

Complexity Model Based Proactive Dynamic Voltage Scaling for Video Decoding Systems

Emrah Akyol and Mihaela van der Schaar, *Senior Member, IEEE*

Abstract—Significant power savings can be achieved on voltage/frequency configurable platforms by dynamically adapting the frequency and voltage according to the workload (complexity). Video decoding is one of the most complex tasks performed on such systems due to its computationally demanding operations like inverse filtering, interpolation, motion compensation and entropy decoding. Dynamically adapting the frequency and voltage for video decoding is attractive due to the time-varying workload and because the utility of decoding a frame is dependent only on decoding the frame before the display deadline. Our contribution in this paper is twofold. First, we adopt a complexity model that explicitly considers the video compression and platform specifics to accurately predict execution times. Second, based on this complexity model, we propose a dynamic voltage scaling algorithm that changes effective deadlines of frame decoding jobs. We pose our problem as a buffer-constrained optimization and show that significant improvements can be achieved over the state-of-the-art dynamic voltage scaling techniques without any performance degradation.

Index Terms—Complexity prediction, dynamic voltage scaling, video decoding.

I. INTRODUCTION

POWER-FREQUENCY reconfigurable processors are already available in wireless and portable devices. Recently, hardware components are being designed to support multiple power modes that enable trading off execution time for energy savings. For example, some mobile processors can change the speed (i.e., frequency) and energy consumption at runtime [1], [2]. Significant power savings can be achieved by adapting the voltage and processing speed for the tasks where early completion does not provide any gains.

Dynamic voltage scaling (DVS) algorithms were proposed for dynamically adapting the operating frequency and voltage. In CMOS circuits, power consumption is described by the following relationship: $Power \propto V^2 \cdot C_{eff} \cdot f$, where V , C_{eff} , f denote the voltage, effective capacitance and the operating frequency, respectively. The energy spent on one task is proportional to the time spent for completing that task and since the time is inversely proportional to frequency, the energy is proportional to the square of the voltage, i.e., $Energy \propto V^2 C_{eff}$ [3].

Manuscript received June 6, 2006; revised April 17, 2007. This work was supported by the National Science Foundation under Grants CCF 0541453 and CSR-EHS 0509522. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Madjid Merabti.

The authors are with the Department of Electrical Engineering, Henry Samuel School of Engineering and Applied Science, University of California, Los Angeles, CA 90095-1594 USA (e-mail: eakyol@ee.ucla.edu; mihaela@ee.ucla.edu).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TMM.2007.906563

The energy spent on one process can be reduced by decreasing the voltage, which will correspondingly increase the delay. The main goal of all DVS algorithms is utilizing this energy-delay tradeoff for tasks whose jobs' completion times are immaterial as long as they are completed before their processing deadline [4]–[8]. An example is real-time video decoding, where an early completion of frame decoding does not provide any benefit as long as the display deadline for that frame is met. A DVS algorithm essentially assigns the operating level (i.e., power and frequency) for each job given the estimated cycle requirement (i.e., the required complexity) and the job deadline. Hence, the success of DVS algorithms depends on the accuracy of the complexity estimation for the various jobs. Overestimating the complexity results in early termination and idle time and hence, resources are unnecessarily wasted. Underestimating the complexity, however, results in insufficient time allocation for the job. In the specific case of video decoding, this can lead to frame drops or frame freezes. To avoid such job misses, current embedded systems often assume “worst-case” resource utilization for the design and implementation of compression techniques and standards, thereby neglecting the fact that multimedia compression algorithms require time-varying resources, which differ significantly from the “worst-case” requirements.

A. Dynamic Voltage Scaling-Related Works

DVS is an especially good fit for multimedia applications because these applications generally involve computationally complex tasks. Recently, several researchers have addressed the problem of efficient DVS for multimedia applications. An intra-job DVS is proposed by gradually increasing the frequency (speed) within the job, while monitoring the instantaneous cycle demands similar to the approach in [5]. In [6], rather than using the worst case complexity, a worst case estimate satisfying a statistical guarantee determined based on the online profiled histogram of the previous frames is used for the frequency assignment of each job. Moreover, a cross layer optimization framework for different layers of the operating system is proposed to minimize the energy consumption for multimedia applications in [6].

Multimedia data consists of a sequence of data units (DU). The data unit could have different granularities, i.e., it can be a video frame, a part of a video frame, a video packet, a collection of video frames, macroblocks, subbands etc. For video decoding, a job represents conventionally the decoding of a frame. Based on the frame rate, there is a worst-case design parameter, T_{max} , that denotes the amount of time available for processing a job. Depending on the time-varying characteristics of the video content, the deployed compression algorithm and parameters

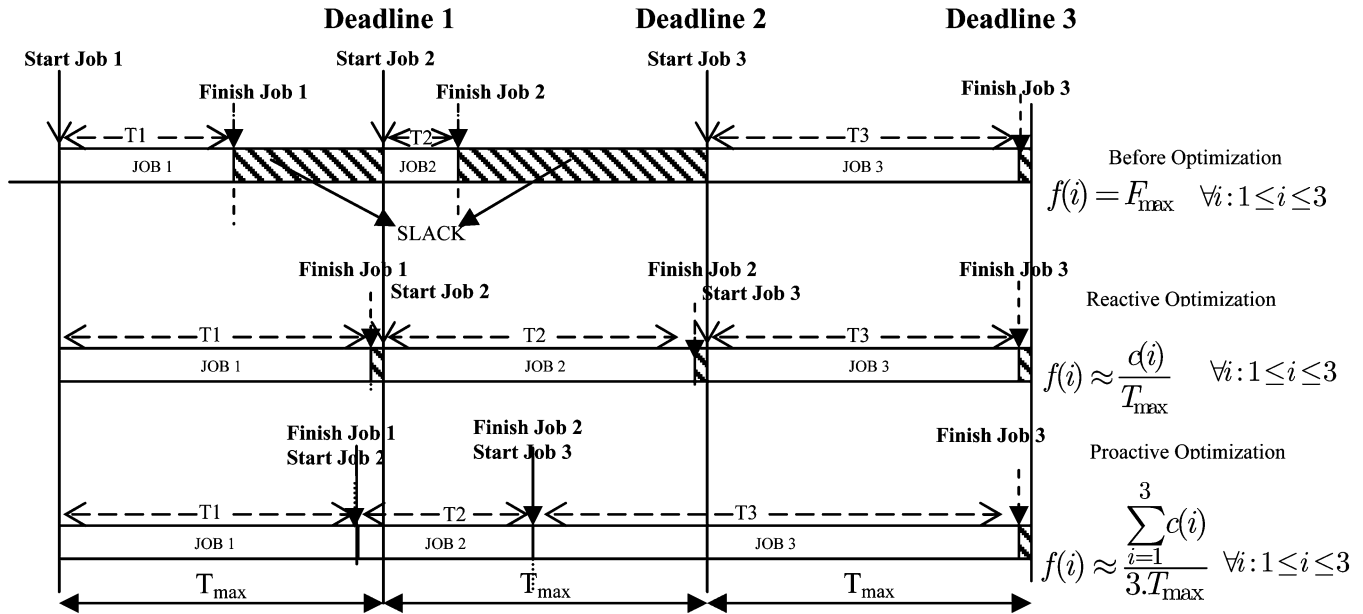


Fig. 1. DVS Strategies: (Top) No DVS, (middle) conventional reactive DVS and (bottom) proposed proactive DVS.

and encoding/transmission bit-rate, not every job needs the entire T_{\max} to complete its execution. Often, the actual completion time of the job is less than T_{\max} . As shown in the top panel of Fig. 1, the first job needs T_1 time units whereas the second job needs T_2 time units. The difference between T_{\max} and the actual completion time is called “slack.” The key idea behind *existing* adaptive systems [5]–[8] is to adapt the operating frequency such that the job is processed exactly in T_{\max} time units (see the middle panel of Fig. 1). In this way, the slack is minimized and the required critical resources (such as energy) are decreased. Thus, the existing approach to adaptive system design is to build on-line models of the multimedia complexity such that the real complexity (in terms of the number of execution cycles) can be accurately predicted, the hardware parameters can be adequately configured, or an optimized scheduling strategy can be deployed to minimize the energy. Clearly, this shows the *reactive* and *greedy* nature of the existing adaptation process—the execution times are predicted based on the time used by previous jobs and, accordingly, the resources are optimized *within a job* for a *fixed* time allocation T_{\max} . However, the DVS gains can be substantially improved when the allocated times (T_1 , T_2 , and T_3) and operating levels (power-frequency pairs) are optimized *jointly* (inter-job optimization) as shown in the bottom panel of Fig. 1.

B. DVS-An Illustrative Example

Assume we have $M = 3$ jobs with complexities $c(1) = (C_{\max}/2)$, $c(2) = (C_{\max}/4)$, $c(3) = C_{\max}$. From now on, we use the term *complexity* to represent the number of execution cycles. As shown in Fig. 1, with “no-DVS,” decoding is performed considering the worst case scenario: at the maximum available frequency for each job F_{\max} and corresponding maximum power P_{\max} . For “conventional DVS,” frequencies are adjusted to finish each job just-in-time i.e., $f(1) = (F_{\max}/2)$, $f(2) = (F_{\max}/4)$, $f(3) = F_{\max}$. If we

assume a power-frequency relationship of $P \propto f^3$ [3], [8], then the power spent on the various jobs will be $p(1) = (P_{\max}/8)$, $p(2) = (P_{\max}/64)$, $p(3) = P_{\max}$. For the “proactive optimization” with modified deadlines, the total complexity is $c(1) + c(2) + c(3) = (7/4)C_{\max}$, and the frequencies are kept constant: $f(1) = f(2) = f(3) = (7F_{\max}/12)$. The total energy spent on the group of jobs equals $E_{total} = \sum_{i=1}^3 (p(i) \cdot c(i) / f(i))$. Hence, the total energy spent for the “no-DVS” case is $E_{no-DVS} = (7/4)E$, for the “conventional DVS” equals $E_{DVS} = (73/64)E$ and for the “proactive DVS” case equals $E_{proactive-DVS} = (343/572)E$, where $E = (P_{\max} \cdot C_{\max} / F_{\max})$. Table I illustrates the parameters associated with each method and each job. Clearly, this example highlights the merit of the proposed proactive DVS algorithm presented in this page since conventional DVS provides 35% energy reduction with respect to no-DVS, whereas proactive DVS provides 66% energy reduction with respect to no-DVS for this simple example. Note that, for the sake of this example, we assumed that the complexities are precisely known beforehand. In practice, it is not possible to obtain an exact estimate of the complexity as this depends not only on the video content and compression algorithm, but also the state of the decoding platform. Hence, existing DVS algorithms with fixed time allocations are conservative in their frequency assignment in order to avoid job misses caused by complexity estimation mismatches. This conservative nature of the previous DVS methods results in redundant slacks at job boundaries.

C. System Assumptions

In this paper, we make similar assumptions to other DVS studies [5]–[8]. Specifically, we assume the following.

- A) The active power is the dominant power, where as the leakage power caused by memory operations, including buffer operations, is negligible. We note that, while a lower CPU speed decreases the CPU energy, it may increase the energy of other resources such as memory [7].

