Energy-Efficient Parallel Real-Time Scheduling on Clustered Multi-Core

Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, Nan Guan, and Zhishan Guo

Abstract—Energy-efficiency is a critical requirement for computation-intensive real-time applications on multi-core embedded systems. Multi-core processors enable intra-task parallelism, and 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 67% through our proposed approach compared to existing methods.

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

1 Introduction

Multi-core processors appear as an enabling platform for embedded systems applications that require real-time guarantees, energy efficiency, and high performance. Intra-task parallelism (a task can be executed on multiple cores simultaneously) enables us to exploit the capability of the multi-core platform, and facilitates a balanced distribution of the tasks among the processors. Such a balanced distribution leads to energy efficiency [1]. Directed Acyclic Graph (DAG) task model [2] is one of the most generalized workload model for representing deterministic intra-task parallelism. Recently, quite some effort has been spent on developing real-time scheduling strategies and schedulability analysis of DAG tasks, few to mention [2], [3], [4], [5], [6], [7], [8].

There are several real-world application that uses the DAG model. For example, the work in [3] studies problems related to scheduling parallel real-time tasks, modeled as DAG, on multiprocessor architectures. In a homogeneous computing environment, a low-complexity compile-time algorithm for scheduling DAG tasks is proposed in [9]. 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 [10], a significant amount of work has considered power minimization for non-parallel tasks on a multi-core platform (refer to [11] for a survey). Regarding parallel tasks, Guo et al. studied energy-efficient real-time scheduling for DAG tasks as an early research effort [12]. 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 [12] 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 [13]. 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 [14]) 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 [15]) 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 [16].

Despite the urgent need, to our knowledge, no work has been done that considered the energy-efficient scheduling of DAG tasks in clustered multi-core platforms, 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 the energy consumption is nonlinear, making it

- *A. Bhuiyan and Z. Guo are with the Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL.*
- *D. Liu is with Yunnan University, Kunming, China.*
- *A. Khan is with Brainco Inc., Somerville, MA.*
- *A. Saifullah is with the Department of Computer Science, Wayne State University, Detroit, MI.*
- *N. Guan is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong.*

highly challenging to minimize energy consumption while guaranteeing real-time correctness, i.e., none of the tasks miss their deadline.

(ii) Existing solution (e.g., [12]) 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 speed of a cluster becomes unpredictable when shared by multiple tasks with sporadic release patterns.

Contribution. In this paper, we propose a novel technique for energy-efficient scheduling of constrained deadline DAG tasks 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. Considering a continuous frequency scheme, we first propose a novel concept of *speed-profile* to present the energy-consumption behavior for each task as well as each cluster, such that they could guide task partitioning in an energy-efficient manner. An efficient *greedy* algorithm is proposed to partition DAG tasks according to the constructed speed-profiles.

• We propose an approach to creating the speed-profile to adapt to the discrete frequency scheme. Also, we extend our approach to apply to the heterogeneous platform.

• 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 [14]. 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 [12], [17]. The simulation results demonstrate that our method can reduce energy consumption by up to 66% compared to the existing ones under the cluster-based platform setting.

Organization. The rest of the paper is organized as follows. Section 2 presents the workload, power, and platform models, , and the problem statement. Section 3, describes the importance of creating a speed-profile for an individual task and the whole cluster. Section 4 discusses the approaches to create the speed-profile (considering both the continuous and discrete frequency mode) for each task. In this section, we also propose a greedy algorithm to allocate multiple tasks in the same cluster. Section 5 and 6 presents the experimental and simulation results. Section 7 discusses related work including a detailed comparison with our existing work [12], [18]. Section 8 concludes this paper.

2 System Model, Problem Statement and Background

2.1 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 τ_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 τ_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 τ_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 τ_i, i.e., in order to make τ_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 τ_i, the condition $T_i \geq L_i$ (implicit deadline tasks) and $D_i \geq L_i$ (constrained deadline tasks) must hold for τ_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., α, β and γ in the power model discussed below) to be identical across different clusters—these parameters can be derived using any curve-fitting method, e.g., [19].

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 αs^γ. Here, the $\alpha > 0$ depends on the effective switching capacitance [20]; $\gamma \in [2, 3]$ 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 [10], [12], [18], [20], [21], [22], [23]. Beside this, recently this model was shown to be highly realistic by showing its similarity with actual power consumption [21]. Figure 1 shows comparison

between the original power consumption results from [24] and the power model in Equation (1).
textbfAssumptions. In this paper, we make the following assumptions: (i) we focus on CPU power consumption, and (ii) Dynamic power management (DPM) is not considered. Appendix B provides the details behind these assumptions, their impacts, and some hints to overcome the drawbacks.

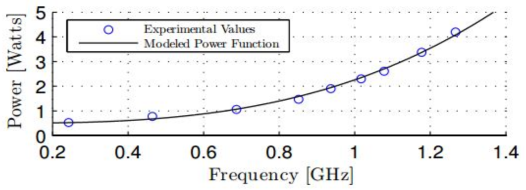


Fig. 1: Comparison of the power model (Equation (1)) with experimental results in [24]. Here, $\alpha = 1.76 Watts/GHz^3$, $\gamma = 3$, and $\beta = 0.5$ Watts. This figure is adopted from [20].

Problem Statement. Considering a constrained deadline sporadic DAG task-set on a clustered multi-core platform, we focus on finding a correct scheduling strategy, while the CPU power consumption is minimized.

2.2 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 τ_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 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 $\mathcal{N}_i^3$ in the 4^{th} segment in Figure 2(b).

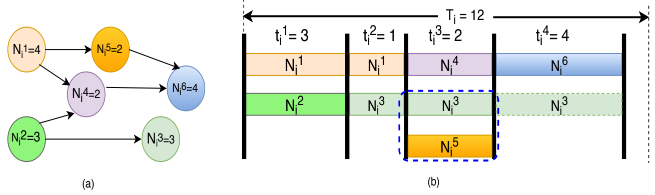


Fig. 2: (a) A DAG, τ_i (b) transformed DAG τ_i after applying task decomposition. Both of them are adopted from [12].

Intra-Task Processor Merging. After applying task decomposition and segment extension upon a DAG task τ_i, some of these cores (where τ_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 [12] 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 [12]) as well as the total number of clusters.

3 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*. We provide the definition of speed-profile and its motivation in Subsection 3.1. Subsection 3.2 describes how speed-profiles are handled when two tasks are partitioned into the same cluster.

