Markov Decision Process Based Energy-Efficient On-Line Scheduling for Slice-Parallel Video Decoders on Multicore Systems

Nicholas Mastronarde, *Member, IEEE*, Karim Kanoun, David Atienza, *Member, IEEE*, Pascal Frossard, *Senior Member, IEEE*, and Mihaela van der Schaar, *Fellow, IEEE*

***Abstract*—We consider the problem of energy-efficient on-line scheduling for slice-parallel video decoders on multicore systems with Dynamic Voltage Frequency Scaling (DVFS) enabled processors. In the past, scheduling and DVFS policies in multi-core systems have been formulated heuristically due to the inherent complexity of the on-line multicore scheduling problem. The key contribution of this paper is that we rigorously formulate the problem as a Markov decision process (MDP), which simultaneously takes into account the on-line scheduling and per-core DVFS capabilities; the power consumption of the processor cores and caches; and the loss tolerant and dynamic nature of the video decoder. The objective of the MDP is to minimize long-term power consumption subject to a minimum Quality of Service (QoS) constraint related to the decoder's throughput. We evaluate the proposed on-line scheduling algorithm in Matlab using realistic video decoding traces generated from a cycle-accurate multiprocessor ARM simulator.**

***Index Terms*—Video decoding, multicore scheduling, dynamic voltage frequency scaling, energy-efficient scheduling, Quality-of-Service, Markov decision process.**

I. Introduction

HIGH-QUALITY video decoding imposes unprecedented performance requirements on energy-constrained mobile devices. To address the competing requirements of high performance and energy-efficiency, embedded mobile multimedia device manufactures have recently adopted MPSoC (multiprocessor system-on-chip) architectures that support Dynamic Voltage Frequency Scaling (DVFS) and Dynamic Power Management (DPM) technologies. DVFS enables dynamic adaptation of each processor's frequency and voltage, and can be exploited to reduce power consumption when the maximum frequency of operation is not required to meet the deadlines of a certain set of tasks [8]. Meanwhile, DPM enables system components such as processors to be dynamically switched on and off when they are not needed [15].

Despite improvements in mobile device technology, energy-efficient multicore scheduling for video decoding remains a challenging problem for several reasons. First, video decoding applications have intense and time-varying stochastic workloads, which have worst-case execution times that are significantly larger than the average case. Second, video applications have sophisticated dependency structures due to predictive coding. These dependency structures, which can be modeled as directed acyclic graphs (DAGs), not only result in different frames having different priorities, but also make it difficult to balance loads across the cores, which is important for energy efficiency [1]. Finally, video applications often have stringent delay constraints, but are considered soft real-time applications [18]. In other words, video frames should meet their deadlines, but when they do not, the application quality (e.g., decoded video frame rate) is reduced.

During the last decade, many energy-efficient multicore scheduling algorithms that exploit DVFS and/or DPM have been proposed, e.g., [2]–[4], [6], [7], [9], [10]. The Largest Task First with Dynamic Power Management (LTF-DPM) algorithm in [3] assumes that frame decoding deadlines are equally spaced in time (e.g., 33 ms apart for 30 frame per second video), and therefore does not support video group of pictures (GOP) structures with B frames; moreover, LTF-DPM will typically have looser deadline constraints than our proposed algorithm because it assigns groups of frames a common "weak" deadline. The Scheduling 2D and Stochastic Scheduling 2D algorithms in [6] and [7], respectively, both consider a periodic directed acyclic graph (DAG) application model that requires a "source" and "sink" node in each period, making the algorithms incompatible with GOP structures where the last B frame in a GOP depends on the I frame in the next GOP (e.g., an IBPB GOP).The Variation Aware Time Budgeting (Var-TB) algorithm in [10] uses a DAG task model and allows for arbitrary complexity distributions; however, the author's propose using a functional partitioning algorithm for parallelizing the video decoder (e.g., pipelining decoder sub-functions such as inverse DCT and motion compensation on different cores). Functional partitioning is known to be suboptimal because

Manuscript received December 03, 2011; revised May 21, 2012; accepted July 08, 2012. Date of publication December 04, 2012; date of current version January 15, 2013. The work of M. van der Schaar and N. Mastronarde was supported in part by the National Science Foundation under Award CNS-0509522. The work of D. Atienza and K. Kanoun was supported in part by the Swiss National Science Foundation, under Grant 200021-127282, and a research grant funded by CSEM SA. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Yiannis Andreopoulos.

N. Mastronarde is with the Department of Electrical Engineering, State University of New York at Buffalo, Buffalo, NY 14260 USA. This work was done in part while he was at the University of California at Los Angeles (UCLA), Los Angeles, CA 90095-1594 USA (e-mail: nmastron@buffalo.edu).

K. Kanoun, D. Atienza, and P. Frossard are with the Institute of Electrical Engineering, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland (e-mail: karim.kanoun@epfl.ch; david.atienza@epfl.ch; pascal.frossard@epfl.ch).

M. van der Schaar is with the Department of Electrical Engineering, University of California at Los Angeles (UCLA), Los Angeles, CA 90095-1594 USA (e-mail: 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.2012.2231668

moving data between cores requires a lot of memory bandwidth [17]. It is shown in [17] that parallelization approaches based on data partitioning (e.g., mapping different frames, slices, or macroblocks to different processors) are superior to functional partitioning approaches. The Global Earliest Deadline First Online DVFS (GEDF-OLDVFS) algorithm in [9] is inappropriate for predictively coded video applications because it assumes that tasks are independent. The so-called SpringS algorithm in [4] uses a task-level software pipelining algorithm called RDAG [5] to transform a periodic dependent task graph (expressed as a DAG) into a set of tasks that can be pipelined on parallel processors. Unfortunately, if this technique is applied to video decoding applications, it will require retiming delays proportional to the GOP size, which may be arbitrarily large. Finally, all of the aforementioned research takes into account processing energy, but does not take into account the power consumption of different cache levels in the memory hierarchy despite multimedia applications being data-access dominated [11].

In summary, although many important advancements have been made, there is still no rigorous multicore scheduling solution that simultaneously considers per-core DVFS capabilities; dynamic processor assignment; the separate power consumption of the processor cores and caches; and loss-tolerant tasks with different complexity distributions, DAG dependency structures (i.e., precedence constraints), and stringent, but soft real-time, constraints. The contributions of this paper are as follows:

- We rigorously formulate the multi-core scheduling problem using a Markov decision process (MDP) that considers the above mentioned properties of the multi-core system and video decoding application. The MDP enables the system to optimally trade-off long-term power and performance, where the performance is measured in terms of a Quality of Service (QoS) metric that is related to the decoder's throughput.
- The MDP solution requires complexity that exponentially increases with both the number of processors and the number of frames in a short look-ahead window. To mitigate this complexity, we propose a novel two-level scheduler. The first-level scheduler determines scheduling and DVFS policies for each frame using frame-level MDPs, which account for the coupling between the optimal policies of parent frames and their childrens' optimal policies. The second-level scheduler decides the final frame-to-processor and frequency-to-processor mappings at run-time, ensuring that certain system constraints are satisfied.
- We validate the proposed algorithm in Matlab using video decoder trace statistics generated from an H.264/AVC decoder that we implemented on a cycle-accurate multiprocessor ARM (MPARM) simulator [14].

The remainder of the paper is organized as follows. We introduce the system and application models in Section II and formulate the on-line multi-core scheduling problem as an MDP. In Section III, we propose a lower complexity solution by approximating the original MDP problem with a two-level scheduler. In Section IV, we present our experimental results. We conclude in Section V.

II. Problem Formulation

We consider the problem of energy-efficient slice-parallel video decoding in a time slotted multicore system, where time is divided into slots of (equal) duration Δt seconds indexed by $t \in \mathbb{N}$. We assume that there are M processors, which we index by $j \in \{1, \ldots, M\}$. In Section II-A, we describe seven important video data attributes. In Section II-B, we propose a sophisticated Markovian traffic model for characterizing video decoding workloads that accounts for the video data attributes introduced in Section II-A. In Sections II-C–II-E we describe the scheduling and frequency actions, the evolution of the video traffic/workload, and the power and Quality of Service (QoS) metrics used in our optimization. In Section II-F, we formulate the multicore scheduling problem as a Markov decision process (MDP).

