Online Energy-Efficient Task-Graph Scheduling for Multicore Platforms

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

***Abstract*—Numerous directed acyclic graph (DAG) schedulers have been developed to improve the energy efficiency of various multicore platforms. However, these schedulers make *a priori* assumptions about the relationship between the task dependencies, and they are unable to adapt online to the characteristics of each application without offline profiling data. Therefore, we propose a novel energy-efficient online scheduling solution for the general DAG model to address the two aforementioned problems. Our proposed scheduler is able to adapt at run-time to the characteristics of each application by making smart foresighted decisions, which take into account the impact of current scheduling decisions on the present and future deadline miss rates and energy efficiency. Moreover, our scheduler is able to efficiently handle execution with very limited resources by avoiding scheduling tasks that are expected to miss their deadlines and do not have an impact on future deadlines. We validate our approach against state-of-the-art solutions. In our first set of experiments, our results with the H.264 video decoder demonstrate that the proposed low-complexity solution for the general DAG model reduces the energy consumption by up to 15% compared to an existing sophisticated and complex scheduler that was specifically built for the H.264 video decoder application. In our second set of experiments, our results with different configurations of synthetic DAGs demonstrate that our proposed solution is able to reduce the energy consumption by up to 55% and the deadline miss rates by up to 99% compared to a second existing scheduling solution. Finally, we show that our DAG flow manager and scheduler have low complexities on a real mobile platform and we show that our solution is resilient to workload prediction errors by using different estimator accuracies.**

***Index Terms*—Adaptive, directed acyclic graph, energy-efficient scheduler, multimedia embedded systems, online.**

Manuscript received September 29, 2013; revised January 18, 2014; accepted March 11, 2014. Date of current version July 15 2014. This work was supported in part by a Joint Research Grant for ESL-EPFL by CSEM, in part by the BodyPoweredSenSE (20NA21 143069) RTD projects evaluated by the Swiss NSF and funded by Nano-Tera.ch through the Swiss Confederation, and in part by Grant NSF CNS 1016081. This paper was recommended by Associate Editor Y. Xie.

K. Kanoun and D. Atienza are with the Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne 1015, Switzerland (e-mail: karim.kanoun@epfl.ch; david.atienza@epfl.ch).

N. Mastronarde is with the Department of Electrical Engineering, State University of New York at Buffalo, Buffalo, NY 14260 USA (e-mail: nmastron@buffalo.edu).

M. van der Schaar is with the Department of Electrical Engineering, University of California at Los Angeles, 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/TCAD.2014.2316094

I. Introduction

EMERGING real time video processing applications such as video data mining, video search, and streaming multimedia (see H.264 video streaming [13] or the new High Efficiency Video Coding standard (HEVC) [17]) have stringent delay constraints, complex Directed Acyclic Graph (DAG) dependencies among tasks, time-varying and stochastic workloads (due to the underlying video source characteristics), and are highly demanding in terms of parallel data computation. Multimedia applications are in general modeled with DAGs where each node denotes a task, each edge from node j to node k indicates that task k depends on task j and each group of tasks has a common deadline d_i. As illustrated in the first layer of Fig. 1, DAG models for applications with dependent tasks can be roughly classified into four types depending on the relationship between the task dependencies and task deadlines. First, applications with independent deadlines are modeled with independent DAGs where each DAG models the dependencies among tasks that share a common deadline. In Fig. 1, we illustrate two different examples of independent DAGs models: DAG model 1 shows a periodic DAG and DAG model 2 illustrates the general case (i.e., aperiodic). Second, applications with dependent deadlines are modeled with multiple connected DAGs where each DAG models the dependencies among tasks that share a common deadline. However, in this DAG model, there are also dependencies between tasks with different deadlines. In Fig. 1, we also illustrate two different examples of dependent DAGs: DAG model 3 shows the widely studied fork-join model [16] where only a single join edge links the last task with deadline d_i to the first task with deadline d_{i+1} and DAG model 4 illustrates the general case where a task's children may have different deadlines than the task itself and its other children.

The number of cores embedded in new mobile platforms is continuously increasing (see Exynos 5 Octa [22], Tegra 4 [23], and Snapdragon 800 [24]). Thus, numerous energy-efficient task-graph scheduling algorithms [5], [8], [10]–[12] that take advantage of dynamic frequency voltage scaling (DVFS) enabled cores embedded in modern mobile platforms have been proposed to schedule the aforementioned DAG models. Approaches [10], [12] also take advantage of dynamic power management (DPM) which is used to switch off unused cores in order to reduce leakage energy. In [29], has been proved that the leakage power also should be considered to minimize the energy consumption. In Fig. 1, we first classify existing solutions based on the applied DAG monitoring solution and

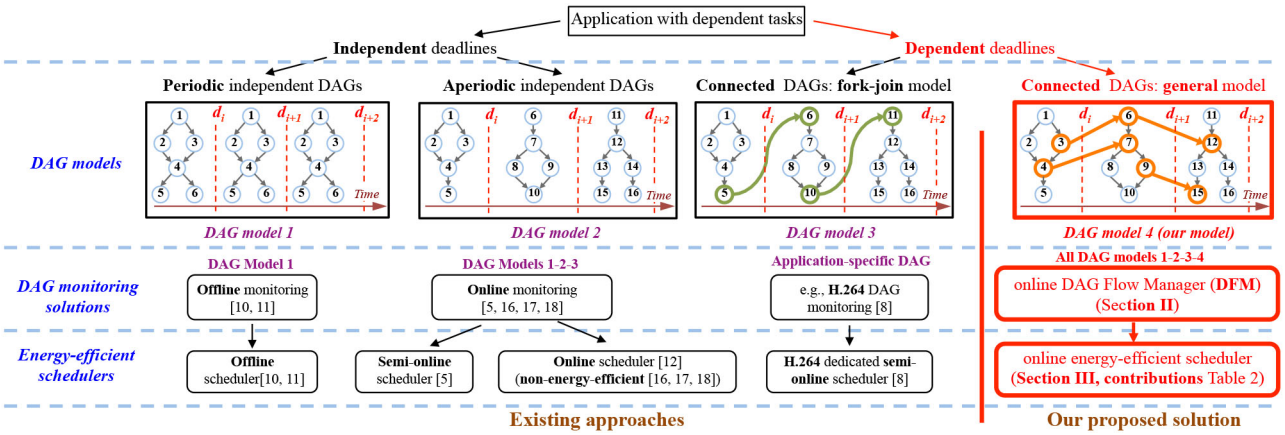

Fig. 1. Comparison of our application model, our online DAG monitoring solution, and our online energy-efficient scheduler among state-of-the-art solutions.

their considered DAG models. We define the DAG monitoring solution as the module used to process and analyze the DAG before scheduling the tasks. While a scheduler is responsible for core assignment and DVFS selection, a DAG monitoring solution is responsible for finding parallelization opportunities, tracking the execution of the DAG and preparing relevant information related to each task in the DAG. Then, we further categorize existing solutions based on their energy-efficient scheduling analysis techniques (i.e., offline, semi-online and online). Offline schedulers determine the scheduling policy (i.e., list of possible scheduling decisions such as a look-up table), core selections, and DVFS assignments at design-time. Semi-online schedulers are similar to offline solutions except that the scheduling decisions are made online based on the current execution status of the application and the offline computed scheduling policy. Finally, online schedulers generate a scheduling decision based only on the current status of the application without the need of any profiling data.

We contend that none of the existing online [12] and semi-online [5], [8] scheduling approaches have considered the general case of the DAG shown in Fig. 1 with DAG model 4, in which a task's children can have different deadlines. Instead, existing online and semi-online approaches convert this type of DAG into a fork-join model, as illustrated in DAG model 3 of Fig. 1. The fork-join parallelism model [16] forces all tasks with deadline d_i to finish processing before any tasks with deadline d_{i+1} can be processed, thereby missing various parallelization opportunities, wasting energy, and potentially increasing deadline miss rates. Only one semi-online approach [8] has considered the multiple connected DAGs model; however, this solution was only optimized for H.264 video decoding and it requires profiling data to build a look-up table for each video stream, which limits its ability to be self-adaptive at run-time to different video workloads. Finally, existing static approaches rely on worst case execution time and assume a periodic DAG (see DAG model 1 of Fig. 1). Thus, they are not suitable for applications that adapt their dependency structure on the fly at run-time and have dynamic workloads (see stream mining applications [14]).

To summarize, each existing scheduler implements its own DAG monitoring solution with several restrictions on the DAG model. Moreover, none of the existing solutions are able to handle the general DAG illustrated in Fig. 1 in DAG model 4, which allows a task's children to have different deadlines. Finally, existing scheduling solutions are unable to adapt online to the characteristics of each application without the need of offline profiling data.

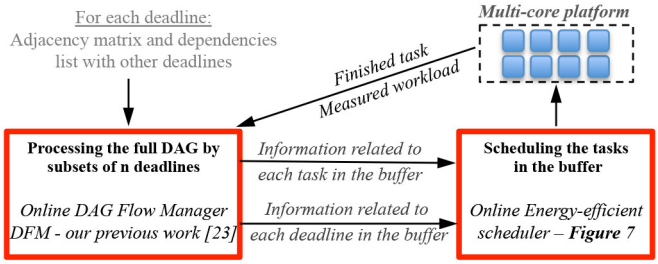

Fig. 2. Overview of our novel scheduling approach for multicore platforms.

To address the aforementioned problems, we propose a novel energy-efficient online scheduler for multicore DVFS- and DPM-enabled platforms for the general DAG model. Our online scheduler integrates the DAG monitoring solution presented in [21] called the DAG flow manager (DFM). Fig. 2 illustrates the interconnection between the DFM and scheduler modules and how they interact online with the application and the platform.

The key contributions of our approach are as follows.

1) Our solution is a low-complexity online technique that is fully independent from the considered DAG model.
2) It does not impose any restrictions on the DAG (see restrictions on deadline dependencies as in the fork-join model). Our scheduler covers online all DAG models (Fig. 1).
3) Our scheduler is self-adaptive to the characteristics of each application and does not require any offline profiling data.
4) Our scheduler efficiently handles execution with very limited resources by avoiding scheduling specific tasks, that are expected to miss their deadlines, in order to reduce the overall deadline miss rates.
5) Our scheduler is resilient to workload prediction error.

