# Energy-Efficient Real-Time Scheduling of DAGs on Clustered Multi-Core Platforms

Zhishan Guo[1]*, Ashikahmed Bhuiyan[1]*, Di Liu[2], Aamir Khan[3], Abusayeed Saifullah[4], Nan Guan[5]
[1]*Department of Electrical and Computer Engineering, University of Central Florida*
[2]*National Pilot School of Software, Yunnan University*
[3]*Hardware Department, BrainCo*
[4]*Department of Computer Science, Wayne State University*
[5]*Department of Computing, Hong Kong Polytechnic University*

***Abstract*—With the growth of computation-intensive real-time applications on multi-core embedded systems, energy-efficient real-time scheduling becomes crucial. Multi-core processors enable intra-task parallelism, and there has been much progress on exploiting that, while there has been only a little progress on energy-efficient multi-core real-time scheduling as yet. In this work, we study energy-efficient real-time scheduling of constrained deadline sporadic parallel tasks, where each task is represented as a directed acyclic graph (DAG). We consider a clustered multi-core platform where processors within the same cluster run at the same speed at any given time. A new concept named *speed-profile* is proposed to model per-task and per-cluster energy-consumption variations during run-time to minimize the expected long-term energy consumption. To our knowledge, no existing work considers energy-aware real-time scheduling of DAG tasks with *constrained deadlines*, nor on a *clustered multi-core platform*. The proposed energy-aware real-time scheduler is implemented upon an *ODROID XU-3* board to evaluate and demonstrate its feasibility and practicality. To complement our system experiments in large-scale, we have also conducted simulations that demonstrate a CPU energy saving of up to 57% through our proposed approach compared to existing methods.**

***Index Terms*—Parallel task, Real-time scheduling, Energy minimization, Cluster-based platform.**

*The authors made equal contribution.

## I. Introduction

There is a trend in moving towards multi/many-core processors for real-time systems—they appear as an enabling platform for embedded systems applications that require real-time guarantees, energy efficiency, and high performance. To exploit the capability of the multi-core platform, considering intra-task parallelism, where an individual task can utilize multiple cores simultaneously, is vital. Intra-task parallelism facilitates a balanced distribution of the tasks among the processors, which leads to energy efficiency [1]. One of the most generalized workload model for representing deterministic intra-task parallelism is Directed Acyclic Graph (DAG) task [2]. Under such method of abstraction, the nodes in a DAG represent various threads of execution while the edges represent their dependencies. In the past five years or so, quite some effort has been spent on developing real-time scheduling strategies and schedulability analysis of DAG tasks [2]–[9].

There are several real-world application that uses the DAG model. For example, the work in [4] studies problems related to scheduling parallel real-time tasks, modeled as DAG, on multiprocessor architectures. A low-complexity compile-time algorithm for scheduling tasks (represented as DAG) within homogeneous computing environments is proposed in [10]. A sporadic task model seems to apply to a system that manipulates live multimedia data, e.g., video conferencing systems [11]. Another example would be systems that control asynchronous devices, such as the local-area network adapters that implement real-time communication protocols.

Since many of those applications are battery-powered, considering energy-efficient approaches for designing such a platform is crucial. Thanks to the fact that modern generation processors support dynamic voltage and frequency scaling (DVFS), where each processor can adjust the voltage and frequency at runtime to minimize power consumption, per-core energy minimization becomes possible during run-time. Despite the hardness of the problem [12], a significant amount of work has considered power minimization for non-parallel tasks on a multi-core platform (refer to [13] for a survey). Regarding parallel tasks, Guo et al. studied energy-efficient real-time scheduling for DAG tasks as an early research effort [14]. They adopted the federated scheduling and task decomposition framework [2] for minimizing system energy consumption via per-core speed modulation. As the only step (that we are aware of) towards energy-aware scheduling of real-time DAG tasks, they targeted an exciting problem and laid some of the foundations of this work. However, the attention of [14] is restricted to implicit deadline tasks with a system model of per-core DVFS.

Unfortunately, per-core DVFS becomes inefficient as it increases the hardware cost [15]. For balancing the energy efficiency and the hardware cost, there is an ongoing trend to group processors into islands, where processors in the same island execute at the same speed. For example, a big.LITTLE platform (e.g., ODROID XU-3 [16]) consists of high performance (but power-hungry) cores integrated into ‘big’ clusters and low-power cores into ‘LITTLE’ clusters. Such a platform executes several real-life applications with heavy computational demands (e.g., video streaming [17]) in an energy-efficient manner. Apart from the energy consumption issue, a multi-core platform enables task execution with high-performance demand and tight deadlines, essential for computation-intensive real-time systems, e.g., autonomous vehicles [18].

Despite the urgent need, to the best of our knowledge, little work has been done that considers both the intra-task parallelism and power consumption issues in clustered multi-core real-time systems, where cores form a group of frequency/voltage clusters. Such kind of system balances the energy efficiency and hardware cost compared to the traditional (with individual frequency scaling feature) multi-core models. The scheduling problem becomes highly challenging on such platforms because:

(i) The relationship between the execution time, frequency, and energy consumption is nonlinear, making it highly challenging to minimize energy consumption while guaranteeing real-time correctness.

(ii) Existing solution (e.g., [14]) relies on the assumption that each processor can freely adjust its speed. That solution performs poorly as the assumption is no longer valid under a more realistic platform model considered in this paper.

(iii) The execution speed of a cluster becomes unpredictable if it is shared by multiple tasks with sporadic release patterns.

**Contribution.** In this paper, we propose a novel technique for energy-efficient scheduling of constrained deadline parallel task in a cluster-based multi-core system. To the best of our knowledge, no work has investigated the energy-efficient scheduling of DAG tasks on such a cluster-based platform. It is also the first work that has addressed the power awareness issue considering constrained deadline DAG tasks. Specifically, we make the following contributions:

• We consider a more practical *cluster-based system model* where the cores must execute at the same speed at any time instant within each cluster.

• To better handle constrained deadlines, one need to capture the gaps between deadlines and upcoming releases, as well as handling sporadic releases. We propose a novel concept of *speed-profile* to present the energy-consumption behavior for each DAG task as well as each cluster, such that they could guide task partition/clustering in an energy-efficient manner. An efficient *greedy* algorithm is proposed to partition DAG tasks according to the constructed speed-profiles.

• To evaluate the effectiveness of our proposed technique, we implement it on the ODROID XU-3 board, a representative multi-core platform for embedded systems [16]. The experiments report that our approach can save energy consumption by 18% compared to a reference approach. For larger-scale evaluation, we perform simulations using synthetic workloads and compare our technique with two existing baselines [14], [19]. The simulation results demonstrate that our method can reduce energy consumption by up to 58% compared to the existing ones under the cluster-based platform setting.

**Organization.** The rest of the paper is organized as follows. Section II presents the workload, power, and platform models, problem statement, and the essential background. In Section III, we describe the importance of creating a speed-profile for an individual task and the whole cluster. Section IV discusses the approaches to create the speed-profile for each task and propose a greedy algorithm to allocate multiple tasks in the same cluster. Experimental and simulation results are presented in Section V and VI. Section VII discusses the assumptions and applicability of our model and evaluation setup. Section VIII discusses related work including a detailed comparison with our existing work [14], and Section IX concludes this paper.

## II. System Model, Problem Statement and Background

### A. System Model and Problem Statement

**Workload model.** We consider a set of sporadic parallel task denoted by $\tau = \{\tau_1, \cdots, \tau_n\}$, where each $\tau_i \in \tau$ $(1 \leq i \leq n)$ is represented as a DAG with a minimum inter-arrival separation (i.e., period) of $T_i$ time units, and a relative deadline of $D_i (\leq T_i)$ time units. An implicit deadline task has the same relative deadline and period, i.e., $D_i = T_i$. As a DAG task, the execution part of task $\tau_i$ contains a total of $N_i$ nodes, each denoted by $\mathcal{N}_i^j (1 \leq j \leq N_i)$. A directed edge from $\mathcal{N}_i^j$ to $\mathcal{N}_i^k (\mathcal{N}_i^j \rightarrow \mathcal{N}_i^k)$ implies that execution of $\mathcal{N}_i^k$ can start if $\mathcal{N}_i^j$ finishes for every instance (precedence constraints). In this case, $\mathcal{N}_i^j$ is called a parent of $\mathcal{N}_i^k$ ($\mathcal{N}_i^k$ is a child of $\mathcal{N}_i^j$). A node may have multiple parents or children. The *degree of parallelism*, $\mathcal{M}_i$, of $\tau_i$ is the number of nodes that can be simultaneously executed. $c_i^j$ denotes the execution requirement of node $\mathcal{N}_i^j$. $C_i := \sum_{j=1}^{N_i} c_i^j$ denotes the worst case execution requirement (WCET) of $\tau_i$.

