MARKOV DECISION PROCESS BASED ENERGY-EFFICIENT SCHEDULING FOR SLICE-PARALLEL VIDEO DECODING

Nicholas Mastronarde[*], *Karim Kanoun*[†], *David Atienza*[†], and *Mihaela van der Schaar*[‡]

[*] Department of EE, University at Buffalo, [†] Institute of EE, EPFL, [‡] Department of EE, UCLA

ABSTRACT

We consider the problem of energy-efficient scheduling for slice-parallel video decoders on multicore systems with Dynamic Voltage Frequency Scaling (DVFS) enabled processors. We rigorously formulate the problem as a Markov decision process (MDP), which simultaneously considers 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 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 scheduling algorithm using traces generated from a cycle-accurate multiprocessor ARM simulator.

Index Terms— Slice-parallel video decoding, multicore scheduling, multicore power management, dynamic voltage scaling, Markov decision process.

1. INTRODUCTION

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 workloads, which have worst-case execution times that are significantly larger than the average case. Second, they 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, they often have stringent delay constraints, but are considered soft real-time applications. 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 Dynamic Voltage Frequency Scaling (DVFS [7]) and/or Dynamic Power Management (DPM [12]) have been proposed, e.g. [2][3][4][6][8]. The Largest Task First with Dynamic Power Management (LTF-DPM) algorithm in [3] assumes that frame decoding deadlines are equally spaced in time, 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 Stochastic Scheduling2D algorithm [6] considers a periodic DAG application model that requires a "source" and "sink" node in each period, making the algorithm 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 [8] uses 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 [13] and parallelization approaches based on data partitioning (e.g. mapping different frames, slices, or macroblocks to different processors) are superior [13]. 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. However, if this technique is applied to video decoding applications, it will require retiming delays proportional to the GOP size, which may be large.

There is no solution that simultaneously considers per-core DVFS capabilities; dynamic processor assignment; and loss-tolerant tasks with different complexity distributions, DAG dependency structures, 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 abovementioned properties. The MDP enables the system to optimally tradeoff long-term power and performance.

• 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 determines scheduling and DVFS policies for each frame using frame-level MDPs, which account for the coupling between the optimal policies of parent and children frames. The second-level decides the final frame- and frequency-to-processor mappings, 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 [11].

The remainder of the paper is organized as follows. We introduce the system and application models in Section 2 and formulate the on-line multi-core scheduling problem as an MDP. In Section 3, we propose a lower complexity solution by approximating the original MDP problem with a two-level scheduler. In Section 4, we present our experimental results. We conclude in Section 5.

2. 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 2.1, we describe seven important video data attributes. In Section 2.2, we propose a sophisticated Markovian traffic/workload model that accounts for the video data attributes introduced in Section 2.1. In Sections 2.3, 2.4, and 2.5 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 subsection 2.6, we formulate the multicore scheduling problem as an MDP.

2.1. Video data attributes

We model the encoded video bitstream as a sequence of compressed data units. 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 [9]. 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 $\text{type}(v_k^g)$.
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 for all frames and is determined by the encoder [9].
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 i.i.d. random variable conditioned on the frame type.
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*: $d^{v_k^g,\text{disp}}$ denotes the final time slot in which v_k^g must be decoded so it can be displayed.
6. *Decoding deadline*: $d^{v_k^g,\text{dec}} \leq d^{v_k^g,\text{disp}}$ 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.
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 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 $(v_{k'}^{g'}, v_k^g) \in \mathcal{E}$ 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 determine which slices can be decoded, how long they will take to decode, when they need to be decoded. 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.

2.2. 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 the *frame working 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 frame working set as all the frames within the STW, i.e. $\mathcal{C}_t = \{v \in \mathcal{V} \mid d^{v,\text{disp}} \in \{t, t+1, \ldots, t + W_t\}\}$. Because the GOP structure is fixed and periodic, $\mathcal{C}_t$ is periodic with some period T. A frame's arrival time t^v (respectively, display deadline $d^{v,\text{disp}}$) is the first (respectively, last) time slot in which it appears in the frame working set, and a frame's decoding deadline $d^{v,\text{dec}}$ 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 itself before its display deadline. Fig. 1 illustrates the STW concept for a simple IBPB GOP structure.

