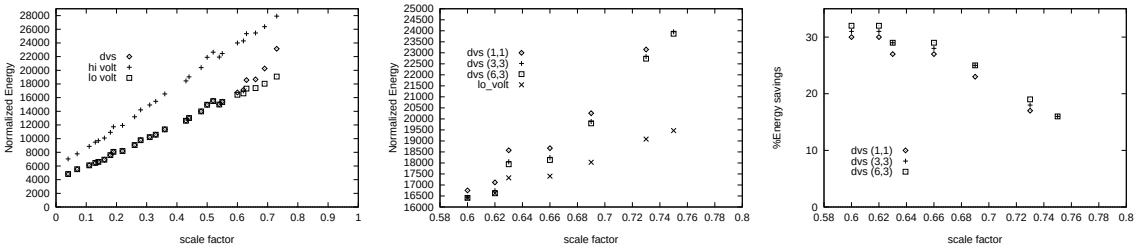
6.4 Hardware parameters: display buffers and processor speed

The results above all used a single model of the client hardware. We now evaluate the impact of changing two important client hardware parameters: the display buffer capacity, and the processor frequency. For the design of resource constrained portable devices, it may be an acceptable expense to provide a small number of display buffer slots, where each slot is large enough to store a fully decoded frame. Increasing the number of buffer slots increases the flexibility of the DVS algorithm in scheduling the frame decoding start times. That may lead to lower energy schedules.

The impact of providing a larger display buffer depends in part on the speed rating of the CPU that is filling it. Frames are placed in the display buffer at a variable rate which depends on the processor speed and the variable number of cycles required to decode each frame. The display buffer empties at a constant rate which is equal to the frame sampling rate and is thus independent of the CPU speed.

For our initial experiments we increased the number of video and audio buffers in the following pair sequences: (1,1), (2,1), (3,1), (2,2), (3,2), (3,3) and (6,3), where the first and second pair elements represent the video (b) and audio (b') buffers respectively. For the Pentium III processor we used, increasing the number of display buffers resulted in small improvements in energy savings (less than 2% improvement at each scale factor). As shown in Figure 5, the Pentium III is fast enough to decode the media clip at maximum resolution without missing deadlines even when half the frames are decoded at low voltage. As a result, it is possible that our initial experiments did not reveal the behavior of the system under the most challenging conditions for the CPU. Therefore, for all the following results we evaluate the impact of adding buffers to a slower Pentium II-based configuration with two core voltage settings: $V_{hi} = 1.7V@300MHz$ and $V_{lo} = 1.4V@225MHz$. With this slower processor, the DVS scheduling algorithm cannot find a feasible decoding schedule for the maximum video frame resolution.

6.4.1 Pentium II: impact of display buffer capacity We start with the $b = 1$ and $b' = 1$ buffer combination and plot the total energy consumption with the Pentium II in Figure 6(a), just as we did with the Pentium III in Section 6.3. For scale factors 0.73 and higher, the DVS algorithm could not find a schedule even when decoding all frames at the highest voltage. Thus the QoS window for which DVS improves the energy-QoS tradeoff is smaller with this hardware configuration, ranging between 0.6 and 0.73.



(a) Total energy: $b = 1$, $b' = 1$

(b) QoS window shifts with buffering

(c) Percentage energy savings vs scale factor (all buffer combinations)

Fig. 6 Pentium II and buffering: $V_{hi} = 1.7$V, $V_{lo} = 1.4$V

We next increase the number of buffers to increase scheduling flexibility. Figure 6(b) shows the energy consumption incurred with the DVS algorithm for different video and audio buffer combinations. The primary observation is that increasing the number of buffers does not significantly improve energy consumption, hence small buffers are sufficient to realize most of the benefits of our DVS algorithm. We examine this issue further in Section 6.4.3. The results also show that extra buffers enable slightly higher quality video to be decoded without missing deadlines. For the (1,1) buffer combination, the QoS window ranges between 0.6 and 0.73. But for the (3,3) and (6,3) combination, the QoS window ranges between 0.62 and 0.75. With more buffers, the DVS algorithm can decode some frames earlier. Having more time for decoding, it can then decode all frames, at $s = 0.6$, at the lowest voltage. Similarly, the algorithm can find an energy efficient schedule at $s = 0.75$. Thus at 0.75 in Figure 6(c), the algorithm saves 16% in energy.

6.4.2 Dynamics of display buffer occupancy We next examine how the display buffers are used over time with both a fixed voltage schedule and DVS scheduling. We examine the case of the maximum scale factor that can be decoded using DVS (i.e. $s = 0.75$). We vary the buffer configuration from (2,2) to (7,7), where for each configuration there is an equal number of slots for audio and video.