A. Video Data Attributes

We model the encoded video bitstream as a sequence of compressed data units with different decoding and display deadlines, source-coding dependencies, priorities, and decoding complexity distributions. In this paper, we assume that a data unit corresponds to one video slice, which is a subset of a video frame that can be decoded independently of other slices within the same frame (see [12] for a good discussion about slice-parallel video decoding). We assume that the video is encoded using a fixed, periodic, GOP structure that contains K frames and lasts a period of T time slots of duration Δt. The set of frames within GOP $g \in \mathbb{N}$ is denoted by $\mathcal{V}^g \triangleq \{v_1^g, v_2^g, \ldots, v_K^g\}$ and the set of all frames is denoted by $\mathcal{V} \triangleq \bigcup_{g \in \mathbb{N}} \mathcal{V}^g$. Each frame v_k^g is characterized by seven attributes:

1. *Type* : Frame v_k^g is an I, P, or B frame. We denote the operator extracting the frame type by $\mathrm{type}(v_k^g)$.
2. *Number of slices*: Frame v_k^g is composed of $l^{v_k^g} \in \{1, \ldots, l^{\max}\}$ slices, where $l^{v_k^g}$ is assumed to be fixed and $l^{\max}$ is the maximum number of slices allowed in any single video frame. The number of slices $l^{v_k^g}$ is determined by the encoder [19].
3. *Decoding complexity*: Slices belonging to frame v_k^g have decoding complexity $w^{v_k^g}$ cycles. We assume that $w^{v_k^g}$ is an exponentially distributed i.i.d. random variable conditioned on the frame type with expectation $\mathbf{E}\left[w^{v_k^g}\right] = \beta^{\mathrm{type}(v_k^g)}$. The assumption of exponentially distributed complexity is inaccurate; however, it is necessary to make the MDP problem formulation tractable. We discuss why we make this assumption in Section II-C; however, due to space limitations, we refer the interested reader to Section 3.3 of our technical report [19] for a more detailed discussion about the assumption's consequences.
4. *Arrival time*: $t^{v_k^g}$ denotes the earliest time slot v_k^g can be decoded (i.e., its arrival time at the scheduler).
5. *Display deadline*: $d^{v_k^g,\mathrm{disp}}$ denotes the final time slot in which v_k^g must be decoded so it can be displayed.
6. *Decoding deadline*: $d^{v_k^g,\mathrm{dec}}$ denotes the final time slot in which v_k^g must be decoded so that frames that depend on it can be decoded before their display deadline. Note that $d^{v_k^g,\mathrm{dec}} \leq d^{v_k^g,\mathrm{disp}}$.

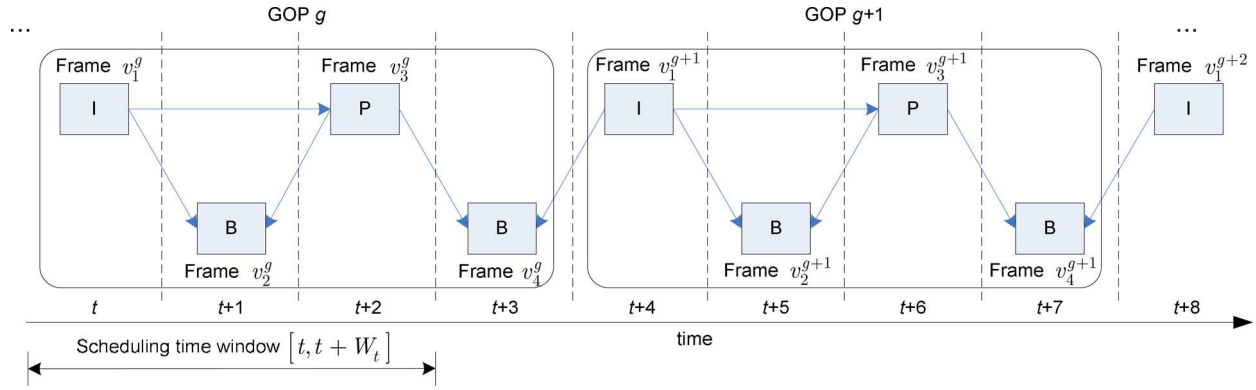

Fig. 1. Illustrative DAG dependencies for an IBPB GOP structure that contains $K = 4$ frames and lasts a period of $T = 4$ time slots of duration $\Delta t = 1/30$ seconds.

7. *Dependency*: The frames must be decoded in decoding order, which is dictated by the dependencies introduced by predictive coding (e.g., motion-compensation). In general, the dependencies among frames can be described by a directed acyclic graph (DAG), denoted by $DAG \triangleq \langle \mathcal{V}, \mathcal{E} \rangle$, with the nodes in $\mathcal{V}$ representing frames and the edges in $\mathcal{E}$ representing the dependencies among frames. We use the notation $v_{k'}^{g'} \prec v_k^g$ to indicate that frame v_k^g depends on frame $v_{k'}^{g'}$ (i.e., there exists a path directed from $v_{k'}^{g'}$ to v_k^g) and therefore v_k^g cannot be decoded until $v_{k'}^{g'}$ is decoded. We write $\left(v_{k'}^{g'}, v_k^g\right) \in \mathcal{E}$ if there is a directed arc emanating from frame $v_{k'}^{g'}$ and terminating at frame v_k^g, indicating that $v_{k'}^{g'}$ is an immediate parent of v_k^g.

These attributes are important because they determine which slices can be decoded, how long they will take to decode, when they need to be decoded, and what the penalty is for not decoding them on time. In the next subsection, we propose a Markovian traffic model that captures the above attributes, enabling us to rigorously formulate the multicore scheduling problem as an MDP.

B. Markovian Traffic Model

We define a *traffic state* $\mathcal{T}_t = (\mathcal{C}_t, \mathbf{x}_t, \mathbf{r}_t)$ to represent the video data that can potentially be decoded in time slot t. This traffic state comprises three components defined in the following paragraphs: the *current frame set* $\mathcal{C}_t \subset \mathcal{V}$, the *buffer state* $\mathbf{x}_t$, and the *dependency state* $\mathbf{r}_t$.

In time slot t, we assume that the set of frames whose deadlines are within the *scheduling time window* (STW) $[t, t+W_t]$ can be decoded. We define the current frame set as all the frames within the STW, i.e., $\mathcal{C}_t = \left\{v \in \mathcal{V} | d^{v,\text{disp}} \in \{t, t+1, \ldots, t+W_t\}\right\}$. Because the GOP structure is fixed and periodic, $\mathcal{C}_t$ is periodic with some period T. Frame v's arrival time t^v, display deadline $d^{v,\text{disp}}$, and decoding deadline $d^{v,\text{dec}}$ are fully determined by the periodic GOP structure. Specifically, it turns out that $t^v \triangleq \min\{t | v \in \mathcal{C}_t\}$, $d^{v,\text{disp}} \triangleq \max\{t | v \in \mathcal{C}_t\}$, and $d^{v,\text{dec}} \triangleq \min\left\{d^{u,\text{disp}} | (v,u) \in \mathcal{E}\right\}$. In words, a frame's arrival time (respectively, display deadline) is the first (respectively, last) time slot in which it appears in the current frame set, and a frame's decoding deadline is the minimum display deadline of its children. Note that the distinction between display and decoding deadlines is important because, even if a frame's decoding deadline is missed, which renders its children undecodable, it is still possible to decode the frame before its display deadline. Fig. 1 illustrates how the current frame sets are defined for a simple IBPB GOP structure. The following example illustrates one way to define the current frame sets for the GOP structure in Fig. 1.

Example: Current frame sets: Let $W_t = W_{t+2} = 2$ and $W_{t+1} = W_{t+3} = 3$. Using the GOP structure in Fig. 1, and a time slot duration of $\Delta t = 1/30$ s, the current frame sets defined by these scheduling time windows are $\mathcal{C}_t = \{v_1^g, v_2^g, v_3^g\}$, $\mathcal{C}_{t+1} = \left\{v_2^g, v_3^g, v_4^g, v_1^{g+1}\right\}$, $\mathcal{C}_{t+2} = \left\{v_3^g, v_4^g, v_1^{g+1}\right\}$, $\mathcal{C}_{t+3} = \left\{v_4^g, v_1^{g+1}, v_2^{g+1}, v_3^{g+1}\right\}$, and $\mathcal{C}_{t+4} = \left\{v_1^{g+1}, v_2^{g+1}, v_3^{g+1}\right\}$. Notice that the GOP structure is periodic with period $T = 4$ such that the current frame sets $\mathcal{C}_t$ and $\mathcal{C}_{t+T}$ contain frames in the same position of the GOP with the same underlying dependency structure.