We define the buffer state $\mathbf{x}_t = (x_t^v \mid v \in \mathcal{C}_t)$, where x_t^v denotes the number of slices of frame v awaiting decoding at time t. Finally, the dependency state $\mathbf{r}_t \triangleq (r_t^v \mid v \in \mathcal{C}_t)$ defines whether or not each frame in the frame working 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.

2.3. Scheduling actions and 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 $\mathbf{y}_t^j = (y_t^{jv} \mid v \in \mathcal{C}_t)$. There are three important constraints on the scheduling actions y_t^{jv} for all $j \in \{1, \ldots, M\}$ and $v \in \mathcal{C}_t$:

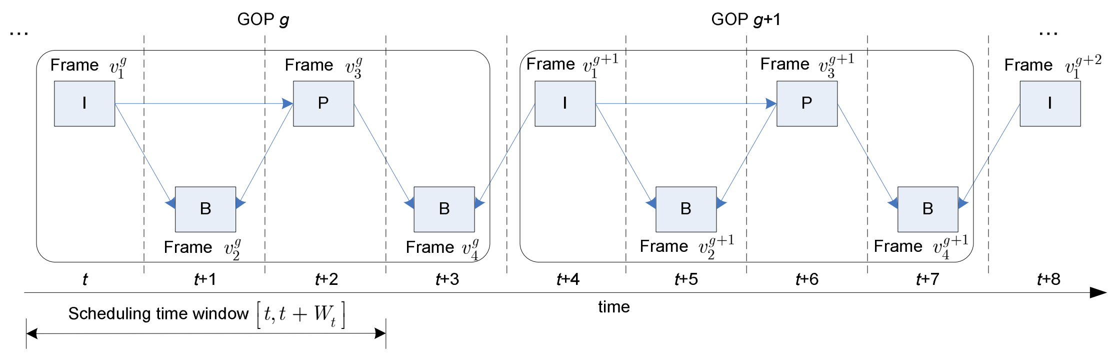

Fig. 1. Illustrative DAG dependencies for an IBPB GOP structure that contains $K = 4$ frames and lasts a period of $T = 4$ time slots of duration $\Delta t = 1 / 30$ seconds.

- *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 its buffer in time 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 time 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 tradeoff processing energy and delay. Let $\mathbf{f}_t = (f_t^1, f_t^2, \ldots, f_t^M) \in \mathcal{F}^M$ 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.

2.4. 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 frame working set from $\mathcal{C}_t$ to $\mathcal{C}_{t+1}$ is independent of the scheduling action; in fact, it is deterministic and periodic for a fixed GOP structure, and therefore the sequence $\{\mathcal{C}_t \mid t \in \mathbb{N}\}$ can be modeled as a deterministic Markov chain.

The transition of the buffer state from x_t^v to x_{t+1}^v depends on the scheduling actions and processor frequencies. Let $z_t^{jv} = z_t^{jv}(f_t^j, y_t^{jv})$ denote the number of slices belonging to frame v that finish decoding on processor j at time t given frequency f_t^j. Note that $z_t^{jv} \leq y_t^{jv}$. Let $p_z(z_t^{jv} \mid f_t^j, y_t^{jv})$ 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 frame working 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 $d^{v,\text{disp}} > t$, 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., $\mathcal{C}_{t+1} \setminus \mathcal{C}_t \triangleq \mathcal{C}_{t+1} - \mathcal{C}_t \cap \mathcal{C}_{t+1}$. Based on this partition, x_{t+1}^v can be determined from x_t^v as follows

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

The sequence $\{x_t^v \mid t \in \mathbb{N}\}$ can be modeled as a controlled Markov chain.

The transition of the dependency state from r_t^v to r_{t+1}^v follows intuitively from the definition of dependency: frame v can be decoded in time slot $t+1$ (i.e., $r_{t+1}^v = 1$) if and only if all of its parents are completely decoded at the end of time slot t. It follows that the sequence $\{r_t^v \mid t \in \mathbb{N}\}$ can be modeled as a controlled Markov chain.