TABLE I
COMPLEXITY, DEADLINES AND FOR EACH ALGORITHM (NO-DVS | CONVENTIONAL DVS | PROACTIVE DVS),
ALLOCATED TIME, FREQUENCY, POWER, AND ENERGY OF EACH JOB

	Complexity, c	Deadline, d	Time Allocation	Frequency f	Power, p	Energy, E
Job 1	$\frac{C_{\max}}{2}$	T_{\max}	$T_{\max} T_{\max} \frac{6T_{\max}}{7}$	$F_{\max} \frac{F_{\max}}{2} \frac{7F_{\max}}{12}$	$P_{\max} \frac{P_{\max}}{8} \frac{343P_{\max}}{1728}$	$\frac{E}{2} \frac{E}{8} \frac{49E}{288}$
Job 2	$\frac{C_{\max}}{4}$	$2T_{\max}$	$T_{\max} T_{\max} \frac{3T_{\max}}{7}$	$F_{\max} \frac{F_{\max}}{4} \frac{7F_{\max}}{12}$	$P_{\max} \frac{P_{\max}}{64} \frac{343P_{\max}}{1728}$	$\frac{E}{4} \frac{E}{64} \frac{49E}{576}$
Job 3	C_{\max}	$3T_{\max}$	$T_{\max} T_{\max} \frac{12T_{\max}}{7}$	$F_{\max} F_{\max} \frac{7F_{\max}}{12}$	$P_{\max} P_{\max} \frac{343P_{\max}}{1728}$	$E E \frac{49E}{144}$

This assumption is reasonable for a computationally complex task like video decoding. We focus on minimizing the CPU energy, although the proposed method can be extended to other resources such as the combined CPU and memory energy. Also, for platforms with significant passive power, (e.g., sensor networks) passive power should be explicitly considered and thus, it should be incorporated into the proposed solution. However, note that this will also result in a more complex optimization problem, since passive energy decreases with frequency whereas active energy increases with it.

- B) The operating frequency changes with the voltage and, at one frequency, there is only one voltage available. Hence, power-frequency tuples also implicitly imply voltage changes, i.e., for each frequency there is only one power consumption level.
- C) The number of execution cycles (i.e., complexity) required for a job does not change with frequency; it only depends on the video, compression algorithm and the decoding platform. Hence, the time spent on one job can be simply written as $t = c/f$, where c is the complexity and f is the operating frequency.
- D) A number of the jobs are available for processing at any time, i.e., we do not assume periodic arrival of jobs as in [5], [6], since in most real-time streaming applications, multiple frames/jobs are already waiting in the decoding buffer. Hence, it is assumed that the processor can start decoding a frame when the decoding of the previous frame is completed.
- E) The processor handles only one task—the video decoding, and there are no other concurrent tasks to be processed.

D. Complexity Estimation

The previous works in complexity estimation for video decoding only consider the frame type and previous frames cycle requirements for estimating the current job (i.e., current frame decoding) complexity [6], [9], [10]. However, our previous studies [11] revealed that with the development of highly adaptive video compression techniques deploying macroblock-based adaptive multiframe prediction, context adaptive entropy coding, adaptive update steps in wavelet coding, etc., modeling the current frame complexity based on previous frames of the same type is no longer accurate. Each individual frame now requires a significantly different amount of video decoding operations yielding a different amount of processing

cycles. As an example, in the state-of-the art video compression standard, H.264/AVC [12], a distinction of frame types does not even exist. Instead, every frame type can include different types of (I, B, or P) macroblocks, and each macroblock requires different amount of processing. Summarizing, it is difficult to achieve the accurate estimates for the current frame decoding complexity merely from the previous frames (i.e., the conventional approach reported in [6]). To overcome this difficulty, we adopt a complexity estimation method for video decoding based on complexity hints that can be easily generated during encoding [13], [15]. We note that, an accurate model of the complexity will increase energy savings for both the proposed method, as well as all other DVS methods.

E. Post-Decoding Buffer

Although the utilized complexity estimation method is significantly more accurate than previous methods based on merely the previous frames' complexity, it is impossible to a priori determine the precise utilized complexity. To avoid the job misses caused by the complexity estimate mismatch (i.e., without unnecessarily being conservative for DVS), we propose to use a post-decoding buffer between the decoding and display devices to store decoded, but not yet displayed frames. Note that, such a post-decoding buffer is already required to store the decoded but not yet displayed frames for video decoders deploying predictions from the future frames (e.g., P-frames are decoded and store beforehand in order to be able to decode B-frames). Using such buffers was also proposed for other purposes like decoding very high complexity frames in real time by introducing some delay [16]. Also, it is important to notice that using this buffer will enable new DVS approaches like changing job execution orders, deadlines and operating frequency according to their estimated cycle demands by mitigating fixed hard deadline constraints. Consider the above example with complexities of second and third jobs interchanged, i.e., $c_{new}(2) = c(3)$ and $c_{new}(3) = c(2)$, i.e., $c(2) > c(3)$. For this case, the modified second job should borrow time slots from the third job for the proposed DVS method. However, in that case, the second job will not meet its deadline which is $d(2) = 2T_{\max}$. Hence, without any buffers to mitigate the deadline requirements, efficient proactive DVS approaches are only possible when the jobs are ordered in increasing complexity. This is generally not the case for most coders and deployed prediction structures. It is important to notice that utilizing this buffer does not introduce any additional delay into the decoding—every job is still completed before its deadline. To conclude, using the post-decoding buffer will serve two purposes: to avoid job misses caused by

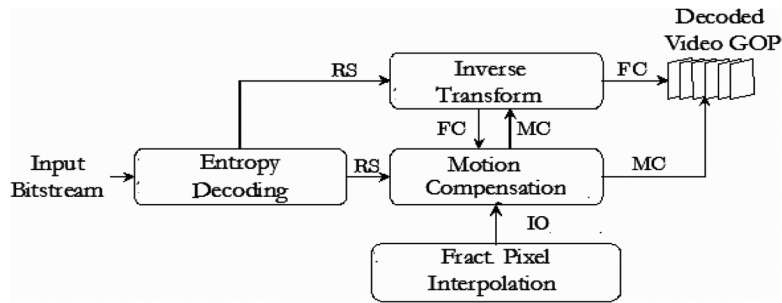


Fig. 2. Basic modules of the proposed generic video decoding complexity modeling framework. The complexity functions are indicated in each module; the generic complexity metrics of each module are indicated on the connecting arrows [13].

the inaccuracy of the complexity estimates and to perform efficient proactive DVS by changing the time allocations through borrowing time allocations from the neighboring jobs.

F. Contributions of This Paper

As mentioned before, none of the previous studies [4]–[8] consider *proactively* changing the time allocations and frequency; instead, they aim at adapting the frequency to *fixed time allocations* in a greedy fashion. We propose a novel DVS algorithm that adapts jobs deadlines by buffering the decoded frames before display. By utilizing this post-decoding buffer, we study the DVS problem into the context of the buffer constrained optimization problem, similar to well studied problems of rate-control with buffer constraints [17]. We also propose an optimal solution for the buffer-constrained power/frequency allocation problem based on dynamic programming. Moreover, we present several low-complexity suboptimal methods. Summarizing, this paper presents two major contributions: i) a practical methodology for complexity estimation that considers the video source, video encoding algorithm and platform specifics to accurately predict execution times with the use of offline generated complexity hints; ii) based on this complexity model, a novel proactive DVS method that changes the effective deadlines (time allocations) considering the buffer constraints.

This paper is organized as follows. Section II describes the video decoding process, complexity estimation and job definitions. The proposed DVS algorithms are explained in Section III. Section IV presents the comparative results and Section V concludes the paper.

II. VIDEO DECODING MODEL

A. Motion Compensated Video Decoding

The basic operational framework for the majority of transform-based motion-compensated video decoders can be summarized by the modules illustrated in Fig. 2. Every video decoder with motion compensation starts with entropy decoding to generate motion and residual information from compressed bits. Then, the inverse transform (DCT or wavelet) is performed to generate reference or residual frames in the spatial domain. Motion compensation with interpolation is employed to inverse the temporal predictions made at the encoder. Hence, the complexity of every video decoding algorithm can be characterized in terms of four basic functional modules: Entropy De-

coding, Inverse Transform, Motion Compensation and Interpolation. Different video coders might have additional modules such as in-loop filters as critical components [14]. Also, other modules, such as inverse quantization, can be easily incorporated in already considered modules. The modules that we illustrated in the paper only capture the most common modules for video decoding systems. For instance, many video coders such as MPEG-2, as well as the coder deployed for experimentation in our paper does not possess an in-loop filter.

B. Background on Generic Complexity Metrics: A Framework for Abstract Complexity Description

In order to represent the different decoding platforms in a generic manner, in our recent work [15], we have deployed a concept that has been successful in the area of computer systems, namely, a virtual machine. We assume an abstract decoder referred to as Generic Reference Machine (GRM), which represents the majority of video decoding architectures. The key idea of the proposed paradigm was that the same bitstream would require/involve different resources/complexities on various decoders. However, given the number of factors that influence the complexity of the decoder such as implementation details, decoding platform, it is impractical to determine at the encoder side the real complexity for every possible decoder. Hence, we adopt a generic complexity model that captures the abstract/generic complexity metrics (GCM) of the employed decoding or streaming algorithm depending on the content characteristics and transmission bitrate [13], [15]. GCMs are derived by computing the number of times the different GRM-operations are executed.

The GRM framework for the transform-based motion-compensated video decoders can be summarized by the modules illustrated in Fig. 2. For any specific implementation of a video decoding algorithm, the decoding complexity may be expressed in terms of generic basic operations and number of execution cycles for each basic operation. Following the approach in [13], we assume basic operations for each module that are easily mapped to real decoding complexity and are indicated by the connecting arrows in Fig. 2. These basic operations are as follows.

- Entropy decoding (g_{ED}): the number of iterations of the “Read Symbol” (RS) function is considered. This function encapsulates the (context-based) entropy-decoding of a symbol from the compressed bitstream.

- Inverse transform (g_{IT}): the number of multiply-accumulate (MAC) operations with nonzero entries is considered, by counting the number of times a MAC operation occurs between a filter-coefficient (FC) and a nonzero decoded pixel or transform coefficient.
- Fractional-pixel interpolation (g_{IO}): the number of MAC operations corresponding to horizontal or vertical interpolation (IO) can characterize the complexity of this module.
- Motion compensation (g_{MC}): the basic motion compensation (MC) operation per pixel is the factor in this module's complexity.

C. Conversion of Generic Complexity Metrics to Real Complexity Using Adaptive Filtering

The GCMs are converted online to the number of execution cycles at the decoder, using a time-varying predictor (w) for each of the GCM functions (RS, FC, MC, IO). We denote this function set $\Psi = \{RS, FC, MC, IO\}$. Let $cr_{k,t}^j(i)$, $c_{k,t}^j(i)$, $g_{k,t}^j(i)$, $w_{k,t}^j(i)$ denote the number of cycles spent (termed here as "real complexity"), their estimates, and the corresponding GCM and linear predictors, respectively, for functions $k \in \Psi$, for DU index i (index within the job), temporal level or frame type t , within job j . The value of each $g_{k,t}^j(i)$ may be determined while encoding and these GCMs can be attached to the bitstream with negligible encoding complexity or bitrate cost [15]. Similar to the linear models in [18], [20], we model the real complexity as

$$cr_{k,t}^j(i) = w_{k,t}^j(i) \cdot g_{k,t}^j(i) + e_{k,t}^j(i), \quad (1)$$

where $e_{k,t}^j(i)$ is the estimation error and the linear prediction (complexity estimate) is

$$c_{k,t}^j(i) = w_{k,t}^j(i) \cdot g_{k,t}^j(i). \quad (2)$$

We note that the linear mapping of the generic complexity metrics to execution times is a kind of least squares matching problem that is well studied in the context of adaptive filtering. Hence, we employ a widely used adaptive filter, normalized LMS, for converting generic metrics to real complexity such as the number of cycles spent on the processor. For this purpose, normalized LMS (NLMS) is shown to be a good choice among the family of adaptive filters since, besides its very low complexity, it replaces the required statistical information with instantaneous approximations [21]. In our problem, statistics are time-varying due to the state of the decoding platform. This makes LMS algorithms a better choice than recursive least squares (RLS) filters, which update the filter from the beginning assuming stationary statistics.