Figure 7 shows the distributions of video and audio display buffer occupancy. The results show that all the buffers are used to some degree in every case. For the fixed voltage schedules, the video buffers are more heavily utilized than the audio buffers. The audio buffer drains while the processor is busy executing the much longer video tasks. For the DVS schedules, buffer occupancy is lower than with a fixed voltage schedule. For example, the video buffer is full nearly always with a fixed voltage

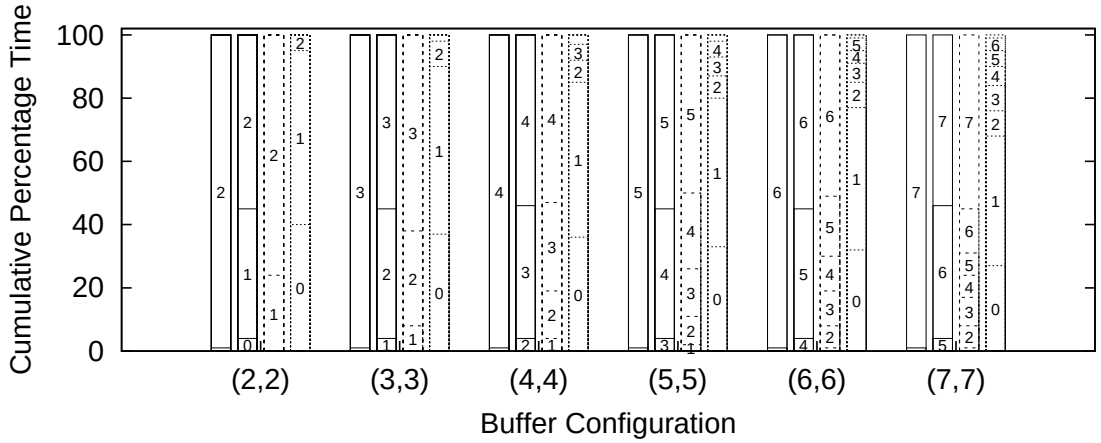
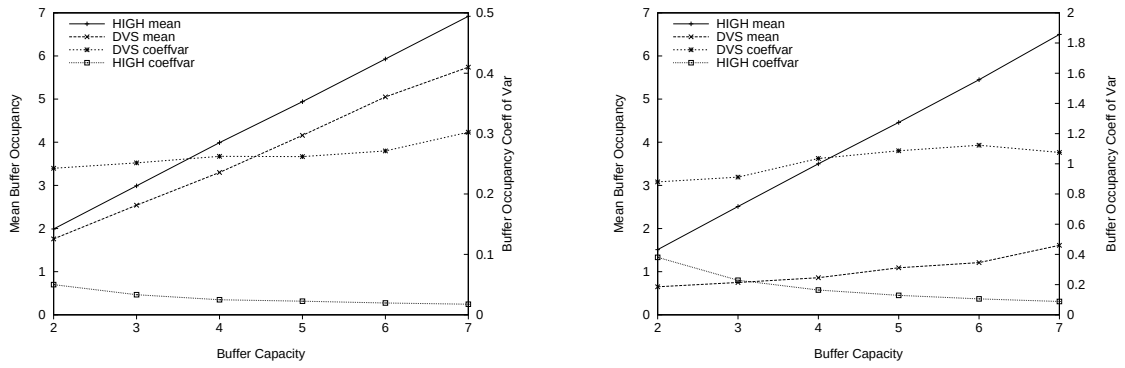


Fig. 7 Buffer occupancy probability distributions. For each buffer configuration, four distributions are shown (from left to right: video buffers with fixed high voltage schedule, audio buffers with fixed high voltage schedule, video buffers with DVS, audio buffers with DVS).



(a) Video display buffer occupancy

(b) Audio display buffer occupancy

Fig. 8 Display buffer occupancy: mean and coefficient of variation (standard deviation divided by mean)

schedule and full between 50% and 80% of the time with DVS. The audio buffer is full or next to full nearly all the time with a fixed voltage schedule. With DVS, it becomes nearly always empty or next to empty. DVS reduces the buffer occupancy compared to a fixed high voltage schedule because it selectively slows down the decoding of frames, thus reducing the rate at which the buffer is filled.

The mean and coefficient of variation of the distributions in Figure 7 are shown in Figure 8. The mean occupancy increases approximately linearly with the buffer capacity, but DVS exhibits a smaller slope than the fixed voltage case. With a fixed voltage schedule, the buffer is nearly always full and thus exhibits much less variation in buffer occupancy than DVS.