2.5. Power cost and slice decoding rate

The power-frequency function $\rho(f_t^j)$ maps the jth processor's speed f_t^j to its expected power consumption (watts). We also consider the expected power consumed by the instruction, data, and L2 cache using a function $\sigma(f_t^j, y_t^{jv}, \text{type}(v))$ (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

$$P(\mathcal{C}_t, f_t^j, \mathbf{y}_t^j) = \rho(f_t^j) + \sum_{v \in \mathcal{C}_t} \sigma\left(f_t^j, y_t^{jv}, \text{type}(v)\right) \text{ (watts).} \quad (2)$$

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

$$Q\left(f_t^j, y_t^{jv}, \text{type}(v)\right) = \sum_{z_t^{jv} \leq y_t^{jv}} p_z(z_t^{jv} \mid f_t^j, y_t^{jv}) z_t^{jv}, \quad (3)$$

which 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 (3) as the *slice decoding rate*. In the remainder of the paper, we will omit the dependence of (2) and (3) on $\text{type}(v)$.

2.6. 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 long-term power consumption subject to a long-term slice decoding rate constraint. The total *discounted* [12] average power consumption and slice decoding rate can be expressed as

$$\bar{P} = \mathbf{E}\left[\sum_{t=0}^{\infty} \sum_{j=1}^{M} \gamma^t P(\mathcal{C}_t, f_t^j, \mathbf{y}_t^j)\right], \text{ and} \quad (4)$$

$$\bar{Q} = \mathbf{E}\left[\sum_{t=0}^{\infty} \sum_{j=1}^{M} \sum_{v \in \mathcal{C}_t} \gamma^t Q(f_t^j, y_t^{jv})\right], \quad (5)$$

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

$$\min_{\mathbf{f}_t,\ [y_t^{jv}]_{jv},\ \forall t \in \mathbb{N}} \bar{P} \ \text{ subject to } \ \bar{Q} \geq \bar{\eta} \quad (6)$$

where $\bar{\eta}$ is the slice decoding rate constraint. Note that the buffer, processor, and dependency constraints defined in Section 2.3 must hold in every time slot; however, we will omit them from our exposition in the remainder of the paper.

Equation (6) can be formulated as an unconstrained MDP by introducing a Lagrange multiplier $\lambda \in \mathbb{R}_+$ associated with the slice decoding rate constraint. 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 actions $[y_t^{jv}]_{jv}$, for all processors $j \in \{1, \ldots, M\}$ and all frames $v \in \mathcal{C}_t$, that minimize the discounted average Lagrangian cost: i.e.,

$$L_\lambda = \min_{\mathbf{f}_t,\ [y_t^{jv}]_{jv},\ \forall t \in \mathbb{N}} \left\{\bar{P} + \lambda\left(\bar{Q} - \bar{\eta}\right)\right\}. \quad (7)$$

3. LOW COMPLEXITY SOLUTION

Solving (6) and (7) is a computationally intractable problem because their complexity increases exponentially with the number of frames in the frame working sets and with the number of processors M. 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 on all of the processor cores. However, 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 (6) and (7): The first-level scheduler determines the optimal scheduling actions and processor frequencies for each frame under the (false) 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 by resolving conflicts in the first-level scheduling decisions.

3.1. First-level scheduler

The first-level scheduler computes a value function $V^v(\mathcal{C}, x^v, r^v)$ for every frame in a GOP, which provides a measure of the expected long-term Lagrangian cost under the optimized scheduling policy. Note that this value function only depends on the frame working set, the frame's buffer state x^v, and the frame's dependency state r^v and is independent of the buffer and dependency states of the other frames in the working set. Importantly, the frame working set indicates the remaining lifetime of a frame and describes the connections to its parents and children; hence, it has a significant impact on the optimal scheduling and DVFS decisions for the frame. To account for the dependencies among frames, we define the v th frame's value function $V^v(\mathcal{C}, x^v, r^v)$ 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 will impact the future performance of their children. We describe the first-level scheduler in more detail in the remainder of this section.

3.1.1. Frame-level value iteration

The first-level scheduler performs the frame-level value iteration algorithm illustrated in Table 1 to compute the optimal value functions $\{V^{v,*} : v \in \mathcal{V}^g\}$. Unlike the conventional value iteration algorithm [10], the proposed algorithm has multiple *coupled* value functions that need to be updated 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 1) differs 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 frame working set (i.e. $v \notin \mathcal{C}$), does not have its dependencies satisfied (i.e. $r^v = 0$), or 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 1). If the frame is in the frame working set, still has undecoded slices, and has its dependencies satisfied (i.e. $v \in \mathcal{C}$, $x^v > 0$, and $r^v = 1$), then 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 v th frame's children. Note that the expected future value of frame v is 0 if $v \notin \mathcal{C}'$; and, the expected future values of the child frames are 0 if $r^{u}{}'$ is not 1 (i.e., if the parent has not been decoded). 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.