If we model the real complexity and generic complexity metrics, as scalar random variables ($c_{k,t}^j(i)$ and $g_{k,t}^j(i)$), then the optimal linear estimator (in the mean square error sense) [21]

$$w_{k,t}^{j,opt} = \arg \min_{w_{k,t}^j} E \left(\left| c_{k,t}^j - w_{k,t}^j \cdot g_{k,t}^j \right|^2 \right) \quad (3)$$

is given by $w_{k,t}^{j,opt} = \mathbf{R}_{g_{k,t}^j}^{-1} \cdot \mathbf{R}_{c_{k,t}^j, g_{k,t}^j}$, where \mathbf{R} denotes the covariance matrix. Since we use a first order predictor, \mathbf{R} reduces to the scalar autocorrelation value. Higher order filters can also

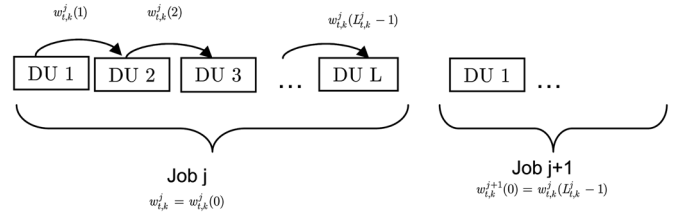


Fig. 3. The update of the predictor for each DU within a job, $i = 0, \dots, L_{t,k}^j - 1$.

be utilized at the expense of an increased complexity overhead [18]. The solution to (3) can be obtained by deploying a steepest descent algorithm that iteratively approximates $w_{k,t}^{j,opt}$:

$$w_{k,t}^j(i) = w_{k,t}^j(i-1) + \mu \left(\mathbf{R}_{c_{k,t}^j, g_{k,t}^j} - \mathbf{R}_{g_{k,t}^j} \cdot w_{k,t}^j(i-1) \right) \quad (4)$$

where μ is a positive step size. We use the instantaneous approximations for the covariance $\mathbf{R}_{c_{k,t}^j, g_{k,t}^j} = cr_{k,t}^j(i-1) \cdot g_{k,t}^j(i-1)$ and $\mathbf{R}_{g_{k,t}^j} = \left(g_{k,t}^j(i-1) \right)^2$ to obtain the well known LMS recursion for each DU i :

$$w_{k,t}^j(i) = w_{k,t}^j(i-1) + \mu \cdot g_{k,t}^j(i-1) \times \left(cr_{k,t}^j(i-1) - w_{k,t}^j(i-1) \cdot g_{k,t}^j(i-1) \right). \quad (5)$$

In this paper, we use the normalized LMS since it has faster convergence and it is independent of the magnitude of the regressor (GCMs), which is

$$w_{k,t}^j(i) = w_{k,t}^j(i-1) + \frac{\mu}{g_{k,t}^j(i-1)} \times \left(cr_{k,t}^j(i-1) - w_{k,t}^j(i-1) \cdot g_{k,t}^j(i-1) \right). \quad (6)$$

Although the predictor is updated at each DU, we update the predictor only at the job boundaries in our optimization framework (as shown in Fig. 3). In other words, updating the job parameters and assigning different operating levels for each DU is not realizable in real time. If job j has $L_{t,k}^j$ DUs of function type k and temporal level (or frame type) t , we use the predictor $w_{k,t}^{j+1}(0) = w_{k,t}^j(L_{t,k}^j - 1)$ for the duration of the execution time of job $j+1$, although we iterate this predictor in job $j+1$ as in (4). Thus, we estimate the complexity job j as

$$c_{t,k}^j = w_{t,k}^j(0) \cdot \sum_{i=0}^{L_{t,k}^j - 1} g_{t,k}^j(i) \quad (7)$$

and the total complexity of job j is the sum of complexities associated with every frame type and function type within that job:

$$c(j) = \sum_{t=1}^T \sum_{k=1}^{\aleph} c_{t,k}^j \quad (8)$$

where \aleph is the cardinality of the function set Ψ , which is in our case $\aleph = 4$.

We also determine an error estimate for the complexity estimation based on the error obtained for the previous job. We assume that errors for each DU decoding can be characterized

as zero mean independent identically distributed Gaussian random variables: $e_{k,t}^j \sim N(0, \sigma^2)$. Given the set of realizations $(e_{k,t}^j(i))$ of this random variable, the variance of the zero mean complexity error estimate of every DU within the job $(\sigma_{t,k}^j)$ can be found by the maximum likelihood method [21] which has closed form expression:

$$(\sigma^2)_{t,k}^j = \frac{\sum_{i=0}^{L_{t,k}^j-1} (e_{k,t}^j(i))^2}{L_{t,k}^j}. \quad (9)$$

Since we do not use the predictor at DU granularity, but rather update it at job intervals, the estimation error associated with the constant predictor throughout the job can be written as

$$\begin{aligned} & \sum_{i=0}^{L_{t,k}^j-1} (cr(i) - w(0)g(i)) \\ &= \sum_{i=0}^{L_{t,k}^j-1} (w(i)g(i) + e(i) - w(0)g(i)) \end{aligned} \quad (10)$$

and using (6)

$$w(i) = w(i-1) + \mu \frac{e(i-1)}{g(i-1)}. \quad (11)$$

Using (9) and (11), we obtain the estimation error variance of the next job as

$$\sigma^2(j+1) = \sum_{k=1}^{\aleph} \sum_{t=1}^T \left(\frac{L_{t,k}^{j+1} (L_{t,k}^{j+1} - 1)}{2} \mu + L_{t,k}^{j+1} \right) (\sigma^2)_{t,k}^j. \quad (12)$$

We define the worst case error (or the risk) of the estimation for job $j+1$ as

$$r(j+1) = k \cdot \sigma(j+1), \quad (13)$$

where the parameter k determines the amount of risk that can be tolerated for a specific task. The worst case error estimate is used in determining the buffer occupancy thresholds (β_1 and β_2) in buffer constrained optimization in Section III.

D. Overhead of the Complexity Estimation

We measure the overhead of the estimation process by the exact count of floating point arithmetic operations (multiplication, addition), herein called “flops” (floating points operations). For normalized LMS, there are 5 flops for each DU. Hence, there are $5 \cdot \sum_{k=1}^{\aleph} \sum_{t=1}^T L_{t,k}^j$ flops required for job j . For the error estimation part, $6 \cdot \sum_{k=1}^{\aleph} \sum_{t=1}^T (L_{t,k}^{j+1})$ flops are required, that can be seen from (9) and (12). Totally, $11 \cdot \sum_{k=1}^{\aleph} \sum_{t=1}^T L_{t,k}^j + 6 \cdot T \cdot \aleph$ flops required for the complexity and error estimation of job j . Note that, error estimation (i.e., the worst case estimate) is only used for determining the risk of the buffer overflow underflow, which, similar to [17], can be heuristically set to some fixed value when the complexity estimation overhead becomes significant.

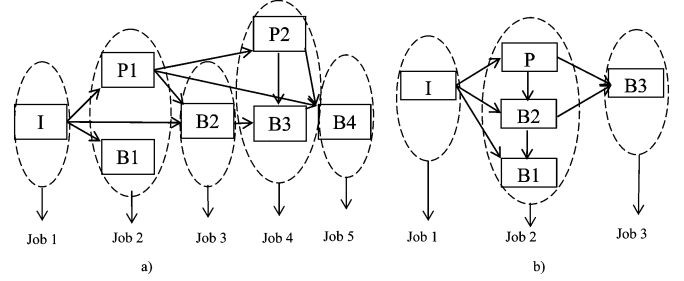


Fig. 4. Directed acyclic dependence graphs for (a) Dependencies between I-B1-B2-P1-B3-B4-P2 frames and (b) Hierarchical B Pictures, I-B1-B2-B3-P.

E. Modified Job Definitions for Adaptive Video Compression

All state of the art video encoders deploy complex temporal predictions from both past and/or future frames. For example, in the latest video compression standard, H.264/AVC [12], B and P type macroblocks are predicted from other macroblocks, some of which are in future frames and thus, must be decoded before their display deadline. Hence, each frame has a *decoding deadline* that is determined by the temporal decomposition structure (temporal dependencies). This deadline is different from the *play-back (display) deadline* determined by the frame rate fr . Let $\mathfrak{R}(n)$ be the set of frames for which frame n is used as a reference. Then, the display and decoding deadlines for frame n can be written as:

$$\begin{aligned} \text{deadline}_{\text{display}}(n) &= \frac{n}{fr}, \\ \text{deadline}_{\text{decoding}}(n) &= \min\{\text{deadline}_{\text{display}}(i)\} : \forall i \in \mathfrak{R}(n). \end{aligned}$$

Unlike previous work that considers the decoding of each individual frame as a task, we combine frames having the same *decoding deadline* (i.e., frames that are dependently encoded) into *one job* of the decoding task. In general, we define every job based on three parameters:

$$\text{job} : \{\text{deadline, complexity, size}\}$$

where

deadline	decoding deadline of job j , $d(j)$;
complexity	estimated number of cycles that job j consumes on a specific platform, $c(j)$;
size	number of decoded <i>original</i> frames when job j finishes, $s(j)$.

The amount and type of complexity vary significantly per job [11]. In the following, we illustrate how to calculate the decoding deadlines and how to define jobs in both traditional predictive coding and motion compensated temporal filtering (MCTF) based coding.

1) *Example-1: Job Structure for Predictive Decoding Schemes:* In predictive coding, frames are encoded with interdependencies that can be represented by a directed acyclic dependence graph (DAG) [25]. Examples are shown in Fig. 4 for two different GOP structures: the conventional I-B-B-P-B-B-P GOP structure and the hierarchical B pictures. Decoding frame I is a job and decoding frames P1 and B1