We define the buffer state $\mathbf{x}_t = (x_t^v | v \in \mathcal{C}_t)$, where x_t^v denotes the number of slices of frame v awaiting decoding at time t. By definition, $x_t^v \leq l^v$, where l^v is the total number of slices belonging to frame v. Finally, the dependency state $\mathbf{r}_t \triangleq (r_t^v | v \in \mathcal{C}_t)$ defines whether or not each frame in the current frame set is decodable in time slot t. In particular, r_t^v is a binary variable that takes value 1 if all of frame v's dependencies are satisfied, i.e., if $x_{u,t} = 0$ for all $u \prec v$, and takes value 0 otherwise.

C. Scheduling Actions and Processor Frequencies

Let $y_t^{jv} \in \mathcal{Y} = \{0, 1\}$ denote the number of slices belonging to frame v that are scheduled on processor j at time t. For notational convenience, we define $\mathbf{Y}_t = \left[y_t^{jv}\right]_{jv}$, $\mathbf{y}_t^v = \left(y_t^{jv} | j \in \{1, \ldots, M\}\right)^T$, and $\mathbf{y}_t^j = \left(y_t^{jv} | v \in \mathcal{C}_t\right)$. There are three important constraints on the scheduling actions y_t^{jv} for all $j \in \{1, \ldots, M\}$ and $v \in \mathcal{C}_t$:

- *Buffer constraint*: $\sum_{j=1}^{M} y_t^{jv} \leq x_t^v$. In words, the total number of scheduled slices belonging to frame v cannot exceed the number of slices in frame v's buffer in slot t.
- *Processor constraint*: $\sum_{v \in \mathcal{C}_t} y_t^{jv} \leq 1$. In words, no more than one slice can be scheduled on processor j in slot t.

- *Dependency constraint*: If $r_t^v = 0$, then $\sum_{j=1}^{M} y_t^{jv} = 0$. In words, all of the vth frame's dependencies must be satisfied before slices belonging to it are scheduled to be decoded.

We assume that each processor can operate at a different frequency in each time slot to trade-off processing energy and delay. Let $\mathbf{f}_t = \left(f_t^1, f_t^2, \ldots, f_t^M\right) \in \mathcal{F}^M$ denote the *frequency vector*, where $f_t^j \in \mathcal{F}$ is the speed of the jth processor in time slot t and $\mathcal{F}$ is the set of available operating frequencies. Recall from Section II-A that slices belonging to frame v have decoding complexity w^v cycles, where w^v is assumed to be exponentially distributed with mean $\mathbf{E}[w^v] = \beta^{\mathrm{type}(v)}$. Consequently, slices belonging to frame v and processed at speed $f_t^j \in \mathcal{F}$ have service time $\tau^v = w^v/f_t^j$, where τ^v is exponentially distributed with mean $\mathbf{E}\left[\tau^v | f_t^j\right] = \beta^{\mathrm{type}(v)}/f_t^j$. Due to the memoryless property of the exponential distribution, if a slice belonging to frame v is scheduled on processor j at time t, then it will finish decoding in time slot t (i.e., in Δt seconds) with probability $\theta^v\left(f_t^j\right) = 1 - \exp\left(-f_t^j/\beta^{\mathrm{type}(v)}\Delta t\right)$, regardless of the number of times it was previously scheduled. In other words, if a slice takes multiple time slots to decode, then the memoryless property implies that it is not necessary to know the number of cycles that were spent decoding the slice in past time slots to predict the distribution of remaining cycles. Hence, assuming exponentially distributed service times greatly reduces the number of states required in our Markovian traffic model (see the appendix of our technical report [19] for more details). This is (implicitly) why a lot of prior research on power management using MDPs assumes exponential service times (e.g., [15], [16]).

D. State Evolution and System Dynamics

To fully characterize the video traffic, we need to understand how the traffic state $\mathcal{T}_t = (\mathcal{C}_t, \mathbf{x}_t, \mathbf{r}_t)$ evolves over time. The transition of the current frame set from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ is independent of the scheduling action; in fact, as illustrated in Fig. 1, it is deterministic and periodic for a fixed GOP structure, and therefore the sequence of current frame sets $\{\mathcal{C}_t | t \in \mathbb{N}\}$ can be modeled as a deterministic Markov chain.

Unlike the current frame set transition, the transition of the buffer state from x_t^v to x_{t+1}^v depends on the scheduling action and processor frequency. Let $z_t^{jv} = z_t^{jv}\left(f_t^j, y_t^{jv}\right)$ denote the number of slices belonging to frame v that finish decoding on processor j at time t. Note that $z_t^{jv} \leq y_t^{jv}$. For notational convenience, we define $\mathbf{Z}_t = \left[z_t^{jv}\right]_{jv}$, $\mathbf{z}_t^v = \left(z_t^{jv} | j \in \{1, \ldots, M\}\right)^T$, and $\mathbf{z}_t^j = \left(z_t^{jv} | v \in \mathcal{C}_t\right)$. Let $p_z\left(z_t^{jv} | f_t^j, y_t^{jv}\right)$ denote the probability that z_t^{jv} slices are decoded on processor j in time slot t given the frequency f_t^j and scheduling action y_t^{jv}.

Before we can write the buffer recursion governing the transition from x_t^v to x_{t+1}^v, we need to define a partition of the current frame set $\mathcal{C}_{t+1}$. The partition divides $\mathcal{C}_{t+1}$ into two sets: a set of frames that persist from time t to $t+1$ because they have display deadlines $d^{v,\mathrm{disp}} > t$, i.e., $\mathcal{C}_t \cap \mathcal{C}_{t+1}$; and, a set of newly arrived frames with arrival times $t^v = t+1$, i.e., $\mathcal{C}_{t+1} \backslash \mathcal{C}_t \triangleq \mathcal{C}_{t+1} - \mathcal{C}_t \cap \mathcal{C}_{t+1}$. Based on this partition, x_{t+1}^v can be determined from x_t^v and $\sum_{j=1}^{M} z_t^{jv}$ as follows:

$$x_{t+1}^v = \begin{cases} x_t^v - \sum_{j=1}^{M} z_t^{jv}, & \text{if } v \in \mathcal{C}_t \cap \mathcal{C}_{t+1} \\ l^v, & \text{if } v \in \mathcal{C}_{t+1} \backslash \mathcal{C}_t. \end{cases} \quad (1)$$

The sequence of buffer states $\{x_t^v | t \in \mathbb{N}\}$ can be modeled as a controlled Markov chain. Note that the buffer state for frame v, i.e., x_t^v, is only defined for $t \in \left[t^v, d^{v,\mathrm{disp}}\right]$. We will refer to this range of times as the *lifetime* of frame v.

The transition of the dependency state from r_t^v to r_{t+1}^v for $v \in \mathcal{C}_t \cap \mathcal{C}_{t+1}$ can be determined as follows:

$$r_{t+1}^v = \begin{cases} 1, & \text{if } x_t^u - \sum_{j=1}^{M} z_t^{ju} = 0 \text{ for all } u \in \mathcal{C}_t \\ & \quad \text{such that } (u, v) \in \mathcal{E} \\ 1, & \text{if } r_t^v = 1 \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

The first line in (2) states that frame v can be decoded in time slot $t+1$ if all of its parents are completely decoded at the end of time slot t. The second line in (2) states that if frame v can be decoded in time slot t then it can also be decoded in time slot $t+1$. Meanwhile, the initial value of dependency state r_{t+1}^v for $v \in \mathcal{C}_{t+1} \backslash \mathcal{C}_t$ can be determined as follows:

$$r_{t+1}^v = \begin{cases} 1, & \text{if } x_t^u - z_t^u = 0 \text{ for all } u \in \mathcal{C}_t \\ & \quad \text{such that } (u, v) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

It follows that the sequence of dependency states $\{r_t^v | t \in \mathbb{N}\}$ can be modeled as a controlled Markov chain. Note that, similar to the buffer state, the dependency state is only defined for the lifetime $t \in \left[t^v, d^{v,\mathrm{disp}}\right]$. Additionally, (2) and (3) imply that, if frame v is an I frame, then $r_{v,t} = 1$ for the frame's entire lifetime.

Because the individual components of the traffic state $\mathcal{T}_t \triangleq (\mathcal{C}_t, \mathbf{x}_t, \mathbf{r}_t)$ evolve as controlled Markov chains, the sequence of traffic states $\{\mathcal{T}_t | t \in \mathbb{N}\}$ can be modeled as a controlled Markov chain.