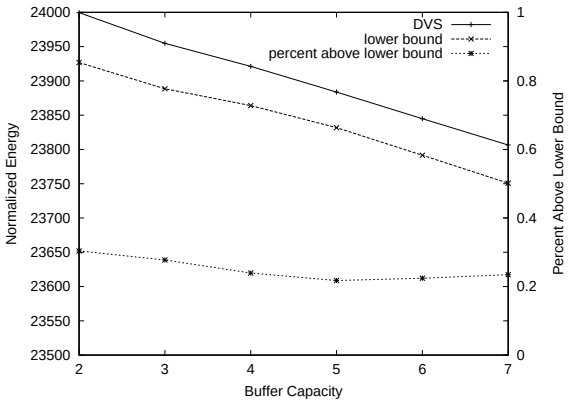


Fig. 9 DVS compared to lower bound: Pentium II, scale factor 0.75, buffer capacity $= b = b'$

6.4.3 DVS vs lower bound on energy consumption As described in Section 6.4.1, the energy savings from adding display buffers appears to be small. Hence low capacity buffers are sufficient to obtain most of the benefits from our DVS algorithm, at least for the workload we examine here. To determine whether the benefits of our algorithm are limited by constraints such as deadlines, precedence constraints, and minimum start times (which are related to the buffer capacity), we next derive a lower bound on energy consumption that ignores those constraints and compare the bound to the energy consumption with our DVS algorithm. The only timing constraint used to obtain the lower bound is that all the video and audio frames must be decoded sometime prior to the display of the last frame.

The lower bound is computed as follows. The first several frames of each stream are assumed to execute at low voltage and fill the buffers prior to the start of any display. The remaining tasks are sorted in increasing order of the *per-unit price* in terms of energy paid when switching the task from low to high voltage in order to save time. For example, a task that expends an extra 6 units of energy to save 2 units of time has a per-unit price of $6/2 = 3$. Ties are broken by favoring shorter tasks. All the sorted tasks are assigned low voltage initially. One by one, the tasks in the sorted order are switched from low voltage to high voltage, until the time constraint described above is met. The energy consumption calculated from the resulting voltage settings is the lower bound.

Figure 9 shows the lower bound energy, the energy with our DVS algorithm, and the percentage difference. The lower bound decreases slightly with increasing buffer capacity, and the DVS result is always well within 1% of the lower bound. Thus constraints such as deadlines, precedence constraints, and the minimum start time do not significantly impede the effectiveness of our DVS algorithm. The

slow decline of the lower bound gives confidence that low capacity buffers are sufficient to get close to ideal energy consumption.

7 Conclusions

In this paper, the impact of dynamic voltage scaling on the tradeoff between low energy consumption and high picture resolution in multimedia decoding was investigated. An efficient offline algorithm was proposed that computes client execution schedules that use DVS on a per-frame basis to minimize energy consumption while satisfying timing and buffering constraints. The experimental results show that the use of DVS significantly reduces energy consumption within a range of high frame resolutions. For a high performance processor (Pentium III), savings of 19% can be achieved at the highest quality, and up to 50% savings are obtained at slightly reduced quality. If CPU accounts for 30% of system energy consumption [23], these savings correspond to system energy savings of between 5 and 15 percent. The main impact of increasing the number of display buffers at the client is to shift upward the range of resolutions for which energy consumption is improved by DVS. Most of the energy savings are realized even with small buffers, partly because the use of DVS tends to reduce display buffer utilization. The energy consumption with our DVS algorithm is within 1% of a derived lower bound with relaxed constraints.

Our proposed offline scheduling algorithm can be applied to MPEG media types such as audio, video, graphics, and text, which together will likely comprise a significant fraction of the workload for future portable devices. Before transmission, the media is stored and pre-processed by the server. At playback, clients are presented options for QoS level, along with corresponding energy consumption information. Throughout playback, users may jump to arbitrary points in the presentation and resume while continuing to follow the DVS schedule. Alternatively, a user may switch to a different schedule at any point in order to effect a different weighting between energy consumption and quality. However, there may be a pause during the transition because of differences in the two schedules.

An important assumption in our algorithm is that the decoding order within each stream is fixed. Subject to that constraint, the algorithm finds the best schedule that accounts for limited display memory at the client and for inter-frame dependencies of the MPEG compression code. The algorithm is also useful for coding schemes that lack frame dependencies, such as JPEG2000 [12],

because the need to account for limited display memory remains. To our knowledge, that aspect has not been addressed by prior investigations [7].

A natural extension to the problem solved in this paper is online scheduling, in which the media is not pre-processed, possibly because it is transmitted live, as it is captured. An online solution that always minimizes energy consumption is impossible, and thus heuristic approaches should be investigated. The offline algorithm proposed in this paper provides a lower bound on energy consumption, to which online results may be compared.