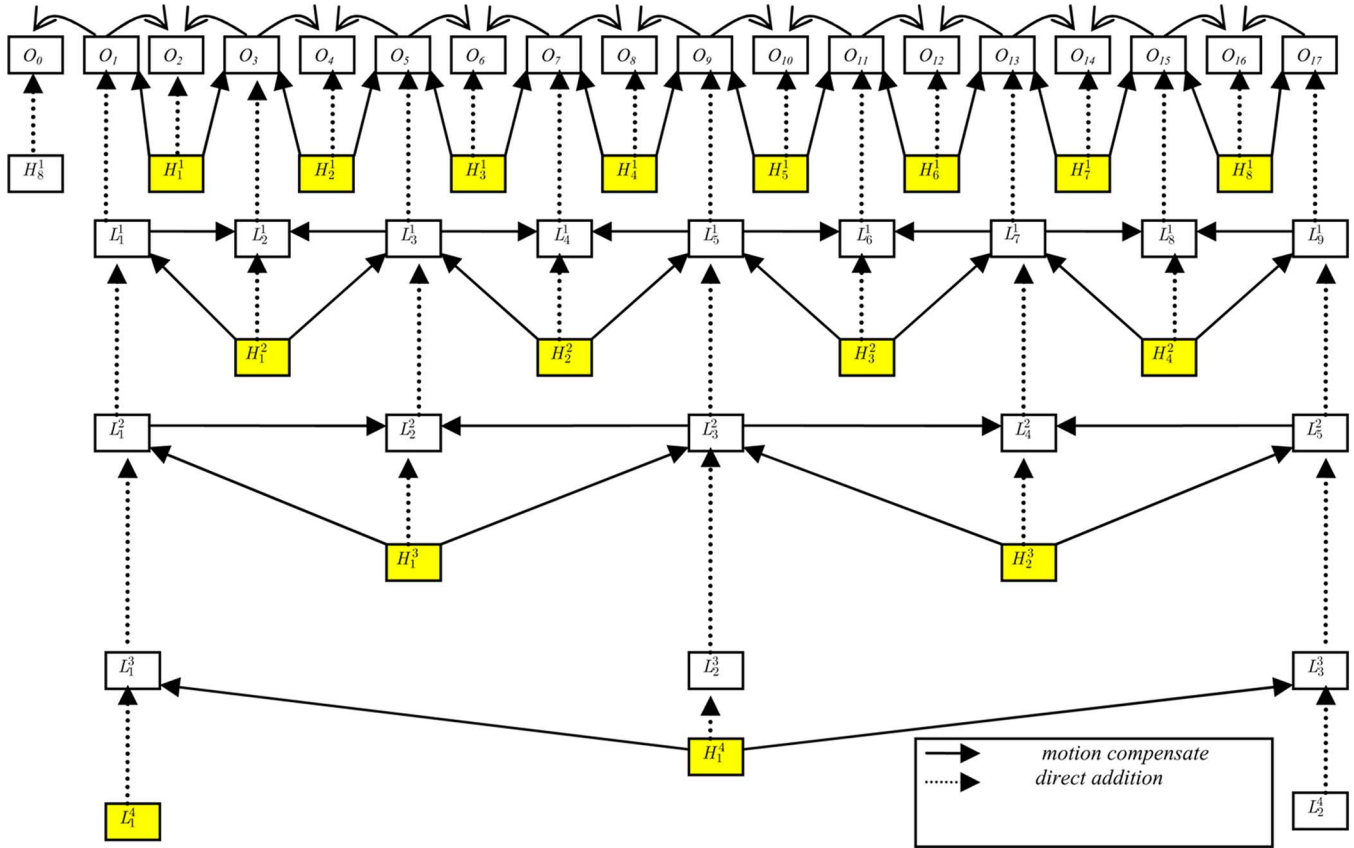


Fig. 5. Inverse 5 – 3 MCTF decomposition in decoding.

jointly represents another job as shown in Fig. 4(a). Prediction structures using hierarchical B pictures as in the H.264/AVC standard lead to the following sizes (s), complexities (c), and deadlines (d): the first job represents the decoding of the frame $I(s(1) = 1, d(1) = \Delta t)$ the second job consists of decoding frames P, B1 and B2 ($s(2) = 3, d(2) = 2\Delta t$) and the last job is the decoding of frame B3 ($s(3) = 1, d(3) = 4\Delta t$). Also, the first job (i.e., decoding I frame) involves only the texture related complexities, whereas the second and third jobs include several bi-directional MC operations. It is important to notice that, both the second and the third job can be viewed from a high level perspective as decoding a B frame. However, the job parameters are substantially different, thereby highlighting the need for encoder-specific complexity estimation.

2) *Example-2: Job Structure for MCTF Decoding Schemes:* In MCTF-based video compression schemes, video frames are filtered into L (low-frequency or average) and H (high-frequency or difference) frames. The process is applied iteratively, first on the original frames (O denotes the sequence of original frames), and then, subsequently, to the resulting L frames. A temporal pyramid of T temporal levels is produced in this manner. We use the notation H_k^t to indicate the k -th H frame of temporal level t , where $1 \leq t \leq T$ and $1 \leq k \leq 2^{T-t}$. Equivalently, the notation L_1^t is used to indicate the remaining L frame at the last level, after the completion of the temporal decomposition.

From the temporal dependencies depicted in Fig. 5, we can see that to display the original frames O_1 and O_0 , the frames

$L_1^4, H_1^1, H_1^2, H_1^3$, and H_1^4 should be decoded¹. This implies that these frames (O_1 and O_0) have the same decoding deadline and thus, they define a job. Table II shows the deadline, complexity ($comp_t$, texture related complexity: entropy decoding and inverse transform; $comp_m$, motion related complexity: interpolation and motion compensation to generate original frames and intermediate low-pass frames) and the size of each job.

Note that the *size* of each frame in the GOP is the same for all the jobs in this example. However, it is related to the temporal decomposition used (5/3 Daubechies filter) and can differ for another decomposition [24].

III. PROACTIVE DYNAMIC VOLTAGE SCALING ALGORITHM

A. Problem Setup

We note that, for each video decoding job, we have a complexity estimate, size, and deadline defined before the job is executed. Let us assume there is a discrete set of operating levels with corresponding frequency and power levels which can be used in our frequency/voltage adaptation $\mathbf{L} = \{l_j : (p_j, f_j), 1 \leq j \leq N \mid l_1, \dots, l_N\}$. Each level has a different power consumption and different frequency, $(\mathbf{P}, \mathbf{F}) = \{(p_1, f_1), \dots, (p_N, f_N) \mid p_1 < \dots < p_N; f_1 < \dots < f_N\}$. Assume there are a total of M jobs with the complexity estimates $\mathbf{C} = \{c(1), \dots, c(M)\}$, the deadlines $\mathbf{D} = \{d(1), \dots, d(M) \mid d(1) < \dots < d(M)\}$ and the sizes

¹Note that H_8^1 of the previous GOP should already be decoded for the frames before O_0

TABLE II
DEADLINE, COMPLEXITY AND SIZE OF EACH JOB FOR DECODING A GOP WITH $T = 4$ LEVEL USING A 5/3 DAUBECHIES FILTER BASED MCTF

	Job-1	Job-2	Job-3	Job-4	Job-5	Job-6	Job-7	Job-8
<i>deadline</i>	t_0	$t_0 + 2\Delta t$	$t_0 + 4\Delta t$	$t_0 + 6\Delta t$	$t_0 + 8\Delta t$	$t_0 + 10\Delta t$	$t_0 + 12\Delta t$	$t_0 + 14\Delta t$
<i>Comp_t</i>	$L_1^4, H_1^3, H_1^2, H_1^1, H_1^4$	H_2^1	H_3^1, H_2^2, H_2^3	H_4^1, H_3^2	H_5^1	H_6^1, H_4^2	H_7^1	H_8^1
<i>Comp_m</i>	L_1^3, L_1^2, L_1^1 O_1, O_0	L_2^3, L_2^1, L_1^3 O_2, O_3	L_4^1, L_5^1, O_4, O_5	L_2^2, L_3^2, O_6, O_7	O_8, O_9	L_6^1, L_7^1, L_4^2 O_{10}, O_{11}	O_{12}, O_{13}	L_8^1, O_{14}, O_{15}
<i>Size</i>	$2, (O_0, O_1)$	$2, (O_2, O_3)$	$2, (O_4, O_5)$	$2, (O_6, O_7)$	$2, (O_8, O_9)$	$2, (O_{10}, O_{11})$	$2, (O_{12}, O_{13})$	$2, (O_{14}, O_{15})$

TABLE III
NOMENCLATURE OF SYMBOLS

<i>Notation</i>	<i>Description</i>	<i>Unit</i>	<i>Notation</i>	<i>Description</i>	<i>Unit</i>
$rc(j)$	Real complexity of the job j	<i>cycles</i>	$d(j)$	Deadline of the job j	<i>seconds</i>
$p(j)$	Power consumption of the job j	<i>Watts</i>	$s(j)$	Size of the job j	<i>frames</i>
$t(j)$	Expected time that the job j will take	<i>seconds</i>	$r(j)$	The worst case error of complexity estimate of the job j	<i>cycles</i>
$E(j)$	Expected energy of the job j	<i>Joules</i>	$B(j)$	Buffer occupancy when the job j is executed	<i>frames</i>
$c(j)$	Complexity estimate of the job j	<i>cycles</i>	$f(j)$	Operating frequency of the job j	<i>cycles/s</i>
f_i	Operating frequency i	<i>cycles/s</i>	fr	Frame rate	<i>frames/s</i>

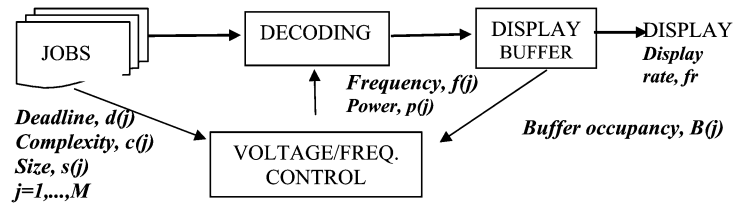


Fig. 6. Proposed buffer controlled DVS.

$\mathbf{S} = \{s(1), \dots, s(M)\}$. To facilitate the reading of the paper, in Table III, we present a summary of the most important symbols and their descriptions in the paper.

The dynamic voltage scaling problem attempts to find the set of operating levels (power and frequency tuple) for each job $\mathbf{l}^{\text{opt}} = \{l(1), \dots, l(M) \mid \forall l \in \mathbf{L}\}$, as shown in (14) at the bottom of the page.

We propose to use a post-decoding buffer between the display device and the decoding platform as shown in Fig. 6. The

operating level of the job j ($p - f$ tuple) is determined considering the parameters of M jobs and buffer occupancy $B(j)$. For each job, the complexity estimates are updated and based on the buffer occupancy, a new operating level is assigned.

We define the buffer occupancy for job j as $B(j)$ recursively as

$$B(j) = \max(B(j-1) + s(j) - \lfloor t(j) \cdot fr \rfloor, 0) \text{ and} \\ B(0) = B_{\text{initial}} \quad (14)$$

DVS Problem :

$$\text{find } \mathbf{l}^{\text{opt}} = \arg \min_{l \in \mathbf{L}} \sum_{j=1}^M E(j) \text{ (energy consumption)}$$

$$\text{such that : } \sum_{m=1}^j t(m) < d(j); \quad \forall j : 1 \leq j \leq M \text{ (delay constraints)}$$

(14)

where fr denotes the frame rate, B_{initial} is the initial state of the buffer, and depends on the initial playback delay which may be zero if no delay is tolerable. We define the buffer size as the number of decoded frames. Even in the zero delay case, the buffer can be occupied with early completions or as a result of a more aggressive (faster) decoding strategy for the initial frames. Then, the DVS problem in (14) becomes the problem of minimizing the total energy under buffer constraints; see (16) at the bottom of the page.

The buffer overflow/underflow constraint: The buffer should never underflow, to avoid any frame freezes. Also, the buffer occupancy cannot grow indefinitely because it is a finite physical buffer. If we assume the maximum buffer size is B_{max} , we need to guarantee that the buffer occupancy is always lower than B_{max} .

B. Optimal Solution Based on Dynamic Programming

The optimal solution can be found by dynamic programming methods, which explicitly consider every possible frequency-buffer pair for each job and check the buffer state for overflow or underflow [22]. A trellis is created with given complexity estimates of the job and possible power-frequency assignments. At every stage, the paths reaching the same buffer occupancy at a higher energy cost are pruned. Note that, since the complexity prediction is updated at each job boundary, this optimization should be performed for each job.

The optimal solution does not assume any property of the power-frequency function such as convexity. However, the complexity of this approach is in the order of $O(B_{\text{max}}M^2)$ for a total of M jobs. This overhead may be not practical for real time decoding systems. Hence, in Section IV, we propose sub-optimal approximations to this optimal solution.