3.1.2. Decomposing frame-level value iteration

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 frame working set, but we still have to address the fact that the optimization in (8) (Line 6 of Table 1) requires a search over an exponential number of scheduling and frequency actions. In this subsection, we discuss how to decompose the monolithic update defined in (8) into M stages (hereafter, *sub-*

Table 1. Frame-level value iteration algorithm performed by the first-level scheduler.

1.	**Initialize:** $V^v_{0,\lambda}(\mathcal{C}, x^v, r^v) = 0$ for all $v \in \mathcal{V}^g$, $\mathcal{C}$, $x^v \in \{0,\ldots,l^v\}$, and $r^v \in \{0,1\}$
2.	**Repeat**
3.	$\Delta \leftarrow 0$
4.	**For each** $v \in \mathcal{V}^g$, $\mathcal{C}$, $x^v \in \{0,\ldots,l^v\}$, and $r^v \in \{0,1\}$
5.	**If** $v \in \mathcal{C}$, $x^v > 0$, and $r^v = 1$ (frame v is in the frame working set, has undecoded slices, and has its dependencies satisfied)
6.	$V^v_{n+1,\lambda}(\mathcal{C}, x^v, r^v) = \min_{\mathbf{f}^{1:M,v}, \mathbf{y}^{1:M,v}} \left\{ \begin{array}{l} \sum_{j=1}^{M} \left[\rho(f^{jv}) + \sigma(f^{jv}, y^{jv}) - \lambda Q(f^{jv}, y^{jv}) \right] \\ \gamma \sum_{\mathbf{z}^{1:M,v} \leq \mathbf{y}^{1:M,v}} \prod_{j=1}^{M} p_z(z^{jv} \mid f^{jv}, y^{jv}) \left[V^v_{n,\lambda}\left(\mathcal{C}', x^v - \left\Vert \mathbf{z}^{1:M,v} \right\Vert_1, r^{v}{}'\right) + \sum_{u \in \mathcal{C}' : v \prec u,\ r^{u}{}' = 1} V^u_{n,\lambda}(\mathcal{C}', l^{u}{}', r^{u}{}') \right] \end{array} \right\}$ (8)
7.	**Else**
8.	$V^v_{n+1,\lambda}(\mathcal{C}, x^v, r^v) = 0$
9	**End**
10.	**End**
11.	$\Delta \leftarrow \max\{\Delta, \mid V^v_{n+1,\lambda}(\mathcal{C}, x^v, r^v) - V^v_{n,\lambda}(\mathcal{C}, x^v, r^v) \mid\}$ and $n \leftarrow n + 1$
12.	**Until** $\Delta < \epsilon$ (a small positive number)
13.	**Output:** $\{V^{v,*} : v \in \mathcal{V}^g\}$

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 j th 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 refer the interested reader to [14] for a derivation of the sub-value iterations.

Sub-value iteration at processor M :

$$V_{n,\lambda}^{M-1,v}\left(\mathcal{C},x^v,r^v\right)= \min_{f^{Mv},\ y^{Mv}}\left\{\begin{array}{l}\rho(f^{Mv})+\sigma(f^{Mv},y^{Mv})-\lambda Q(f^{Mv},y^{Mv})+\\ \gamma\mathbf{E}_{z^{Mv}|f^{Mv},y^{Mv}}\left[\begin{array}{l}V_{n,\lambda}^{v}\left(\mathcal{C}',x^v-z^{Mv},r^{v}{}'\right)+\\ \displaystyle\sum_{u\in\mathcal{C}':v\prec u,\ r^{u}{}'=1}V_{n,\lambda}^{u}(\mathcal{C}',l^{u}{}',r^{u}{}')\end{array}\right]\end{array}\right\} \quad (9)$$

The M th processor's sub-value iteration estimates the value of being in traffic state $\mathcal{T}^v=(\mathcal{C},x^v,r^v)$ under the assumption that only processor M exists in the current time slot, while all processors exist thereafter. This value is calculated as the sum of (i) the immediate cost incurred by processor M for processing slices belonging to frame v, (ii) the expected discounted future value of frame v, and (iii) the expected discounted future value of frame v 's children. The output of the M th processor's sub-value iteration is used as input to the $(M-1)$ st processor's sub-value iteration.