E. Power Cost and Slice Decoding Rate

The power-frequency function $\rho\left(f_t^j\right)$ maps the jth processor's speed f_t^j to its expected power consumption (watts). We assume that the power-frequency function is a strictly convex and increasing function of the frequency f and that it is the same for each processor. We also consider the expected power consumed by the instruction, data, and L2 cache using a function $\sigma\left(f_t^j, y_t^{jv}, \mathrm{type}(v)\right)$, which maps the jth processor's speed f_t^j, the scheduling action y_t^{jv}, and frame type $\mathrm{type}(v)$ to power consumption (watts). Thus, the total expected power consumed by processor j (and the associated accesses to the various caches) at time t can be written as

$$P\left(\mathcal{C}_t, f_t^j, \mathbf{y}_t^j\right) = \rho\left(f_t^j\right) + \sum_{v \in \mathcal{C}_t} \sigma\left(f_t^j, y_t^{jv}, \mathrm{type}(v)\right) \text{ (watts)}. \quad (4)$$

We consider the following QoS metric in each time slot t:

$$Q\left(f_t^j, y_t^{jv}, \text{type}\,(v)\right) = \sum_{z_t^{jv} \le y_t^{jv}} p_z\left(z_t^{jv} | f_t^j, y_t^{jv}\right) z_t^{jv}. \quad (5)$$

This QoS metric is simply the expected number of slices belonging to frame v that will be decoded on processor j in time slot t. We will refer to (5) as the *slice decoding rate* for frame v on processor j. For notational simplicity, in the remainder of the paper, we will omit the functional dependence of (4) and (5) on $\text{type}\,(v)$.

F. Markov Decision Process Formulation

In this subsection, we formulate the problem of energy-efficient slice-parallel video decoding on M processors. In each time slot t, the objective is to determine the scheduling action y_t^{jv}, for all $j \in \{1, 2, \ldots, M\}$ and $v \in \mathcal{C}_t$, and the frequency vector $\mathbf{f}_t$, in order to minimize the total average power consumption subject to a constraint on the average slice decoding rate. In this paper, for mathematical convenience, we use *discounted*[1] average power consumption and slice decoding rate, which can be expressed as follows

$$\bar{P} = \mathbf{E}\left[\sum_{t=0}^{\infty}\sum_{j=1}^{M}\gamma^t P\left(\mathcal{C}_t, f_t^j, \mathbf{y}_t^j\right)\right], \quad \text{and} \quad (6)$$

$$\bar{Q} = \mathbf{E}\left[\sum_{t=0}^{\infty}\sum_{j=1}^{M}\sum_{v\in\mathcal{C}_t}\gamma^t Q\left(f_t^j, y_t^{jv}\right)\right], \quad (7)$$

respectively, where $\gamma \in [0, 1)$ is the discount factor, and the expectation is over the sequence of traffic states $\{\mathcal{T}_t | t \in \mathbb{N}\}$. Stated more formally, the optimization objective and constraints are as follows:

$$\min_{\mathbf{f}_t, \mathbf{Y}_t,\ \forall t\in\mathbb{N}} \bar{P}$$

Subject to:

Slice decoding rate constraint:

$$\bar{Q} \ge \bar{\eta}$$

Buffer constraint:

$$\sum_{j=1}^{M} y_t^{jv} \le x_t^v,\ \forall v \in \mathcal{C}_t,\ \forall t \in \mathbb{N}$$

Processor constraint:

$$\sum_{v\in\mathcal{C}_t} y_t^{jv} \le 1,\ \forall j \in \{1, \ldots, M\},\ \forall t \in \mathbb{N}$$

Dependency constraint:

$$\text{if } r_t^v = 0, \text{then } \sum_{j=1}^{M} y_t^{jv} = 0,\ \forall v \in \mathcal{C}_t,\ \forall t \in \mathbb{N} \quad (8)$$

where $\bar{\eta}$ is the discounted slice decoding rate constraint.

[1]In this paper, for mathematical convenience, we use the *discounted* averages instead of conventional averages; however, the problem can be formulated using non-discounted averages. We refer the interested reader to [15] for an intuitive justification for using discounted averages.

The constrained optimization defined in (8) can be formulated as an unconstrained MDP by introducing a Lagrange multiplier $\lambda \in \mathbb{R}_+$ associated with the slice decoding rate constraint. Note that the buffer, processor, and dependency constraints defined in (8) must still hold in every time slot, however, for notational simplicity, we will omit them from our exposition in the remainder of the paper. We can define the Lagrangian cost function:

$$c_\lambda\left(\mathcal{C}_t, \mathbf{f}_t, \mathbf{Y}_t\right) = \sum_{j=1}^{M} P\left(\mathcal{C}_t, f_t^j, \mathbf{y}_t^j\right) - \lambda\left(\sum_{j=1}^{M}\sum_{v\in\mathcal{C}_t}\gamma^t Q\left(f_t^j, y_t^{jv}\right) - \bar{\eta}\right). \quad (9)$$

For a fixed λ, in each time slot t, the unconstrained problem's objective is to determine the frequency vector $\mathbf{f}_t$ and scheduling matrix $\mathbf{Y}_t$ in order to minimize the average Lagrangian cost. The discounted average Lagrangian cost can be expressed as

$$L_\lambda = \min_{\mathbf{f}_t, \mathbf{Y}_t,\ \forall t\in\mathbb{N}} \mathbf{E}\left[\sum_{t=0}^{\infty}\gamma^t c_\lambda\left(\mathcal{C}_t, \mathbf{f}_t, \mathbf{Y}_t\right)\right]. \quad (10)$$

Letting $p\left(\mathcal{T}' | \mathcal{T}, \mathbf{f}, \mathbf{Y}\right)$ denote the traffic state transition probability function, the problem of minimizing (10) can be mapped to the following dynamic programming equation:

$$V_\lambda^*\left(\mathcal{T}\right) = \min_{\mathbf{f},\mathbf{Y}}\left\{\begin{array}{l} c_\lambda\left(\mathcal{C}, \mathbf{f}, \mathbf{Y}\right) + \\ \gamma\sum_{\mathcal{T}'} p\left(\mathcal{T}' | \mathcal{T}, \mathbf{f}, \mathbf{Y}\right) V_\lambda^*\left(\mathcal{T}'\right)\end{array}\right\} \quad (11)$$

which can be solved using the well-known value iteration algorithm [13] as follows:

$$V_{n+1,\lambda}\left(\mathcal{T}\right) = \min_{\mathbf{f},\mathbf{Y}}\left\{\begin{array}{l} c_\lambda\left(\mathcal{T}, \mathbf{f}, \mathbf{Y}\right) + \\ \gamma\sum_{\mathcal{T}'} p\left(\mathcal{T}' | \mathcal{T}, \mathbf{f}, \mathbf{Y}\right) V_{n,\lambda}\left(\mathcal{T}'\right)\end{array}\right\} \quad (12)$$

where n is the iteration index, $V_{0,\lambda}\left(\mathcal{T}\right)$ is initialized to 0 for all $\mathcal{T}$, and $V_{n,\lambda}\left(\mathcal{T}\right)$ approaches $V_\lambda^*\left(\mathcal{T}\right)$ as $n \to \infty$ [13].

III. Low Complexity Solution

Unfortunately, solving (12) directly is a computationally intractable problem for two reasons. First, the number of traffic states exponentially increases with the number of frames in the current frame set. Second, the action-space exponentially increases with the number of processors M because $\mathbf{f} \in \mathcal{F}^M$ and, accounting for the processor constraint defined in Section II-C and the fact that all processors are homogeneous, we have to consider at most $2^M = \left|\{0, 1\}^M\right|$ scheduling actions $\mathbf{Y}$ [19].