We validate our approach against existing solutions [8], [12]. In the first set of experiments, our results for the H.264 video decoder demonstrate that our proposed low-complexity solution

for the general DAG model reduces the energy consumption by up to 15% compared to a sophisticated state-of-the-art scheduler [8] that was specifically built for H.264 video decoding. In the second set of experiments, our results with different configurations of synthetic DAGs demonstrate that our proposed solution is able to reduce the energy consumption by up to 55% and the deadline miss rates by up to 99% compared to a second existing scheduler [12]. Finally, we show that our DFM and scheduler have low complexities on an Apple A6 SoC [25] and we show that our solution is resilient to workload prediction errors by using different estimator accuracies.

The remainder of this paper is organized as follows. In Section II, we first introduce our system and application model and present the first phase of our approach namely our online DFM to address limitations related to processing a general DAG model online. Then in Section III, we describe the second phase of our approach, namely, our online adaptive energy-efficient scheduling solution and how it exploits the data prepared by our DFM. In Section IV, we present our experimental results on the H.264 video decoder [1] and also on different configurations of synthetic DAGs [6]. In Section V, we describe the limitations of existing DAG monitoring solutions and energy-efficient schedulers. Finally, we summarize the main conclusions in Section VI.

II. Phase 1: Online DAG Flow Manager (DFM)

A. Application and Platform Power Model

We model computationally intensive applications (see [13] and [17]) as a DAG $G =< \mathcal{N}, \mathcal{E} >$ of dependent tasks t_j with nondeterministic workload w_j and coarse-grained soft deadlines. $\mathcal{N}$ is the node set containing all the tasks. $\mathcal{E}$ is the edge set, which models the dependencies among the tasks. Each node in the DAG denotes a task t_j. e_j^k denotes that there is a directed edge from t_j to t_k indicating that task k depends on task j. Each task t_j is characterized with its index j and a deadline d_i. Our solution allows coarse-grained deadlines where a deadline can be assigned to a subset of tasks indexed by i. Our model covers all general DAG models (see all DAG models of Fig. 1) including the general case where a task's children may have different deadlines than the task itself and its other children (i.e., DAG model 4 of Fig. 1).

Our targeted multicore platform has M processors. The major sources of power dissipation from each processor can be broken down into dynamic power P_{dyn} and leakage power P_{leak} [26]. The dynamic power consumption can be controlled by the selected frequency and the supply voltage (see using DVFS) while the leakage power can be minimized with power gating techniques (see using DPM). In our model, we assume that each core has DVFS capability to trade off energy consumption and delay. Each processor can operate at a different frequency $f_i \in F$, where F denotes the set of available operating frequencies and $f_i < f_{i+1}$. Finally, we assume that each processor has two different modes namely, active and sleep modes. In the active mode, the processor runs normally (i.e., full leakage power consumption) while in the sleep mode the processor is power gated (i.e., inactive with reduced leakage power consumption). Each time a processor is switched to sleep mode, it requires X_{switch} clock cycles to wake up and switch to active mode. Our power model is based on the case study realized in [26].

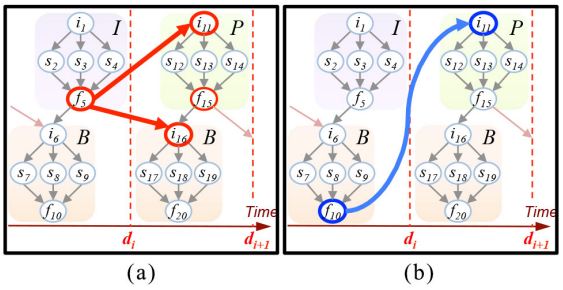

Fig. 3. H.264 decoder DAG model [13]. (a) Original model. (b) Fork-join model.

B. Limitations of Existing DAG Models

The H.264 decoding process is characterized by coarse-grained deadlines. An example of these deadlines and the dependencies between I, P, and B frames is illustrated in Fig. 3(a) where I-frames are compressed independently of the other frames, P-frames are predicted from previous frames, and B-frames are predicted from previous and future frames [13]. Each frame is composed of three types of tasks, namely, initialization (see i_1), slice decoding (here we illustrate three slices per frame, see s_2, s_3, and s_4) and the deblocking filter (see f_5). In the example shown in Fig. 3(a) with four frames (I-B-P-B), there are two deadlines corresponding to the display deadlines of the two B frames. These deadlines are imposed by the frame rate and the underlying dependency structure. For instance, if the decoder is running at 30 frames per second, then frame k has to be displayed at $k/30$ s (display deadline). However, if frame k depends on frame $k+l$ (with $l > 0$), then both frames will have their deadlines set to the minimum one, i.e., $k/30$ s (decoding deadline). Finally, in this DAG model, a task's children may have different deadlines (see $[f_5 \rightarrow i_6, f_5 \rightarrow i_{16}]$) as in the 4th DAG model in Fig. 1.

Existing approaches (see Section V) do not consider DAG models where a single task's children can have different deadlines [see Fig. 3(a)]. Therefore, they are not able to correctly handle such DAGs without applying additional modifications. In fact, existing approaches are forced to use the fork-join DAG model as presented in Fig. 3(b) where critical edges (i.e., edges linking a task to other tasks having different deadlines) are removed and replaced by a single join edge that links the last task with deadline d_i to the first task with deadline d_{i+1}. Although the fork-join model preserves the dependency coherency between tasks, it restricts the scheduler to operate one deadline at a time (i.e., the earliest deadline). Hence, several parallelization opportunities are missed.

C. Our Proposed Online DAG Flow Manager

In our solution, we integrate the DFM that we proposed in [21] to monitor general DAG models and to prepare a set of outputs for our scheduler. In this section, we only provide a brief description about the DFM applied task decomposition and the outputs it generates for our scheduler. Full technical details about the DFM are provided in [21].

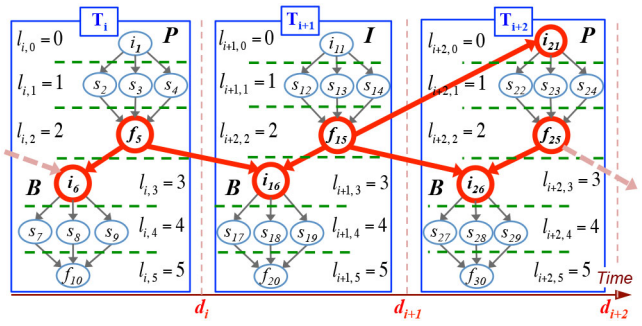

Fig. 4. Decomposition applied by our DFM on H.264 DAG [13].

We define T_i as the subset of tasks having the same deadline d_i (we refer to T_i as a deadline task set). We also define the Working Set *WS* as a look-ahead window buffer with N T_is. The DFM processes the full DAG of the application using this *WS* buffer where only a limited number of deadlines are monitored at a time. When all of the tasks in deadline task set T_i finish executing, the DFM requests the next T_i input from the application. Each T_i is associated with an adjacency matrix, a deadline and a list of edges connecting it with T_{i+l} (with $l \neq 0$). This information must be provided to the DFM by the application. Analyzing the full DAG of an application by subsets of N deadlines (see T_is) is the key to having a low complexity online DAG monitoring solution.

We denote by $l_{i,k}$ the group of tasks t_j having the same depth level $\delta_i^j = k$ in T_i. Note that, for all tasks in T_i, tasks at depth level $k+1$ (see $l_{i,k+1}$) can only be scheduled after tasks at depth level k (see $l_{i,k}$) are finished. Fig. 4 illustrates in detail the difference between t_j, T_i, and $l_{i,k}$ after applying the DFM algorithm on a *WS* buffer containing part of the DAG of the H.264 video decoder from Fig. 3. The DFM provides an output structure called the Priority Table to assist online schedulers with immediate parallelization opportunities and the priorities of the available tasks. Each entry in the Priority Table corresponds to a task t_j, which is characterized by its estimated workload w_j (clock cycles), its fixed deadline d^i (seconds), with $j \in T_i$, and the dependency status r_j (see the total number of incomplete parent tasks that it still depends on). The tasks in the Priority Table are sorted by the DFM according first to their deadline d_i, then refined to their depth level $l_{i,k}$ in T_i and finally to their estimated workload in case of a tie. Several workload estimation methods [3], [20] with negligible overhead have been proposed for multimedia applications. In our DFM, we use a workload predictor based on the Kalman filter [3] to estimate the workload of each task.

The DFM indicates to the scheduler which tasks are entry nodes in the remaining tasks set using the dependency status r_j of each task t_j. Nodes with $r_j = 0$ in the Priority Table are potential starting tasks for parallelization. The DFM provides also output to track the overall progress of each group of tasks T_i such as total workload, executed workload, scheduled workload. Finally, for each T_i in the working set buffer, it provides information related to each $l_{i,k}$ in T_i, namely, the total workload, the maximum number of allowed cores and the minimum amount of parallelizable workload.

III. Phase 2: Online Energy-Efficient Adaptive Scheduler

A. Problem Statement

Given a multicore platform with M cores, k frequencies and two power modes (i.e., active and sleep mode) per core, and the output of the DFM, namely, the Priority Table, the dependency status r_j, the depth values δ_i^j, the estimated workload w_j of each task t_j and the overall progress of each group of tasks T_i, the goal is to find an online schedule that is fully adaptive to the different application characteristics that minimizes the deadline miss rate and the total energy consumption of the available cores, without the need of any offline profiling data. The proposed scheduler should be able to efficiently schedule multiple deadlines simultaneously (i.e., frequency selection and core assignment for each task) and to decide when to switch cores to sleep mode. Finally, the proposed scheduler should automatically detect when a system is very congested and avoid scheduling tasks that are expected to miss their deadlines and do not have an impact on future deadlines in order to reduce the deadline miss rate.