C. Properties of Buffer Constrained Proactive DVS Problem

We approximate the buffer constraints for a look-ahead window of ($W \ll M$) jobs: Starting from the job j

for the next W jobs, we aim to have the buffer occupancy at the equilibrium (the half size of the buffer ²) $B(j) + \sum_{m=j}^{j+W} [s(m) - t(m) \cdot fr] = B_{\text{max}}/2$. We redefine the expected energy spent and expected delay (time spent) for job j as

$$E(j) = \frac{p(j)c(j)}{f(j)} \text{ and } D(j) = \frac{c(j)}{f(j)}. \quad (17)$$

Then DVS problem for the next W jobs starting from job j can be approximated as described below. See equation (18) at the bottom of the page.

Although there is a finite number of operating levels, the intermediate operating levels are achievable by changing the frequency/power within the job similar to the approach in [4], [5]. Fig. 7 shows the energy-delay curve for jobs j and $j+1$.

1) *Proposition 1:* If we neglect the transition cost from one frequency to another [5], the frequency change within the job corresponds to piecewise bilinear interpolation power-frequency points as shown in Fig. 7.

Proof: Assume the desired frequency for the job j f' ($f_i < f' < f_{i+1}$) can be achieved by performing c_1 cycles at frequency f_i and c_2 cycles at f_{i+1} such that $c_1 + c_2 = c(j)$, i.e., for some α such that $c_1 = \alpha c(j)$ and $c_2 = (1 - \alpha)c(j)$. Also, let $E_i - D_i$ and $E_{i+1} - D_{i+1}$ be the corresponding E-D points to f_i and f_{i+1} .

$$\begin{aligned} \text{Clearly, } \frac{c(j)}{f'} &= \alpha \frac{c(j)}{f_i} + (1 - \alpha) \frac{c(j)}{f_{i+1}} \\ \text{since } \frac{c(j)}{f'} &= \frac{c_1}{f_i} + \frac{c_2}{f_{i+1}}, \text{ or equivalently} \\ D' &= \alpha D_i + (1 - \alpha) D_{i+1}. \end{aligned} \quad (19)$$

The energy spent for the job becomes

²A more accurate equilibrium buffer size can be determined by considering complexity estimates in the long run, i.e., considering the relative amount of complexity of the look-ahead window with respect to total complexity of M jobs.

Buffer Constrained DVS Problem :

$$\begin{aligned} \text{find } \mathbf{l}^{\text{opt}} &= \arg \min_{l \in \mathbf{L}} \sum_{j=1}^M E(j) \text{ (energy consumption)} \\ \text{such that : } & 0 \leq B(j) \leq B_{\text{max}}; \forall j : 1 \leq j \leq M \text{ (buffer overflow/underflow constraint)} \end{aligned} \quad (16)$$

Approximation of the Buffer Constrained DVS Problem :

$$\begin{aligned} \text{find } \mathbf{l}^{\text{opt}}(j) &= \arg \min_{l \in \mathbf{L}} \sum_{m=j}^{j+W} E(m) \text{ (energy consumption)} \\ \text{such that : } & \sum_{m=j}^{j+W} D(m) = \frac{B(j) - B_{\text{max}}}{2} + \sum_{m=j}^{j+W} s(m) \text{ (delay constraint)} \end{aligned} \quad (18)$$

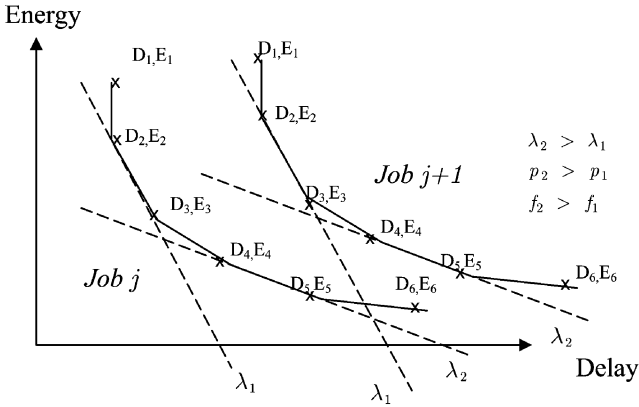


Fig. 7. Energy-Delay curve of two different jobs with D,E pairs corresponds to different frequency-power levels of the processor. Intermediate levels are achieved by frequency changes within the job. Greater values of the slope means greater energy spent with smaller delay yielding a higher frequency-power choice. Every job has different E-D points but the slopes are identical for every job.

$$E' = \frac{p_j \cdot c_1}{f_j} + \frac{p_{j+1} \cdot c_2}{f_{j+1}}, \text{ or equivalently,}$$

$$E' = \alpha E_i + (1 - \alpha) E_{i+1}. \quad (20)$$

Equations (19) and (20) show the linear relationship for the points interpolated between E-D points. ■

2) *Proposition II:* The slope between two E-D points only depends on the power and frequency values but not on complexity of the job.

Proof: The E-D slope between points E_k, D_k and E_{k+1}, D_{k+1} for the job j is

$$\lambda_k(j) = \frac{E_{k+1} - E_k}{D_{k+1} - D_k}, \quad 1 \leq k \leq N - 1. \quad (21)$$

Replacing (17) into (21), we obtain

$$\lambda_k(j) = \frac{p_{k+1} \cdot f_k - p_k \cdot f_{k+1}}{f_k - f_{k+1}}. \quad (22)$$

Equation (21) shows that the slopes (λ_k) depend only on power-frequency tuples, hence they are identical for all jobs. ■

3) *Proposition III:* From a set of given power-frequency points, only the ones that generate a convex set of E-D points should be used for the proposed proactive DVS.

Proof: The proof is based on the Jensen's Inequality [26]. If there is an E-D point E_k, D_k which does not lie on the convex hull of E-D points, then an intermediate point E', D' with the same delay ($D' = D_k$) can be achieved by bilinear interpolation of E_{k-1}, D_{k-1} and E_{k+1}, D_{k+1} such that if for some constant α , $D_k = \alpha D_{k-1} + (1 - \alpha) D_{k+1}$ then $E' = \alpha E_{k-1} + (1 - \alpha) E_{k+1}$ from Proposition I. However, since E_k, D_k does not lie on the convex hull, if there exists a constant α , $D_k = \alpha D_{k-1} + (1 - \alpha) D_{k+1}$ then $E_k < \alpha E_{k-1} + (1 - \alpha) E_{k+1}$ [26]. Hence, E-D points that are not on the convex hull should not be used since there is an intermediate point with the same delay, leading to lower energy. ■

Generation of the Convex Hull of E-D Points: The convexity of power-frequency values does not guarantee the convexity of the E-D curve. For example, $p = \alpha \cdot f^k$ for $1 < k < 2$ is a

convex function of frequency but does not provide convex E-D points. Hence, before any optimization, the power-frequency values which do not provide complex E-D points should be pruned, i.e., a convex hull of E-D points, from a possible set of E-D points should be generated. Note that, since the slopes are identical for all jobs (Proposition II), this pruning is done only once for all jobs as shown in Table V.

4) *Proposition IV:* The optimal operating level assignment will result in equal slopes for every job:

$$\lambda^{opt}(i) = \lambda^{opt}(j); \quad 1 \leq j \leq M, \quad 1 \leq i \leq M.$$

Proof: The DVS problem (18) can be converted to an unconstrained problem by introducing a Lagrange multiplier into the cost function of the problem. Lagrangian solutions are based on the fact that for additive convex cost functions, the slope of the individual cost functions should be equal. This is a direct extension of the Theorem-1 of [23]. Hence, to minimize energy, the E-D slope of all jobs should be identical. ■

From the Propositions II and IV, the optimal frequency considering a look-ahead window of W jobs can be found considering a total processing time $t = (B(j) - (B_{max}/2) + \sum_{m=j}^{j+W} s(m)/fr)$ and a total complexity $c = \sum_{m=j}^{j+W} c(m)$. Then, the frequency can be found by $f = c/t$:

$$f^{opt}(j) = \frac{fr \cdot \sum_{m=j}^{j+W} c(m)}{B(j) - \frac{B_{max}}{2} + \sum_{m=j}^{j+W} s(m)}. \quad (23)$$

This result is also intuitive because, by utilizing the buffer we consider a collection of W jobs as one job. In other words, by utilizing the post-decoding buffer, we smooth the complexities of jobs within the group of W jobs.

We note that, although we assumed that all intermediate points are achievable by continuously changing the frequency within the job, in practice there may only be a limited number of frequency transition opportunities within a that job [6], i.e., the choice of c_1 and c_2 in Proposition I is not arbitrary but limited to some number determined by the used platform. Also, note an intermediate point (E', D' such that $E_k < E' < E_{k+1}, D_k < D' < D_{k+1}$) does not provide any energy savings beyond using the given power frequency points (p_k, f_k and p_{k+1}, f_{k+1}) since the intermediate point is on the linear curve, not strictly convex. Let us consider an example where two jobs with identical complexities are to be processed at a frequency $f' = (f_k + f_{k+1})/2$. We can achieve this frequency by processing half of the cycles at f_k and the other half of the cycles at f_{k+1} for both jobs. However, processing one job at frequency f_k and the other job at f_{k+1} will result in identical energy spent and identical total delay. The only difference created by using intermediate frequencies is decreasing the instantaneous delay (the delay caused by the first job in the given example) which is already tolerated by the buffer. Also considering the frequency transition cost within a job, we conclude that the intermediate frequencies should not be used in the proposed DVS method. In Section IV, we

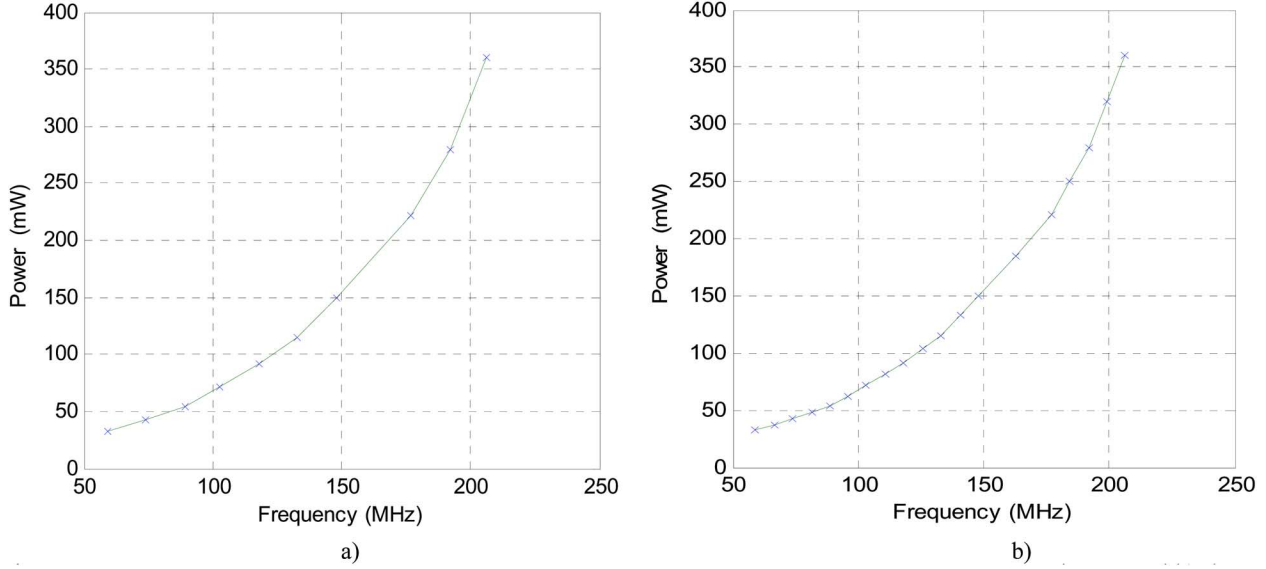


Fig. 8. Frequency power values of Strong-Arm processor analyzed in [19] (a) original values which are used (b) the piecewise linearly interpolated values that are used for comparison.