Clearly, the reason for the exponential growth in the state space (respectively, action space) is that the optimization simultaneously considers the states (respectively, scheduling actions and processor frequencies) of multiple frames. However, carefully studying the optimization objective and constraints defined in (8), it is clear that the only reason these need to be optimized jointly is the processor constraint, which ensures that only one slice is assigned to each processor in each time slot. Motivated by this weak coupling among tasks, we propose a two-level scheduler to approximately solve (8): The first-level scheduler

TABLE I
FRAME-LEVEL VALUE ITERATION ALGORITHM PERFORMED BY THE FIRST-LEVEL SCHEDULER

1.	**Initialize:** $V^v_{0,\lambda}\left(\mathcal{C},x^v,r^v\right)=0$ for all $v\in\mathcal{V}^g$, $\mathcal{C}$, $x^v\in\left\{0,\ldots,l^v\right\}$, and $r^v\in\left\{0,1\right\}$
2.	**Repeat**
3.	$\Delta\leftarrow 0$
4.	**For each** $v\in\mathcal{V}^g$, $\mathcal{C}$, $x^v\in\left\{0,\ldots,l^v\right\}$, and $r^v\in\left\{0,1\right\}$
5.	**If** $v\in\mathcal{C}$, $x^v>0$, and $r^v=1$ (i.e. frame v is in the current frame set, still has undecoded slices, and has its dependencies satisfied)
6.	$V^v_{n+1,\lambda}\left(\mathcal{C},x^v,r^v\right)=$ $\min\limits_{\mathbf{f}^{1:M,v},\mathbf{y}^{1:M,v}}\left\{\begin{array}{l}\sum\limits_{j=1}^{M}\left[\rho\left(f^{jv}\right)+\sigma\left(f^{jv},y^{jv}\right)-\lambda Q\left(f^{jv},y^{jv}\right)\right]\\ \gamma\sum\limits_{\mathbf{z}^{1:M,v}\le\mathbf{y}^{1:M,v}}\prod\limits_{j=1}^{M}p_z\left(z^{jv}\mid f^{jv},y^{jv}\right)\left[\begin{array}{l}V^v_{n,\lambda}\left(\mathcal{C}',x^v-\left\|\mathbf{z}^{1:M,v}\right\|_1,r^{v\prime}\right)+\\ \sum\limits_{\substack{u\in\mathcal{C}':v\prec u\\ r^{u\prime}=1}}V^u_{n,\lambda}\left(\mathcal{C}',l^{u\prime},r^{u\prime}\right)\end{array}\right]\end{array}\right\}$ (13)
7.	**Else**
8.	$V^v_{n+1,\lambda}\left(\mathcal{C},x^v,r^v\right)=0$
9	**End**
10.	**End**
11.	$\Delta\leftarrow\max\left(\Delta,\left\|V^v_{n+1,\lambda}\left(\mathcal{C},x^v,r^v\right)-V^v_{n,\lambda}\left(\mathcal{C},x^v,r^v\right)\right\|\right)$
12.	$n\leftarrow n+1$
13.	**Until** $\Delta<\epsilon$ (a small positive number)
14.	**Output:** $\left\{V^{v,*}:v\in\mathcal{V}^g\right\}$

determines the optimal scheduling actions and processor frequencies for each frame under the assumption that each frame has exclusive access to the M processors. Given the results of the first-level scheduler, the second-level scheduler determines the final slice- and frequency-to-processor mappings.

A. First-Level Scheduler

The first-level scheduler computes a value function $V^v\left(\mathcal{C},x^v,r^v\right)$ for every frame in a GOP. This value function only depends on the current frame set, the frame's buffer state x^v, and the frame's dependency state r^v. Note that the current frame set indicates the remaining lifetime of a frame and describes the connections to its parents and children. Hence, the current frame state will have a significant impact on the optimal scheduling and DVFS decisions for the frame. To account for the dependencies among frames, we define the vth frame's value function $V^v\left(\mathcal{C},x^v,r^v\right)$ so that it includes the values of its children. In this way, frames with many children (e.g., I frames) can account for how their scheduling and frequency decisions impact the future performance of their children. We describe the first-level scheduler in more detail in the remainder of this section.

1) Frame-Level Value Iteration: The first-level scheduler performs the frame-level value iteration algorithm illustrated in Table I to compute the optimal value functions $\left\{V^{v,*}:v\in\mathcal{V}^g\right\}$. Unlike the conventional value iteration algorithm, the proposed algorithm has multiple *coupled* value functions that need to be updated. Note that the coupling exists because the value of a frame depends on the values of its children. Due to this coupling, the form of the value function update (lines 5–9 in Table I) is different from the conventional value iteration algorithm.

If it is not possible to make any decisions for a frame in the current traffic state, then we set the frame's value to 0 in that state. Hence, if a frame is not in the current frame set (i.e., $v\notin\mathcal{C}$), does not have its dependencies satisfied (i.e., $r^v=0$), or is in the current frame set, but is already fully decoded (i.e., $v\in\mathcal{C}$ and $x^v=0$), then we set the frame's value to 0 (line 8 in Table I). The more interesting case is when the frame is in the current frame set, still has undecoded slices, and has its dependencies satisfied (i.e., $v\in\mathcal{C}$, $x^v>0$, and $r^v=1$). In this case, the value function update comprises four distinct terms: the power consumed by each processor in the current state; the expected slice decoding rate on each processor in the current state; the expected future value of frame v; and the sum of the expected future values of the vth frame's children. Note that the expected future value of frame v, i.e., $\gamma V^v_{n,\lambda}\left(\mathcal{C}',x^v-\left\|\mathbf{z}^{1:M,v}\right\|_1,r^{v\prime}\right)$, is 0 if $v\notin\mathcal{C}'$; and, the sum of the expected future values of the children's frames, i.e., $\gamma\sum_{u\in\mathcal{C}':v\prec u,\ r^{u\prime}=1}V^u_{n,\lambda}\left(\mathcal{C}',l^{u\prime},r^{u\prime}\right)$, is 0 if $x^v-\left\|\mathbf{z}^{1:M,v}\right\|_1$ is not 0 (because $x^v-\left\|\mathbf{z}^{1:M,v}\right\|_1$ must be 0 for $r^{u\prime}$ to be 1). In other words, the parent frame's value function is coupled with the children's value functions only if the parent frame gets fully decoded.

2) Decomposing the Monolithic Frame-Level Value Iteration Update: The frame-level value iterations allow us to eliminate

the exponential growth of the state space with respect to the number of frames in the current frame set, but we still have to address the fact that the optimization in (13) (Line 6 of Table I) requires a search over an exponential number of scheduling and frequency vectors. In this subsection, we discuss how to decompose the monolithic update defined in (13) into M stages (hereafter, *sub-value iterations*), each corresponding to a local scheduling problem on a single processor. These M sub-value iterations can be performed iteratively, using the output of the jth processor's sub-value iteration as the input to the $(j-1)$st processor's sub-value iteration. Importantly, decomposing the monolithic update into M sub-value iterations significantly reduces the computational complexity of the update. Due to space limitations, we omit the derivation of the sub-value iterations and refer the interested reader to our technical report [19] for details.

Let $\mathbf{f}^{1:j,v}$, $\mathbf{y}^{1:j,v}$, $\mathbf{z}^{1:j,v}$, for $1 < j \leq M$, be vectors denoting the frequencies, scheduling actions, and number of decoded slices for frame v on processors 1 through j. Let $\left\|\mathbf{z}^{1:j,v}\right\|_1 = \sum_{i=1}^{j} z^{iv}$ denote the ℓ_1 -norm of the vector $\mathbf{z}^{1:j,v}$. Equipped with this new notation, the sub-value iteration at processor $j = M$ is defined as follows:

Sub-value iteration at processor M: See equation (14) at the bottom of the page. The Mth processor's sub-value iteration estimates the value of being in traffic state $\mathcal{T}^v = (\mathcal{C}, x^v, r^v)$ conditioned on processors 1 through $M-1$ successfully decoding $\left\|\mathbf{z}^{1:M-1,v}\right\|_1 \in \{0,1,\ldots,M-1\}$ slices. This value is calculated as the sum of (i) the immediate cost incurred by processor M for processing slices belonging to frame v, i.e., $\rho\left(f^{Mv}\right) + \sigma\left(f^{Mv}, y^{Mv}\right) - \lambda Q\left(f^{Mv}, y^{Mv}\right)$, (ii) the expected discounted future value of frame v transitioning to state $\mathcal{T}^{v'} = \left(\mathcal{C}', x^v - \left\|\mathbf{z}^{1:M-1,v}\right\|_1 - z^{Mv}, r^{v\prime}\right)$, and (iii) the expected discounted future value of the vth frame's children, i.e., $\gamma \mathbf{E}_{z^{Mv}|f^{Mv},y^{Mv}}\left[\sum_{u\in\mathcal{C}':v\prec u, r^{u\prime}=1} V^{u}_{n,\lambda}\left(\mathcal{C}', l^{u\prime}, r^{u\prime}\right)\right]$. The output of the Mth processor's sub-value iteration, i.e.,

$$\left\{ \begin{array}{l} V^{M-1,v}_{n,\lambda}\left(\mathcal{C}, x^v, r^v \middle| \left\|\mathbf{z}^{1:M-1,v}\right\|_1\right): \\ x^v \in \{0,\ldots,l^v\}, \left\|\mathbf{z}^{1:M-1,v}\right\|_1 \in \{0,1,\ldots,M-1\} \end{array} \right\},$$

is used as input to the $(M-1)$st processor's sub-value iteration defined below.