B. Motivational Example: Scheduling General DAG Models with Dependent Deadlines

We compare our scheduling solution to an online scheduler [12] that applies the least possible restrictions on its application model compared to other state-of-the-art solutions. We use a general DAG model with dependent deadlines as illustrated in Fig. 5(a). In [12], an online scheduling approach called was proposed for multimedia applications, where the earliest deadline is scheduled with limited consideration of future tasks' deadlines and workloads. Indeed, based on the derived estimated duration of all pending tasks, a new virtual deadline is set in order to have more balanced workload distribution over the time. The number of required active cores for the next deadline is computed based on the estimated energy consumption to complete available deadlines in the buffer for different cores configurations. Then, a largest task first (LTF) schedule is applied and each core is given a minimum frequency that meets its assigned workload requirement for the next deadline. Fig. 5(b) shows how the M cores-Largest Task First-Dynamic Power Management (MLTF-DPM) algorithm [12] schedules the task set illustrated in Fig. 5(a) along with the tasks' workloads and deadlines (in time slots). For this example, we assume that there are two cores in the platform and each core has two frequency/voltage pairs ($f_1^k = \frac{f_{max}}{2}, V_1$) and ($f_2^k = f_{max}, V_2$) with $V_1 < V_2$ and $k = (1 \text{ or } 2)$. For instance, task 1 (i.e., t_1) will require one time slot when executed with f_2 and two time slots when executed with f_1. As shown in Fig. 5(b), one out of the two deadlines is missed and all the scheduled workload is executed with f_{max}. Clearly, schedulers assuming a fork-join model are unable to produce a balanced workload, which results in several wasted gaps (i.e., when a core is idle and waiting for another task to finish). In the next section we present an overview of our proposed scheduling solution and we show how it can efficiently schedule the general DAG in Fig. 5(a).

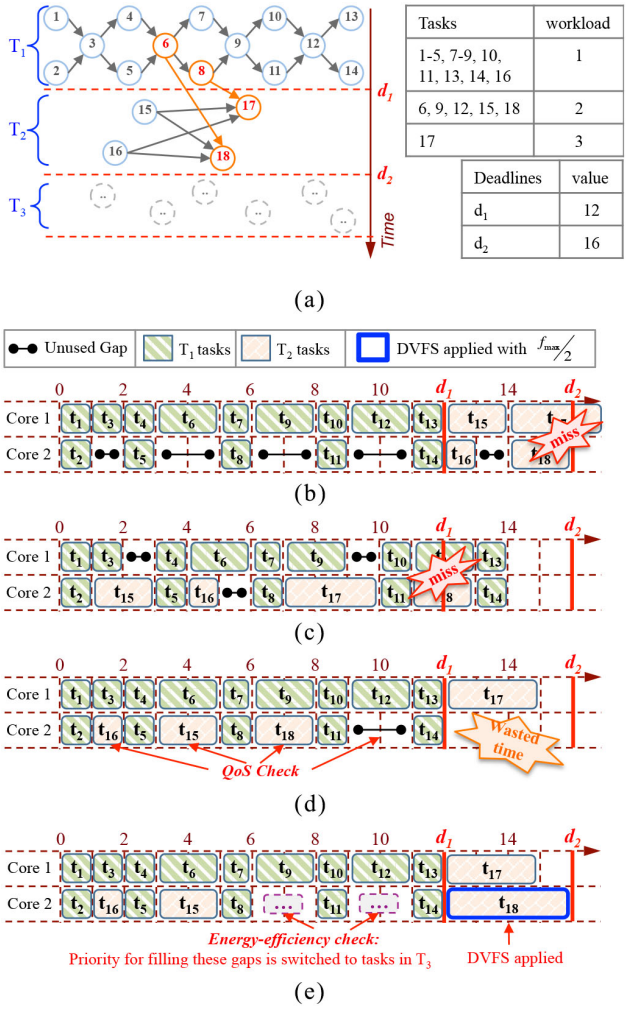

Fig. 5. Motivational example for our proposed energy-efficient scheduler. (a) Motivational example: dependent deadlines DAG (general case). (b) Fork-join scheduling model [12]. (c) Filling the gaps without QoS and energy efficiency checks. (d) Filling the gaps: QoS check applied. (e) Filling the gaps: QoS and energy-efficiency checks applied.

C. Overview of the Proposed Scheduling Solution

For a working set buffer with N deadlines monitored by our DFM, we define T_e as the subset of tasks with the earliest deadline d_e and T_{e+l} (with $0 < l < N$) as the subset of tasks with a future deadline d_{e+l}. When all the tasks in T_e are scheduled then T_{e+1} will become T_e. We identify then N subsets of tasks to schedule (i.e., T_e, T_{e+1},..., T_{e+N-1}). Given this working set decomposition provided by our DFM output, first we tune the deadline value of each T_i in the *WS* buffer in order to create more balanced workload over the time. Then, and as illustrated in Fig. 6, we propose two different scheduling algorithms running at the same time and cooperating together to schedule available tasks in the *WS* buffer. While, the first scheduling algorithm is responsible for mapping tasks with the earliest deadline (i.e., T_e), the second scheduling algorithm is responsible for mapping tasks from future deadlines (i.e., T_{e+l} with $0 < l < N$) to the gaps generated by the first scheduler.

The first online scheduling algorithm dedicated to T_e is a single priority scheduler. In fact, given the critical path workload of the remaining tasks in T_e, we determine the minimum operating frequency such that the earliest deadline d_e is likely to be met. Moreover, and with the support of our DFM output, our first scheduling algorithm is also responsible for automatically detecting when a system is very congested and selecting online from T_e the appropriate tasks drop when their deadlines are expected to be missed. In fact, the proposed scheduling solution is designed for soft real-time tasks where deadline misses are tolerable, but degrade the quality of service (QoS). However, because of the dependencies among tasks, not all tasks are equally important, so it is possible to improve overall performance by selectively dropping unimportant tasks. The earliest deadline scheduler is illustrated in Fig. 6 by the modules to the right of the T_e task set arrow.

The schedule for tasks in T_e may create several gaps, which can be filled with ready tasks that have later deadlines, namely, ready tasks in T_{e+l} with $0 < l < N$. Filling these gaps has the effect of reducing the workload intensity and relaxing the dependency constraints for future tasks. However, if a ready task in T_{e+l} takes longer than the gap to finish, then it will delay the processing of remaining tasks in T_e, possibly leading to deadline misses. Fig. 5(c) illustrates an example of a scheduler filling a gap without comparing the workload to the size of the available gap. The first deadline is then missed. To avoid this, we propose a second scheduling algorithm to fill the gaps with tasks from T_{e+l}, for $0 < l < N$. As illustrated in Fig. 6 by the modules to the right of the T_{e+l} task set arrows, our T_{e+l} scheduler provides first a QoS check in order to make the decisions coherent with the T_e scheduler choices. Fig. 5(d) illustrates an example of our QoS check module applied on the schedule of Fig. 5(a). With the new generated schedule there are no more deadlines misses, however, there is wasted time (or slack time) between d_1 and d_2 that can be used more efficiently. Thus, the scheduling algorithm for filling these gaps should not be overly aggressive (i.e., it should not fill the gaps all the time) because tasks in T_{e+l} will have a dedicated time in the future to be processed (i.e., when T_e becomes empty and tasks from T_{e+l} migrate to T_e set), which should be used efficiently.

To avoid this slack time and reduce the energy consumption, our T_{e+l} scheduler provides also an energy-efficiency check module, which is applied before filling a gap. This module takes into account: 1) previous QoS check and 2) the execution status of each available T_i in the predecoding buffer provided by the DFM to estimate the maximum allowed frequency for an energy-efficient execution. In fact, tasks in T_{e+l} will be allocated dedicated time in the future to be processed (i.e., when T_e becomes empty and tasks from T_{e+l} migrate to T_e), which should be used efficiently. Scheduling a gap with a task using a higher frequency than it will require if it is scheduled in its future allocated time could be less energy efficient. Fig. 5(e) illustrates an example of the energy efficiency check applied on top of the previous QoS check. For gaps 6–8 and 9–11, we notice that our scheduler switched the priority for filling these gaps to tasks from T_3. Our scheduler detected that scheduling t_{17} in its original allocated time (i.e., between d_1 and d_2) is more energy efficient (i.e., using lower frequency with a lower voltage) than scheduling it in the gap. By applying the QoS and energy efficiency checks together, our approach is then able to remove the deadline miss and to reduce the energy consumption as well. We apply these checks

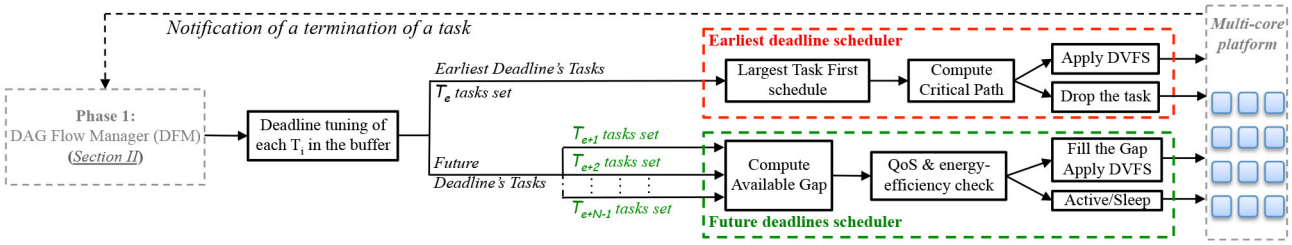

Fig. 6. Complete overview of our novel online energy-efficient scheduler.

Algorithm 1 Computing the Critical Path Workload

1: **for** m in cores **do**
2: $\varphi_{e,0,m} \leftarrow 0$
3: **end for**
4: **for** $l_{e,k}$ in T_e **do**
5: **for** tasks t_j in $l_{e,k}$ **do**
6: $m \leftarrow argmin_m \varphi_{e,k,m}$
7: $\varphi_{e,k,m} \leftarrow \varphi_{e,k,m} + w_j$
8: **end for**
9: $\varphi_{e,k} \leftarrow \max_m \varphi_{e,k,m}$
10: **for** m in cores **do**
11: $\varphi_{e,k+1,m} \leftarrow \varphi_{e,k}$
12: **end for**
13: **end for**
14: **return** $\varphi_{e,last}$

Algorithm 2 Earliest Deadline Scheduler

1: Optimizing the deadline values d_e^v (Similar to [12] but applied to all the deadlines in the *WS* buffer)
2: Compute critical path workload $\varphi_{e,last}$ of remaining tasks in T_e using Algorithm 1
3: **if** $\varphi_{e,last} \leq d_e^v$ **then**
4: Schedule the task with

$$f^{DVFS} = \underset{f_n \in F}{\text{argmin}} \left\{ f_n : \frac{\varphi_{e,last}}{d_e^v - (current\ time)} \leq f_n \right\} \quad (1)$$

5: **else if** $d_e^v \leq \varphi_{e,last} \leq d_e$ **then**
6: Schedule the task with f_{max}
7: **else if** $d_e \leq \varphi_{e,last}$ **then**
8: **if** all dependencies with other T_is are cleared out **then**
9: Drop the remaining tasks in T_e
10: Cancel the execution of currently running tasks in T_e
11: **else**
12: Schedule the task with f_{max}
13: **end if**
14: **end if**

each time before a task is scheduled and using updated information (generated by our DFM) related to all the tasks in the buffer creating foresighted decisions from the scheduler. This guarantees continuous adaptation to each application's time-varying workload at run-time.