3.1 Speed-Profile for Each DAG

Interesting energy-saving techniques (e.g., segment extension) have been proposed in [12] for the implicit deadline 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 τ_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 [12] becomes *non-convex* (in a convex function, we can find the global maximum or minimum, for some variables of this function, which does not hold for a non-convex function) in a cluster-based platform (see Lemma 1).

Lemma 1. *In a cluster-based platform, the convex optimization problem constructed in [12] 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 τ_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, the work in [12] considered that the execution speed of a node, $\mathcal{N}_i^l$, is *constant* within its scheduling window (from b_i^l to d_i^l), and can be represented by a function of nodes execution requirement and its scheduling window. Also, the work in [12] considered that a single DAG executes at a time, and, hence the execution speed of a node is not affected by the execution speed of other nodes (of other tasks). In this work, we consider the cluster-based platform, and the execution speed of a node depends on the execution speed of other nodes (of other tasks) in the same cluster. As a result, we cannot express the execution speed of a node as a function of its execution requirement, resulting in quadratic inequality constraints (Equation (2)). This makes the optimization problem *non-convex*. ■

Due to the characteristics of 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 (w.r.t. 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 τ_1 and τ_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 τ_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*

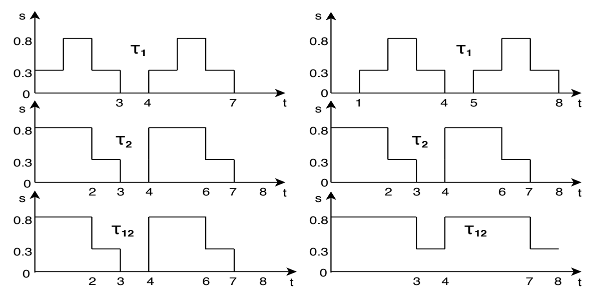


Fig. 3: When two tasks τ_1 and τ_2 with fixed speed patterns each are partitioned on to the same cluster, the resultant speed pattern (τ_{12}) of the cluster may vary for their (τ_1 and τ_2) different release offsets. In order to satisfy platform model restrictions while guaranteeing the correctness, the processors (of the same cluster) must run at the maximum/larger of the two individual speeds at each instant.

this example, from τ_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, τ_2 requires no execution over [5,7)), we may not be able to reduce the cluster speed at all.

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. *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 a speed from the finite set of possible speeds, and $f_{\mathcal{S}}(s)$ represents the portions of the time (likelihoods) when it is running at speed s.*

Example 2. *Let us consider a task τ_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 $\mathcal{S}_i = \begin{pmatrix} 0.6 & 0.5 \\ 5/15 & 10/15 \end{pmatrix} = \begin{pmatrix} 0.6 & 0.5 \\ 0.33 & 0.67 \end{pmatrix}$. 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 τ_i remains the same for an initial phase (release offset) $\phi_i \geq 0$. Regarding inter-task combinations, we assume uniform distribution for the phase of any task; i.e., $\phi_i \sim U[0, T_i)$.*

Subsection 4.1 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.

3.2 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 τ_i and τ_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*

$$\mathcal{S}_i \odot \mathcal{S}_j = \begin{pmatrix} 6 & 6 & 6 & 5 \\ 0.16 & 0.24 & 0.24 & 0.36 \end{pmatrix} = \begin{pmatrix} 6 & 5 \\ 0.64 & 0.36 \end{pmatrix}.$$

1. Although the appearance of the proposed operator is identical to [25], the calculation is quite different. This is due to the "larger value dominating" nature of the platform model considered in this paper.

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 τ_i and τ_j are allocated to the same cluster, we use $\mathcal{S}_{ij}$ to denote the speed-profile of the cluster (see Example 3).

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. 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.

4 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 3) 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 4.1. Then, Subsection 4.4 describes the task-to-cluster partitioning algorithm.

4.1 Creating the Speed-Profile of a Task

Given a DAG task τ_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 2), 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 τ_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., $D_i \le T_i$), 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 τ_i. At j^{th} segment, we calculate $s_{i,j}$ and $p_{i,j}$ as follows:

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

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 τ_i with $T_i = 15, D_i = 12$ and $C_i = 6.5$. Let, the task is partitioned into three segments of length 5, 7 and 3 time units respectively, where the processor is executing at a (maximum) speed of 0.6 in the first segment, speed of 0.5 in the second segment, and remain idle in the third segment 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 τ_i, then $\mathcal{S}_i$ also represents the cluster speed-profile. If τ_i and τ_j (or more tasks) are executing on the same cluster, then the technique described in Subsection 3.2 needs to be applied before making any choices. The greedy choosing approach for task partition is detailed in Subsection 4.4.

Approach B: A Single Speed Throughout. Theorem 4 of [12] 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 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 τ_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:

$$w_{i,j,k} = s_{i,j,k} \times t_j^c.$$

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 τ_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:*

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

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

2. Note that [12] 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.

Proof. It has been observed in [12] that the following constraint guarantees the real-time correctness:

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

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 [12], 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 τ_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. ■

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 5 considering implicit deadline. 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:

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

4.2 Discretization of the Speed-Profile

In Section 4.1, we have described two approaches to create the speed-profile for an individual task. While creating the speed-profiles, those approaches assume a continuous frequency scheme. From a practical point of view, discrete frequency mode should be preferred over the continuous frequency mode, because a real platform supports only a set of frequencies. Now, we describe the technique to discretize all the speeds available in a speed-profile (assuming that the speed-profile is already created).

Suppose, we execute a task τ_i (and its speed-profile is $\mathcal{S}_i$) in a real-platform, and this platform supports only those speeds available on a speed-set $\mathcal{Z}$. Note that the content of $\mathcal{Z}$ is dependent on the platform. For example, ODROID XU-3 supports a frequency range of 200-1400 MHz (LITTLE cluster) and 200-2000 MHz (big cluster) with scale steps of 100 MHz). Now, for each entry $s_{i,j} \in \mathcal{S}_i$, we find the minimum speed $\mathcal{Z}_k \in \mathcal{Z}$, where $\mathcal{Z}_k \geq s_{i,j}$. Once, we find an appropriate $\mathcal{Z}_k$; we set the value of $s_{i,j}$ as $s_{i,j} = \mathcal{Z}_k$.