The sub-value iterations for processors $j = 2,\ldots,M-1$ are defined as follows:

Sub-value iteration at processors $j \in \{2,\ldots,M-1\}$: See equation (15) at the bottom of the page. The jth processor's sub-value iteration estimates the value of being in traffic state $\mathcal{T}^v = (\mathcal{C}, x^v, r^v)$ conditioned on processors 1 through $j-1$ successfully decoding $\left\|\mathbf{z}^{1:j-1,v}\right\|_1 \in \{0,1,\ldots,j-1\}$ slices. This value is calculated as the sum of the immediate cost incurred by processor j and an expectation over the value calculated by the $(j+1)$ st processor's sub-value iteration. The output of the jth processor's sub-value iteration, i.e.,

$$\left\{ \begin{array}{l} V^{j-1,v}_{n,\lambda}\left(\mathcal{C}, x^v, r^v \middle| \left\|\mathbf{z}^{1:j-1,v}\right\|_1\right): \\ x^v \in \{0,\ldots,l^v\}, \left\|\mathbf{z}^{1:j-1,v}\right\|_1 \in \{0,1,\ldots,j-1\} \end{array} \right\},$$

is used as input to the $(j-1)$ st processor's sub-value iteration.

Finally, the sub-value iteration at processor $j = 1$ is defined as follows:

Sub-value iteration at processor 1:

$$V^{v}_{n+1,\lambda}\left(\mathcal{C}, x^v, r^v\right) = \min_{f^{1v},\, y^{1v}} \left\{ \begin{array}{l} \rho\left(f^{1v}\right) + \sigma\left(f^{1v}, y^{1v}\right) - \lambda Q\left(f^{1v}, y^{1v}\right) \\ +\mathbf{E}_{z^{1v}|f^{1v},y^{1v}}\left[V^{1,v}_{n,\lambda}\left(\mathcal{C}, x^v - z^{1v}, r^v | z^{1v}\right)\right] \end{array} \right\}. \tag{16}$$

The output of the first processor's sub-value iteration

$$\left\{V^{v}_{n+1,\lambda}\left(\mathcal{C}, x^v, r^v\right) : x^v \in \{0,\ldots,l^v\}\right\}$$

includes (i) the immediate power costs incurred by all processors, (ii) the slice decoding rate of all processors, (iii) the expected discounted future value of frame v, and (iv) the expected future discounted value of frame v's children, and it is used as input to the Mth processor's sub-value iteration during iteration $n+1$.

Performing the M sub-value iterations for frame v on a single traffic state $\mathcal{T}^v = (\mathcal{C}, x^v, r^v)$ only requires a search over the (scalar) scheduling actions $y^{jv} \in \{0,1\}$ and frequencies $f^{jv} \in \mathcal{F}$ for each processor $j \in \{1,\ldots,M\}$ and each possible value of $\left\|\mathbf{z}^{1:j-1,v}\right\|_1$. Therefore, using the proposed decomposition of the monolithic value function update significantly reduces the action-selection complexity.

$$V^{M-1,v}_{n,\lambda}\left(\mathcal{C}, x^v, r^v \middle| \left\|\mathbf{z}^{1:M-1,v}\right\|_1\right) = \min_{f^{Mv},\, y^{Mv}} \left\{ \begin{array}{l} \rho\left(f^{Mv}\right) + \sigma\left(f^{Mv}, y^{Mv}\right) - \lambda Q\left(f^{Mv}, y^{Mv}\right) + \\ \gamma \mathbf{E}_{z^{Mv}|f^{Mv},y^{Mv}}\left[\begin{array}{c} V^{v}_{n,\lambda}\left(\mathcal{C}', x^v - \left\|\mathbf{z}^{1:M-1,v}\right\|_1 - z^{Mv}, r^{v\prime}\right) + \\ \sum\limits_{u\in\mathcal{C}':v\prec u,\, r^{u\prime}=1} V^{u}_{n,\lambda}\left(\mathcal{C}', l^{u\prime}, r^{u\prime}\right) \end{array}\right] \end{array} \right\} \tag{14}$$

$$V^{j-1,v}_{n,\lambda}\left(\mathcal{C}, x^v, r^v \middle| \left\|\mathbf{z}^{1:j-1,v}\right\|_1\right) = \min_{f^{jv},\, y^{jv}} \left\{ \begin{array}{c} \rho\left(f^{jv}\right) + \sigma\left(f^{jv}, y^{jv}\right) - \lambda Q\left(f^{jv}, y^{jv}\right) + \\ \mathbf{E}_{z^{jv}|f^{jv},y^{jv}}\left[V^{j,v}_{n,\lambda}\left(\mathcal{C}, x^v - \left\|\mathbf{z}^{1:j-1,v}\right\|_1 - z^{jv}, r^v \middle| \left\|\mathbf{z}^{1:j,v}\right\|_1\right)\right] \end{array} \right\} \tag{15}$$

TABLE II
DETERMINING AN APPROXIMATELY OPTIMAL POLICY FOR FRAME v

1.	**Input:** $V_{\lambda}^{v,*}\left(\mathcal{C}, x^v, r^v\right)$ for all $v \in \mathcal{V}^g$
2.	**For each** $\mathcal{C}$, $x^v \in \left\{0,\ldots,l^v\right\}$, and $r^v \in \left\{0,1\right\}$
3.	Obtain $\left(f^{1v,*}, y^{1v,*}\right)$ as the argument that maximizes the 1st processor's sub-value function (Eq. (16)).
2.	**For each** $j \in \left\{2,\ldots,M\right\}$
4.	Approximate $\left\|\mathbf{z}^{1:j-1,v}\right\|_1$ with $\overline{Z}^{1:j-1,v,*} = \left\lfloor \mathbf{E}_{\mathbf{z}^{1:j-1,v}\mid \mathbf{f}^{1:j-1,v,*},\mathbf{y}^{1:j-1,v,*}}\left[\left\|\mathbf{z}^{1:j-1,v}\right\|_1\right]\right\rfloor$
	Obtain $\left(f^{jv,*}, y^{jv,*}\right)$ as the argument that maximizes the j th processor's sub-value function (Eq. (14) or (15)) given the optimal future value.
5.	**End**
7.	$\pi^{v,*}\left(\mathcal{C}, x^v, r^v\right) \leftarrow \left(\mathbf{f}^{1:M,v,*}, \mathbf{y}^{1:M,v,*}\right)$
8.	**End**
14.	**Output:** $\pi^{v,*}\left(\mathcal{C}, x^v, r^v\right)$

3) Determining the Approximately Optimal Policy: We define the *policy* $\pi^v : (\mathcal{C}, x^v, r^v) \rightarrow (\mathbf{f}^v, \mathbf{y}^v)$ as a mapping from the vth frame's traffic state $\mathcal{T}^v = (\mathcal{C}, x^v, r^v)$ to a pair of scheduling and frequency vectors $(\mathbf{f}^v, \mathbf{y}^v)$. If we know the optimal frame-level value functions $\{V^{v,*} : v \in \mathcal{V}^g\}$, then we can determine the optimal action to take in each traffic state, and therefore the optimal policy, by finding the scheduling and frequency vectors that optimize (13). However, as we discussed earlier, this requires searching over an exponential number of scheduling and frequency vectors. Fortunately, it turns out that we can use the sub-value iterations proposed in Section III-A2 to find an approximately optimal policy. An algorithm for doing this is summarized in Table II.

The key idea behind the algorithm in Table II is to find the (scalar) scheduling and frequency actions that optimize the sub-value functions defined in (14), (15), and (16) for each processor. However, notice that the sub-value iterations for processors $j \in \{2,\ldots,M\}$ require knowledge of the number of slices that finish decoding on processors 1 through $j-1$, but we need to select the (scalar) scheduling action and processor frequency on processor j before $\left\|\mathbf{z}^{1:j-1,v}\right\|_1$ is known. To work around this problem, the algorithm in Table II first selects the optimal (scalar) scheduling action and frequency for processor 1. Then, to select the optimal (scalar) scheduling actions and frequencies for processors $j \in \{2,\ldots,M\}$, the algorithm approximates $\left\|\mathbf{z}^{1:j-1,v}\right\|_1$ with the floor of its expected value (the floor of X, denoted by $\lfloor X \rfloor$, is the largest integer value that is less than X), which depends on the optimal actions selected by processors 1 through $j-1$.