D. Optimizing the Deadlines

An optimized selection of the DVFS value of each mapped core requires a balanced workload distribution over time [12]. Therefore, in our approach, we set new virtual deadlines, respecting the original ones, based on the solution presented in [12]. In [12], only the earliest deadline is tuned once before scheduling all its tasks and based on the derived estimated workload of all pending T_is in the *WS* buffer. This approach is only efficient if the scheduler assigns tasks from earliest deadline only. However, in our approach, we schedule tasks from multiple T_i at the same time. Thus, we tune the deadline value of each of the available T_is in the *WS* buffer each time a core becomes available. This will guarantee a continuous adaption to the execution status of the applications. In fact, we use the updated information provided by the DFM (see Section II-C) namely: total workload and executed workload of each T_i in the *WS* buffer to estimate the remaining workload of each T_i. We denote the virtual deadline value of T_i with d_i^v. The number of clock cycles allocated to each of the available T_is in the *WS* buffer is then $(d_i^v - d_{i-1}^v) * f_{max}$.

E. Earliest Deadline Tasks Scheduler

In the algorithm presented in this section, we define $\varphi_{e,k}$ (with $k = \delta_e^j$ and e refers to the earliest deadline d_e) as the number of clock cycles after which all the tasks t_j in $l_{e,k}$ have finished their execution. We call $\varphi_{e,k}$ a global synchronization point between $l_{e,k}$ and $l_{e,k+1}$ tasks (i.e., it indicates the completion of $l_{e,k}$ tasks and the start of $l_{e,k+1}$ tasks). We also use $\varphi_{e,k,m}$ to indicate the number of clock cycles after which the core m becomes available in depth level $l_{e,k}$ of T_e.

By using the Priority Table, the depth level information $l_{e,k}$, and each task's estimated workload w_j (all provided by our DFM as described in Section II-C), the scheduler simulates a LTF schedule of all the remaining tasks for each of the $l_{e,k}$, with $k \geq 0$, in order to compute the critical path workload among all the cores. We describe this procedure in Algorithm 1. In lines 1–3, we initialize a temporary synchronization point $\varphi_{e,0,m}$ (cycles) for each core to 0. In lines 5–8, we perform an LTF schedule for each $l_{e,k}$. In lines 9–12, the algorithm selects the core with the maximum workload obtained from the simulated schedule of $l_{e,k}$ tasks and uses it as the new global synchronization starting point for the next level $l_{e,k+1}$. By repeating these steps to all the remaining $l_{e,k}$ for d_i (line 4), the total critical path workload among all the cores is then obtained. The returned number of cycles $\varphi_{e,last}$ (line 14, "*last*" representing the last depth level in T_e) is then the critical path workload among the cores.

We illustrate an example of the execution of the critical path workload computation algorithm in Fig. 7, where the total

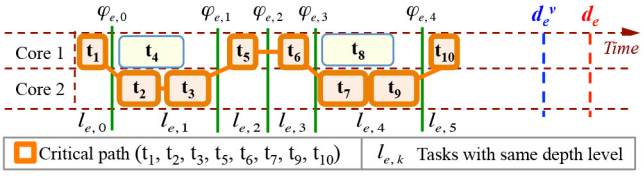

Fig. 7. Finding the critical path workload among the cores in the platform.

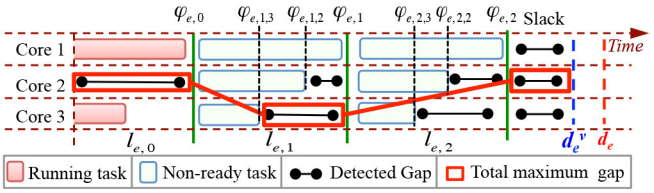

Fig. 8. Computing the maximum gap size among the cores in the platform.

critical path workload is obtained from both core 1 ($l_{e,0}$, $l_{e,2}$, $l_{e,3}$ $l_{e,5}$) and core 2 ($l_{e,1}$, $l_{e,4}$).

As shown in Algorithm 2 (line 3–14), our scheduler either schedules the next available task or drops all the remaining ones in T_e depending on the current execution status of T_e. In fact, if $\varphi_{e,last} < d_e^v$ then the task is scheduled with f^{DVFS} (line 4), which guarantees the execution of the critical path workload before the virtual deadline d_e^v. However, If $d_i^v < \varphi_{e,last} < d_e$ then the task is scheduled with the maximum available frequency. Finally, if $\varphi_{e,last} > d_e$ then there is a high probability that deadline d_e will be missed. Therefore, we check for the dependency status of T_e to decide if we can safely avoid scheduling the rest of the T_e task set. We use then the DFM output to check if the dependencies of T_e have been already cleared out with all T_js for $j \neq e$. If it is the case, then we avoid scheduling all the remaining tasks of T_e and we cancel the execution of all currently running tasks in T_e. Otherwise, we schedule the task with the maximum frequency.

Due to all the uncertainties in the workload estimation and the generated gaps that are used later by the second scheduler, the f^{DVFS} value is calculated each time a core and a task in T_e are available. Once T_e becomes empty, remaining tasks of the next earliest deadline set T_{e+1} will migrate to T_e.

F. Gap Detection

Tasks in T_e with deadline d_e that are currently running have the same depth level $l_{e,0}$ because there are no dependencies between tasks at the same depth. Therefore, If a core becomes available and there are no tasks ready to be scheduled from T_e, a gap is then detected on the available core as all the remaining tasks of T_e start from $l_{e,k}$ with $k \geq 1$ and depend on at least one of the tasks currently running in $l_{e,0}$. We calculate the gap's maximum size, which respects previous DVFS frequency decision f_e^m made by core m on tasks with deadline d_e, as follow: we compute the first synchronization point $\varphi_{e,0}$ (cycles) based on the tasks currently running from $l_{e,0}$ with their corresponding selected frequencies and we initialize then the total gap value with $\varphi_{e,0}$. Then, starting from $\varphi_{e,0}$, we simulate an LTF schedule on the remaining unscheduled tasks of T_e, very similar to the one described in Algorithm 1 but taking into account the previously computed f^{DVFS} (see Section III-E) on the workload of each task in the simulated schedule in order to respect previous DVFS frequency decision. To this end, in Algorithm 1 we change line 7 with $\varphi_{e,k,m} \leftarrow \varphi_{e,k,m} + w_j \times (f_{max}/f^{DVFS})$. Once the LTF schedule is simulated we can easily compute the available gap at each depth level knowing $\varphi_{e,k}$ and $\varphi_{e,k,m}$ provided by algorithm 1. For each $l_{e,k}$ we only consider the maximum gap with $gap_{e,k} \leftarrow \max_m(\varphi_{e,k} - \varphi_{e,k,m})$. Note that, the gap occurring in $l_{e,k}$ is a flexible gap that can happen either at the beginning or at the end of the schedule of $l_{e,k}$. Therefore, we add it to the gap detected in $l_{e,k-1}$ in order to give more flexibility to the T_{e+l} task set scheduler proposed in the next section. Therefore, starting from $\varphi_{e,0}$, two consecutive gaps are concatenated until either the last level $l_{e,last}$ is reached or if at least one level $l_{e,k}$ has a number of tasks more or equal than the number of cores. For instance, in Fig. 8, the gap occurring in $l_{e,2}$ was not taken into account because $l_{e,1}$ already has three tasks and the number of cores is 3. Finally, the last part that we add to the total gap size is the slack time occurring at the end of the deadline which can be easily deduced with $slack = d_e^v - \varphi_{e,last}$. We illustrate an example of the calculation of the gap's maximum size in Fig. 8, where the maximum total gap is the sum of the gaps detected in core 2 for $l_{e,0}$, core 3 for $l_{e,1}$ and the slack time $d_e^v - \varphi_{e,2}$.