Example 7. *Consider a task τ_i with the same speed-profile from Example 4. Let us assume that we will execute τ_i in a platform where $\mathcal{Z} = \{0, 0.2, 0.4, 0.55, 0.75,$ and $1\}$, i.e., this platform supports only six discrete speeds, and all the speeds are normalized w.r.t. the maximum speed supported by this platform. Considering the speed-profile $\mathcal{S}_i$ (from Example 4) and the speed-set $\mathcal{Z}$, we find that:*
(a) $s_{i,1} \leq \{\mathcal{Z}_5$ and $\mathcal{Z}_6\}$
(b) $s_{i,2} \leq \{\mathcal{Z}_4, \mathcal{Z}_5$ and $\mathcal{Z}_6\}$, and
(c) $s_{i,3} \leq \{\mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3, \mathcal{Z}_4, \mathcal{Z}_5$ and $\mathcal{Z}_6\}$.
Now, we choose the minimum $\mathcal{Z}_k \in \mathcal{Z}$ such that $\mathcal{Z}_k \geq s_{i,j}$. So, we assign $\mathcal{Z}_5$ to $s_{i,1}$, $\mathcal{Z}_4$ to $s_{i,2}$, and $\mathcal{Z}_1$ to $s_{i,3}$. Now, the updated (i.e., discretized) speed-profile becomes

$$\mathcal{S}_i = \begin{pmatrix} 0.75 & 0.55 & 0 \\ 0.33 & 0.47 & 0.2 \end{pmatrix}.$$

Theorem 1. *When a task executes with its discretized speed-profile, it guarantees that the task will not miss the deadline.*

Proof. We have shown in Lemma 2 that a task τ_i will not miss the deadline if executed according to its speed-profile $\mathcal{S}_i$. If we discretize τ_i's speed-profile and execute τ_i according to this speed-profile, then the task still guarantees the real-time correctness. This is because any speed $s_{i,j}$ of the discretized speed-profile is greater than or equal to $s_{i,j}$ when it was continuous. ■

4.3 Handling Platform Heterogeneity

In this section, we discuss a specific type of multi-core platform with diverse computing abilities: heterogeneous multi-core platform. We first discuss different types of heterogeneous platforms, and then explain how our 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 [26] 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 execute at the same speed if allocated on the same processor, but at a different speed on different processors. So, the execution speed of a task 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 [27] for any 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.

Our proposed approach covers both directions. First, we use speed profile to identify efficient core/cluster to task mapping and then try to reduce the overall cluster speed as much as possible. It works for an identical heterogeneous platform (a.k.a., homogeneous multiprocessor) as task-to-core mapping does not impact energy consumption much.

TABLE 1: Estimated parameters for different cluster of an ODROID XU-3 board. This table is adopted from [15].

Cluster Type	$\beta(W)$	$\alpha(W/MHz^{\gamma})$	γ
big	0.155	3.03×10^{-9}	2.621
LITTLE	0.028	2.62×10^{-9}	2.12

Now, we extend our approach 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 γ 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 clusters. We report the estimated values of $\alpha, \beta,$ and γ in Table 1. These parameters are adopted from [15]. The work in [15] estimated these parameters for the ODROID XU-3 board using the real power measurements along with a curve fitting method. They have also assumed that there is another contributor to the total power consumption of a cluster, i.e., the "uncore" power consumption (reported in Table 2). The "uncore" power consumption introduced in the system from some components other than a processor, e.g., a shared cache. Similar to the dynamic power consumption, the "uncore" power consumption also depends on the processor frequency. However, unlike the dynamic power consumption, there is always some "uncore" power consumption as long as the cluster remains on (even if there is no workload on a processor). Considering all the parameters from Table 1 and

TABLE 2: The "uncore" power consumption for different cluster of an ODROID XU-3 board. This table is adopted from [15].

Freq(GHz)	2	1.8	1.6	1.4	1.2	1.0
big cluster(W)	0.8	0.528	0.39	0.309	0.244	0.182
Freq(GHz)	1.4	1.2	1.0	0.8	0.6	0.4
LITTLE cluster(W)	0.04	0.04	0.04	0.04	0.04	0.04

Table 2, we bring the following modification in Equation (1):

$$P(s) = N_p\beta + \alpha s^{\gamma} + P_s(f), \tag{6}$$

Here, N_p denotes the number of cores per cluster, and $P_s(f)$ denotes the "uncore" power consumption.

We have a different power model for the "big" and the "LITTLE" cluster, but we still don't know what the basis of assigning a task to a cluster is. Recall that, while creating the speed-profile, some vital information (e.g., the speed of a core at a specific time) were known to us (Subsection 4.1). If the execution speed of a task is greater than a certain threshold at any point from its release to its deadline, then we assign this task to the big cluster. Else, we assign this task to the LITTLE cluster. For the platform we consider (ODROID XU-3), we set the threshold to 0.7. It is the ratio of the maximum speed supported by the big cluster and the LITTLE cluster (see Table 2).

4.4 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.