A *critical path* is a directed path with the maximum total execution requirements among all other paths in a DAG. $L_i$ is the sum of the execution requirements of all the nodes that lie on a critical path. It is the minimum make-span of $\tau_i$, i.e., in order to make $\tau_i$ schedulable, at least $L_i$ time units are required even when number of cores is unlimited. Since at least $L_i$ time units are required for $\tau_i$, the condition $T_i \geq L_i$ (implicit deadline tasks) and $D_i \geq L_i$ (constrained deadline tasks) must hold for $\tau_i$ to be schedulable. A schedule is said to be feasible if upon satisfying the precedence constraints, all the sub-tasks (nodes) receive enough execution from their arrival times, i.e., $C_i$ within $T_i$ (implicit deadline) or $D_i$ (constrained deadline) time units. These terms are illustrated in Figure 2(a).

**Platform Model.** We consider a clustered multi-core platform, where processors within the same cluster run at the same speed (frequency and supply voltage) at any given time. Such additional restriction comparing to traditional multi-core platform makes the model more realistic in many senarios. For example, our experiment is conducted on the ODROID XU-3 platform with one 'LITTLE' cluster of four energy-efficient ARM Cortex-A7 and one 'big' cluster of four performance-efficient ARM Cortex-A15. Note that we do not restrict the hardware-dependent energy parameters (e.g., $\alpha, \beta$ and $\gamma$ in the power model discussed below) to be identical across different clusters—these parameters can be derived using any curve-fitting method, e.g., [20].

**Energy Model.** Assuming frequency (speed) of a processor at a specific instant $t$ is $s(t)$ (in short, denoted as $s$), then its power consumption $P(s)$ can be calculated as:

$$P(s) = P_s + P_d(s) = \beta + \alpha s^{\gamma}, \tag{1}$$

Here, $P_s$ and $P_d(s)$ respectively denote the static and dynamic power consumption. Whenever a processor remains on, it introduces $P_s$ in the system (due to leakage current). Switching activities introduce $P_d(s)$ which is frequency dependent and represented as $\alpha s^{\gamma}$. Here, the $\alpha > 0$ depends on the effective switching capacitance [21]; $\gamma \in [2, 3]$ is a fixed parameter determined by the hardware; $\beta > 0$ represents the static part of power consumption. From this model, the energy consumption over any given period $[b, f]$ is calculated as $E(b, f) = \int_b^f P(s(t))\, dt$.

Our motivation behind selecting this power model comes from the fact that it complies with many existing works in the community, few to mention [12], [14], [21]–[25]. Beside this, recently this model was shown to be highly realistic by showing its similarity with actual power consumption [22]. Figure 1 shows comparison between the original power consumption results from [26] and the power model in Equation (1).

**Assumptions.** In this paper, we make the following assumptions: (i) frequency scaling is continuous on the theory side, (ii) task-level energy heterogeneity is not considered, (iii) we focus on CPU power consumption, and (iv) *Dynamic power management (DPM)* is not considered. For those that are interested, Section VII provides detailed discussions behind these assumptions, their impacts, and some hints to overcome the drawbacks.

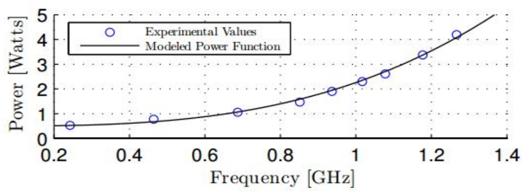

Fig. 1. Comparison of the power model (Equation (1)) with experimental results in [26]. Here, $\alpha = 1.76 Watts/GHz^3$, $\gamma = 3$, and $\beta = 0.5$ Watts. This figure is adopted from [21].

**Problem Statement.** Considering a set of constrained deadline sporadic DAG tasks (to be scheduled) on a clustered multi-core platform, we focus on finding a correct scheduling strategy such that all tasks receive enough execution by their deadlines, while the overall system power consumption is minimized.

### B. *Background and Existing Concepts*

In this section, we describe some existing concepts and techniques for handling real-time parallel task scheduling, and that constitute an initial step for our proposed work.

**Task Decomposition.** The well-known task decomposition technique [2] transforms a parallel task $\tau_i$ into a set of sequential tasks as demonstrated in Figure 2(b). Upon task decomposition, each node $\mathcal{N}_i^l \in \tau_i$ is converted into an individual sub-task with its scheduling window (defined by its own release time and deadline) and execution requirement ($c_i^l$). The allocation of release time and deadline respect all the dependencies (represented by edges in the DAG). Considering that a task is allowed to execute on an unlimited number of cores, starting from the beginning, a vertical line is drawn at every time instant where a node $\mathcal{N}_i^l$ starts or ends. So the DAG is partitioned into several segments which may contain single/multiple thread(s). Threads assigned to the same segment share equal amount of execution length; e.g., $\mathcal{N}_i^3$, $\mathcal{N}_i^4$, and $\mathcal{N}_i^5$ all have 2-time units assigned to the $3^{rd}$ segment, as demonstrated in Figure 2(b).

**Segment Extension.** The deadline for each node via task decomposition may be unnecessarily restrictive, e.g., the decomposition of the DAG in Figure 2(a) will restrict $\mathcal{N}_i^3$ within the $2^{nd}$ and $3^{rd}$ segment. To systematically eliminate such unnecessary restriction and allow $\mathcal{N}_i^3$ to execute in the $4^{th}$ segment, segment extension should be applied, e.g., the green rectangle for node $\mathcal{N}_i^3$ in the $4^{th}$ segment in Figure 2(b).

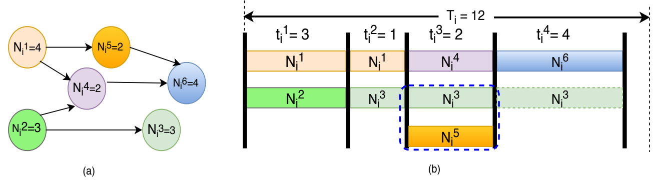

Fig. 2. (a) A DAG, $\tau_i$ (b) transformed DAG $\tau_i$ after applying task decomposition. Both of them are adopted from [14].

**Intra-Task Processor Merging.** After applying task decomposition and segment extension upon a DAG task $\tau_i$, some of these cores (where $\tau_i$ is allocated) can be very lightly loaded. Those core cause massive leakage power consumption in the long run and should be avoided when necessary. Intra-task merging [14] seeks to merge those cores to gain overall energy efficiency by reducing the total number of active cores. For example, in Figure 2(b), the third core (executing $\mathcal{N}_i^5$) is lightly loaded, and thus it is better to merge all the execution into the second core and shut it off completely. Such a reduction on the number of active cores minimizes leakage power consumption (see Equation (1) and Figure 2 in [14]) as well as the total number of clusters.

## III. Speed-Profile for Task and Cluster

This section discusses how different tasks share a cluster where all processors in a cluster execute at the same speed. When multiple tasks share a cluster, they may not align well due to sporadic releases and different periods. In a cluster-based platform, the processor having the maximum speed dominates the others in the same cluster. Hence, existing energy-saving techniques may perform poorly in a cluster-based platform. To tackle this problem, we propose a new concept called *speed-profile*. Subsection III-A highlights the motivation of bringing this new concept and provides its definition. Subsection III-B describes how speed-profiles are handled when two tasks are partitioned into the same cluster.

### A. *Speed-Profile for Each DAG*

Interesting energy-saving techniques (e.g., segment extension) have been proposed in [14] for the implicit deadline

DAG tasks. For the constrained deadline tasks, this technique becomes incompetent because of the non-negligible idle gaps between the task deadline and its next release. For example, consider the task $\tau_i$ in Figure 2(b) with $D_i = 10$ and $T_i = 12$. Segment extension can stretch $\mathcal{N}_i^3$ to the end of the $4^{th}$ segment but cannot utilize the idle time of 2 units. Besides, the sub-optimal solution provided in [14] becomes *non-convex* (see Lemma 1) in a cluster-based platform.

**Lemma 1.** *In a cluster-based platform, the convex optimization problem constructed in [14] becomes non-convex.*

*Proof.* The following set of constraints ensure the real-time correctness for each node $\mathcal{N}_i^l \in \tau_i$, i.e., $\mathcal{N}_i^l$ receives enough time to finish execution within its scheduling window.

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{j=b_i^l}^{d_i^l} t_j^c s_{i,j} \geq c_i^{\mathcal{N}_i^l}. \tag{2}$$

We introduce the following inequalities to bound the total length for all segments in task $\tau_i$:

$$\sum_{j=1}^{Z} t_j^c \leq T_i. \tag{3}$$

Any value of execution speed and segment length ensures real-time correctness if Equation (2) and (3) are respected. However, unlike [14], the execution speed of a node is not *constant* within its scheduling windows. It depends on the execution speed of other nodes (of other tasks) in the same cluster. As a result, we cannot substitute the variable $s_{i,j}$ as a function of nodes execution requirement, resulting in quadratic inequality constraints (Equation (2)). This makes the optimization problem *non-convex*. ∎

Due to the $\boxed{\text{maximum-dominating}}$ nature in a clustered platform, at each instant, all cores in a cluster must execute at the speed of the fastest one. If these tasks are not well aligned (concerning their execution speed), the cluster as a whole will perform poorly regarding energy efficiency. Assigning tasks with similar speed shape on the same cluster may not be an energy efficient option (due to their sporadic releases pattern). Figure 3 and Example 1 demonstrates one such scenario.