Algorithm 3 QoS and Energy Efficiency Checks Before Filling the Gaps with Nonearliest Deadline Tasks T_{e+l}

1: Compute available gap g_j (see Section III-F)
2: **for** each T_{e+l} in the *WS* buffer with $l > 0$ **do**
3: Compute the minimum required frequency for the critical path workload of T_{e+l} to meet d_{e+l}

$$f_{e+l,cp} = \underset{f_n \in F}{\text{argmin}} \left\{ f_n : \frac{\varphi_{e+l,last}}{d_{e+l}^v - d_{e+l-1}^v} \leq f_n \right\} \quad (2)$$

4: **for** each available tasks t_j with workload w_j and deadline d_{e+l} from the priority table (i.e., DFM) **do**
5: **if** $g_j \geq w_j$ **then**
6: Compute the lowest frequency such as w_j can be completed within the gap g_j

$$f^{j,QoS} = \underset{f_k \in F}{\text{argmin}} \left\{ f_k : \frac{w_j}{g_j} \leq \frac{f_k}{f_{\max}} \right\} \quad (3)$$

7: **if** $f^{j,QoS} \geq f_{e+l,cp}$ **then**
8: Fill the gap with t_j at $f^{j,QoS}$
9: **return** 1
10: **end if**
11: **end if**
12: **end for**
13: **end for**
14: **if** $g_j \geq w_j$ **then**
15: Switch the core having the gap to sleep mode
16: **end if**

G. Energy-Efficient Tasks Scheduler to Fill the Gaps

We describe this procedure in Algorithm 3. Our algorithm goes first through each of the available T_{e+l} in the buffer in ascending order of their deadlines (line 2). Then, as shown

with 2 in Algorithm 3, we compute the critical path frequency $f_{e+l,cp}$ which denotes the minimum frequency that meets the deadline requirement of the critical path workload $\varphi_{e+l,last}$ (cycles) of T_{e+l} during the time $d^v_{e+l} - d^v_{e+l-1}$. The critical path workload $\varphi_{e+l,last}$ is computed using Algorithm 1 with T_{e+l} tasks as presented in Section III-E. Once $f_{e+l,cp}$ is calculated. We go through each available task t_j in T_{e+l} in the order given by the priority table provided by the DFM module (line 4). We compare then the gap size g_j and the task workload w_j. If the task workload is greater or equal to the gap size, then scheduling the current task t_j may cause some of the remaining tasks in T_e (i.e., tasks with earliest deadline) to miss their deadline. Therefore, in this case, we request the next available task from the Priority Table until the condition $(g_j \geq w_j)$ is met. If no available task in the buffer satisfies this QoS condition then the gap is not filled. However, if a task t_j, satisfying the previous condition (i.e., $(g_j \geq w_j)$ exists, then we proceed with the energy efficiency check which allows us to take a foresighted decision based on future deadlines' execution status. To this end, and as shown in 3 of Algorithm 3, we compute the minimum allowed frequency $f^{j,QoS}$ to use in the gap for the given workload w_j. In words, $f^{j,QoS}$ is the lowest frequency at which a task of workload size w_j can be completed within a gap of size g_j. Finally, by comparing $f_{e+l,cp}$ and $f^{j,QoS}$ we can determine if scheduling the task in T_{e+l} in the available gap at frequency $f^{j,QoS}$ is energy-efficient or not. If $f^{j,QoS} > f_{e+l,cp}$, then it will cost more energy to schedule t_j in the gap at frequency $f^{j,QoS}$ than to schedule it later at its allocated time (i.e., between d^v_{e+l-1} and d^v_{e+l}) at frequency $f_{e+l,cp}$ (note that in both cases t_j will meet its deadline). Therefore, if the energy efficiency check is met then the gap is filled, otherwise we move to the next available task in the Priority Table and we apply the QoS and energy efficiency checks again. If there is no task that satisfies these two conditions and the gap meets the minimum duration requirements for a core to be efficiently switched to sleep mode (i.e., $g_j \geq X_{switch}$) then the core is switched to sleep mode for the duration of the gap in order to reduce the leakage power (line 14–16).

The gap-filling algorithm is the main module in the scheduler responsible for reducing the dynamic power and leakage power. In fact, filling gaps with future deadlines creates a more balanced workload distribution among the cores and makes it easier to meet the deadline constraints with the lowest possible frequency. However, for processors that have significant leakage power, using the lowest processor speed may in fact increase the overall energy consumption. The proposed approach can be easily adapted to this situation by specifying to the algorithm the minimum frequency starting from which the leakage power of the considered processor is not significant for the estimated duration of the task.

IV. Experimental Results

We have implemented in C our task-graph scheduling approach, namely, the DFM, the deadline tuning module, the T_e task set scheduler (including the task dropping feature) and the T_{e+l} task set scheduler (i.e., the gap-filling algorithm) as presented in Sections II-C and III. We have also implemented a module that simulates if a core is active or depending on the mapped task and its DVFS value (i.e., the execution status of the application). This last module requires then accurate measurement of the workload of each task in order to detect when a core finishes the execution of its mapped tasks. Therefore, we use accurate statistics generated from the tested application executed on a sophisticated multiprocessor virtual platform simulator. In fact, in this paper, we use the multiprocessor ARM (MPARM) virtual platform simulator [4], 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. In our experiments, we have generated with MPARM the workloads and the dynamic power consumption statistics of each slice decoding task using ARM9 (90nm technology) power consumption figures with DVFS support (300MHZ at 1.07V, 400MHZ at 1.24V, and 500MHZ at 1.6V). We use the estimated average dynamic power consumption generated by MPARM per task per voltage as well as the execution time of each task. Finally our leakage power model and sleep mode model are based on a case study of an ARM946 realized in [26]. Thus, we assume that the leakage power consumption is 12% of the maximum dynamic power consumption. The leakage power is cut by 96% when the processor is switched to sleep mode and the processor requires a few hundred clock cycles to wake up.

We demonstrate the advantages of our online energy-efficient scheduler compared to existing scheduling approaches [8], [12] in terms of energy-efficiency, deadline miss rates, overhead and workload prediction error resiliency on a set of experimental benchmarks.

1) A real multimedia application: H.264 video decoder [1].
2) Multiple different configurations of synthetic DAGs generated with GGEN tool [6].

A. Comparison with State-of-the-Art Schedulers Using the H.264 Video Decoder Application

1) Experimental Setup: In this first set of experiments, we compare our general scheduling solution to a sophisticated approach [8] specifically designed for the H.264 video decoder application and based on a Markov decision process (MDP) formulation. Even though the approach states that it is an online solution, the online part is only restricted to a look up table generated at design time for each platform configuration or new video input. Indeed, the approach proposed in [8] generates an optimized scheduling policy offline and tuned individually to each video input at design time using offline profiling data. The scheduler proposed in [8] is then a semi-online solution. We only compare to its dynamic energy consumption as [8] does not take into account leakage energy consumption.

In our benchmark setup, we also consider a second online scheduler with DPM and DVFS capability namely the MLTF-DPM approach presented in [12] and which we have already described in Section III-B. In [12], future tasks' deadlines and workloads are taken into account before scheduling T_e. The MLTF-DPM approach models a multimedia application as a sequence of jobs that must be executed one after the other.

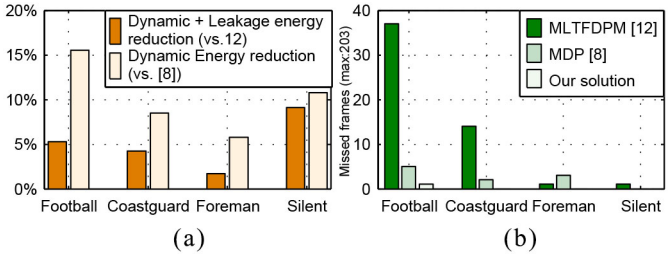

Fig. 9. H.264 decoder benchmarks on four different video stream with WS = 4 and #cores = 4. (a) Energy reduction resulted by our scheduler with respect to [12] and [8]. (b) Number of missed frames out of 203 decoded frames.

Each job J_i can be further partitioned into n_i sub-jobs running on several cores in parallel with the same deadline d_i. If we compare the MLTF-DPM application model to our application model, each job J_i is similar to $l_{i,k}$ except that in our general DAG model $l_{i,k}$ does not have a fixed deadline (in Section II-C, we defined $l_{i,k}$ as the group of tasks t_j having the same depth level $\delta_i^j = k$ in T_i). To make the MLTF-DPM scheduler a generic approach, we need then to add a virtual deadline to each $l_{i,k}$ in the working set buffer. First, we optimize the deadline value of each T_i and compute the minimum required number of active cores as described in [12]. Then, we calculate an optimized virtual deadline for each $l_{e,k}$ in T_e. To this end, we compute the maximum allowed deadline for each $l_{e,k}$ to finish its execution before the tuned deadline of T_e. We apply then an LTF schedule on T_e starting from its last depth level and going back in the time from the tuned deadline of T_e to set the maximum value of each global synchronization point $\varphi_{e,k}$. Finally, we apply the technique used in MLTF-DPM on the previously computed $\varphi_{i,k}$ to set the final optimized virtual deadline of each $l_{i,k}$ as described in [12].

For our multimedia benchmark, we use the Joint Model reference software (JM 17.2) of an H.264 video decoder [1]. The DAG model that we consider for our benchmark is similar to the one shown in Figs. 3(a) and 4 with an IBPB GOP structure. However, we have used eight slices per frame instead of three slices. We validate our proposed solution on four CIF resolution video sequences namely: *Foreman, Silent, Football*, and *Coastguard*. These sequences have different motion characteristics, which impact the workload intensity and variation. We measure the workload and the energy consumption of tasks using the aforementioned H.264 video decoder that we have parallelized and executed on MPARM. The workload measurements are used only to detect when a core is active or idle and they are not used in the decision of our solution.

Finally, in our DFM we have used a workload predictor based on the Kalman filter [2], [3] to estimate the workload of each task in the application. Each type of task in the DAG is handled with its own workload predictor. By using different estimator accuracies in our experiments in Section IV-C, we show that our scheduler is resilient to prediction error.

2) Comparison of the Multimedia Benchmarks—Energy Consumption and Deadline Miss Rates: In Fig. 9, we show the total energy reduction and the deadline miss rates obtained by our proposed solution compared to the MDP [8] and the MLTF-DPM [12] approaches on a four-core platform for the *Foreman, Silent, Football*, and *Coastguard* sequences. For our scheduler, we have processed the H.264 DAGs by a working set buffer size of four deadlines (i.e., 80 nodes and 136 edges for this example). As shown in Fig. 9, our scheduler outperforms the sophisticated MDP approach [8] and the online MLTF-DPM [12] in all tested scenarios. In fact, compared to [8], our approach has reduced the dynamic energy consumption by up to 15% for the football sequence and we have even reduced the number of missed frames from 5 to 1. This is because the MDP approach assumes that the system is time slotted and that slice-scheduling and DVFS decisions are determined at the beginning of each time slot. The discrete-time assumption drives it to underestimate the amount of slices that can be decoded in each time slot, and therefore drives it to select higher processor frequencies than necessary, which leads to suboptimal energy consumption. Moreover, each frame ignores the fact that other frames might require system resources (i.e., each frame-level MDP assumes that all of the cores are available for that frame until it is finished decoding, which is not true). This assumption drives the MDP to select lower than optimal processor frequencies. In contrast, in our new work, we consider an event-driven system in which scheduling and DVFS decisions are made every time a processor core becomes available. Moreover, the task scheduling and DVFS decisions account for the resources being used by other tasks. This results in reduced unused gaps, more energy-efficient operation, and less deadline violations.