Let, we decide for each task whether it should be allocated on a LITTLE or a big cluster using the technique described in Subsection 4.3. To select a (task) pair that will share the same cluster, we greedily choose the pair that provides maximum power saving, as depicted in Algorithm 1. Note that we allow the pairing of two DAGs that are not merged previously. Also, if any task uses more cores than what 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 $\mathcal{S}_i$ (computed using
   approach A or approach B) 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:        $\bar{\mathcal{S}} \leftarrow \bar{\mathcal{S}} - \mathcal{S}_{kx} - \mathcal{S}_{xk} - \mathcal{S}_{ky} - \mathcal{S}_{yk}$
14:    end for
15: end while
16: return $\tilde{\mathcal{S}}$.
```

Algorithm 1 creates 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, calculate 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 (calculated using Equation (6) and Equation (10) from [12]) 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).

Theorem 2. *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 τ_i will not miss the deadline if executed according to its speed-profile $\mathcal{S}_i$. If τ_j share a cluster with another task τ_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 2. *For n tasks, the time complexity to generate all possible speed-profiles, $\bar{\mathcal{S}}$, is $O(n^2Z)$, 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 $\bar{\mathcal{S}}$ 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 methods (Subsection 4.1) to create the speed-profile for a constrained-deadline DAG. 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 3, we could evaluate and compare all potential pairs of the combination by calculating the cluster speed-profile after merging. Finally, Subsection 4.4 discussed how to use these speed-profiles to find suitable partners to share a cluster greedily.

5 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. In this experiment, we generate two task sets each with 300 DAGs, and use the widely used Erdos-Renyi [29] method to generate a DAG. We tune a parameter p, that denotes the probability of having an edge between two nodes. In this experiment, we set p to 0.25 generate DAGs with an uncomplicated structure. If a disconnected DAG is generated, we add the fewest number of edges to make it connected. For experimentation, we have considered arbitrary task 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 τ_i, calculated according to the definition of L_i (refer to Section 2).

After generating the topology of each DAG of a set, we partition them into two subsets according to the proposed approach, one to the "big" 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. rt-app simulates 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 4), 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 (up) the frequency of the big cluster takes at most $60\ (40)ms$, respectively. On the LITTLE cluster, both the

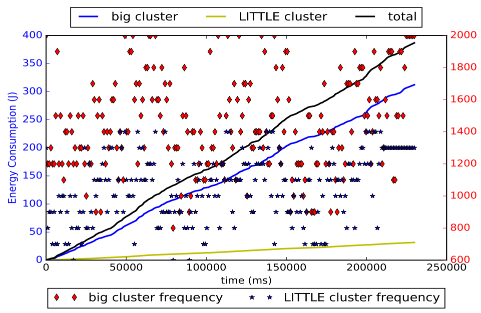


Fig. 4: The energy consumption and the frequency variation of our proposed approach on ODROID XU-3.

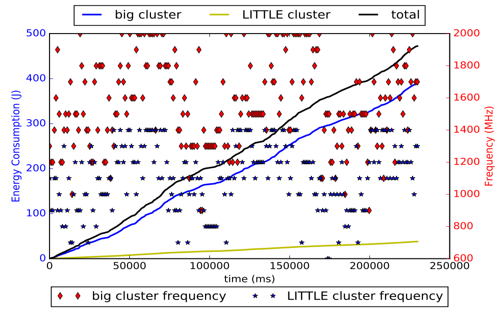


Fig. 5: The energy consumption and the frequency variation of the reference approach on ODROID XU-3.

operation takes at most $15ms$. Due to this delay, the hyper-period of all DAGs becomes large ($230s$, in this experiment). We detail the reasons behind this delay in Appendix B.2.

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 4.

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 3 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

TABLE 3: Summary of experimental results.

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

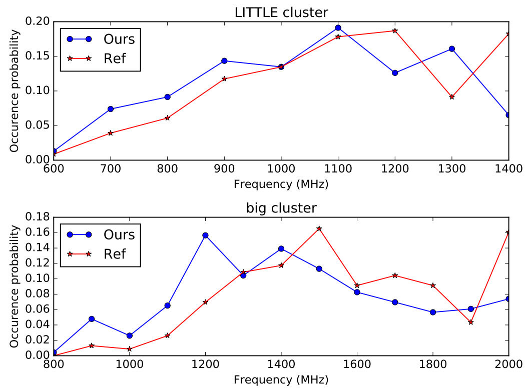


Fig. 6: Frequency occurrence probabilities.

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 fine-grained frequency adjustment at each segment (Section 4.1), and could scale down the frequency if required. Figure 6 presents the frequency occurrence probability of two clusters which is recorded per second by emoxu3. We observe that within the same time interval the reference approach has a higher probability to execute at a higher frequency, while our approach is more likely to execute at the lower frequencies, thus reducing the energy consumption.

Remark 3. *Each heavy DAG ($C_i > T_i$) needs two or more cores while executing and the ODROID XU-3 board contains four cores per cluster. So, in this experimental setup, we can not execute more than four heavy DAGs at a time. Such a restriction is not applicable to the light DAGs ($C_i \leq T_i$). We also consider that a heavy DAG cannot be allocated in multiple clusters.*

6 Simulations

For large-scale evaluation, we perform simulations and compare the results with existing baselines. We generate DAGs using the Erdos-Renyi method (Section 5). We consider two types of task periods; *(a) harmonic periods*, where the task period T_i is enforced to be an integral power of 2. We define T_i as $T_i = 2^\alpha$, where α is the minimum value such that $2^\alpha \geq L_i$, where L_i is the critical path length of τ_i *(b) arbitrary periods*, T_i is determined using Gamma distribution (see Section 5).

We compare our approaches with some existing baselines studied in [12], [17], [33]. Total power consumption by our approach and by these baselines are calculated using Equation (6). As mentioned earlier, [12] 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 [12], when a task is allocated on some cores at any time instant t, we choose the maximum speed among all these cores. We consider [12] 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 [12] 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 [17] 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 [17] 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 [17]). 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, we chose [17] as a baseline because they studied a *GSS* approach for energy minimization. They considered the inter-dependent sequential tasks and their dependencies was represented by a DAG, which is similar to our task model.