B. Second-Level Scheduler

Given the optimal policies calculated by the first-level scheduler (i.e., $\pi^{v,*}(\mathcal{C}, x^v, r^v)$, for all $v \in \mathcal{V}^g$), it is very likely that slices belonging to different frames in the current frame set will want to be scheduled on the same processor in the same time slot, thereby violating the processor constraint defined in (8). To avoid this problem, the second-level scheduler determines the final slice-to-processor and frequency-to-processor mappings using an Earliest Deadline First (EDF) policy. Specifically, frame $v^{j,*}$ gets scheduled on processor j at frequency $f^{jv,*}$ if $v^{j,*}$ is the solution to the following optimization:

$$v^{j,*} = \arg \min_{v: y^{jv,*} \neq 0} d^{v,\mathrm{dec}} \tag{17}$$

where $d^{v,\mathrm{dec}}$ is the frame's decoding deadline and ties are broken randomly.

Finally, if a slice finishes decoding before the first-level scheduler's time quantum is up, then the second-level scheduler will start decoding another slice (from the same frame) during the "slack" time, which is the time between the beginning of the next time quantum and the time that the originally scheduled slice finished decoding.

We refer the interested reader to our technical report [19] for a detailed discussion about the complexity of our proposed algorithm.

IV. Experiments

In this section, we describe our experimental framework in detail and evaluate our proposed algorithm. We note that we did not have access to a decoder that supports the sophisticated level of slice parallel decoding that our algorithm is designed to exploit. Specifically, the available decoder implementation can decode slices belonging to the same frame in parallel, but it cannot decode slices from different frames in parallel. Because the latter capability is essential to our proposed algorithm, we use Matlab to evaluate it, instead of the multiprocessor ARM (MPARM) simulator.

In order to validate our optimized multi-core scheduling approach in Matlab, we use accurate profiling/statistics generated from an H.264/AVC decoder executed on a sophisticated multiprocessor virtual platform simulator. Specifically, we have extended and customized the MPARM virtual platform simulator in [14], which is a complete SystemC simulation environment for MPSoC architectural design and exploration. MPARM provides cycle-accurate and bus signal-accurate simulation for different processors. Due to space limitations, we refer the interested reader to our technical report [19] for an in depth description of the modifications that we made to the MPARM simulator to support the execution of our multimedia benchmark (the Joint Model reference software version 17.2 of an H.264 encoder) and to support the gathering of accurate profiling statistics.

To generate our experimental results, we implemented the two-level scheduling algorithm proposed in Section III in Matlab. This algorithm, together with slice-level data traces recorded from MPARM, allowed us to determine on-line scheduling and DVFS policies for the Silent and Foreman sequences (CIF resolution, 30 frames per second, 8 slices per frame) with an IBPB GOP structure as illustrated in Fig. 1. In our Matlab simulations, we assume a time slot duration of 1/90 s, which is one-third of the frame period. We divide each GOP into 12 current frame sets to capture the dependencies among frames. These 12 current frame sets are generated from the four unique current frame sets given in the example in Section II-B by repeating each for three consecutive time slots.[2] The system, application, and other parameters used in our experiments are given in Table III. Importantly, although our MDP model assumes that the slice decoding complexities are exponentially distributed, we use the actual slice decoding times from the MPARM simulator when we simulate the scheduling and DVFS policies.

[2]In the example of Section II-B, the time slot duration was equal to the frame duration (i.e., 1/30 s). Because we are now using a time slot duration equal to one-third of the frame duration (i.e., 1/90 s), we must repeat each of the current frame sets in example 1 three times.

TABLE III
Simulation Parameters

Parameter	Value(s)
No. slave cores (M)	1, 2, 4, 8
Frequency set ($\mathcal{F}$)	{125, 166, 250, 500} MHz
Sequence	Foreman (220 frames), Silent (300 frames)
Resolution	CIF (352 x 288)
GOP Structure	'IBPB'
Frame rate	30 frames per second
Time slot duration	1/90 s
No. current frame sets	12
No. slices per frame	8
Lagrange multiplier (λ)	0, 50, 100, 200, 400, 800

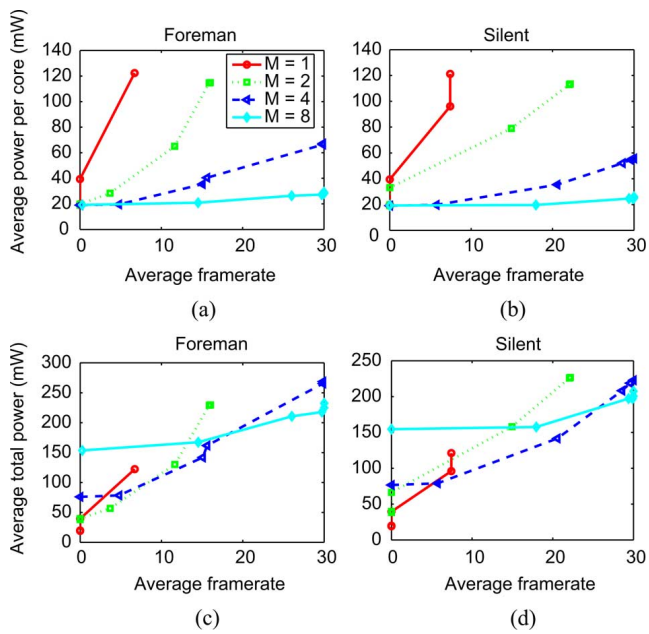

Fig. 2. Power consumption versus the average decoded frame rate. (a, b) Per core power. (c, d) Total power.

A. Trade-Off Between Power Consumption and Quality of Service

The optimization proposed in (8) allows the system to trade-off power consumption and a QoS metric, namely, the slice decoding rate, which is roughly proportional to the frame rate. This trade-off can be made by adapting the Lagrange multiplier λ in the cost function defined in (9). Fig. 2 shows the trade-off between the average power consumption and average frame rate for the values of λ given in Table III and $M = 1, 2, 4$, and 8 processors.

Figs. 2(a) and 2(b) show the average power consumption *per core* versus the average decoded frame rate for the Foreman and Silent sequences, respectively. The power-QoS pairs in the lower left of these two figures occur when $\lambda = 0$ and correspond to a scheduling policy that never schedules any tasks and a

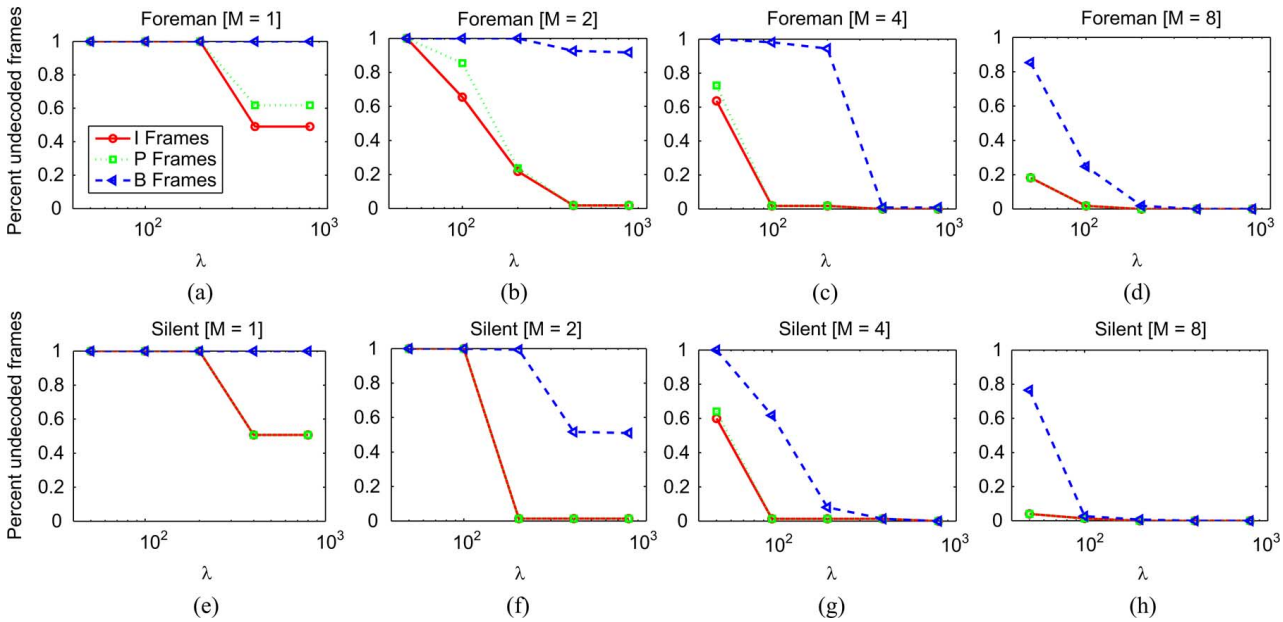

Fig. 3. Fractions of I, P, and B frames that miss their display deadline as a function of the parameter λ. (a,b,c,d) Foreman sequence. (e,f,g,h) Silent sequence.