When comparing to [12], we have also reduced the total energy consumption by up to 9% for the silent sequence and reduced the number of missed frames from 37 to 1 in the *Coastguard* sequence with 5% of total energy reduction. Finally, if we consider only *Foreman* and *Silent*, where at most three frames were missed for all schedulers, we observe a higher energy reduction when compared with MDP, thus the MLTF-DPM consumes less energy than MDP. This is because, in the MLTF-DPM approach, the DPM and the deadline tuning optimization module assume a balanced workload which is the case here as we have four cores and eight slices per frames that is two slices per core for each frame. However, and as shown in Fig. 9(b), the MLTF-DPM scheduler fails to efficiently schedule *Football* and *Coastguard* as they have different motion characteristics resulting into different workload variation topologies. Unlike the solutions proposed in existing approaches, in our scheduler, we tune all the deadlines available in the working set at the same time, and we use the gap efficiently in order to create more balanced workload distribution among the cores and relax future deadline constraints. Finally we have also shown that our generic online solution was able to adapt to the workload variation characteristics of each of the four tested video sequences without the need of any profiling data.

B. Generalizing the Results with Synthetic DAG Models

1) Experimental Setup: Existing online energy-efficient schedulers assume either periodic task model or no dependencies between different T_is tasks (see Sections I and V). These schedulers are then unable to directly work with any general DAG that we consider in our application model

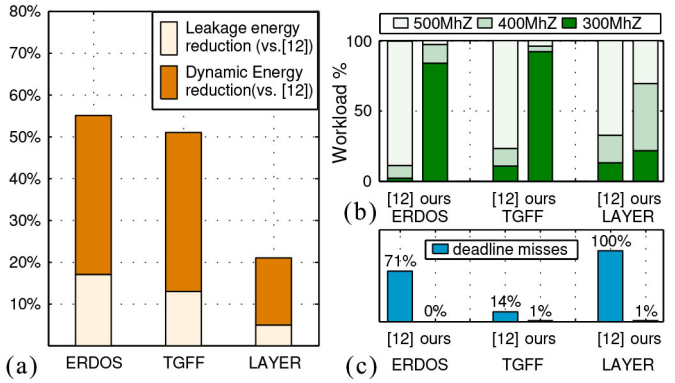

Fig. 10. Synthetic DAGs with deadlines set 10% less than the critical path workload ($\beta = -0.1$). Comparison between our solution and an adapted version of [12]. (a) Leakage and dynamic power consumption reduction. (b) Frequency usage. (c) Deadline miss rates.

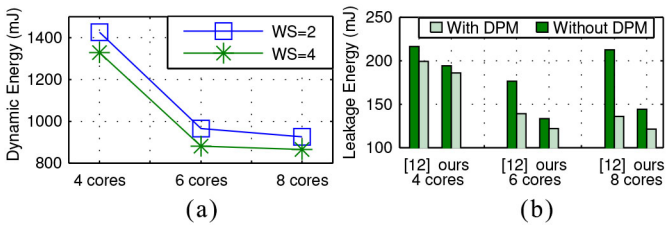

Fig. 11. Synthetic DAGs: comparison of the results generated by our solution for different buffer sizes and different number of cores in the platform (with $\beta = -0.2$). (a) Dynamic energy consumption. (b) Leakage energy consumption with and without DPM technique.

(see Section II-A). Therefore, we compare our results to the adapted version of MLTF-DPM, as described in Section IV-A.

To generate different synthetic DAG configurations for our general DAG model benchmark, we use the GGEN [6] tool to model an application with 100 connected DAGs. We connect these DAGs by randomly adding m edges in a way that some tasks in DAG g depend on some other tasks in DAG $g-1$. For the workload, we assume that an application of n tasks has k types of workloads. We assign then each task t_j with a random type number a_j between 1 and k, and we compute the workload with $w_j = (1+\alpha)w(a_j)$ where $w(a_j)$ is the minimum workload value of all the tasks with type a_j and $\alpha \in [0, 0.5]$. α represents the workload variation. Finally, to assign realistic deadline to each DAG, we compute the critical path workload w_i^{cp} of each DAG i by simulating an LTF schedule on a six-core platform and using Algorithm 1. The final deadlines (in seconds) are assigned with $d_i = d_{i-1} + w_i^{cp} * \frac{1+\beta}{f_{max}}$ with $\beta \in [-0.25, 0.25]$. Negative β is used to simulate tight deadlines and positive β is used to simulate relaxed deadlines.

2) Comparison of the Synthetic DAGs Benchmarks: Energy Consumption and Deadline Miss Rates: In this section, we first compare our solution to MLTF-DPM [12] presented in the previous section, for different configurations of synthetic DAG models. Then, we discuss how our solution scales with respect to the number of considered deadlines in the buffer and the number of cores in the platform. Finally, we illustrate the benefit of avoiding scheduling tasks that are expected to miss their deadlines and do not have an impact on future deadlines.

In the first set of general DAG experiments, we generate the synthetics DAGs with Erdos, Task Graphs For Free (TGFF), and Layers DAG generation methods as presented in [6]. We set previously described parameters to $n = 25$, $k = 5$, $\beta = -0.1$ (i.e., deadlines are 10% less than the critical path workload), $\alpha = 0.4$ and $m \in [5, 10]$. For TGFF method, we set the maximum number of ingoing and outgoing edges per node to 4, for Erdos and Layer we set the probability of an edge to appear in each DAG to 0.5, and for the Layer method we set the number of layers to 4. We choose these parameters in order to simulate a slightly congested system. Fig. 10(a) shows the dynamic and leakage power reduction compared to MLTF-DPM [12]. Our solution has reduced the total energy consumption by up to 55% for the Erdos DAG model and 51% for the TGFF DAG model. The energy reduction for the Layer DAG model is less significant than the other two DAG models with up to 21% of energy reduction. This is is due to the dependency topology that we have set in our Layer DAG model. In fact in our Layer DAG generation method, we generate 25 tasks distributed into four layers, which is around six tasks per layers. Scheduling this DAG on a six-core platform provides a better balanced workload distribution among the cores and generates less gaps than the other models creating then a more congested application.

To better understand how this significant energy reduction is obtained for different DAG models, in Fig. 10(b), we show the distribution of the frequency usage of the total workload assigned by [12] and our solution. A higher fraction of workload processed at lower frequencies is desirable because it indicates lower dynamic energy consumption. We also compare the deadline miss rates in Fig. 10(c). Our solution significantly reduces the usage of the maximum frequency by up to 86% and the deadline miss rates by up to 99% (with 0% to 1% miss rates overall). Even though, the deadline miss rates reduction is less significant for TGFF, we still have a total energy reduction by up to 51%. The MLTF-DPM solution fails to schedule these three configurations of synthetic DAG models for two main reasons. First, it assumes a balanced workload distribution among the cores when selecting the number of cores to be activated and when tuning the deadline values, however, this is not correct in reality because of the dependencies, which make it difficult to balance the load. Second, the deadlines of these DAG models were set to be 10% less than their original deadlines (the original values were set based on an LTF schedule on a six-core platform and using Algorithm 1). Thus, when a deadline is missed, it will be difficult for the MLTF-DPM solution to catch up on the following deadline because it does not fill the available gaps. Unlike MLTF-DPM, in our solution we efficiently use the gap-filling algorithm to relax the workload for future deadlines and to create a real balanced workload distribution making then the assumption made when tuning all the deadlines more realistic and correct. Filling the gaps with tasks with future deadlines significantly reduces the usage of the maximum frequency resulting in less energy consumption. Moreover, our solution implements a finer grained technique to select which cores to switch to sleep mode based on the current execution status of each core as described in Section III-G.

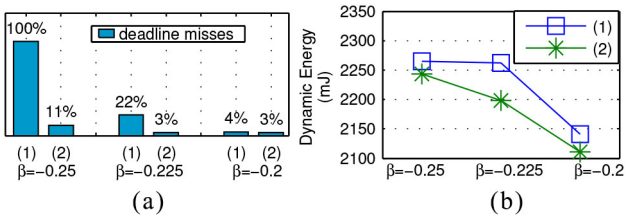

Fig. 12. Synthetic DAGs: comparison of the results generated by: 1) our solution scheduling all the tasks and 2) our solution avoiding scheduling tasks missing their deadlines for very congested systems with $\beta = -0.2, -0.225$, and -0.25. (a) Deadline miss rates. (b) Energy consumption.

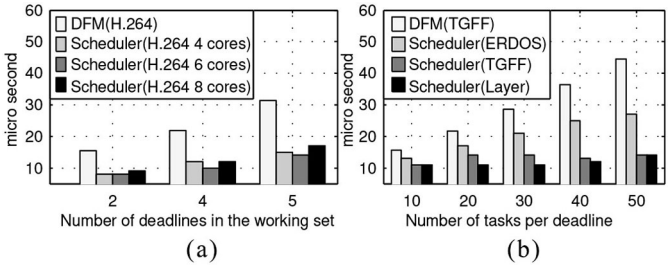

Fig. 13. Average execution time per DFM and scheduler call. (a) Variation of the number of deadlines per working set and the number of cores (with 20 tasks per deadline and using the H.264 video decoder DAG model). (b) Variation of the number of tasks per deadline and the dependency topology (with four deadlines per working set and using six cores).

Then, in the second set of synthetic DAGs experiments, we illustrate how our solution scales with respect to the number of considered deadlines in the buffer and the number of cores in the platform. We use the DAG Layer model with 20 tasks per deadline divided into six layers. We set β to 0.2 (i.e., deadlines are set to 20% less than the critical path workload). We vary the buffer size ($ws = 2$ deadlines and $ws = 4$ deadlines) and the number of cores. We limit the number of cores to 4, 6, and 8 as the deadlines are originally set based on the critical path workload among the cores of an LTF schedule simulated on the generated DAG on a six-core platform (i.e., we generate a DAG designed to run on a platform with at least six cores and reduce its tasks' deadlines by 20% of their original values). We show then how the scheduler scales when it has two cores less or more than the minimum required ones. Fig. 11(a) shows the dynamic energy consumption for each configuration while Fig. 11(b) shows the leakage energy consumption obtained by our scheduler with and without DPM. We do not show the deadline miss rates in the figure, however, they vary between 2% and 4% depending on the configuration. Thus, our scheduler was able to efficiently schedule, on a four-core platform, a DAG that was supposed to run on at least six-core platform thanks to the management of multiple deadlines at the same time in the *WS* buffer. In fact, it was not possible to schedule such a DAG when using a buffer size $ws = 1$ even on a six-core platform as shown in the previous set of experiments with MLTF-DPM scheduler. Thus, considering more than one deadline in the working set buffer can remove the need of having more cores. Our solution can be used then in systems to determine how many cores to use at run-time.