We also consider [33] as a baseline because this work considered scheduling a set of independent periodic applications, where each application is modeled as a DAG. They proposed an approach for energy minimization combining the DVFS and the DPM policy. Similar to [12] and [17], the work in [33] considered per-core DVFS.

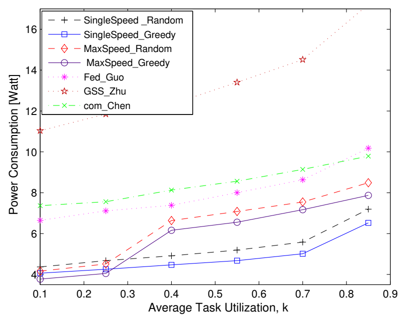
We compare power consumption by varying two parameters for each task: *task periods (utilization)* and *the number of nodes*. We randomly generate 25 sets of DAG tasks and compare the average power consumption.

Notations of Referenced Approaches. For the task partitioning step, either we randomly choose any two and allocate them to the same cluster, or greedily choose the ones with lowest speed as proposed. Regarding speed-profile calculation, there are also two options (Approaches A and B in Section 4.1). Combining these options in two steps lead to four baselines: *MaxSpeed_Greedy*, *SingleSpeed_Greedy*, *MaxSpeed_Random*, *SingleSpeed_Random*. Also, three baselines mentioned above are included for comparison:

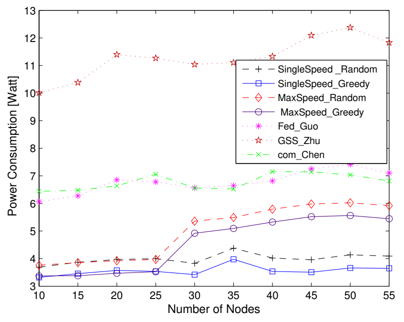
• Federated scheduling with intra-task processor merging [12], denoted by *Fed_Guo*;
• GSS algorithm [17], denoted by *GSS_Zhu*.
• DVFS and DPM combination [33], denoted by *com_Chen*.

6.1 Uniform Heterogeneous Platform with a Continuous Frequency Scheme

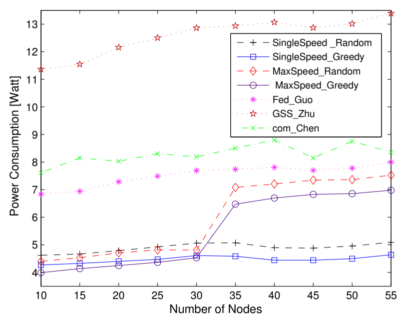
In this section, considering the uniform heterogeneous platform and a continuous frequency scheme, we report the power consumption comparison for different approaches



(a) Under Varying Utilization



(b) Under Varying Number of Nodes (Harmonic Period)



(c) Under Varying Number of Nodes (Arbitrary Period)

Fig. 7: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **continuous** frequency scheme on the **uniform** heterogeneous platform.

mentioned earlier. Under such a platform, different clusters no longer share the same power model and we use the power model described in Equation (6). We present the power comparison results in an identical heterogeneous platform (from [34]) in Appendix A.

6.1.1 *Constrained Deadline Task*

Here, we report the power consumption under the scheme for constrained deadline tasks mentioned in Section 4. 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 τ_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 5. We also use the following equation (according to [12]) to ensure that the value of T_i satisfies the range ($L_i \leq T_i \leq C_i$).

$$T_i = L_i + (1-k)(C_i - L_i) \tag{7}$$

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(a). 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. In this experiment, the number of nodes is fixed to 30. Figure 7(a) shows that *SingleSpeed_Greedy* approach performs better for a higher utilization value. On average, the *SingleSpeed_Greedy* approach leads to a power saving of at least 30.23% and 60.2% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively. 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 reduce energy efficiency (see Figure 3 and Example 1).

Effect of Varying the Numbers of Nodes. Now we vary the number of nodes (10 to 55) (T_i is fixed) and report the average power consumption. We report the average power consumption for harmonic deadline tasks in Figure 7(b) and arbitrary deadline tasks in Figure 7(c). We observe that the power consumption pattern does not change that much, i.e., SingleSpeed_Greedy approach outperforms other approaches especially when the number of nodes (in each DAG) are high, 35 or higher. Specifically, under harmonic task periods, the SingleSpeed_Greedy incurs 40.19% and 65.9% less power on average compared to Fed_Guo and GSS_Zhu; under arbitrary task periods, the savings potential are 33.43% and 61.96%, respectively.

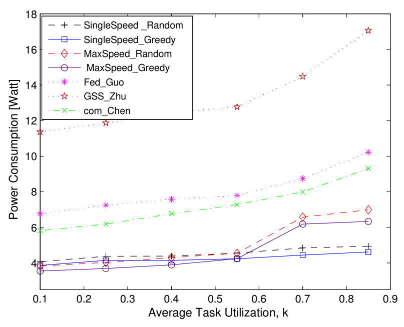
6.1.2 *Implicit Deadline Task*

Effect of Varying Task Periods (Utilization). Using previous setup (Section 6.1.1), We observe that the average energy consumption is directly proportional to the average task utilization. Figure 8(a) shows that *SingleSpeed_Greedy* approach performs better for a higher utilization value and on average, saves at least 35.21% and 62.52% compared to *Fed_Guo* and *GSS_Zhu* approaches, respectively.

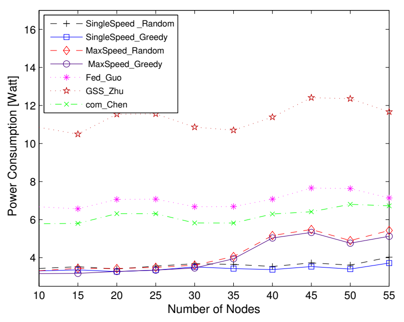
Effect of Varying the Numbers of Nodes. Figure 8(b) and 8(c) report the average power consumption for the harmonic and arbitrary deadline tasks, respectively. We observe that the SingleSpeed_Greedy approach outperforms other approaches when the number of nodes (in each DAG) are high. Under harmonic task periods, the SingleSpeed_Greedy incurs 44.84% and 67.55% less power on average compared to Fed_Guo and GSS_Zhu; under arbitrary task periods, the savings potential are 42.33% and 67.19%, respectively.

6.2 Uniform Heterogeneous Platform With a Discrete Frequency Scheme

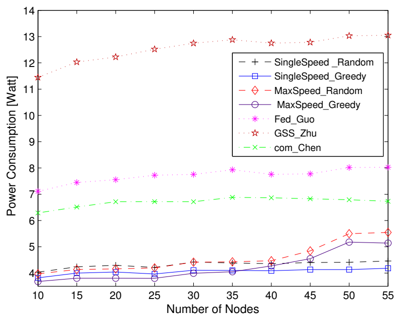
In this section, we report the power consumption comparison for the (previously mentioned) approaches considering the uniform heterogeneous platform and a discrete frequency scheme. Under such a platform, we discretize the frequency using the technique described in Section 4.2.



(a) Under Varying Utilization

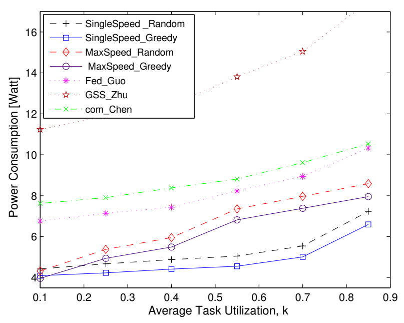


(b) Under Varying Number of Nodes (Harmonic Period)

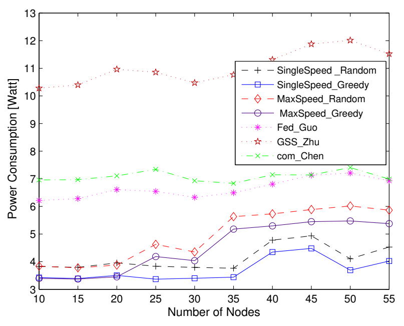


(c) Under Varying Number of Nodes (Arbitrary Period)

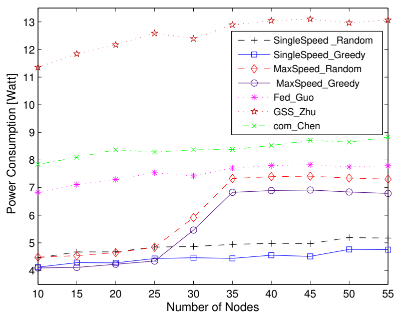
Fig. 8: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **continuous** frequency scheme on the **uniform** heterogeneous platform.



(a) Under Varying Utilization



(b) Under Varying Number of Nodes (Harmonic Period)



(c) Under Varying Number of Nodes (Arbitrary Period)

Fig. 9: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **discrete** frequency scheme on the **uniform** heterogeneous platform.

6.2.1 *Constrained Deadline Task*

Here, we consider the constrained deadline tasks and report their average power consumption by changing two parameters: task period (or utilization) and the number of nodes.