**Example 1.** *In this example, we describe how the sporadic arrival pattern of a task influences the energy efficiency of the whole cluster. Consider two tasks $\tau_1$ and $\tau_2$ with the predefined necessary speed of execution on two processors each, to be partitioned on to the same cluster (of four processors). Figure 3(a) shows the synchronous release case, where the whole cluster could run at 0 speed between [3,4) and [7,8). While Figure 3(b) shows the scenario when $\tau_1$'s initial release is delayed by one-time unit, where the whole cluster will need to run at a higher speed (of 0.8) most (75%) of the time and thus consumes more energy. In this example, from $\tau_2$'s perspective, direct energy reduction with existing per-task WCET based techniques may not help much, as it may be another task dominating the speed of the whole cluster most of the time. The critical observation is that, due to the extra restriction of the more realistic platform model, the speed of a cluster is determined by the heavier DAG running on it, as well as how synchronous are the releases, which could be entirely random. Moreover, even a task finishes its execution early (say, $\tau_2$ requires no execution over [5,7)), we may not be able to reduce the cluster speed at all.*

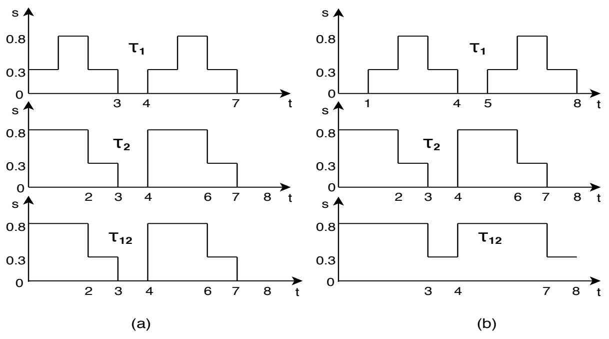

Fig. 3. When two tasks $\tau_1$ and $\tau_2$ with fixed speed patterns each are partitioned on to the same cluster, the resultant speed pattern ($\tau_{12}$) of the cluster may vary when they experience different release offsets. Note that in order to satisfy platform model restrictions while guaranteeing the correctness, the processors (of the same cluster) must run at the maximum/larger of the two individual speeds at each instant.

To address this issue, we propose a novel concept of speed-profile to capture the energy consumption behavior of all possible alignment scenarios.

**Definition 1.** ***(Speed-Profile)*** *The speed-profile of a task describes the percentage/likelihood of all possible speeds that the task may execute at over a period. It is a random variable $\mathcal{S}$ with an associated probability function (PF) $f_{\mathcal{S}}(s) = \mathbb{P}(\mathcal{S} = s)$, where $s$ is the finite set of possible speeds, and $f_{\mathcal{S}}(s)$ represents the portions of the time (likelihoods) when it is running at those speeds respectively.*

**Example 2.** *Let us consider a task $\tau_i$ with $T_i = 15$ executing at a speed of $0.6$ for $5$ time units (not necessarily to be continual), and at a speed of $0.5$ for the rest of the time. The speed-profile of the task is thus $\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.5 \\ 0.33 & 0.67 \end{pmatrix}$. At any specific time, $t$, there is about 33% probability that the cores are running at the speed of 0.6 unit and about 67% probability that the cores are running at the speed of 0.5 unit.*

It is evident that from another task's point of view, the speed-profile provides probabilistic information on how the task of interest would restrict the lower bound to the speed of the cluster over time. As the alignment of releases between any two tasks is unknown, we assume in the future analysis that any phase difference is of equal chance over the long run.

**Remark 1.** *The speed-profile $\mathcal{S}_i$ of a given task $\tau_i$ remains the same for an initial phase (release offset) $\phi_i > 0$.*

**Remark 2.** *A similar concept, pWCET-profile, was created in task modeling for probabilistic real-time scheduling, which is for an entirely different purpose—see [27] for more details.*

Subsection IV-A details the calculation for task speed-profile. Here, we describe the calculation of the cluster speed-profile when two tasks are combined on to the same cluster.

## B. Speed-Profile for the Cluster

As stated earlier, the property of the clustered platform and sporadic arrival pattern of a task makes the exact speed of the cluster unpredictable at a specific time instant (see Figure 3 and Example 1). As a result, when two tasks $\tau_i$ and $\tau_j$ (with speed-profiles) are being considered allocating to the same cluster, we need to construct the merged speed-profile of the cluster (executing them both). To perform such calculation, we introduce a special $\odot$ operator that takes the maximum of the two profiles on a probability basis[1].

**Definition 2.** *The special operator $\odot$ operates on two (or more) random variables $\mathcal{X}$ and $\mathcal{Y}$. During this operation, each entry $\mathcal{X}_i \in \mathcal{X}$ is compared with each entry $\mathcal{Y}_j \in \mathcal{Y}$ and the value $\mathcal{Z}_{ij}$ is calculated as $\mathcal{Z}_{ij} = max(\mathcal{X}_i, \mathcal{Y}_j)$, with a combined (multiplied) probability. If there are multiple occurrences of an entry, all of them are merged into a single entry, and their associated probability are summed together.*

**Example 3.** *Let $\mathcal{S}_i = \begin{pmatrix} 6 & 5 \\ 0.4 & 0.6 \end{pmatrix}$, $\mathcal{S}_j = \begin{pmatrix} 6 & 2 \\ 0.4 & 0.6 \end{pmatrix}$. Then* $\mathcal{S}_i \odot \mathcal{S}_j = \begin{pmatrix} 6 & 6 & 6 & 5 \\ 0.16 & 0.24 & 0.24 & 0.36 \end{pmatrix} = \begin{pmatrix} 6 & 5 \\ 0.64 & 0.36 \end{pmatrix}$.

Note that we allocate two different DAGs (with same/different periods) to the same cluster. The speed-profile indicates how long a DAG executes at different speeds within its deadline, i.e., the probability that a DAG executes at a specific speed. The task's period becomes irrelevant as speed-profile is a probability-based measurement. Once $\tau_i$ and $\tau_j$ are allocated to the same cluster, we use $\mathcal{S}_{ij}$ to denote the speed-profile of the cluster (see Example 3 for the details).

In summary, energy minimization in a cluster-based platform is challenging because of sporadic release pattern and the idle gaps between a task deadline and its period (constrained deadline task). To tackle these problems, we have introduced the concept of speed-profile for both an individual task and a cluster where multiple tasks can be allocated.

## IV. Task Partitioning Algorithm

The ultimate goal of the paper is to partition all DAGs into clusters, such that overall platform energy consumption is minimized. Recall that on a clustered multiprocessor platform, at a given instant, all processors in the same cluster must execute at the same speed. Due to this property of a cluster-based platform, if two tasks that are not well-aligned (in terms of execution speed) are allocated to the same cluster, it will result in reduced energy efficiency. So, we have proposed the concept of speed-profiles (refer to Section III) which is a tool to measure the potential long-term energy saving of a cluster when partitioning any pair of DAGs into this cluster. So far we have discussed the importance of the concept of speed-profile but did not mention how to create them given a DAG task, which is the focus on Subsection IV-A. Then, Subsection IV-B describes the task-to-cluster partitioning algorithm.

[1]Note that although the appearance of the proposed operator is identical to the one in [27], the calculation is quite different. This is due to the "larger value dominating" nature of the platform model considered in this paper.

## A. Creating the Speed-Profile of a Task

Given a DAG task $\tau_i$, we provide two approaches to create the speed-profile $\mathcal{S}_i$.

**Approach A: Considering the Maximum Speed from all the Cores.** Upon applying the task decomposition, segment extension, and intra-task processor merging techniques (Section II), some vital information (e.g., the speed of a core at a specific time and number of cores required) becomes available. This information plays a role to calculating the speed-profile $S_i$ of task $\tau_i$. At any time instant $t$, we consider the maximum speed from all the cores available. It ensures the sufficient speed so that even the heaviest node can finish execution within its scheduling window (defined after task decomposition). We consider constrained deadline (i.e., $D_i \leq T_i$), so the task must have to finish by $D_i$ and rest of the time $(T_i - D_i)$ it remains idle. For each segment $j \in \tau_i$, (summation of the length of these segments is equal to $D_i$), we create a pair $(s_{i,j}, p_{i,j})$. For the $j^{th}$ segment, $s_{i,j}$ and $p_{i,j}$ respectively denote the maximum execution speed and the probability that the cluster will run at this speed. Let, $M$ cores are allocated to $\tau_i$. At $j^{th}$ segment, we calculate $s_{i,j}$ and $p_{i,j}$ as follows:

$$s_{i,j} = \max_{k \leq M}\{s_{i,j,k}\}, p_{i,j} = \frac{t_j^c}{T_i}.$$

Here, $s_{i,j,k}$ denotes the speed of $k^{th}$ core at $j^{th}$ segment and $t_j^c$ is the length of $j^{th}$ segment. The speed-profile $\mathcal{S}_i$ will be:

$$\mathcal{S}_i = \begin{pmatrix} s_{i,1} & s_{i,2} & \cdots & s_{i,z} & 0 \\ p_{i,1} & p_{i,2} & \cdots & p_{i,z} & (T_i - D_i)/T_i \end{pmatrix}.$$

The last pair reflects the fact that the core remains idle for the $(T_i - D_i)$ time units at the end of each period.