Fig. 11 shows two main results. First, the dynamic energy consumption decreases with respect to the number of cores. The energy reduction is less significant from six to eight cores than from four to six cores, which means that the available parallelizations in the DAG require no more than six cores. Thus, more gaps are likely to be unused for six- and eight-core platforms. Our scheduler is able to detect these unused gaps and switch their cores to sleep mode. This is illustrated by the leakage energy consumption of our solution with DPM in Fig. 11(b) with respect to the number of cores. In fact, the DPM solution implemented in our scheduler efficiently switches the cores to sleep mode and keeps the same leakage level as with the six cores configuration. Second, the dynamic energy consumption decreases with respect to the number of deadlines in the working set buffer, as more parallelization opportunities can be exploited. This is also illustrated in Fig. 11(b). The figure shows that for the case where the DPM is not used, the leakage energy consumption is significantly reduced between $ws = 2$ and $ws = 4$ as more gaps are then efficiently filled and the leakage power is then reduced. Our solution efficiently adapts then to different platform configurations.

Finally, in Fig. 12, we illustrate the benefit of avoiding scheduling tasks that are expected to miss their deadlines. We use the Layer DAG generation method to generate DAGs with parameters similar to the ones generated previously but using a very congested systems. We set $m \in [1, 5]$ and $\beta = -0.2, -0.225$, and -0.25. Fig. 12(a) shows the deadline miss rates generated by: 1) our solution scheduling all the available tasks and 2) our solution using the task dropping feature. We do not show the results generated by [12] as this solution was not able to schedule this type of DAG even for $\beta = -0.1$ [see Fig. 10(b)]. Our results show that by partially avoiding scheduling some tasks, the task dropping feature can significantly improve the deadline miss rates in very congested systems. In fact, for $\beta = -0.25$, by selectively dropping some tasks in six different deadline task sets, we decreased the miss rates from 99% to 11%. Moreover, for $\beta = -0.225$ and -0.2, by selectively dropping some tasks in two different deadline task sets, we decreased the miss rates from 22% and 4% to only 3% for both cases. Additionally, our task dropping feature reduces the energy consumption by up to 2.8% as shown in Fig. 12(b).

C. Computation Overhead Analysis

To measure the overhead of our proposed solution, we run the full algorithm on the Apple A6 SoC using the iPhone 5 mobile platform [25] and we measure the average execution time per DFM call and per scheduler call. Fig. 13 shows the average time spent by the DFM and the scheduler each time a task finishes its execution for different DAG and platform configurations. In Fig. 13(a), the measured average execution time slightly increases with respect to the number of deadlines in the working set. In fact, the decomposition technique that we have applied in our working set buffer allows each T_i to be processed separately. Thus, it made it possible to increase the number of deadlines in the working set with an acceptable overhead. For the scheduler, the measured execution time also slightly fluctuates when we vary the number of cores because the gap pattern changes, which has an effect on the number of times the gap filling module is called. During the

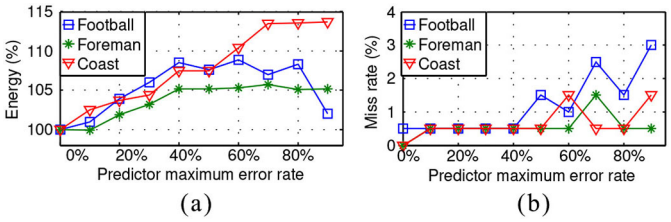

Fig. 14. Evaluation of our algorithm with respect to different workload estimator error rates. (a) Energy consumption. (b) Deadline miss rates.

full execution of an application, the DFM and scheduler are called as many times as the number of tasks in the application. Therefore, if we consider the example of the H.264 video application running at 30 frames/s with 10 tasks per frame (initialization + eight slice decoding + deblocking filter), there are $10 \times 30 = 300$ tasks to schedule in 1 s. Thus, our module will be called 300 times in 1 s. As shown in Fig. 13(a), for the six-core configuration, $32\mu s$ are needed for the DFM and the scheduler to be executed once. $9.6ms$ ($32\mu s \times 300$ tasks) are then required by the DFM and the scheduler to map the tasks of 30 frames which is less than 1% of the available 1 s for decoding 30 frames.

We also illustrate the average execution time of our DFM and scheduler for each DAG topology (i.e., Erdos, TGFF and Layer) in Fig. 13(b) with respect to the number of tasks per deadline and using a six-core configuration. For clarity, in Fig. 13(b), we show the execution time of the DFM only for the TGFF case as we measured similar results for other DAG models. The average execution time of the DFM increases linearly with respect to the number of tasks per deadline. For the scheduler, the average execution time depends on the dependency topology of the DAG. Our scheduler spends more time with Erdos DAG than other models. In fact, Erdos produces more gaps than the other considered DAG models, as we found that the GAP filling method was called up to 2.2 times the number of times it was called in other DAG models depending on the number of tasks considered per deadline. Finally, the execution time measured with the Layer DAG model for the 20 tasks configuration is very similar to the value measured for the H.264 video decoder example with six cores and four deadlines (recall that there are two frames per deadline, i.e., 20 tasks, in the H.264 case).

D. Prediction Error Resiliency

In this section, we run our algorithm with different workload estimator accuracies. We use then a random estimator that estimates a workload value in $[ActualExecutionTime \times (1 - x), ActualExecutionTime \times (1 + x)]$ with $x = \{0, 0.1, 0.2, \ldots, 0.9\}$. Each execution uses a different x. We do not compare our solution to MLTF-DPM because it cannot schedule the generated DAGs [as shown in Fig. 10(c)] even with a good workload estimator.

Fig. 14 shows the energy consumption (compared to the energy consumed with $x = 0$) and the deadline miss rates of each execution with respect to all the x values for the H.264 video benchmark with four cores and four deadlines per working set. Compared to the energy consumed with $x = 0$ (i.e., known workload values), our scheduler is able to decode the video sequences with less than 1.5% deadline miss rates and consuming no more than 6% of energy for $x \leq 0.5$. Moreover, for the Football video sequence, the execution with $x = 0.9$ consumes less energy than the case with $x = 0.2$. In fact, the scheduler decided to drop five frames (out of 203 frames) due to wrong workload predictions (so there is less scheduled workload overall). The scheduler is then able to efficiently adapt to prediction errors.

We have also applied the same experiment to the synthetic DAG model on a six-core platform. However, due to the limited space we do not show the results in a figure. We run the Layer DAG model on a congested system where the deadlines are set 10% less than their original values. For $0 \leq x \leq 0.3$ the DAG is scheduled with less than 5% more energy and no more than 2% deadline miss rates. For $0.3 \leq x \leq 0.6$ the DAG is scheduled with less than 11% more energy and no more than 5% deadline miss rates. Overall, the obtained results for the Layer model are very similar to the ones obtained for the *Coastguard* video sequence in Fig. 14. Finally when applying the same experiments on Erdos and TGFF DAG models, we notice that varying x (i.e., the estimator accuracy parameter) has almost no effect on the scheduler efficiency as these DAGs were scheduled with less than 3% more energy and no more than 1% of deadline miss rates. In fact, a lot of relative big gaps are generated in the schedule of these DAG models, and by scheduling tasks from future deadlines in advance, our gap-filling module allows the scheduler to be resilient to prediction error even for high x values.

The decision of the earliest deadline scheduler uses the critical path workload. Thus, if in one T_i, half of the tasks are for instance estimated with an average of 30% more workload and the other half with an average of 25% less workload. The prediction error for the scheduler decision will be only around 5%. However, the gap-filling module decision is based on the size of the gap, the task workload (the one to be scheduled) and the critical path workload. Therefore, the gap-filling algorithm is less resilient to prediction error. This explains also why in the Layer DAG model (and also the H.264 decoder case), where the system is much more congested than other DAG models, the prediction error on the gap size and the task workloads had a larger effect on the scheduler efficiency.

TABLE I
ENERGY-EFFICIENT SCHEDULER FEATURES: COMPARISON OF EXISTING APPROACHES TO OUR SOLUTION

energy-efficient schedulers	profiling data required	self-adaptive	tasks dropping feature
[10], [11], [20], [21]	yes	no	no
[5]	yes	yes (limited)	no
MLTF [12]	no	yes (limited)	no
MDP [8]	yes	yes (limited)	only H.264
Our solution	**no**	**yes**	**yes**

V. RELATED WORK

In Table I, we summarize different features considered by state-of-the-art energy-efficient schedulers and we compare them to our proposed solution. Moreover, in our previous paper

[21], we discussed in detail the drawbacks and shortcomings of DAG monitoring solutions and their application models.

In offline solutions, static schedules are generated at design time and rely on worst-case execution time estimates to compute the DVFS and DPM actions used to scale the voltage and frequency and to switch on/off cores, respectively [10], [11], [18], [19]. Synchronous dataflow (SDF) and cyclo-static dataflow (CSDF) are also known to be powerful modeling tools for static compile-time scheduling onto single and multicore processors [27], [28]. scheduling solutions based on data flow models are designed for hard real-time tasks. For this reason, these solutions must guarantee that all tasks can be completed before their deadlines under their worst-case execution times. In contrast, the proposed energy-efficient scheduler is designed for soft real-time tasks, where deadline misses are tolerable. For the application model that we consider, these static scheduling approaches are efficient if all the tasks start and finish as planned (see if the workload and the starting time of each task is fixed and known). However, they are unsuitable for multimedia applications with dynamic workload. In fact, modeling a nondeterministic workload with periodic tasks and worst-case execution time leads to significant slack time and inefficient resource utilization.