This work takes a first step towards analyzing the QoS-energy tradeoff for multimedia applications. Although we have concentrated on one QoS metric (frame resolution) and one application (MPEG), other media parameters such as frame rate, display brightness, or spectral frequency range present similar quality-energy tradeoffs for MPEG and other compression techniques. The progressive coding standard JPEG2000, for example, is likely well suited for such exploration, since coding for dynamic changes in frame rate and resolution are part of the standard. We envision a future scenario in which the user may adjust energy consumption dynamically through a software knob, and in response the system dynamically adjusts various media and system parameters throughout the presentation to maximize the perceived quality for a desired level of energy consumption.

Acknowledgements We thank the anonymous reviewers for their insightful comments which helped us improve this paper. We thank Tajana Šimunić for initial helpful discussions.

References

1. http://www.linuxvideo.org/devel/dl.html.
2. http://www.flaskmpeg.net.
3. Chandrakasan, A., Sheng, S., and Brodersen, R. W. Low-power CMOS digital design. *IEEE Journal of Solid-State Circuits 27*, 4 (April 1992), 473–84.
4. Fleischmann, M. Crusoe power management- reducing the operating power with LongRun. In *Hot Chips 12* (Aug 2000).
5. Govil, K., Chan, E., and Wasserman, H. Comparing algorithms for dynamic speed-setting of a low-power CPU. In *MOBICOM 95* (1995), pp. 13–25.
6. Haskell, B. G., Puri, A., and Netravali, A. M. *Digital Video: An Introduction to MPEG-2*. Kluwer Academic Publishers, 1996.

7. Hong, I., Kirovski, D., Qu, G., Potkonjak, M., and Srivastava, M. Power optimization of variable voltage core-based systems. In *Proc. Design Automation Conf.* (June 1998), pp. 176–81.
8. Hughes, C. J., Kaul, P., Adve, S. V., Jain, R., Park, C., and Srinivasan, J. Variability in the execution of multimedia applications and implications for architecture. In *Proceedings of the 28th International Symposium on Computer Architecture* (June 2001).
9. Intel. *Mobile Pentium II Processor in Micro-PGA and BGA Packages at 400 MHz, 366 MHz, 300 MHz, 300 PE, and 266PE MHz*, 1999. Order Number 245103-003.
10. Intel. *Pentium III processor for the PGA370 Socket at 500 Mhz to 1 Ghz*, 2000. Order Number 245264-007.
11. Ishihara, T., and Yasuura, H. Optimization of supply voltage assignment for power reduction on processor-based systems. In *7th Workshop on Synthesis and System Integration of Mixed Technologies* (Dec 1997), pp. 51–58.
12. JPEG. *Motion JPEG 2000 Committee Draft 1.0.* ISO, 2000. http://www.jpeg.org/public/cd15444-3.pdf.
13. Lawler, E., and Moore, J. A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 1 (Sep 1969), 77–84.
14. Lin, Y.-R., Hwang, C.-T., and Wu, A. C.-H. Scheduling techniques for variable voltage low power designs. *ACM Transactions on Design Automation of Electronic Systems 2*, 2 (April 1997), 81–97.
15. Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *JACM 20*, 1 (Jan 1973), 46–61.
16. MPEG. *ISO/IEC 13818-2 Generic coding of moving pictures and associated audio information: video.* ISO, 1996.
17. MPEG. *ISO/IEC 13818-10:1999 Generic coding of moving pictures and associated audio information – Part 10: Conformance extensions for Digital Storage Media Command and Control (DSM-CC).* ISO, 1999.
18. Rabaey, J. M. *Digital Integrated Circuits.* Prentice Hall Electronics and VLSI Series, 1996.
19. Raje, S., and Sarrafzadeh, M. Variable voltage scheduling. In *ACM Low Power Design Symp.* (April 1995), pp. 9–14.
20. Shin, Y., and Choi, K. Power conscious fixed priority scheduling for hard real-time systems. In *Design Automation Conference* (June 1999), pp. 134–139.
21. Simunic, T., Benini, L., and De Micheli, G. Cycle-accurate simulation of energy consumption in embedded systems. In *Proc. Design Automation Conf.* (June 1999), pp. 867–72.
22. Steinmetz, R. Human perception of jitter and media synchronization. *IEEE Journal on Selected Areas in Communications 14*, 1 (January 1996), 61–72.

23. Viredaz, M. A., and Wallach, D. A. Power evaluation of a handheld computer: a case study. Tech. Rep. WRL-TR-2001.1, Compaq Western Research Laboratory, May 2001.

24. Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced CPU energy. In *1st Symp. on Operating Systems Design and Implementation* (Nov 1994), pp. 13–23.

25. Yao, F., Demers, A., and Shenker, S. A scheduling model for reduced CPU energy. In *IEEE Annu. Foundations of Comput. Sci.* (Oct 1995), pp. 374–82.