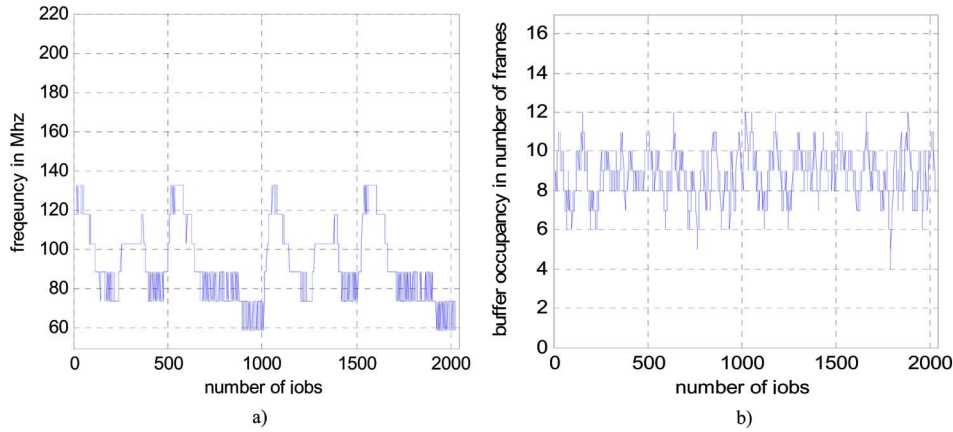


Fig. 9. Frequency assignment (a) and buffer fullness(b) for Algorithm 1 for one test video.

experimentally validate this approach. Then, the frequency, $f^*(j)$, becomes the closest frequency to $f^{opt}(j)$ chosen from a set that generate convex E-D points. Let φ be the frequency set that generate convex E-D points, the operating frequency for job j can be found as:

$$f^*(j) = \arg \min_{f \in \varphi} |f - f^{opt}(j)|. \quad (24)$$

D. Suboptimal Low Complexity Algorithms

Next, we propose a fast suboptimal approximation to the optimal solution, as performing this frequency assignment for each job as illustrated in Table VI.

The complexity of the Algorithm 1 is $(2W + 5)M$ (from (23)) flops for M jobs, since for each job we have to repeat this frequency assignment. Algorithm 1 performs the delay-energy optimization for every job considering a look ahead window of W jobs. A fast extension of this algorithm, Algorithm 2, is to perform this optimization and assign the same frequency for W jobs and reperform this optimization when the buffer level

gets lower or higher than the specified thresholds, i.e., perform it when $B(j) \leq \beta_1$ or $B(j) \geq \beta_2$. These thresholds can be determined based on the risk that buffer will overflow or underflow. Hence, unless the buffer is at risk of overflow or underflow, we use the same frequency for the whole duration of W jobs for Algorithm 2. $r(j)$ is the maximum expected error in the complexity estimation part, in terms of real complexity. In the worst case, the buffer will loose $((c(j) + r(j)) \cdot fr / f_{min})$ frames while decoder generates $s(j)$ frames in job j . Similarly, if buffer is close to overflow; in the worst case, the buffer will loose $((c(j) - r(j)) \cdot fr / f_{max})$ frames. Hence,

$$\beta_1 = \left\lceil \frac{(c(j) + r(j)) \cdot fr}{f_{min}} - s(j) \right\rceil \text{ and} \\ \beta_2 = B_{max} - \left\lfloor s(j) - \frac{(c(j) - r(j)) \cdot fr}{f_{max}} \right\rfloor \quad (25)$$

where $r(j)$ is the standard deviation of the expected error in the complexity estimation process, defined in (6).

Algorithm 2 requires in the order of $2M$ flops for M jobs since it performs this optimization for each look-ahead window

TABLE IV
OPTIMAL SOLUTION-DYNAMIC PROGRAMMING

1.	For each job j ; $1 \leq j \leq M$,
2.	Create Trellis: For each job m ; $j \leq m \leq M$,
3.	Find <i>expected energy</i> ($E(m)$) and <i>buffer occupancy</i> ($B(m)$) for each <i>power-frequency tuple</i> $(p_1, f_1), \dots, (p_N, f_N)$ using the <i>complexity estimate</i> ($c(m)$)
4.	Record the <i>cumulative energy</i> $\sum_{k=1}^{k=m} E(k)$ and <i>buffer occupancy</i> $B(m)$ on the trellis diagram for each branch. For branches with identical $B(m)$, keep the one with the lowest cumulative energy
5.	Proceed to job $m+1$ (i.e., go to Step 3)
6.	Find the path resulting in the lowest cumulative energy
7.	Assign the frequency for that job according to the path found in Step 6
8.	Proceed to job $j+1$ (i.e., go to Step 1)

TABLE V
GENERATING THE SET OF POWER-FREQUENCY TUPLES THAT RESULT IN CONVEX E-D POINTS

1.	For the set of ordered $(p_k, f_k) \in (\mathbf{P}, \mathbf{F})$ pairs; $1 \leq k \leq N-1$, initialize $k=1$.
2.	If $\lambda_k = \frac{p_{k+1} \cdot f_k - p_k \cdot f_{k+1}}{f_k - f_{k+1}} < \lambda_{k+1} = \frac{p_{k+2} \cdot f_{k+1} - p_{k+1} \cdot f_{k+2}}{f_{k+1} - f_{k+2}}$, remove (p_k, f_k) from set (\mathbf{P}, \mathbf{F}) , go to Step 1.
3.	Else, proceed to the next (p, f) tuple (i.e., set $k = k+1$) and go to Step 2.

TABLE VI
LOW COMPLEXITY ALGORITHM 1

1.	Choose set of frequency-power tuples that create convex E-D points according to Table V.
2.	For each job j ; $1 \leq j \leq M$,
3.	Find the frequency according to Eq-24, execute at that frequency.
4.	Proceed to job, i.e., set $j = j+1$.

TABLE VII
LOW COMPLEXITY ALGORITHM 2

1.	Choose set of frequency-power tuples that create convex E-D points according to Table V
2.	For each look-ahead window of size W ,
3.	For each job j ; $1 \leq j \leq W$ within the window,
4.	Find the frequency according to Eq-24.
5.	Execute the job with the assigned frequency and check the buffer state $B(j)$.
6.	If $B(j) \leq \beta_1$ or $B(j) \geq \beta_2$, change the frequency (i.e., go to Step 4), and proceed to next job in the window, i.e., set $j = j+1$.
7.	Else, continue with same frequency (i.e., go to Step 5), and proceed to next job in the window, i.e., set $j = j+1$.
8.	Proceed to the next window

of size W instead of performing it for every job, as explained in Table VII. A least complex approximation, Algorithm 3, would be changing the frequency only when there is a risk of overflow or underflow. Then, the buffer occupancy, $B(j)$, will oscillate

between β_1 and β_2 . Algorithm 3 may be also preferred when the complexity (time or power) cost of changing the frequency is not negligible. The complexity of this algorithm is less than $2M$ flops for a total of M jobs.

TABLE VIII
 LOW COMPLEXITY ALGORITHM 3

1.	Choose set of frequency-power tuples that create convex E-D points according to Table V.
2.	For job j ; $1 \leq j \leq M$,
3.	Find the frequency according to Eq-24.
4.	Execute the job with the assigned frequency and check the buffer state $B(j)$.
5.	If $B(j) \leq \beta_1$ or $B(j) \geq \beta_2$, change the frequency (i.e., go to Step 3), and proceed to next job, i.e., set $j = j + 1$.
6.	Else, continue with same frequency (i.e., go to Step 4), and proceed to next job, i.e., set $j = j + 1$.

 TABLE IX
 OVERVIEW OF THE JOINT COMPLEXITY ESTIMATION AND DVS OPTIMIZATION

1.	GCM Generation:Generate generic complexity metrics g (GCM)at the encoder and tag the bitstream.
2.	Initialization:Set $w_{k,t}^1(0) = 0; 1 \leq t \leq T, 1 \leq k \leq N$.
3.	For job j set $w_{k,t}^j(0) = w_{k,t}^{j-1}(L_{t,k}^{j-1} - 1)$, estimate complexity of next M jobs according to Eq-6.
4.	Update buffer occupancy $B(j)$ according to Eq-15.
5.	Perform one of the proposed optimization algorithms to find $f(j)$, start executing job
6.	For each DU i ; $0 \leq i \leq L_{t,k}^j$,
7.	Monitor the real complexity $rc(i)$, update the predictors of each complexity function
8.	Proceed to next job, i.e., set $j = j + 1$.

 TABLE X
 CONVENTIONAL DVS

1.	For every job j ; $1 \leq j \leq M$, arriving with T_{\max} intervals
2.	Create a worst case complexity estimate for each job $c(j)$
3.	Execute at frequency $f(j) = c(j)/T_{\max}$, rounded to the closest higher frequency
4.	When finish go to idle time, wait for the next job

E. Overview and Overhead of the Combined Complexity Estimation -Optimization Algorithm

As explained in Section II-C, we iterate the linear estimator of complexity within the job but we update the complexity estimates of the next W or M jobs (depending on the optimization algorithm) only at job boundaries since DVS optimization is performed at job boundaries. The overview of combined complexity estimation-DVS optimization can be seen in Table IX.

The proposed optimization algorithms have very low complexity and the dominant factor in total complexity of the proposed joint complexity estimation-DVS optimization is the complexity estimation part.

F. Relation to the Conventional DVS Algorithms

The major difference of this work and the conventional DVS algorithms [5]–[8] is that previous works only investigated the frequency/power adaptation to the fixed time allocations within a single job execution (intra-job), whereas we propose an inter-job optimization framework. In that sense, previous works on multimedia DVS [5], [6] can be seen as *complementary* to the proposed proactive DVS method.

Previous works assume job arrival times are periodic with period T_{\max} , and access to only one job in one period is allowed. This approach aims to optimize the operating level in a greedy fashion, given fixed deadlines; it tries to match the frequency to uniformly allocated time slots in an adaptive manner. Here, the delay constraint is

$$t(j) < T_{\max}; \quad \forall j : 1 \leq j \leq M. \quad (26)$$

Then, the optimal operating level is the one corresponding to the frequency, $f(i) = c(i)/T_{\max}$, as this will yield the minimum energy spent due to convex nature of the power-frequency function. We call this method *conventional DVS* [4], [7]. This algorithm is illustrated in Table X.

Note that, the worst case based allocation may waste energy whereas average based allocation may result in job misses (i.e., frame freezes). To this effect, an intra-job adaptation mechanism is proposed in [5], [6] called *statistical DVS*. In this method, each job starts slowly and accelerates as it progresses; frequency is increased when the job consumes its cycles corresponding to that frequency. Since most jobs require much less than their worst case complexity, this intra-job adaptation yields less idle

TABLE XI
INTRA-JOB ADAPTATION IN CONVENTIONAL DVS (STATISTICAL DVS)

1.	Create the histogram of real (observed) complexities for each frame type based on the previous frames.
2.	For every frame (job) j ; $1 \leq j \leq M$,
3.	Create an increasing set of frequency schedule $\{f(1), \dots, f(K)\}$ corresponding to worst case complexity groups $\{c(1), \dots, c(K)\}$
4.	For each group m , $1 \leq m \leq K$,
5.	Execute at frequency $f(m)$ for $c(m)$ cycles.
6.	If job is finished or deadline is reached stop go to Step 2, else set $m = m + 1$ and go to Step 4

time compared to conventional DVS without any intra-job adaptation. The statistical DVS algorithm is given in Table XI.