**Example 4.** *Consider a task $\tau_i$ with $T_i = 15$, $D_i = 12$ and $C_i = 6.5$. Now, it is executing at a speed of 0.6 for 5-time units (not necessarily to be continual), at a speed of 0.5 for 7-time units, and it remains idle for the rest of the 3 $(T_i - D_i)$ time units. The speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.5 & 0 \\ 0.33 & 0.47 & 0.2 \end{pmatrix}.$$

Note that, if a cluster contains a single task $\tau_i$, then $\mathcal{S}_i$ also represents the cluster speed-profile. If $\tau_i$ and $\tau_j$ (or more tasks) are executing on the same cluster, then the technique described in Subsection III-B needs to be applied before making any choices. The greedy choosing approach for task partition is detailed in Subsection IV-B.

**Approach B: A Single Speed Throughout.** Theorem 4 of [14] shares a valuable insight: *The total energy consumption (assuming processor remains on) is minimized in any scheduling window when execution speed remains uniform (the same) throughout the interval.* Motivated by it[2], we propose another

[2]Note that [14] considered that the speed remains constant within a scheduling slot for each processor. Also, they assumed per core speed scaling and calculated the speed within each scheduling slot through a convex optimization method. This paper considers the clustered platform where the objective function becomes *non-convex* (see Lemma 1) and thus the existing approach is inefficient.

approach of selecting a single speed for a DAG task (job) during the whole duration from its release until its deadline.

In this approach, we consider the maximum workload (or the execution requirement) from all the cores available and determine the aggregated workload. Upon dividing the aggregated workload by the deadline, we get the desired single speed. Let $M$ cores be allocated to task $\tau_i$. At $j^{th}$ segment, the execution requirement of the $k^{th}$ core is denoted by $w_{i,j,k}$, which is calculated as follows:

$$w_{i,j,k} = s_{i,j,k} \times t^c_j.$$

We determine the maximum execution requirement as follows:

$$w_{i,j} = \max_{k \le M}\{w_{i,j,k}\}.$$

Let $Z$ denotes the total number of segments in $\tau_i$. The maximum total workload $w_i$ and the desired single speed $s_i$ is calculated using the following equations:

$$w_i = \sum_{j=1}^{Z} w_{i,j}, \quad s_i = \frac{w_i}{D_i}. \tag{4}$$

Other than the idle pair $(0, (T_i - D_i)/T_i)$, we consider a single speed throughout the deadline so only a single pair $(s_i, p_i)$ is required, where $s_i = w_i/D_i$ and $p_i = D_i/T_i$.

**Example 5.** *Consider the task described in Example 4 ($T_i = 15$, $D_i = 12$ and $C_i = 6.5$). It must finish 6.5 unit of workloads within 12-time units. Using this approach its speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.54 & 0 \\ 0.8 & 0.2 \end{pmatrix}.$$

**An Efficient Approach for Implicit Deadline System.** By adopting simple modification in Equation (4), it is possible to apply the process mentioned above for the implicit deadline tasks also. The workload $w_i$ should be divided by the period instead of the deadline. We consider the same speed through the task period, so only a single pair $(s_i, p_i)$ is required, where $s_i = w_i/T_i$ and $p_i = 1$.

**Example 6.** *Now we create the speed-profile for the task described in Example 4 and Example 5. This time we consider it is an implicit deadline task. So it has $T_i = D_i = 15$ and $C_i = 6.5$. Let's assume that it is executed at a speed of 0.6 for 5-time units, at a speed of 0.35 for 10-time units. According to Approach A, the speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.35 \\ 0.33 & 0.67 \end{pmatrix},$$

*and according to Approach B, the speed-profile is:*

$$\mathcal{S}_i = \begin{pmatrix} 0.43 \\ 1 \end{pmatrix}.$$

## *B. Task Partition: Greedy Merging with Speed-Profiles*

We are now equipped with tools (speed-profiles) to measure the potential long-term energy saving of a cluster when partitioning any pair of DAG tasks into it. This subsection describes the scheme for selecting pair by pair so that the total number of clusters can be significantly smaller than the total number of tasks.

To select a pair that will share the same cluster, we greedily choose the pair that provides maximum power saving, as depicted in Algorithm 1. We allow only the pairing of two DAGs that are not part of any merging previously. Also, if any task uses more cores than it is available in a cluster, that task cannot be merged with that cluster.

**Algorithm 1** Greedy Merging

```
1:  Input: Task-set $\tau$, with speed-profile $\mathcal{S}_i$ for each task
2:  Output: Speed-profile $\tilde{\mathcal{S}}$ (with processor power saving).
3:  $\bar{\mathcal{S}}, \tilde{\mathcal{S}} \leftarrow \emptyset$        ▷ All the possible/selected speed-profiles
4:  for $i = 1$ to $n$ do
5:      for $j = i + 1$ to $n$ do
6:          $\mathcal{S}_{ij} \leftarrow \mathcal{S}_i \odot \mathcal{S}_j$; $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} \cup \mathcal{S}_{ij}$;
7:      end for
8:  end for
9:  while $\exists \mathcal{S}_{xy} \in \bar{\mathcal{S}}$ and $\mathcal{S}_{xy}$ provides non-zero power saving do
10:     $\mathcal{S}_{xy} \leftarrow$ the pair from $\bar{\mathcal{S}}$ with maximum power saving
11:     $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \mathcal{S}_{xy}$
12:     for $k = 1$ to $n$ do
13:         $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} - \mathcal{S}_{kx} - \mathcal{S}_{ky}$
14:     end for
15: end while
16: return $\tilde{\mathcal{S}}$.
```

The algorithm starts by creating two empty lists $\bar{\mathcal{S}}$ and $\tilde{\mathcal{S}}$ that will contain all the possible and selected speed-profiles (Line 3). Lines 4–8, calculates all the possible speed-profiles and insert them into $\bar{\mathcal{S}}$. We greedily select a pair of DAGs that provide the maximum power saving and update the list $\bar{\mathcal{S}}$ by removing the pair from any further merging (Lines 9–15). The list $\tilde{\mathcal{S}}$ is also updated by adding the selected pair (Line 11). We conclude by returning the updated list $\tilde{\mathcal{S}}$ (Line 16).

**Lemma 2.** *If a task $\tau_i$ executes according to the speed-profile $\mathcal{S}_i$, it guarantees real-time correctness.*

*Proof.* It has been observed in [14] that the following constraint guarantees the real-time correctness:

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{k=b_i^l}^{d_i^l} t_k^c S_k^{\mathcal{M}_i^l} \ge c_i^{\mathcal{N}_i^l}. \tag{5}$$

Here, $b_i^l$ and $d_i^l$ denotes the release time and deadline of $\mathcal{N}_i^l$, $\mathcal{M}_i^l$ denotes the node-to-processor mapping and $S_k^{\mathcal{M}_i^l}$ is the speed of the processor (where $\mathcal{N}_i^l$ is allocated) at $k^{th}$ segment. Unlike to [14], at any time instant $t$, we choose either the maximum speed from all the cores running on the same cluster (Approach A) or a single speed that can guarantee the maximum execution requirement for the whole duration up to $\tau_i$'s deadline (Approach B). So, at any time instant, the cluster speed is larger or equals to the speed of any individual core. Considering Equation (2) and (5) we can deduce that:

$$\forall l : \mathcal{N}_i^l \in \tau_i :: \sum_{k=b_i^l}^{d_i^l} t_k^c s_{i,k} \geq \sum_{k=b_i^l}^{d_i^l} t_k^c S_k^{\mathcal{M}_i^l} \geq c_i^{\mathcal{N}_i^l}.$$

So, we conclude that Executing a task with speed according to the speed-profile $\mathcal{S}_i$ guarantees real-time correctness. ■

**Theorem 1.** *Executing a task with a speed according to the cluster speed-profile guarantees real-time correctness.*

*Proof.* We have shown in Lemma 2 that a task $\tau_i$ will not miss the deadline if executed according to its speed-profile $\mathcal{S}_i$. If $\tau_i$ share a cluster with another task $\tau_j$ and executes according to the merged (i.e., cluster) speed-profile $\mathcal{S}_{ij}$, then it still guarantees the real-time correctness, because $\mathcal{S}_{ij} \geq \mathcal{S}_i$ holds at any time instant. ■

**Remark 3.** *For $n$ tasks, the time complexity to generate all possible speed-profiles, $\bar{\mathcal{S}}$, is $O(n^2 Z)$, where $Z$ is the maximum number of segments of all DAGs in the set after decomposition (related to the structure and size of the DAGs). Algorithm 1 greedily choose a speed-profile by iterating through $\mathcal{S}$ and then update, which takes $O(n^2)$ time as well. Thus the total complexity of the proposed method is $O(n^2)$.*

In summary, we have proposed two approaches (Subsection IV-A) to create the speed-profile for any constrained-deadline DAG task. We also show that if a task executes according to the speed-profile, it ensures real-time correctness. According to the techniques provided in Section III, we could evaluate and compare all potential pairs of the combination by calculating the cluster speed-profile after merging. Finally, Subsection IV-B discussed how to use these speed-profiles to find suitable partners to share a cluster greedily.