DVFS policy that always selects the lowest operating frequency, thereby achieving a QoS of zero frames per second. The minimum power consumption per core, which is approximately 20 mW, is due to leakage power. If we were to introduce DPM into our optimization framework, then this minimum power would be significantly lower. Clearly, as λ increases, the QoS is improved at the expense of power; as the number of processors increases, less power is required per processor to decode at a given QoS; and, depending on the video source characteristics (e.g., Foreman vs. Silent), the achievable QoS varies for a given power consumption (in this case, Silent receives a higher QoS than Foreman for the same power consumption because Silent is a lower activity sequence).

Figs. 2(c) and 2(d) show the average *total* power consumption versus the average decoded frame rate for the Foreman and Silent sequences, respectively. It is interesting to note that, as the decoded frame rate decreases, having less processors results in less overall power consumption. This is due to the large leakage power incurred by each processor, which, as mentioned before, could be significantly reduced using DPM in addition to DVFS. It is clear from Fig. 2 that the proposed scheduling algorithm exploits the loss-tolerant nature of video decoding tasks to achieve lower decoded frame rates when the energy-budget does not allow for full frame rate decoding.

B. Display Deadline Miss Rates

Fig. 3 shows the fractions of I, P, and B frames that miss their display deadline as a function of the parameter λ (for the values of λ listed in Table III). The results show that the proposed on-line scheduling and DVFS optimization has a very desirable property: as minimizing power becomes more important (i.e., λ decreases), B frames are the first to miss their deadlines, followed by P frames, and then I frames. In other words, due to the proposed scheduling algorithm, the QoS (i.e., frame rate) decreases slowly with the power consumption. In contrast, a scheduling policy that allows P frames to be lost before B frames, or I frames before P frames, is inherently suboptimal because a deadline miss by one I or P frame induces deadline misses of dependent frames, adversely impacting the QoS.

We refer the interested reader to our technical report [19] for additional experimental result showing the impact of video resolution on the system performance and power consumption. In [19], we have also included a detailed comparison (both qualitative and quantitative) of our proposed algorithm to the so-called Optimum Minimum-Energy Multicore Scheduling algorithm (OPT-MEMS [2]), which we omit here due to space limitations.

V. Conclusion

We propose a Markov decision process based on-line scheduling algorithm for slice-parallel video decoders on multicore systems. To mitigate the complexity of solving the optimal on-line scheduling and DVFS policy, we proposed a novel two-level scheduler. The first-level scheduler determines scheduling and DVFS policies independently for each frame and the second-level decides the final frame-to-processor and frequency-to-processor mappings at run-time. We validated the proposed algorithm in Matlab using accurate video decoder trace statistics generated from an H.264/AVC decoder that we implemented on a cycle-accurate MPARM simulator. Our experimental results indicate that the proposed algorithm effectively trades-off power consumption and QoS by ensuring that a limited energy-budget is allocated to decoding the most important frames (e.g., I and P frames) before the less important frames (e.g., B frames).

References

[1] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proc. 17th Int. Symp. Parallel and Distributed Proc. (IPDPS '03)*, Apr. 2003.

[2] W. Y. Lee, Y. W. Ko, H. Lee, and H. Kim, "Energy-efficient scheduling of a real-time task on DVFS-enabled multi-cores," in *Proc. 2009 Int. Conf. Hybrid Inform. Technol. (ICHIT '09)*, 2009, pp. 273–277.

[3] Y.-H. Wei, C.-Y. Yang, T.-W. Kuo, S.-H. Hung, and Y.-H. Chu, "Energy-efficient real-time scheduling of multimedia tasks on multi-core processors," in *Proc. 2010 ACM Symp. Appl. Comput. (SAC '10)*, 2010, pp. 258–262.
[4] H. Liu, Z. Shao, M. Wang, and P. Chen, "Overhead-aware system-level joint energy and performance optimization for streaming applications on multiprocessor systems-on-Chip," in *Proc. 2008 Euromicro Conf. Real-Time Syst. (ECRTS '08)*, Jul. 2008, pp. 92–101.
[5] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1–6, pp. 5–35, 1991.
[6] R. Xu, R. Melhem, and D. Mosse, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS '07)*, pp. 25–38.
[7] R. Xu, "Energy-aware scheduling for streaming applications" Ph.D. dissertation, Univ. Pittsburgh, Pittsburgh, PA. [Online]. Available: http://etd.library.pitt.edu/ETD/available/etd-03082010- 201840/unrestricted/XuRuibin20100104.pdf.
[8] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001.
[9] D. Zhang, F. Chen, and S. Jin, "Global EDF-based online, energy-efficient real-time scheduling in multi-core platform," in *Proc. 2011 IEEE Int. Conf. Comput. Sci. and Automation Eng. (CSAE)*, Jun. 10-12, 2011, vol. 2, pp. 666–670.
[10] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," in *Proc. Conf. Design, Automation and Test in Europe (DATE '09)*, 2009, pp. 411–416.
[11] F. Catthoor, E. de Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Norwell, MA: Kluwer, 1998.
[12] M. Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding," in *Proc. 7th ACM & IEEE Int. Conf. Embedded Software*, 2007, pp. 269–278.
[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction*. Cambridge, MA: MIT Press, 1998.
[14] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri, "MPARM: Exploring the multi-processor SoC design space with systemc," *J. VLSI Signal Process. Syst.*, vol. 41, no. 2, pp. 169–182, Sep. 2005.
[15] L. Benini, A. Bogliolo, G. A. Paleologo, and G. D. Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 18, no. 6, pp. 813–833, Jun. 1999.
[16] D. Niyato, S. Chaisiri, and L. B. Sung, "Optimal power management for server farm to support green computing," in *Proc. 9th IEEE/ACM Int. Symp. Cluster Comput. and the Grid*, 2009.
[17] E. B. v. d. Tol, E. G. Jaspers, and R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," *Proc. SPIE* , pp. 707–718, May 2003.
[18] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets, "GRACE-1: Cross-layer adaptation for multimedia quality battery energy," *IEEE Trans. Mobile Comput.*, vol. 5, no. 7, pp. 799–815, Jul. 2006.
[19] N. Mastronarde, K. Kanoun, D. Atienza, P. Frossard, and M. v. d. Schaar, Markov Decision Process Based Energy-Efficient on-Line Scheduling for Slice-Parallel Video Decoders on Multicore Systems. [Online]. Available: http://arxiv.org/abs/1112.4084.

Nicholas Mastronarde (S'07–M'11) received the B.S. and M.Sc. degrees in electrical engineering from the University of California at Davis in 2005 and 2006, respectively, and the Ph.D. degree in electrical engineering from the University of California at Los Angeles (UCLA) in 2011.

He is currently an Assistant Professor in the Department of Electrical Engineering at the State University of New York at Buffalo.

Karim Kanoun received the M.Sc. degree in computer science from the ENSIMAG School of Engineering in Informatics and Applied Mathematics in Grenoble, France. He is currently pursuing the Ph.D. degree in embedded systems at the Federal Institute of Technology in Lausanne, Switzerland.

David Atienza (M'05) received the M.Sc. and Ph.D. degrees in computer science and engineering from Complutense University of Madrid (UCM), Spain, and Inter-University Micro-Electronics Center (IMEC), Belgium, in 2001 and 2005, respectively.

Currently, he is Professor and Director of the Embedded Systems Laboratory (ESL) at EPFL, Switzerland, and Adjunct Professor at the Computer Architecture and Automation Department of UCM.

Pascal Frossard (S'96–M'01–SM'04) received the M.Sc. and Ph.D. degrees, both in electrical engineering, from the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 1997 and 2000, respectively.

Since 2003, he has been a faculty at EPFL, where he heads the Signal Processing Laboratory (LTS4).

Mihaela van der Schaar (F'10) is Chancellor's Professor of Electrical Engineering at the University of California, Los Angeles. She is the founding director of the UCLA Center for Engineering Economics, Learning, and Networks (see netecon.ee.ucla.edu).