Frequency-complexity schedule is created based on the histogram of complexities of frames within the same frame type. Statistical DVS adjusts the execution frequency for each process job based on its demand distribution. The frequency schedule assumes an ideal CPU that supports a continuous range of frequencies with power proportional to the cube of the frequency ($p \propto f^3$). Then, the calculated frequency is rounded to the available frequencies. Skipping the derivation for space constraints, if function $F(b)$ is the cumulative distribution function (cdf) of complexity of group b within the job, i.e., if the probability of complexity being smaller than $c(b)$ is $F(b)$, then the frequency assignment for the group m is given as [5], [6]:

$$f(m) = \frac{\sum_{j=1}^K c(j) \sqrt[3]{1 - F(j)}}{T_{\max} \sqrt[3]{1 - F(m)}}. \quad (27)$$

In Section IV, we compare the proposed proactive DVS algorithm to these conventional DVS methods in terms of energy savings and the number of frequency changes.

IV. RESULTS

A. Complexity Estimation

In this section, we present comparative results on the complexity estimation. We quantify the estimation performance gain that we can obtain by considering generic complexity metrics. Note that the previous algorithms [5]–[8] are based on finding the worst case execution time, since the conventional DVS algorithms require completing the jobs before their deadline. In our experiments, we used four different test sequences, *foreman*, *mobile*, *coastguard* and *silence*, at CIF resolution and at a frame rate of 30 fps. We used a scalable wavelet video coder [27] that utilizes motion-compensated temporal filtering using the 5/3 Daubechies filter, which was analyzed in terms of job definitions in Section II-D Example 2. We used four level temporal and five level spatial decompositions, in spatial domain ME/MC mode. We generated two sets of decoded video with high and low complexities at two rates, 512 kbps and 1024 kbps. High complexity compression includes adaptive update and quarter pixel motion compensation whereas the update step is skipped and only half pixel MC is used for the low complexity case.

The proposed complexity estimation method is compared to the conventional methods based on the complexity of frames with the same frame type [6]. The statistical *worst case* method described in [6] monitors the real complexity and determines a complexity estimate such that some percent (ρ) of the complexity values observed till instant of estimation lie below that estimate. In our experiments, as in [6], we set $\rho = 95\%$. The *average* complexity estimation is obtained by averaging the complexities of all of the previous frames with the same frame type. We implemented the proposed complexity estimation method as described in Section II. We used spatial subband granularity (DU) for ED and IT complexity functions and macro-block granularity (DU) for IO and MC complexity functions. The update step size of the predictor is set to $\mu = 0.1$. We used job definitions explained in Section II-D Example-2 for every estimation method including the conventional methods, and comparison is in job granularity.

The comparative results in Tables XII and XIII show the superiority of the proposed complexity estimation method over the conventional methods like averaging or worst case estimation for different bitrates, video content and complexity. The proposed method yields significantly less estimation error than the averaging method (8.5% less error for high complexity encoded videos and 6.63% less error for low complexity encoded videos).

B. DVS Comparative Results

In this section we compare the proposed DVS method to conventional DVS methods. We use four different test sequences, *foreman*, *mobile*, *coastguard* and *silence*, 256 frames at CIF resolution and frame rate 30 fps, encoded at two different compression specifications (low complexity and high complexity as explained in Part A), and decode at two different rates, 512 kbps and 1024 kbps. To obtain statistically meaningful results, we concatenated the resulting 16 videos in 12 different orders, resulting in a set of 12 long videos with 3072 frames each. We present the average results of 12 videos with different decoding traces. We scaled the real complexity values to make real time decoding possible for the used frequency range, similar to the approach in [16]. Power and frequency values that we used are shown in Fig. 8(a). We set the coefficient k for the risk explained in Section II-C as $k = 2$. We present results on i) the comparison of the different optimization algorithms (Algorithms

TABLE XII
 SCALED ABSOLUTE ERRORS $((c_{\text{real}} - c_{\text{estimate}}/c_{\text{real}}) \times 100)$ IN DIFFERENT COMPLEXITY ESTIMATION METHODS FOR HIGH COMPLEXITY ENCODED VIDEOS

Silence	Mobile		Foreman		Coastguard			
	512kbps	1024kbps	512kbps	1024kbps	512kbps	1024kbps		
Proposed	6.61	6.75	9.87	9.80	6.46	6.51	22.45	21.85
Average	14.14	13.78	13.78	13.53	20.62	20.31	23.34	22.89
Worst Case	25.19	24.10	17.91	17.43	39.69	39.27	38.26	37.43

TABLE XIII
 SCALED ABSOLUTE ERRORS $((c_{\text{real}} - c_{\text{estimate}}/c_{\text{real}}) \times 100)$ IN DIFFERENT COMPLEXITY ESTIMATION METHODS FOR LOW COMPLEXITY ENCODED VIDEOS

Silence	Mobile		Foreman		Coastguard			
	512kbps	1024kbps	512kbps	1024kbps	512kbps	1024kbps		
Proposed	7.72	8.85	10.58	11.18	11.08	10.97	23.34	21.29
Average	16.42	15.60	15.02	14.30	27.70	26.56	21.98	20.52
Worst Case	37.32	34.41	21.74	21.04	50.22	48.81	35.14	32.47

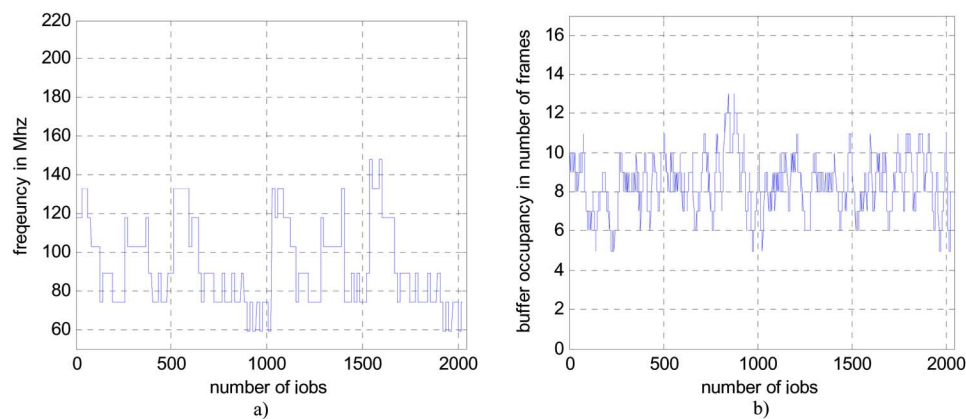


Fig. 10. (a) Frequency assignment and (b) buffer fullness for Algorithm 2 for one test video.

1–2–3); ii) the comparison of DVS with different buffer sizes B_{max} and look-ahead window size W ; iii) the comparison of the proposed and conventional DVS methods; iv) the investigation of the complexity estimation accuracy in the proposed DVS method v) the investigation of combined inter-intra job (intermediate frequencies) DVS within the proposed framework.

We compare the fast algorithms, Algorithm 1, 2, and 3 in terms of the buffer utilization, the number of frequency transitions and energy savings. In our cost function in (13), we assumed frequency transitions have no complexity (time or energy) cost. However, in practice, the cost of frequency transition may not be negligible as reported in [7]. In the following, we present energy savings, the number of frequency changes and complexities of the algorithms given in Section III along with the comparisons to the conventional DVS algorithms.

Figs. 9–11 shows the buffer utilization and frequency changes within Algorithm 1, Algorithm 2, and Algorithm 3, respectively. As can be seen from the frequency assignment plot, the highest

number of frequency changes occur in Algorithm 1 compared to other algorithms whereas the least buffer level variation is observed.

Algorithm 3 is the most conservative in terms of the number of frequency changes and as expected provides the least energy savings among the proposed three algorithms. The performance of Algorithm 2 is between that of Algorithm 1 and 3 in the sense of energy savings, buffer utilization, and the number of frequency changes.

As the buffer size and the look-ahead window size increases, energy savings of the proposed buffer controlled DVS algorithms increase as expected intuitively. The gap between the energy savings of Algorithm 1 and Algorithm 3 decrease as the buffer size gets large. This result shows the tradeoff between the buffer size, energy savings and number of frequency changes. If the buffer size is large, energy savings close to the savings of Algorithm 1 can be achieved with less frequency transitions using Algorithm 3. However, when the buffer size is small, the difference in en-

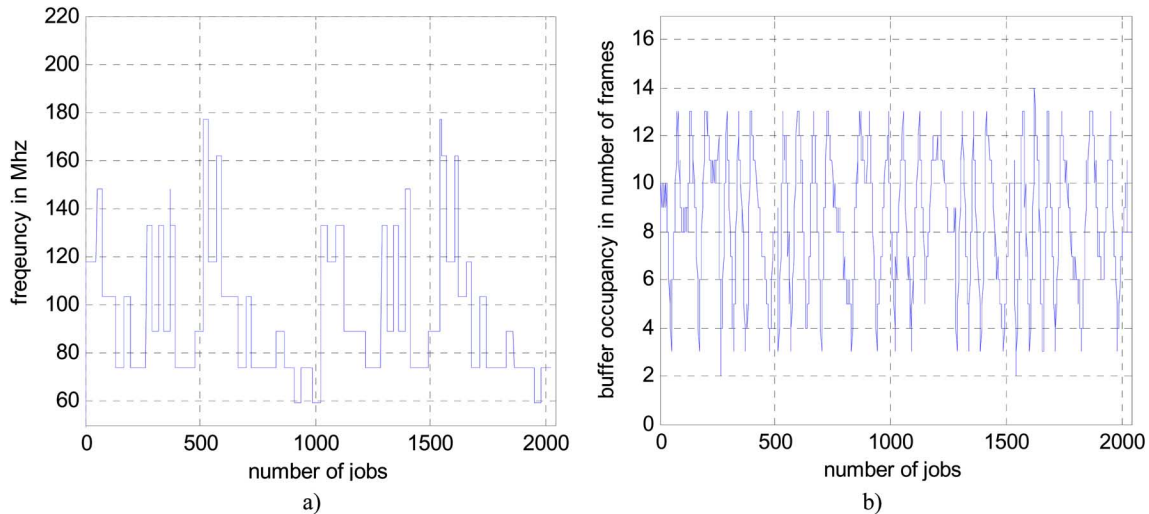


Fig. 11. (a) Frequency assignment and (b) buffer fullness for Algorithm 3 for one test video.

TABLE XVI

COMPARISON OF DIFFERENT DVS ALGORITHMS IN TERMS OF SCALED ENERGY SPENT AND NUMBER OF FREQUENCY CHANGES PER JOB. BENCHMARK (%100 ENERGY SPENT CASE) IS NO-DVS SCENARIO $B_{\max} = 8$ $W = 8$ $B_{\max} = 16$ $W = 16$

	Scaled Energy	# freq. changes	Scaled Energy	#of freq. changes
Conventional Lower Bound	45.21	1519.8	45.21	1519.8
Conventional	60.43	1688.5	60.43	1688.5
Conventional Statistical	54.05	5338.8	54.05	5338.8
Optimal Proactive	36.93	1753.7	36.63	1747.3
Algorithm 1	40.36	655.91	40.23	545.67
Algorithm 2	40.60	161.0	40.58	82.75
Algorithm 3	42.35	121.17	41.66	57.92

TABLE XV

EFFECT OF THE COMPLEXITY ESTIMATION ACCURACY ON BUFFER CONTROLLED DVS

	$B_{\max} = 8$ $W = 8$		$B_{\max} = 16$ $W = 16$	
	Scaled Energy	# freq. changes	Scaled Energy	#of freq. changes
Exact (Oracle)	40.34	675.25	40.20	562.5
Algorithm 1	40.36	655.91	40.23	545.67

ergy savings of Algorithm 3 and Algorithm 1 can be significant. We also simulated the conventional DVS methods in three ways. The first method, as explained in Section III-D, Table X, utilizes a statistical worst case complexity estimate denoted here as *Conventional*. Another method is the optimal conventional DVS for benchmark purposes which is named as *Conventional Lower Bound*. We assumed the complexities of each job are known *a priori* and find the optimal theoretical DVS gain achievable by conventional DVS methods. When the complexities are known, the optimal frequency is the minimum of frequency from the set of given frequencies which completes the job before its deadline. *Conventional Statistical DVS* is the state-of-the-art DVS method for multimedia processing [6], given in Table XI where we set the number of partitions in one job to $K = 3$ with groups that have identical complexities, i.e., if complexity of the job is c then $c(1) = c(2) = c(3) = c/3$, similar to [6]. *Optimal Proactive* is the optimal dynamic programming based algorithm given in Table IV, assuming an exact knowledge of complexities before the actual decoding is performed. Optimal Proactive DVS shows the limit of energy savings given the buffer size. Table XIV illustrates the comparative energy savings and number of frequency transitions for different DVS methods.