## V. System Experiments

In this section, we present experimental results conducted on an ODROID XU-3 board. The platform runs on Ubuntu 16.04 LTS with Linux kernel 3.10.105. It is fabricated with Samsung Exynos5422 Octa-core SoC, consisting of two quad-core clusters, one 'LITTLE' cluster with four energy-efficient ARM Cortex-A7 and one 'big' cluster with four performance-efficient ARM Cortex-A15. Four TI INA231 power sensors are integrated onto the board to provide real-time power monitoring for the A-7 and A-15 clusters, GPU, and DRAM. An energy monitoring script, `emoxu3` [28], is used to log energy consumption of the workloads.

**DAG Generation.** We use the widely used Erdos-Renyi [29] method to generate a DAG. In this experiment, we generate two task sets each with 300 DAGs. A parameter $p$ is used to denote the probability of having an edge between two nodes. Recall that, ODROID XU-3 board has two quad-core clusters and all these DAGs will be allocated on this two clusters. So we generate DAGs with an uncomplicated structure and set $p$ to 0.25. This method may generate a disconnected DAG. If it happens, we add the fewest number of edges to make it connected. For experimentation, we have considered arbitrary task

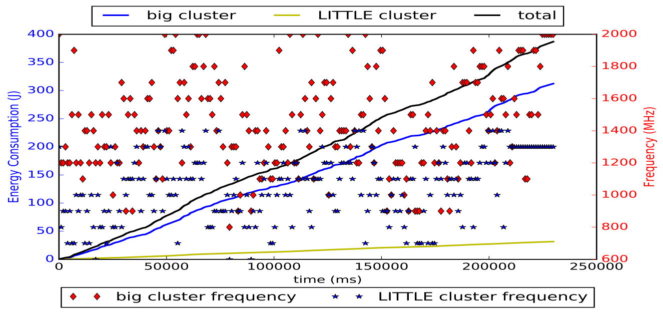

Fig. 4. The energy consumption and the frequency variation of our proposed approach on ODROID XU-3.

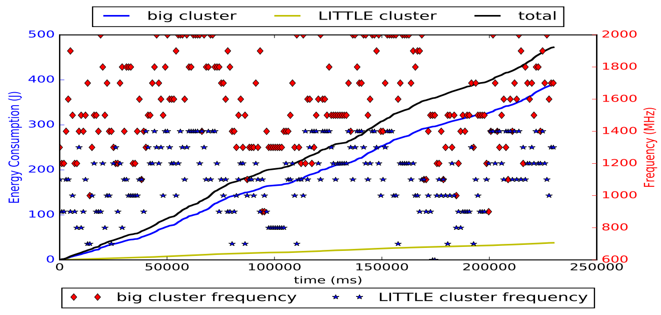

Fig. 5. The energy consumption and the frequency variation of the reference approach on ODROID XU-3.

periods, and it is determined using Gamma distribution [30]. We set the periods with $T_i = L_i + 2(C_i/m)(1 + \Gamma(2, 1)/4)$ [2]. Here, $L_i$ is the critical path length of $\tau_i$, calculated according to the definition of $L_i$ (in Section II).

After generating the topology of each DAG task of a set, we partition them into two subsets according to the proposed approach, one to the big cluster and the other one to the LITTLE cluster, and measure the energy consumption over the hyper-period of all DAGs. We use `rt-app` [31] to emulate the workload for each node. It is a test-application to simulate a real-time periodic load and utilizes the POSIX threads model to call and execute threads. For each thread, an execution time needs to be assigned. In this experiment, for each node, we randomly select an execution time ranged between $[300ms, 700ms]$. rt-app itself has a latency that varies randomly between $13 - 150ms$ per thread. Therefore, we add the maximum latency of rt-app, i.e., $150ms$, to the execution time of each thread from an analytical point of view.

**DAG Scheduling.** We use the Linux built-in real-time scheduler `sched_FIFO` to schedule the DAGs. Compared to the other system tasks, DAGs are assigned with higher priorities so that they can execute without interference. Our approach is also applicable to other preemptive schedulers which feature the work-conserving property.

**Frequency Scaling.** According to the frequency/speed-profile (Section IV), we use `cpufreq-set` program from `cpufrequtils` package to change the system's frequency online. We use the ODROID XU-3 board, where scaling-down and scaling-up the frequency of the big cluster takes at most $60ms$ and $40ms$, respectively. On the LITTLE cluster, both the operation takes at most $15ms$. Due to this delay, the hyper-

TABLE I
SUMMARY OF EXPERIMENTAL RESULTS.

| | Ours ($J$) | Ref ($J$) | Energy Saving (%) |
|---|---|---|---|
| big cluster | 312 | 389 | 20 |
| LITTLE cluster | 32 | 38 | 16 |
| Total | 387 | 472 | 18 |

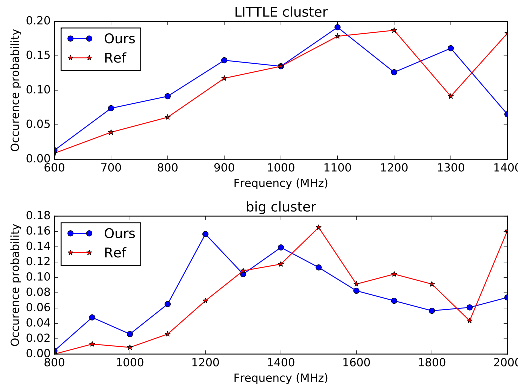

Fig. 6. Frequency occurrence probabilities.

period of all DAGs becomes large ($230s$, in this experiment). We detail the reasons behind this delay in Subsection VII-C.

**The Reference Approach**. Since no work has studied the same problem considered in this paper, we do not have a direct baseline for comparison. So, we propose a reference approach based on the studies for energy-efficient scheduling of sequential tasks [32]. They assigned an operational frequency to each task, and at run-time, schedule them according to their frequency. In this reference approach, we compute an operational frequency for each DAG. This frequency stretches out execution length of these DAGs as much as possible without violating their deadlines. As stated earlier, the reference approach executes the DAGs with the same partition, but without the merging techniques proposed in Section IV.

**Results.** The experimental results are plotted in Figure 4 and 5. In these figures, we show (i) the energy consumption over the hyper-period ($230s$), where the three lines show the energy consumption of the big and LITTLE cluster, and the total system; and (ii) frequency variation during the run-time, where the diamond and star marks denote the operational frequency of the big and the LITTLE cluster at a specific time instant, respectively. Note that the GPU and DRAM also contribute the energy consumption of the total system. Hence, the total energy consumption is a bit higher than the summation of the contribution of the big and the LITTLE cluster, but it is observed that there is a negligible difference for the energy consumption of GPU and DRAM between the two approaches. Besides, it is worth noticing that this energy consumption also accounts for energy consumption of the operating system.

Table I summarizes the comparison of the experimental results, where the energy consumption of the two clusters and the total system is presented, and the energy saving from our approach is given. As can be seen, our approach consumes $312J$ and $32J$ on the big and the LITTLE cluster, respectively. Comparing to the reference approach, we save energy consumption by $20\%$ and $16\%$. In total, our approach saves energy consumption by $18\%$.

The result can be justified as the reference approach changes the frequency for each DAG, while ours have a more fine-grained frequency adjustment at each segment (Subsection IV-A). For each DAG, during scheduling, our approach could scale down the frequency if required. Figure 6 presents the frequency occurrence probability of two clusters which is recorded at each second by emoxu3. We observe that within the same time interval the reference approach has a higher probability to execute at a higher frequency. On the other hand, our approach is more likely to execute at the lower frequencies, thus reducing the energy consumption.

**Remark 4.** *ODROID XU-3 board contains four cores per cluster. As each heavy DAG ($C_i > T_i$) is allocated to two or more cores while executing, using this experimental setup, it is not possible to execute more than four heavy DAGs at a time. However, there is no such restriction for the light DAGs ($C_i \leq T_i$). In this work, we consider that a heavy DAG cannot be allocated in multiple clusters.*

## VI. SIMULATIONS

For large-scale evaluation, we perform simulations and compare the results with existing baselines. We generate DAGs using the approach discussed in Section V. We consider two types of task periods. For *harmonic periods*, $T_i$ is enforced to be an integral power of 2. We find the smallest value $\alpha$ such that $2^\alpha \geq L_i$, where $L_i$ is the critical path length of task $\tau_i$ and then set $T_i$ to be $2^\alpha$. For *arbitrary periods*, $T_i$ is determined using Gamma distribution as described in Section V.

We compare our approaches with some existing baselines studied in [14], [19]. Total power consumption by our approach and by these baselines are calculated using the power model in Section II. As mentioned earlier, [14] considered per-core DVFS, i.e., each core individually is an island of the cluster-based platform. For a fair comparison, according to the scheduling policy of [14], when a task is allocated on some cores at any time instant $t$, we choose the maximum speed among all these cores. We consider [14] as a baseline because that work is closely related to ours. Although they have considered per-core DVFS and restrict their attention only to implicit deadline tasks, the task and the power model are same. Besides, although this work and [14] propose different approaches to power saving, the initial (preparation) steps of both approaches are based on commonly known techniques like task decomposition, task merging, etc.