Semi-online schedulers [5], [8] have also been proposed where the scheduling policy is computed offline and the scheduling decision for the core assignment and the DVFS selection are made online based on the current execution status and an offline computed policy. In [5] and [8], both algorithms construct a scheduling table at design time. In [5], in the online phase, the lookup table provides multiple scheduling options for each task depending on the execution status, and a dynamic slack reclamation is performed as well. However, this approach does not consider the DAG model with dependent deadlines as shown in the DAG model 4 of Fig. 1 which limits its applicability. Recently, a new scheduling approach [8], that takes into account the DAG model presented in Fig. 3(a), has been proposed. However, this solution is limited to work only with the H.264 video decoder application and it is not applicable to the general DAG model. In [8], a look up table is built at design-time based on a MDP. The resulting scheduling policy offers the possibility of dropping appropriate tasks during execution to achieve lower deadline miss rates. The MDP formulation was applied for slice-parallel video decoders and solved offline using a two-level approach to reduce the complexity of the algorithm. The scheduling policy, computed offline and based on profiling data for each new video stream, is applicable only for the H.264 DAG model at the slice level without considering other type of tasks (i.e., initialization and deblocking filter). Even though the MDP-based optimization proposed in [8] showed promising results, it could not be used with arbitrary DAG dependencies (i.e., other than the H.264 DAG model).

Semi-online solutions [5], [8] are unable to adapt at runtime to different application dynamics because they require profiling information prepared at design-time for each targeted application. To address the limitations of semi-online solutions, an online solution for energy-efficient task scheduling on multicore platforms [12], that we have already presented in Section III-B, has been proposed. Even though the proposed scheduler does not require any profiling data, this solution is restricted to schedule tasks from one deadline at a time limiting then its ability to adapt to application characteristics. Moreover, this scheduler is not able to generate a balanced workload distribution over the cores resulting in several wasted gaps.

VI. Conclusion

In this paper, we have proposed a novel energy-efficient online scheduler for general DAG models for multicore DVFS- and DPM-enabled platforms. The key contributions of our approach are as follows.

1) Our solution is a low-complexity online technique that is fully independent from the considered DAG model.
2) Our scheduler does not impose any restrictions on the DAG and it covers online all DAG models.
3) Our scheduler is fully self-adaptive to the characteristics of each application and it does not require any offline profiling data.
4) Our scheduler is able to efficiently handle execution with very limited resources by detecting online the appropriate tasks to drop in order to reduce the deadline miss rates.
5) Our scheduler is resilient to workload prediction error. Our results for the H.264 video decoder have demonstrated that our proposed low-complexity solution for the general DAG model reduces the energy consumption by up to 15% with a lower deadline miss rates compared to a sophisticated state-of-the-art scheduler [8] that was specifically built for H.264 video decoding.

Moreover, our results with different configurations of synthetic DAGs have demonstrated that our proposed solution is able to reduce the energy reduction by up to 55% and the deadline miss rates by up to 99% compared to an existing online scheduling solution [12]. We have also shown how our solution efficiently adapts with respect to the DAG type, and scales well with the number of cores and the number of deadlines considered in the buffer. The low complexity of our proposed solution has been validated with a real execution of the full algorithm on an Apple A6 SoC [25]. Finally, we showed that our solution is resilient to workload prediction errors.

References

[1] *H.264/14496-10 AVC Reference Software Manual (revised for JM 17.1)*.
[2] G. Welch *et al.*, "An introduction to the Kalman filter," in *Proc SIGGRAPH*, 2001.
[3] S.-Y. Bang, K. Bang, S. Yoon, and E.-Y. Chung, "Run-time adaptive workload estimation for dynamic voltage scaling," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1334–1347, Sep. 2009.
[4] 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.
[5] J. Cong and K. Gururaj, "Energy efficient multiprocessor task scheduling under input-dependent variation," in *Proc. DATE*, Nice, France, 2009.
[6] D. Cordeiro *et al.*, "Random graph generation for scheduling simulations," in *Proc. SIMUTools*, 2010.
[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.
[8] 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. Multimedia*, vol. 15, no. 2, pp. 268–278, Feb. 2013.

[9] O. Sinnen, *Task Scheduling for Parallel Systems*. Hoboken, NJ, USA: Wiley, 2007.
[10] Y. Wang *et al*., "Overhead-aware energy optimization for real-time streaming applications on multiprocessor system-on-chip," *ACM Trans. Design Autom. Electron. Syst.*, vol. 16, no. 2, pp. 14:1–14:32, Apr. 2011.
[11] Y. Wang, D. Liu, M. Wang, Z. Qin, and Z. Shao, "Optimal task scheduling by removing inter-core communication overhead for streaming applications on MPSoC," in *Proc. RTAS*, Stockholm, Sweden, 2010.
[12] 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. ACM SAC*, New York, NY, USA, 2010.
[13] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[14] R. Ducasse, D. S. Turaga, and M. van der Schaar, "Adaptive topologic optimization for large-scale stream mining," *IEEE J. Sel. Topics Signal Process.*, vol. 4, no. 3, pp. 620–636, Jun. 2010.
[15] M. Qamhieh, S. Midonnet, and L. George, "A parallelizing algorithm for real-time tasks of directed acyclic graphs model," in *Proc. RTAS*, Apr. 2012.
[16] *OpenMP* [Online]. Available: http://openmp.org
[17] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.
[18] M. Ruggiero, D. Bertozzi, L. Benini, M. Milano, and A. Andrei, "Reducing the abstraction and optimality gaps in the allocation and scheduling for variable voltage/frequency MPSoC platforms," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 28, no. 3, pp. 378–391, Mar. 2009.
[19] J. Luo and N. K. Jha, "Power-efficient scheduling for heterogeneous distributed real-time embedded systems," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 26, no. 6, pp. 1161–1170, Jun. 2007.
[20] Y. Andreopoulos and M. van der Schaar, "Adaptive linear prediction for resource estimation of video decoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 6, pp. 751–764, Jun. 2007.
[21] K. Kanoun, D. Atienza, N. Mastronarde, and M. van der Schaar, "A unified online directed acyclic graph flow manager for multicore schedulers," in *Proc. ASP-DAC*, Singapore, 2014.
[22] *Samsung Exynos 5 Octa* [Online]. Available: http://www.samsung.com/exynos/
[23] *Nvidia Tegra 4* [Online]. Available: http://www.nvidia.com/object/tegra-4-processor.html
[24] *Snapdragon 800* [Online]. Available: http://www.qualcomm.com/snapdragon/processors
[25] *Apple A6* [Online]. Available: http://www.apple.com
[26] S. Idgunji, "Case study of a low power MTCMOS based ARM926 SoC: Design, analysis and test challenges," in *Proc. ITC*, Santa Clara, CA, USA, 2007.
[27] A. K. Singh, A. Das, and A. Kumar, "Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems," in *Proc. DAC*, Austin, TX, USA, 2013.
[28] J. Zhu, I. Sander, and A. Jantsch, "Energy efficient streaming applications with guaranteed throughput on MPSoCs," in *Proc. EMSOFT*, Atlanta, GA, USA, 2008.
[29] F. Kong, W. Yi, and Q. Deng, "Energy-efficient scheduling of real-time tasks on cluster-based multicores," in *Proc. DATE*, Grenoble, France, 2011.

Karim Kanoun received the M.Sc. degree in computer science from the École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble School of Engineering in Informatics and Applied Mathematics, Grenoble, France. He is currently pursuing the Ph.D. degree in embedded systems with École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

His current research interests include machine learning and energy efficient schedulers for streaming multimedia applications on mobile multicore platforms and large-scale systems.

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

He is an Assistant Professor with the Department of Electrical Engineering at the State University of New York at Buffalo, Buffalo, NY, USA. For more information, please visit his Research Laboratory's website: http://www.eng.buffalo.edu/~nmastron/.

David Atienza (M'05–SM'13) received the M.Sc. and Ph.D. degrees in computer science and engineering from UCM, Madrid, Spain, and IMEC, Leuven, Belgium, in 2001 and 2005, respectively.

He is an Associate Professor of Electrical Engineering and Director of the Embedded Systems Laboratory at École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. His current research interests include system-level design methodologies for high-performance multi-processor system-on-chip (MPSoC) and low-power embedded systems, including new 2-D/3-D thermal-aware design for MPSoCs, ultra-low power system architectures for wireless body sensor nodes, HW/SW reconfigurable systems, dynamic memory optimizations, and network-on-chip design. He has co-authored over 200 publications in peer-reviewed international journals and conferences, several book chapters, and eight U.S. patents in these fields.

Dr. Atienza has earned several best paper awards and he is an Associate Editor of IEEE TRANSACTIONS ON COMPUTERS, IEEE DESIGN AND TEST, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and *Integration* Elsevier. He received the IEEE CEDA Early Career Award in 2013, the ACM SIGDA Outstanding New Faculty Award in 2012 and a Faculty Award from Sun Labs at Oracle in 2011. He is a Distinguished Lecturer of the IEEE CASS during 2014–2015, and a Senior Member of ACM.

Mihaela van der Schaar (F'10) received the Ph.D. degree from Eindhoven University of Technology, Eindhoven, The Netherlands, in 2001.

She is a Chancellor's Professor of Electrical Engineering at the University of California, Los Angeles (UCLA), Los Angeles, CA, USA. Her current research interests include engineering economics and game theory, multiagent learning, online learning, decision theory, network science, multiuser networking, big data and real-time stream mining, and multimedia. She was a Distinguished Lecturer of the Communications Society during 2011–2012, the Editor-in-Chief of IEEE TRANSACTIONS ON MULTIMEDIA from 2011 to 2013. She holds 33 granted U.S. patents. She is also the Founding and Leading director of the UCLA Center for Engineering Economics, Learning, and Networks (see netecon.ee.ucla.edu).

Prof. van der Schaar was an Editorial Board Member of the IEEE JOURNAL ON SELECTED TOPICS IN SIGNAL PROCESSING in 2011. She received an NSF CAREER Award in 2004, the Best Paper Award from IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2005, the Okawa Foundation Award in 2006, the IBM Faculty Award in 2005, 2007, 2008, the Most Cited Paper Award from *EURASIP: Image Communications Journal* in 2006, the Gamenets Conference Best Paper Award in 2011 and the 2011 IEEE Circuits and Systems Society Darlington Best Paper Award. She received three ISO Awards for her contributions to the MPEG video compression and streaming international standardization activities.