Sub-value iteration at processors $j\in\{2,\ldots,M-1\}$:

$$V_{n,\lambda}^{j-1,v}\left(\mathcal{C},x^v,r^v\right)= \min_{f^{jv},\ y^{jv}}\left\{\begin{array}{l}\rho(f^{jv})+\sigma(f^{jv},y^{jv})-\lambda Q(f^{jv},y^{jv})+\\ \mathbf{E}_{z^{jv}|f^{jv},y^{jv}}\left[V_{n,\lambda}^{j,v}\left(\mathcal{C},x^v-z^{jv},r^v\right)\right]\end{array}\right\}. \quad (10)$$

The j th processor's sub-value iteration estimates the value of being in traffic state $\mathcal{T}^v=(\mathcal{C},x^v,r^v)$ under the assumption that only processors $j,\ldots,M$ exist in the current time slot, while all processors exist thereafter. 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 j th processor's sub-value iteration is used as input to the $(j-1)$ st processor's sub-value iteration.

Sub-value iteration at processor 1:

$$V_{n+1,\lambda}^{v}(\mathcal{C},x^v,r^v)= \min_{f^{1v},\ y^{1v}}\left\{\begin{array}{l}\rho(f^{1v})+\sigma(f^{1v},y^{1v})-\lambda Q(f^{1v},y^{1v})+\\ \mathbf{E}_{z^{1v}|f^{1v},y^{1v}}\left[V_{n,\lambda}^{1,v}(\mathcal{C},x^v-z^{1v},r^v)\right]\end{array}\right\}. \quad (11)$$

The output of the first processor's sub-value iteration 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. $V_{n+1,\lambda}^{v}$ is used as input to the M th processor's sub-value iteration during iteration $n+1$.

Performing the M sub-value iterations for frame v in 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 $j\in\{1,\ldots,M\}$. Therefore, using the proposed decomposition significantly reduces the optimization complexity.

Finally, at run-time, we determine the approximately optimal actions $(f^{jv,*},y^{jv,*})$ to take in each state $\mathcal{T}^v=(\mathcal{C},x^v,r^v)$ by taking the arguments that minimize the right-hand sides of (9), (10), and (11). For complete details on the action selection procedure, we refer the interested reader to [14].

3.2. Second-level scheduler

Given the optimal actions calculated by the first-level scheduler, it is likely that slices belonging to different frames in the frame working set will want to be scheduled on the same processor in the same time slot, thereby violating the processor constraint in (6). To avoid this problem, the second-level scheduler determines the final slice- 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 $v^{j,*}$ has the minimum decoding deadline $d^{v,\mathrm{dec}}$ of all of the frames scheduled on processor j (with ties 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 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.

4. EXPERIMENTS

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 cycle-accurate and bus signal-accurate MPARM simulator [11]. We implemented the two-level scheduling algorithm proposed in Section 3 in Matlab. This algorithm, together with slice-level data traces recorded from MPARM, allowed us to determine 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. The relevant parameters used in our experiments are given in Table 2.

In Fig. 2, we compare our proposed algorithm to the Optimum Minimum-Energy Multicore Scheduling algorithm (OPT-MEMS [2]), and to a modification of our algorithm where we require all processors to operate at the same frequency (i.e., coordinated DVFS). We note that OPT-MEMS supports both DPM and coordinated DVFS; however, we only compare against the DVFS part to achieve a fair comparison. (Although DPM can be integrated into our proposed solution, we omitted it here to simplify the exposition.)

OPT-MEMs uses a frame's worst-case execution complexity and its deadline to determine a DVFS schedule that multiplexes between two frequencies in time in order to execute exactly the worst-case number of cycles before the task's deadline. There are four important limitations of OPT-MEMS. First, OPT-MEMS does not consider characteristics and requirements of future tasks (e.g. deadlines, complexities, dependencies) when deciding the DVFS schedule for the current task. Second, OPT-MEMS does not provide a scheduling technique to allocate tasks to processor cores; instead, it assumes that each task is perfectly divisible among an arbitrary number of cores. This corresponds to the case of perfect load balancing, which can only be achieved in practice if the number of slices per frame is exactly the number of cores, and each slice has exactly the same decoding complexity. Third,

Table 2. 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. frame working sets	12
No. slices per frame	8
Lagrange multiplier (λ)	400

OPT-MEMS does not provide a mechanism for scheduling slices belonging to different frames at the same time. This leads to some inefficiency because fully parallelized decoding (which accounts for frame dependencies) is not possible. Forth, OPT-MEMS uses coordinated DVFS. This leads to inefficiency in practice because tasks are not the same size and therefore cannot be perfectly load balanced with a single frequency for all cores.