The work in [19] studied a greedy slack stealing (GSS) scheduling approach considering inter-dependent sequential tasks. It considered the DAG model to represent dependencies among the tasks. In GSS, the *slack* (unused time in actual computation requirement of a task) is reclaimed by one task by shifting others towards the deadline. They did not consider repetitive tasks; hence it can be regarded as scheduling a single task. Besides this, the power and graph model used in [19] is different from ours. To ensure a fair comparison, we execute the GSS algorithm using the power model in Equation (1) and assume that once introduced in the system; a processor remains active. We also consider a minimum inter-arrival separation for a DAG. That work considered three different kinds of nodes: *AND, OR*, and *Computation* nodes (Subsection 2.1 in [19]). A computation node has both the maximum and average computation requirement. To comply with our work where the focus in energy reduction while guaranteeing worst-case temporal correctness, we execute the GSS algorithm considering only the computation nodes with their maximum computation requirement. We made all the changes in order to provide a fair comparison. Despite these differences, [19] is chosen as a baseline because it studied a *GSS* scheduling approach for energy minimization. It applies to applications consisting of inter-dependent sequential tasks and their dependencies was represented by a DAG, which is similar to our task model.

We compare power consumption by varying two parameters for each task: *task periods (utilization)* and *the number of nodes*. We randomly generate 25 sets of DAG tasks and compare the average power consumption.

**Notations of Referenced Approaches.** For the task partitioning step, one may either randomly choose any two and allocate them to the same cluster, or greedily choose the ones with lowest current speed as proposed. Regarding speed-profile calculation, there are also two options (Approaches A and B in Subsection IV-A). Combining these options in two steps lead to four baselines: *MaxSpeed_Greedy*, *SingleSpeed_Greedy*, *MaxSpeed_Random*, *SingleSpeed_Random*. Also, two baselines mentioned above are included for comparison:

- Federated scheduling with intra-task processor merging [14], denoted by *Fed_Guo*;
- GSS algorithm [19], denoted by *GSS_Zhu*.

## A. *Constrained Deadline Task*

Here, we report the power consumption under the scheme for constrained deadline tasks mentioned in Section IV. We evaluate the efficiency of our proposed method by changing two parameters; task period (utilization) and the number of nodes in the task.

**Effect of Varying Task Periods (utilization).** Here we control the average task utilization through varying the task period. In order to make the task schedulable, the critical path length $L_i$ of task $\tau_i$ should not exceed its deadline $D_i$. We vary the period in a range ($L_i \leq T_i \leq C_i$). The parameter $L_i$ and $C_i$ are measured once the DAG is generated according to the technique described in Section V. We also use the following equation (according to [14]) to ensure that the value of $T_i$ satisfies the range ($L_i \leq T_i \leq C_i$).

$$T_i = L_i + (1 - k)(C_i - L_i) \tag{6}$$

Here, $k \in$ [0, 1] is task utilization. As we are considering the constrained deadline tasks, $D_i$ is randomly picked from the range ($L_i \leq D_i \leq T_i$). The results are presented in Figure 7. Note that when any parameter (e.g., number of nodes in a DAG, task utilization) changes, savings in energy randomly vary within a small range and we consider the minimum value among them. The results indicate a proportional relationship between the average power consumption and average task utilization. It happens because a higher task utilization imposes tighter real-time restrictions. It restricts (refer to Figure 2(b)) the space for the segment length optimization. Figure 7 shows that *SingleSpeed_Greedy* approach outperforms the others and leads to a power savings of at least 16.67% and 56.24% compared to the *Fed_Guo* and *GSS_Zhu* approaches. In *SingleSpeed_Greedy* approach, a task executes with a single speed throughout the deadline. During the task partitioning step, a suitable partner (with similar speed-profile) leads to energy efficiency. However, for the other approaches task speed may vary throughout the deadline. In that case, evil alignment and a significant variation in the speed may lead to reduced energy efficiency. Refer to Figure 3 and Example 1 for details.

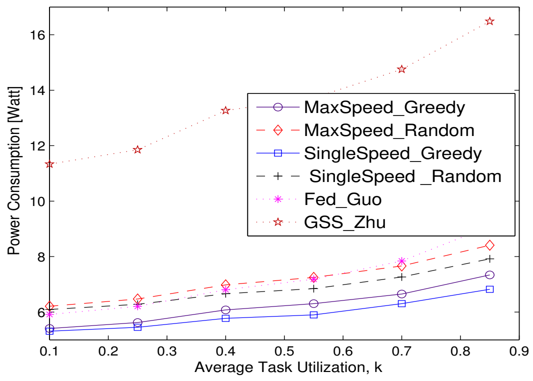

Fig. 7. Under various utilization, this graph compares power consumption for the constrained deadline tasks for different approaches with a fixed number of nodes (30 in this experiment).

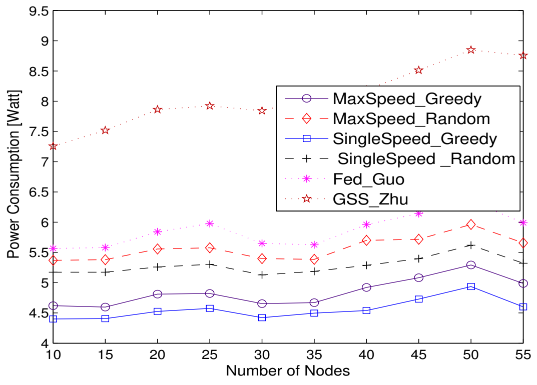

Fig. 8. For the harmonic task period, this graph compares power consumption for the constrained deadline tasks considering different approaches with a various number of nodes.

**Effect of Varying the Numbers of Nodes.** Here we vary the number of nodes ($T_i$ remains fixed and $D_i \leq T_i$) and report the average power consumption. We consider both harmonic and arbitrary periods (reported in Figures 8 and 9). For both of these settings, we randomly generate 100 tasks; and vary the number of nodes in each task between 10 and 55.

Compared to the previous set of experiments (varying task utilization with a fixed number of nodes), we observe similar improvements in power consumption, i.e., choosing a single speed over the whole deadline leads to more power savings. Especially, when considering harmonic task periods the *SingleSpeed_Greedy* approach uses on average 22.25% and 43.56% less power compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively. If we consider arbitrary task periods, the savings become 12.39% and 54.57%, respectively.

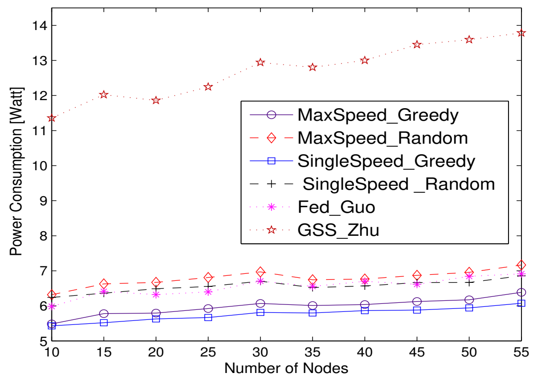

Fig. 9. For the arbitrary task period, this graph compares power consumption for the constrained deadline tasks considering different approaches with a different number of nodes.

## B. *Implicit Deadline Task.*

Now we consider the Implicit deadline tasks and show their average power consumption by changing two parameters: task period (or utilization) and the number of nodes.

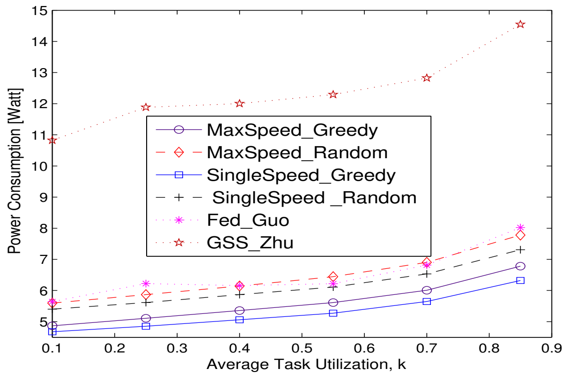

Fig. 10. Under various utilizations, this graph compares power consumption for the implicit deadline tasks for different approaches with a fixed number of nodes (30 in this experiment).

**Effect of Varying Task Periods (Utilization).** In this experiment, we fix the number of nodes to 30. We vary the task period using Equation (6) and report the average power consumption in Figure 10. Similar to the phenomenon we observed in the last experiment, average energy consumption for implicit deadline tasks is directly proportional to the average task utilization. Figure 10 also shows that adopting the *SingleSpeed_Greedy* approach results in reduced power consumption. On average, the *SingleSpeed_Greedy* approach leads to a power saving of at least 18.44% and 57.3% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively.

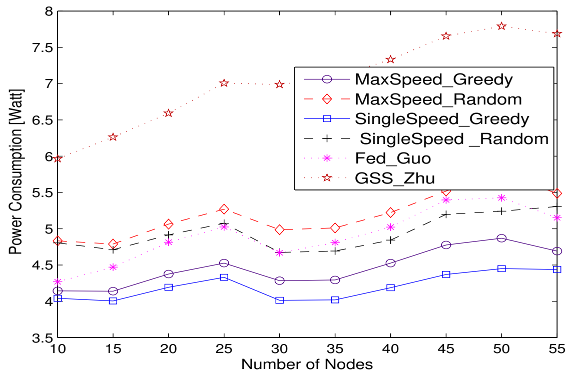

Fig. 11. For the harmonic task period, this graph compares power consumption for the implicit deadline tasks considering different approaches with a different number of nodes.