Effect of Varying Task Periods (Utilization). Similar to the Figure 7(a), and 8(a), we observe that the (i) average energy consumption is directly proportional to the average task utilization. (ii) *SingleSpeed_Greedy* approach consumes less power than other approaches (see Figure 9(a)).

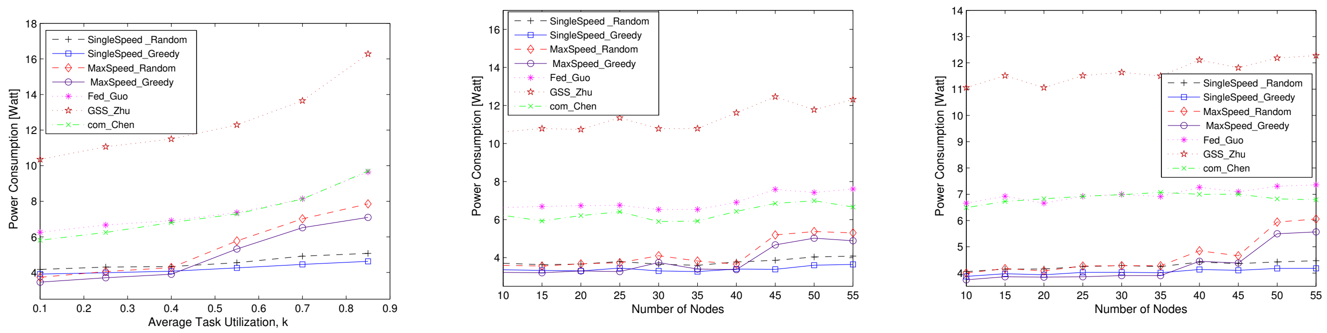
Effect of Varying the Numbers of Nodes. We vary the number of nodes (10 to 55) and report the average power consumption for harmonic (arbitrary) deadline tasks in Figure 9(b) (Figure 9(c)). Similar to the Figure 7 and 8, we observe that the (i) Performance of *SingleSpeed_Random*, *SingleSpeed_Greedy*, *MaxSpeed_Greedy*, and *MaxSpeed_Random* does not vary that much for a small number of nodes (typically 10 to 25) per DAG. (ii) *SingleSpeed_Greedy* approach performs better (i.e., consume less power) than other approaches when the number of nodes per DAG is high.

6.2.2 *Implicit Deadline Task*

Now, we report the average power consumption using the same setup as described in Section 6.2.1, i.e., (a) for a fixed number of nodes (30) per task, change their utilization value, and (b) vary the number of nodes (10 to 55) per task, while keeping T_i fixed. We report the average power consumption in Figure 10. From this figure, we observe that the (i) Performance of *SingleSpeed_Random*, *SingleSpeed_Greedy*, *MaxSpeed_Greedy*, and *MaxSpeed_Random* does not vary that much for a smaller task utilization or when the number of nodes per DAG is small (typically 10 to 35). (ii) *SingleSpeed_Greedy* approach performs better (i.e., consume less power) compared to the other approaches when the number of nodes per DAG is high.

7 Related Work

Much work has been done aimed at energy-efficient scheduling of sequential tasks in a homogeneous multi-core platform (see [11] for a survey). Considering the mixed-criticality task model and varying-speed processors, the works on [23], [35], [36], [37], [38] proposed an approach to handle the energy minimization problem. The work in [27], [39], [40], [41], [42], [43] presented an energy-efficient approach for the heterogeneous platform. Considering the real-time tasks in clustered heterogeneous platforms, the work in [39] studied the partitioned EDF scheduling policy, while [42] proposed an optimal task-core mapping technique that is fully-migrative. Considering the heterogeneous multi-core platform, a two-phase algorithm was proposed by [27]. In the first phase, they proposed a tasks-core allocation approach with the aim of reducing the dynamic energy consumption, while the second phase seeks for a better sleep state to reduce the leakage power consumption.



(a) Under Varying Utilization

(b) Under Varying Number of Nodes (Harmonic Period)

(c) Under Varying Number of Nodes (Arbitrary Period)

Fig. 10: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **discrete** frequency scheme on the **uniform** heterogeneous platform.

A low overhead, DVFS-cum-DPM enabled energy-aware approach, HEALERS, was proposed by [43]. However, none of them considered the intra-task parallelism. Considering a clustered heterogeneous MPSoC platform, a migrative cluster scheduling approach was proposed by [15]. In this approach, run-time migration (within different cores in the same cluster) for a task is allowed to improve resource utilization. The work in [44] studied the technique to utilize the parallelism in a hard real-time streaming application (represented as a Synchronous Data Flow (SDF) graph) in a clustered heterogeneous platform.

Till date, considering both the intra-task parallelization and power minimization has received less attention. A greedy slack stealing algorithm is proposed in [17] that deals with task represented by graphs but did not consider the periodic DAGs. Assuming per-core DVFS, [33] provided the technique to combine DVFS and DPM. Considering the real-time jobs (represented as a DAG) in cloud computing systems and in a heterogeneous multi-core platform, the work in [45], [46] studied a QoS-aware and energy-efficient scheduling strategy. They proposed a scheduling policy that utilizes per-core DVFS. With the aim of improving energy-efficiency in a heterogeneous real-time platform, [47] proposed a combined approach considering the approximate computation and bin packing strategy. [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 [12], [18] is most related to ours. However, these works are significantly different from ours w.r.t the task model, platform, real-time constraints (deadlines), solution techniques, and the evaluation. *First*, the work in [12] considered a simplified model where only one DAG task executes at a time, while the work in [18] extends this work by allowing inter-task processor sharing. However, both of these works assumed that the number of cores are unlimited. *Second*, Both the works in [12], [18] 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 [13]. *Third*, Both of these works have studied only the implicit deadline tasks and did not consider the *constrained* deadline tasks. Hence, the non-negligible idle gaps between the task deadline and its next release remain un-utilized. *Finally*, the evaluations in [12], [18] were done based on simulations without any implementation on a real platform.

8 Conclusion

In this paper, we have studied real-time scheduling of a set of implicit and constrained deadline sporadic DAG tasks. We schedule these tasks on the cluster-based multi-core platforms with the goal of minimizing the CPU 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, i.e., *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, this is the first work that has investigated energy-efficient scheduling of DAGs on clustered multi-core platform. Also, no work considered energy-aware real-time scheduling of constrained deadline DAG tasks.

We have implemented our result on an ODROID XU-3 board 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. In the future, we plan to consider other components that may affect the total power consumption, e.g., cache misses, context switches, I/O usage, etc. We also plan to study the effect of tasks sporadic release patterns (to the overall power consumption) and propose a task reallocation scheme.