As illustrated in Fig. 2(a) and Fig. 2(b), for M = 1 or 2 processors, all algorithms achieve approximately the same frame rates and power consumptions for a given sequence. This is because, even at the highest operating frequency, there are not enough resources to decode all frames. For M = 4 or 8 processors, Fig. 2(a) and Fig. 2(b) show that all algorithms achieve the full frame rate (or very close to the full frame rate); however, Fig. 2(c) and Fig. 2(d) show that the proposed algorithm achieves lower overall power consumption. For M = 4 cores, the proposed algorithm reduces power by approximately 24% for Foreman and 36% for Silent, relative to OPT-MEMS. The improvements are more modest for M = 8 cores because each core runs at a much lower operating frequency than with M = 4 cores, so there is less opportunity to reduce power consumption.

5. 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 validate 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.

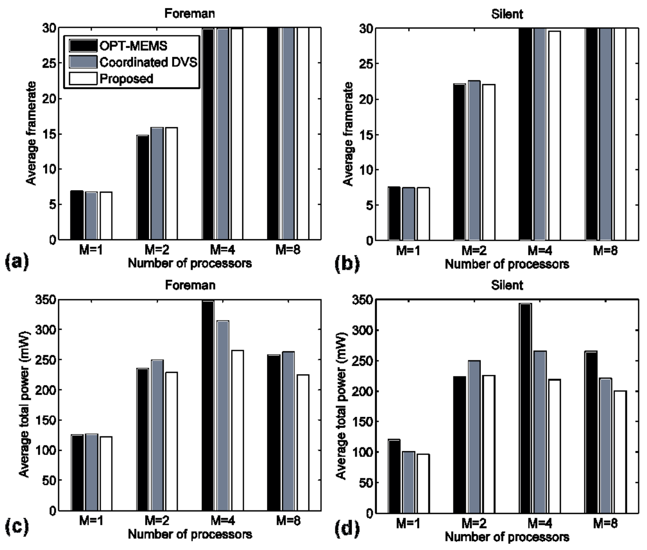

Fig. 2. Experimental comparisons (a,b) Avg. decoded frame rates for Foreman and Silent, respectively. (c,d) Avg. total power consumption for Foreman and Silent, respectively.

6. REFERENCES

[1] H. Aydin and Q. Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," *Proc. of the 17th International Symposium on Parallel and Distributed Processing* (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," *Proc. of the 2009 Intl. Conf. on Hybrid Information Technology (ICHIT '09),* pp. 273-277, 2009.

[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," *Proc. of the 2010 ACM Symp. on Applied Computing (SAC '10)*, pp. 258-262, 2010.

[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," *Proc. of the 2008 Euromicro Conference on Real-Time Systems (ECRTS '08)*, pp. 92-101, July 2008.

[5] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1-6, pp 5–35, 1991.

[6] R. Xu, "Energy-aware scheduling for streaming applications," Ph.D. Dissertation: http://preview.tinyurl.com/c6uul4m

[7] 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.

[8] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," *Proc. of the Conference on Design, Automation and Test in Europe (DATE '09)*, pp. 411-416, 2009.

[9] M. Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding," *Proc.of the 7th ACM & IEEE International Conference on Embedded Software*, pp. 269-278, 2007.

[10] R. S. Sutton and A. G. Barto, "Reinforcement learning: an introduction," Cambridge, MA:MIT press, 1998.

[11] 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, Sept. 2005.

[12] L. Benini, A. Bogliolo, G. A. Paleologo, G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, June 1999.

[13] E. B. van der Tol, E. G. Jaspers, R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," *Proc. of the SPIE* (May 2003), pp. 707-718.

[14] N. Mastronarde, K. Kanoun, D. Atienza, P. Frossard, and M. van der Schaar, "Markov decision process based energy-efficient on-line scheduling for slice-parallel video decoders on multicore systems," *IEEE Trans. on Multimedia*, vol. 15, no. 2, pp. 268-278, Feb. 2013.