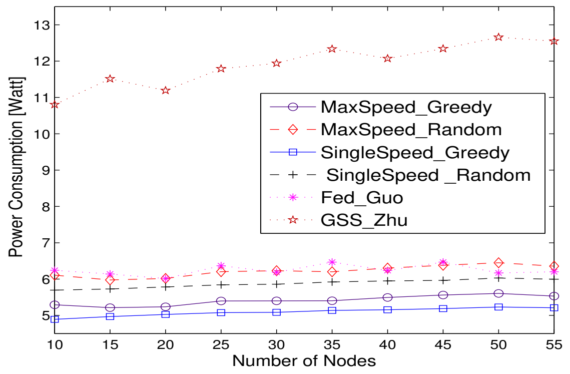

Fig. 12. For the arbitrary task period, this graph compares power consumption for the implicit deadline tasks Considering different approaches with a different number of nodes.

**Effect of Varying the Numbers of Nodes.** Now we measure the average power consumption by varying the number of nodes (10 to 55) with fixed $T_i$. We randomly generate 100 tasks with harmonic and arbitrary deadline. We report the average power consumption in Figure 11 and 12 for harmonic and arbitrary deadline tasks, respectively. Again, we observe similar improvements in power consumption, i.e., choosing a single speed over the whole deadline outperforms other approaches. Instead of task period, we use the term deadline because we are considering constrained deadline tasks. Specifically, under harmonic task periods, the *SingleSpeed_Greedy* incurs 14.05% and 40% less power on average compared to *Fed_Guo* and *GSS_Zhu*; under arbitrary task periods, the savings potential are 18.39% and 57.27%, respectively.

## VII. Discussions: Assumptions and Applicability

We have mentioned some assumptions we have made in the paper. The first two subsections in this section discuss the validity of these assumptions, their impacts and potential solutions to overcome these impacts. Then, in Subsection VII-C we detail the reasons behind the measurement overheads (see Section V) and the applicability of our proposed approaches.

## A. Assumptions Behind the Power Model

**Continuous Frequency Scheme.** In this paper, we consider a continuous frequency scaling, and it can be rounded up to achieve the discrete frequency level. So, compared to the system with discrete frequency levels, the applicability of our approach is not affected. Modern processors have a relatively fine-grained step to scale frequency. For example, the ODROID XU-3 board (used in the system experiment) has a frequency range of $0.2 - 1.4GHz$ for LITTLE cores) and $0.2 - 2GHz$ for big cores, with a scale step of $0.1GHz$. With such fine-grained steps, the energy consumption with continuous assumption becomes very close to actual scenario, as reported in our system experiments.

**Components Behind the Overall Power Consumption.** While some other factors such as cache miss, bus accesses, context-switches, and I/O usage also affect the power consumption, CPU power consumption is one of the major contributors to the overall power consumption. Power consumption may largely be dominated by any of these factors depending on the application/benchmark (e.g., power consumption is dominated by the radio/network in some communication-oriented applications [33], [34]). In this work, we target to minimize the CPU power consumption only. While minimizing the CPU power consumption, our approach does not increase the power consumption that is influenced by other factors, as our technique does not introduce additional preemption/frame-related overhead compared to any existing DAG schedulers (DAG decomposition based). It would build the foundation for more complicated analysis that considers all other factors of overall power consumption in the future.

**Dynamic Power Management.** DPM explores idle slot of a processor and puts the processor to a low power mode to reduce the static power consumption. Switching to low power mode (and backward) incurs additional energy consumption and is beneficial only when the idle slot is longer than a threshold, known as the *break-even time (BET)* [3] [35], [36]. In this paper, the available idle slot may not be longer than the BET for two reasons. First, we focus on clustered multiprocessor platform, where processors within each cluster must execute at the same speed, i.e., the maximum speed necessary for the demand on each processor at a given instant. As a result, unless all processors within a cluster are idle, the cluster cannot be switched into any sleep mode. For sporadic releases, idle slots of each processor are unlikely to be synchronized. Moreover, cluster-wide idle slots tend to be relatively short. Second, while executing a task, a uniform execution speed significantly reduces the overall energy consumption (Theorem 4 of [14]), which is the goal of our proposed approaches—this leads to further reduction of idle slots. For example, a study using Intel Strong ARM SA-1100 processor has shown a transition time of 160ms to switch from sleep mode back to run mode [37], which can be larger than many task periods in avionics. As triggering mode switches becomes more energy consuming in general and may overwhelm the gain in energy savings, DPM is considered out of the scope of this work. DPM could be a valid option under certain scenario, and we leave the further exploration along this direction as future work.

[3]BET is the minimum duration for the processor to stay at the sleep mode.

## B. Heterogeneity

This section discusses a specific type of multi-core platform with diverse computing abilities: heterogeneous multi-core platform. We first discuss different types of heterogeneous platforms, then explain the reason for not considering heterogeneity, and finally discuss how the proposed techniques can be extended to handle heterogeneity.

In a heterogeneous platform, different cores have different computational capabilities. In terms of speed, Funk defined a widely-accepted classification of the heterogeneous platform [38] as follows, where the speed of the processor denotes the work completed (while executing a task) in a single-time unit by this processor.

(i) *Identical multiprocessors*: On Identical multiprocessors, all tasks are executed at the same speed on any processor;

(ii) *Uniform multiprocessors*: On Uniform multiprocessors, all the tasks allocated on a given processor are executed at the same speed, but at different speeds on different processors, i.e., a tasks execution speed depends on the processor where the task is allocated;

(iii) *Unrelated multiprocessors*: On Unrelated multiprocessors, execution speeds of different tasks may vary on the same processor, i.e., a tasks execution speed depends on both the task itself and the processor where it is allocated.

In a heterogeneous platform, each core is designed with a different computational capability, and an efficient task to core mapping improves the system resource efficiency. In the context of energy efficiency, two major directions have been mentioned in [39] for any type of heterogeneous platform:

(i) Find an appropriate core/cluster for task mapping to reduce the overall power consumption of the whole platform.

(ii) Deploy energy-aware scheduling techniques on each core/cluster to reduce power consumption.

The proposed approach (described in Sections III and IV) covers both directions, where we first use speed profile to identify efficient core/cluster to task mapping, and then try to reduce the overall cluster speed as much as possible. A moments thought should convince the reader that such an approach works for an identical heterogeneous platform (a.k.a., homogeneous multiprocessor) as task-core mapping does not impact energy consumption much. Other benefits of considering the identical multiprocessor platform are its reusability and simpler design [40].

Our approach can be extended to apply to the uniform heterogeneous platform by modifying the parameters in the power model in Equation (1), i.e., setting different $\alpha, \beta,$ and $\gamma$ values for the 'big' and 'LITTLE' cluster. Under such consideration, different clusters no longer share the same power model, and the same task may have different execution requirements on different processors. In the future, we will extend our approach to adopt the unrelated heterogeneity.

### C. A Note on the Overhead Delay

We have mentioned that the scaling-down (up) the frequency of the big cluster takes at most 60 $(40)ms$, while both these operations take at most $15ms$ on the LITTLE cluster (see Section V). we use cpufreq-set module to change the system's frequency, and this module accounts for microsecond-scale transition delay (usually 100-200$\mu s$), which is typically incorporated into the WCET. In our case, the delay is much higher because (i) we used a Python script to measure the delay; and (ii) there is some user-level delay caused by I/O operations and file logging, e.g., time-stamp storage before and after each run. Time-stamp storage detects the arrival and completion of nodes which could be avoided when one does not need to track system behavior in a precise manner (which is the normal scenario). Considering the potential overhead issue, in Subsection IV-A, we proposed Approach-B, where a task executes at a single speed (so, there is no frequency changing overhead) for the whole duration from its release to the deadline. Experimental study (Section VI) also shows excellent performance of such approach when WCETs of sub-jobs are short. Note that, we can not entirely avoid the speed changing overhead for Approach-A in Subsection IV-A. However, we can reduce the number of frequency changes by partitioning the tasks (into a cluster) according to Algorithm 1. Thus, an efficient partitioning can reduce the frequency changing overhead.

## VIII. Related Work

Much work has been done aimed at energy-efficient scheduling of sequential tasks in a homogeneous multi-core platform (see [13] for a survey). Considering the mixed-criticality task model and varying-speed processors, the works on [24], [41]–[44] proposed an approach to handle the energy minimization problem. [45] and [46] presented an energy efficient approach for the heterogeneous multi-core platform. To minimize the energy consumption under timing constraint and heterogeneous platform, [47] considered the problem of allocating real-time tasks onto the cores.

Till date, considering both the intra-task parallelization and power minimization has received less attention. A greedy slack stealing algorithm is proposed in [19] that deals with task represented by graphs but did not consider the periodic DAGs. Assuming per-core DVFS, [36] provided the technique to combine DVFS and DPM. [48] investigated the energy awareness for cores that are grouped into blocks, and each block shares the same power supply scaled by DVFS. Benefits of (in terms of power saving) intra-task parallelism is proven theoretically in [1]. Considering the fork-join model, [49] reported an empirical evaluation of the power savings in a real test-bed. Based on level-packing, [50] proposed an energy efficient algorithm for implicit deadline tasks with same arrival time and deadline.