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 TPDS*, vol. 25, no. 12, pp. 3242–3252, 2014.

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

[4] 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.

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

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

[7] 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.

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

[9] 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.

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

[11] M. Bambagini, M. Marinoni, H. Aydin, and G. Buttazzo, "Energy-aware scheduling for real-time systems: A survey," *ACM TECS*, vol. 15, no. 1, p. 7, 2016.

[12] 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.

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

[14] 2017, http://www.hardkernel.com/.

[15] 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.

[16] 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.

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

[18] A. Bhuiyan, Z. Guo, A. Saifullah, N. Guan, and H. Xiong, "Energy-efficient real-time scheduling of DAG tasks," *ACM TECS*, vol. 17, no. 5, p. 84, 2018.

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

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

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

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

[23] 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.

[24] 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.

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

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

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

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

[34] Z. Guo, A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, and N. Guan, "Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms," in *RTAS*. IEEE, 2019.

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

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

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

[38] A. Bhuiyan, S. Sruti, Z. Guo, and K. Yang, "Precise scheduling of mixed-criticality tasks by varying processor speed," in *RTNS*. ACM, 2019.

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

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

[41] 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.

[42] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," in *RTSS*. IEEE, 2015.

[43] S. Moulik, R. Devaraj, and A. Sarkar, "Healers: a heterogeneous energy-aware low-overhead real-time scheduler," *IET Computers & Digital Techniques*, vol. 13, no. 6, pp. 470–480, 2019.

[44] J. Spasic, D. Liu, and T. Stefanov, "Energy-efficient mapping of real-time applications on heterogeneous mpsocs using task replication," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2016.

[45] G. L. Stavrinides and H. D. Karatza, "Energy-aware scheduling of real-time workflow applications in clouds utilizing DVFS and approximate computations," in *FiCloud*. IEEE, 2018.

[46] ——, "An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Generation Computer Systems*, vol. 96, pp. 216–226, 2019.

[47] ——, "Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 977–988, 2012.

[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.

[51] 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.

[52] 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.

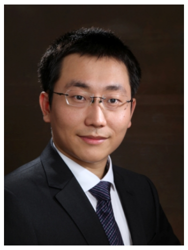
Ashikahmed Bhuiyan is a Ph.D. candidate in the Department of Electrical and Computer Engineering at the University of Central Florida (UCF), under the supervision of Zhishan Guo and Abusayeed Saifullah (from the Wayne state University). He is a member of the Real-Time & Intelligent Systems Lab at UCF. He received his Bachelor of Science degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Bangladesh, in 2013. His research focuses on improving energy efficiency in real-time embedded systems, parallel computing, and mixed-criticality scheduling. He has received the Best Student Paper Award at the 40th IEEE Real-Time Systems Symposium (RTSS 2019).

Di Liu received the B.Eng. and M.Eng. degrees from Northwestern Polytech- nical University, China, in 2007 and 2011, respectively, and the Ph.D. degree from Leiden University, The Netherlands, in 2017. He is currently an Assistant Professor with the School of Software, Yunnan University, China. His research interests include the fields of real-time systems, energy-efficient multicore/many core systems, and cyber-physical systems.

Aamir Khan is an Embedded Systems Engineer at BrainCo Tech in Boston. He received his M.S. in 2018 from the Computer Engineering Department at Missouri University of Science and Technology in Rolla. His areas of interest includes real-time systems and Body-Machine Interface products.

Abusayeed Saifullah is an assistant professor of the Computer Science Department at Wayne State University. He received PhD in Computer Science and Engineering with Turner Dissertation Award from Washington University in St Louis in 2014. His research primarily concerns Internet-of-Things, cyber-physical systems, real-time systems, embedded systems, and low-power wide-area networks. He received 7 Best Paper Awards/Nominations in highly competitive conferences including ACM SenSys (2016 nomination), IEEE RTSS (2019, 2014, 2011), IEEE ICII (2018), and IEEE RTAS (2012 nomination). He also received multiple young investigator awards including the CAREER award (2019) and the CRII award (2016) of the National Science Foundation (NSF). He is serving as the program chair of IEEE ICESS 2020, served as a track chair of IEEE ICCCN 2019 and as a program committee member for various conferences including ACM SenSys, IEEE RTSS, ACM/IEEE IoTDI, IEEE RTAS, ACM/IEEE ICCPS, ACM MobiHoc, IEEE INFOCOM, EWSN, and ACM IWQoS. He also served as a guest editor of IEEE Transactions on Industrial Informatics, and is currently an editor of Elsevier Pervasive and Mobile Computing journal.