As the results show, all of the proposed methods significantly outperform the conventional algorithms in terms of energy savings and the number of frequency changes. Note that, the proposed method provides energy savings exceeding the upper bound of the conventional methods which is based on the exact knowledge of the complexity. This result clearly shows the superiority of the proposed proactive DVS method. Note that, buffer size of 16 decoded frames is enough for the proposed method, although larger buffer sizes increase energy savings but only marginally. In the following, we present the effect of several factors on the proposed and conventional DVS methods.

1) *Effect of Complexity Estimation Accuracy in Buffer Controlled DVS*: To investigate the robustness of the proposed DVS method to complexity estimate mismatches, we simulated the proposed method with three other complexity estimates. The first estimate is the oracle case where the complexity is exactly known beforehand. We use this estimate to investigate whether there may be possible energy savings with a more accurate estimate. The second case is using the average estimate explained in complexity estimation results. Third complexity estimation method is the worst case estimate. Table XV shows the

TABLE XVI
ENERGY AND NUMBER OF FREQUENCY CHANGES USING A PIECEWISE INTERPOLATED SET OF FREQUENCIES FOR $B_{\max} = 16$ AND $W = 16$

	<i>original set of frequencies</i>		<i>interpolated set of frequencies</i>	
	Scaled Energy	# freq. changes	Scaled Energy	#of freq. changes
Conventional	60.43	1688.5	58.65	1931.8
Alg-1	40.23	545.67	40.25	469.67
Alg-2	40.58	82.75	40.60	86.5
Alg-3	41.66	57.92	41.73	60.00

effect of complexity estimation accuracy on the proposed DVS method.

The estimate using averaging³ and worst case were shown to be too coarse for our application, they did not satisfy the buffer constraints, for buffer sizes as much as $B_{\max} = 64$ ⁴. This experiment shows that using an accurate estimate is vital for even buffer controlled DVS. Hence, there is a tradeoff between buffer size and complexity estimation accuracy, as the estimation error increases a larger buffer should be used to avoid job misses. Also, notice that the proposed complexity estimation method is good enough for buffer controlled DVS: As expected the exact estimate (oracle) improved the results in the sense of energy savings but only marginally.

2) *Experimental Validation of not Using the Intermediate Points:* To validate our claim of not using the intermediate power-frequency values, we hypothetically generated additional points by piecewise linear interpolation of given power-frequency points. As stated in Proposition 1, this interpolation corresponds to changing the frequency within the job. We repeated the same experiments with this enhanced set of power-frequency values, given in Fig. 8(b). From the results given in Table XVI, we can see that intermediate points which improve the performance of conventional DVS methods, are not very useful for buffer controlled DVS. Although we present the results for only $B_{\max} = 16$ $W = 16$ case, we observed similar results for different buffer and look-ahead window sizes. The main reason is the proposed method can accommodate instantaneous delay with the help of the buffer where as the conventional methods cannot tolerate any instantaneous delay, hence intermediate frequencies provide improvement over conventional methods.

V. CONCLUSION

In this paper, we proposed a novel DVS method for video decoding that considers a complexity model and video compression algorithm specifications. The proposed DVS algorithm achieves energy savings that exceed the upper bound of energy savings that conventional DVS algorithms can provide. We explored the fundamental energy vs. delay tradeoff for video decoding systems and proposed new tradeoffs such as buffer

³We did not update the averaging method for scene changes, i.e., more accurate estimates can be achieved by averaging if scene changes are considered. Since the purpose of this experiment was to show the degradation of the DVS performance when estimation is not accurate, we did not consider scene changes for averaging.

⁴Look-ahead window size is set to 8, $W = 8$ for every experiment, since a GOP size of 16 frames, which is used in our experiments, correspond to 8 jobs in one GOP. Also, since there is no error estimation, β_1 and β_2 are set heuristically as: $\beta_1 = 0.1B_{\max}$ and $\beta_2 = 0.9B_{\max}$.

size vs. complexity estimation accuracy, energy savings and the number of frequency changes that the platform can handle. We also derived several important properties of the proposed DVS framework that enable highly efficient energy optimizations. The principles and the main trade-offs proposed can be applied to video encoding with small modifications. We note that, for real time encoding scenario, the DVS problem is much more complex because, complexity of encoding may also be adjustable in real-time, by utilizing a complexity scalable encoder. In the future, we will analyze DVS for real time encoding and explore the effect of the leakage (passive) power on DVS for multimedia.

REFERENCES

- [1] Intel XScale Technology Intel Inc [Online]. Available: <http://www.intel.com/design/intelxscale>
- [2] AMD PowerNow!™ Technology Platform Design Guide for Embedded Processors AMD, Inc. [Online]. Available: <http://www.amd.com/epd/processors>
- [3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer, 1997.
- [4] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int. Symp. Low-Power Electronics and Design.*, Monterey, CA, 1998.
- [5] J. Lorch and A. Smith, "Improving dynamic voltage scaling algorithms with PACE," in *Proc. ACM SIGMETRICS 2001 Conf.*, Jun. 2001.
- [6] W. Yuan, K. Nahrstedt, S. Adve, D. Jones, and R. Kravets, "GRACE: Cross-layer adaptation for multimedia quality and battery energy," *IEEE Trans. Mobile Comput.*, to be published.
- [7] V. Raghunathan, C. Pereira, M. Srivastava, and R. Gupta, "Energy aware wireless systems with adaptive power-fidelity tradeoffs," *IEEE Tran. VLSI Syst.*, vol. 13, no. 2, Feb. 2005.
- [8] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta, "An overview of the competitive and adversarial approaches to designing dynamic power management strategies," *IEEE Trans. VLSI Syst.*, vol. 13, no. 12, pp. 1349–1362, Dec. 2005.
- [9] M. Ravasi and M. Mattavelli, "High-abstraction level complexity analysis and memory architecture simulations of multimedia algorithms," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 15, no. 5, pp. 673–684, May 2005.
- [10] A. C. Bavier, A. B. Montz, and L. Peterson, "Predicting MPEG execution times," in *Proc. ACM SIGMETRICS*, 1998, pp. 131–140.
- [11] G. Landge, M. van der Schaar, and V. Akella, "Complexity metric driven energy optimization framework for implementing MPEG-21 scalable video decoders," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, Philadelphia, PA, Apr. 2005.
- [12] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560–576, Jul. 2003.
- [13] Y. Andreopoulos and M. van der Schaar, "Complexity-constrained video bitstream shaping," *IEEE Trans. Signal Process.*, to be published.
- [14] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC baseline profile decoder complexity analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 704–716, Jul. 2003.
- [15] M. van der Schaar and Y. Andreopoulos, "Rate-distortion-complexity modeling for network and receiver aware adaptation," *IEEE Trans. Multimedia*, vol. 7, no. 3, pp. 471–479, Jun. 2005.
- [16] S. Regunathan, P. A. Chou, and J. Ribas-Corbera, "A generalized video complexity verifier for flexible decoding," in *Proc. IEEE Int. Conf. Image Processing*, Sep. 2003, vol. 3, pp. 289–292.

- [17] A. Ortega, K. Ramchandran, and M. Vetterli, "Optimal trellis-based buffered compression and fast approximations," *IEEE Trans. Image Processing*, vol. 3, no. 1, Jan. 1994.
- [18] Y. Andreopoulos and M. van der Schaar, "Adaptive linear prediction for resource estimation of video decoding," *IEEE Trans. Circuits Syst. Video Technology*, to be published.
- [19] A. Sinha and A. P. Chandrakasan, "Jouletrack: A web based tool for software energy profiling," in *Proc. IEEE/ACM DAC*, 2001, pp. 220–225.
- [20] P. A. Dinda and D. R. O'Hallaron, "An evaluation of linear models for host load prediction," in *Proc. IEEE InV. Symp. High Perf. Distrib. Comput.*, Aug. 1999, pp. 87–96.
- [21] A. H. Sayed, *Fundamentals of Adaptive Filtering*. New York: Wiley, 2003.
- [22] D. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, MA: Athena Scientific, 1995, vol. 2.
- [23] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 36, no. 9, pp. 1445–1453, Sep. 1988.
- [24] J. R. Ohm, M. van der Schaar, and J. W. Woods, "Interframe wavelet coding—Motion picture representation for universal scalability," *Signal Process.: Image Commun.*, vol. 19, no. 9, pp. 877–908, Oct. 2004.
- [25] P. A. Chou and Z. Miao, "Rate-distortion optimized streaming of packetized media," *IEEE Trans. Multimedia*, to be published.
- [26] J. L. W. V. Jensen, "Sur les fonctions convexes et les inegalites entre les valeurs moyennes," *Acta Math.*, vol. 30, pp. 175–193, 1906.
- [27] Y. Andreopoulos, A. Munteanu, J. Barbarien, M. van der Schaar, J. Cornelis, and P. Schelkens, "In-band motion compensated temporal filtering," *Signal Process.: Image Commun.*, vol. 19, no. 7, pp. 653–673, Aug. 2004.
- [28] Intel StrongARM Processors Intel Inc [Online]. Available: <http://developer.intel.com/design/strong/>

Emrah Akyol received the B.S. degree in electrical engineering in 2003 from Bilkent University, Ankara, Turkey and the M.S. degree in electrical and computer engineering in 2005 Koc University, Istanbul, Turkey.

He is currently pursuing the Ph.D. degree the University of California, Los Angeles.

He is an intern at NTT DoCoMo Communication Laboratories, Palo Alto, CA. His research areas include image-video compression and transmission. Between June 2006 and November 2006, he was a research intern at HP Labs, Palo Alto, where he worked on complexity related problems of media compression. He has coauthored several publications and one pending patent application.

Mihaela van der Schaar (SM'04) is currently an Associate Professor in the Electrical Engineering Department at the University of California, Los Angeles (UCLA). Since 1999, she was an active participant to the ISO MPEG standard to which she made more than 50 contributions and for which she received three ISO recognition awards.

Dr. van der Schaar received the NSF CAREER Award in 2004, IBM Faculty Award in 2005, Okawa Foundation Award in 2006, Best IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY Paper Award in 2005, and Most Cited Paper Award from EURASIP Journal Signal Processing: Image Communication between the years 2004–2006. She holds 28 granted U.S. patents. She is currently an associate editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, IEEE SIGNAL PROCESSING LETTERS and *IEEE Signal Processing e-Newsletter*. She is also the editor (with Phil Chou) of the book *Multimedia over IP and Wireless Networks: Compression, Networking, and Systems*.