None of these works allows intra-task processor sharing considering the sporadic DAG task model. The recent work in [14] is most related to ours. However, our work is significantly different from [14] regarding the task model, platform, real-time constraints (deadlines), solution techniques, and also evaluation (real platform experiment instead of simulation). *First*, the work in [14] considered a simplified model where only one DAG task executes at a time, and the number of cores was unlimited. *Second*, [14] assumed per-core speed scaling. However, many of the existing platforms (e.g., ODROID XU-3) do not support such speed scaling—speeds of processors under the same cluster must execute at the same speed. As the number of cores fabricated on a chip increases, per-core speed scaling design is less likely to be supported due to the inefficiency on hardware levels [15]. *Third*, the work in [14] did not consider *constrained* deadline tasks (leaving the non-negligible idle gaps between the task deadline and its next release un-utilized) and focused only on implicit deadline tasks. *Finally*, the evaluations in [14] were done based on simulations without any implementation on a real platform.

## IX. Conclusion

The evolution of multi-core embedded systems and the rapid growth of computation-intensive time-sensitive applications call for considerable attention for energy-efficient real-time multi-core scheduling. In this paper, we have studied real-time scheduling of a set of implicit and constrained deadline sporadic DAG tasks which is considered as the most representative model of a deterministic parallel task. We schedule these tasks on the cluster-based multi-core platforms with the goal of minimizing the overall system power consumption. In a clustered multi-core platform, the cores within the same cluster run at the same speed at any given time. Such design better balances energy efficiency and hardware cost and appears in many systems. However, from the resource management point of view, this additional restriction leads to new challenges. By leveraging a new concept named *speed-profile*, which models energy consumption variations during run-time, we can conduct scheduling and task-to-cluster partitioning while minimizing the expected overall long-term CPU energy consumption. To our knowledge, no work considers energy-aware real-time scheduling of DAG tasks with constrained deadlines. Also, this is the first work that has investigated energy-efficient scheduling of DAGs on clustered multi-core platform.

We have implemented our result on an ODROID XU-3 board and have experimented to demonstrate its feasibility and practicality. We have also complemented our system experiments on a larger scale through realistic simulations that demonstrate an energy saving of up to 57% through our proposed approach compared to existing methods. In this work, we have restricted our attention mainly to the CPU power consumption which is one of the major contributors to the total power draw. In the future, we plan to consider other components that may affect the total power consumption, e.g., cache misses, bus accesses, context switches, and I/O usage.

## Acknowledgment

The authors would like to thank the shepherd for the kind efforts in helping us improving the paper. This work is partially supported by National Science Foundation CNS-1850851 and CNS-1742985.

## References

[1] A. Paolillo, J. Goossens, P. M. Hettiarachchi, and N. Fisher, "Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies," in *RTCSA*. IEEE, 2014.

[2] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. D. Gill, "Parallel real-time scheduling of dags," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, pp. 3242–3252, 2014.

[3] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-time analysis of conditional DAG tasks in multiprocessor systems," in *ECRTS*. IEEE, 2015.

[4] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF scheduling of directed acyclic graphs on multiprocessor systems," in *RTNS*. ACM, 2013.

[5] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *RTSS*. IEEE, 2012.

[6] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global EDF for parallel tasks," in *ECRTS*. IEEE, 2013.

[7] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic DAG task model," in *ECRTS*, 2013.

[8] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *ECRTS*. IEEE, 2014.

[9] S. Baruah, V. Bonifaci, and A. Marchetti-Spaccamela, "The global EDF scheduling of systems of conditional sporadic DAG tasks," in *ECRTS*, 2015.

[10] T. Hagras and J. Janecek, "A high performance, low complexity algorithm for compile-time job scheduling in homogeneous computing environments," in *Parallel Processing Workshops*. IEEE, 2003.

[11] K. Jeffay, D. L. Stone, and F. D. Smith, "Kernel support for live digital audio and video," *Computer communications*, vol. 15, no. 6, pp. 388–395, 1992.

[12] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *IPDPS*. IEEE, 2003.

[13] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 1, p. 7, 2016.

[14] Z. Guo, A. Bhuiyan, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient multi-core scheduling for real-time dag tasks," in *LIPIcs-Leibniz International Proceedings in Informatics*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[15] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ISLPED*. IEEE, 2007.

[16] 2017, http://www.hardkernel.com/.

[17] D. Liu, J. Spasic, G. Chen, and T. Stefanov, "Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs," in *ESTIMedia*. IEEE, 2015.

[18] J. Kim, H. Kim, K. Lakshmanan, and R. R. Rajkumar, "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car," in *ICCPS*. ACM, 2013.

[19] D. Zhu, D. Mosse, and R. Melhem, "Power-aware scheduling for and/or graphs in real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 9, pp. 849–864, 2004.

[20] W. M. Kolb, *Curve fitting for programmable calculators*. Imtec, 1984.

[21] S. Pagani and J.-J. Chen, "Energy efficient task partitioning based on the single frequency approximation scheme," in *RTSS*. IEEE, 2013.

[22] ——, "Energy efficiency analysis for the single frequency approximation (SFA) scheme," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 5s, p. 158, 2014.

[23] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient DVFS scheduling for mixed-criticality systems," in *EMSOFT*. IEEE, 2014.

[24] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," in *RTAS*. IEEE, 2016.

[25] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient real-time scheduling of dag tasks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 5, p. 84, 2018.

[26] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries *et al.*, "A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and DVFS for performance and power scaling," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, 2011.

[27] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *RTSS*. IEEE, 2013.

[28] 2017, https://github.com/tuxamito/emoxu3.

[29] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *ICST*, 2010.

[30] 2017, http://en.wikipedia.org/wiki/Gamma distribution.

[31] 2017, https://github.com/scheduler-tools/rt-app/.

[32] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," in *RTCSA*. IEEE, 2007.

[33] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, "Software-based on-line energy estimation for sensor nodes," in *Proceedings of the 4th workshop on Embedded networked sensors*. ACM, 2007, pp. 28–32.

[34] Y. Liu, W. Zhang, and K. Akkaya, "Static worst-case energy and lifetime estimation of wireless sensor networks," in *IPCCC*. IEEE, 2009.

[35] H. Cheng and S. Goddard, "Online energy-aware i/o device scheduling for hard real-time systems," in *Proceedings of the conference on Design, automation and test in Europe:*. European Design and Automation Association, 2006.

[36] G. Chen, K. Huang, and A. Knoll, "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, p. 111, 2014.

[37] H.-Y. Zhou, D.-Y. Luo, Y. Gao, and D.-C. Zuo, "Modeling of node energy consumption for wireless sensor networks," *Wireless Sensor Network*, vol. 3, no. 01, p. 18, 2011.

[38] S. H. Funk, *EDF scheduling on heterogeneous multiprocessors*. University of North Carolina at Chapel Hill, 2004.

[39] M. A. Awan, D. Masson, and E. Tovar, "Energy efficient mapping of mixed criticality applications on unrelated heterogeneous multicore platforms," in *SIES*. IEEE, 2016.

[40] A. Sethi and H. Kushwah, "Multicore processor technology-advantages and challenges," *International Journal of Research in Engineering and Technology*, vol. 4, no. 09, pp. 87–89, 2015.

[41] S. Baruah and Z. Guo, "Mixed-criticality scheduling upon varying-speed processors," in *RTSS*. IEEE, 2013.

[42] ——, "Scheduling mixed-criticality implicit-deadline sporadic task systems upon a varying-speed processor," in *RTSS*. IEEE, 2014.

[43] Z. Guo and S. Baruah, "The concurrent consideration of uncertainty in wcets and processor speeds in mixed-criticality systems," in *RTNS*. ACM, 2015.

[44] S. Sruti, A. A. Bhuiyan, and Z. Guo, "Work-in-progress: Precise scheduling of mixed-criticality tasks by varying processor speed," in *RTSS*. IEEE, 2018.

[45] J.-J. Chen, A. Schranzhofer, and L. Thiele, "Energy minimization for periodic real-time tasks on heterogeneous processing units," in *IPDPS*. IEEE, 2009.

[46] C. Liu, J. Li, W. Huang, J. Rubio, E. Speight, and X. Lin, "Power-efficient time-sensitive mapping in heterogeneous systems," in *PACT*. ACM, 2012.

[47] A. Colin, A. Kandhalu, and R. Rajkumar, "Energy-efficient allocation of real-time applications onto heterogeneous processors," in *RTCSA*, 2014.

[48] X. Qi and D.-K. Zhu, "Energy efficient block-partitioned multicore processors for parallel applications," *Journal of Computer Science and Technology*, vol. 26, no. 3, p. 418, 2011.

[49] A. Paolillo, P. Rodriguez, N. Veshchikov, J. Goossens, and B. Rodriguez, "Quantifying energy consumption for practical fork-join parallelism on an embedded real-time operating system," in *RTNS*. ACM, 2016.

[50] H. Xu, F. Kong, and Q. Deng, "Energy minimizing for parallel real-time tasks based on level-packing," in *RTCSA*. IEEE, 2012.