Nan Guan is currently an Assistant Professor in Department of Computing, The Hong Kong Polytechnic University. He got his PhD in Uppsala University, Sweden and worked in Northeastern University, China before joining The Hong Kong Polytechnic University. His research interests include the design and analysis of real-time systems, embedded systems, cyber-physical systems and Internet-of-Things (IoT) systems. He received the EDAA Outstanding Dissertation Award in 2014, the CCF Outstanding Dissertation Award in 2013, the Best Paper Award of IEEE RTSS in 2009, the Best Paper Award of DATE in 2013, the Best Paper Award of ACM e-Energy 2018 and the Best Paper Award of IEEE ISORC 2019. He served as the TPC co-Chair of EMSOFT 2015, ICESS 2017, SETTA 2019, the TPC Track Chair of RTAS 2018.



Zhishan Guo is an assistant professor in the Department of Electrical and Computer Engineering at The University of Central Florida. He received his B.E. degree (with honor) in computer science and technology from Tsinghua University, Beijing, China, in 2009, his M.Phil. degree in mechanical and automation engineering from the Chinese University of Hong Kong in 2011, and his Ph.D. degree in computer science at the University of North Carolina from the University of North Carolina at Chapel Hill in 2016. His research and teaching interests include real-time scheduling, cyber-physical systems, and neural networks and their applications.

Appendix A
Identical Heterogeneous Platform With a Continuous Frequency Scheme

In this section, we report the power consumption comparison considering the identical heterogeneous platform and a continuous frequency scheme. Under the identical heterogeneous platform, both the ("big" and the "LITTLE") clusters share the same power model as described in Equation (1).

A.1 Constrained Deadline Task

Similar to the settings described in Section 6.1.1, we consider the constrained deadline tasks and report their average power consumption under varying task period (or utilization) and the number of nodes.

Effect of Varying Task Periods (utilization). Here, the number of nodes is fixed to 30. We observe the similar trend, i.e., average energy consumption is directly proportional to the average task utilization. Figure 11(a) 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.

Effect of Varying the Numbers of Nodes. Here we vary the number of nodes (T_i remains fixed) and report the average power consumption. We consider both harmonic and arbitrary periods (reported in Figures 11(b) and 11(c)). 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.

A.2 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.

Effect of Varying Task Periods (Utilization). In this experiment, we fix the number of nodes to 30. We vary the task period using Equation (7) and report the average power consumption in Figure 12(a). 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 12(a) 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.

Effect of Varying the Numbers of Nodes. Now we measure the average power consumption by varying the number of nodes with fixed T_i. We randomly generate 100 tasks with harmonic and arbitrary deadline. We report the average power consumption in Figure 12(b) (12(c)) for harmonic (arbitrary) deadline tasks. Again, we observe similar improvements in power consumption, i.e., choosing a single speed over the whole deadline outperforms other approaches. Instead of 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.

Appendix B
Discussions: Assumptions and Applicability

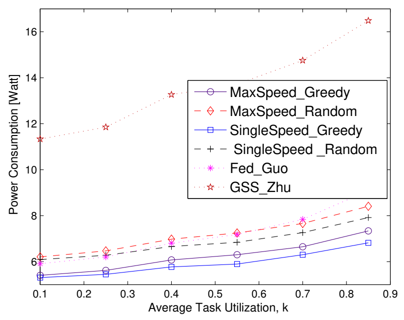
We have mentioned the assumptions adopted in this paper. The first two subsections in this section discuss the validity of these assumptions, their impacts and potential solutions to overcome these impacts. Then, in Subsection B.2 we detail the reasons behind the measurement overheads (see Section 5) and the applicability of our proposed approaches.

B.1 Assumptions Behind the Power Model

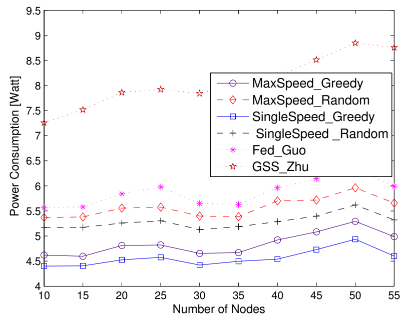
Components Behind the Overall Power Consumption. While some other factors such as cache miss, bus accesses, context-switches, and I/O usage also affect the power consumption, CPU power consumption is one of the major contributors to the overall power consumption. Power consumption may largely be dominated 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 [51]). 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 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] [33]. 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 [12]), 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 [52], which can be larger than many task periods in avionics. As triggering mode switches becomes more energy consuming in general

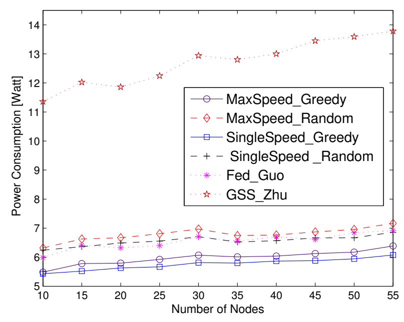
3. BET is the minimum duration for the processor to stay at the sleep mode.



(a) Under Varying Utilization

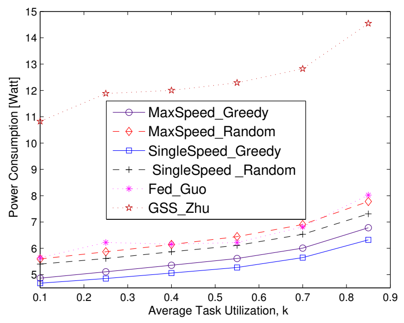


(b) Under Varying Number of Nodes (Harmonic Period)

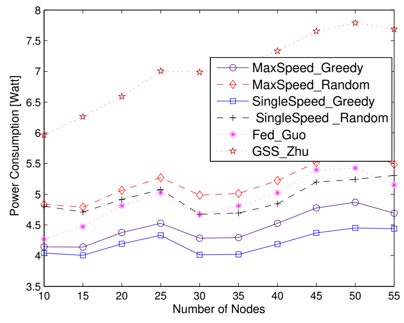


(c) Under Varying Number of Nodes (Arbitrary Period)

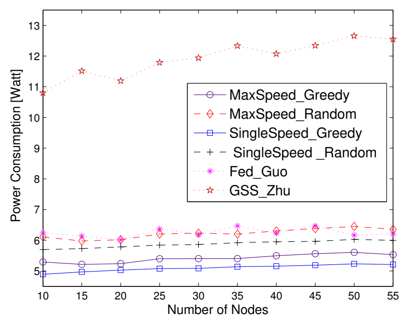
Fig. 11: Power consumption comparison between different approaches for the **constrained** deadline tasks considering a **continuous** frequency scheme on the **identical** heterogeneous platform.



(a) Under Varying Utilization



(b) Under Varying Number of Nodes (Harmonic Period)



(c) Under Varying Number of Nodes (Arbitrary Period)

Fig. 12: Power consumption comparison between different approaches for the **implicit** deadline tasks considering a **continuous** frequency scheme on the **identical** heterogeneous platform.

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.

B.2 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 5). we use cpufreq-set module to change the system's frequency, and this module accounts for microsecond-scale transition delay (usually 100-200μs), 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 4.1, 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 6) 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 4.1. 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.