



CPU Energy-Aware Parallel Real-Time Scheduling

Abusayeed Saifullah 

Wayne State University, USA
saifullah@wayne.edu

Sezana Fahmida 

Wayne State University, USA
fahmida.sezana@wayne.edu

Venkata P. Modekurthy 

Wayne State University, USA
modekurthy@wayne.edu

Nathan Fisher 

Wayne State University, USA
fishern@wayne.edu

Zhishan Guo 

University of Central Florida, USA
zsguo@ucf.edu

Abstract

Both energy-efficiency and real-time performance are critical requirements in many embedded systems applications such as self-driving car, robotic system, disaster response, and security/safety control. These systems entail a myriad of real-time tasks, where each task itself is a parallel task that can utilize multiple computing units at the same time. Driven by the increasing demand for parallel tasks, multi-core embedded processors are inevitably evolving to many-core. Existing work on real-time parallel tasks mostly focused on real-time scheduling without addressing energy consumption. In this paper, we address hard real-time scheduling of parallel tasks while minimizing their CPU energy consumption on multicore embedded systems. Each task is represented as a directed acyclic graph (DAG) with nodes indicating different threads of execution and edges indicating their dependencies. Our technique is to determine the execution speeds of the nodes of the DAGs to minimize the overall energy consumption while meeting all task deadlines. It incorporates a frequency optimization engine and the dynamic voltage and frequency scaling (DVFS) scheme into the classical real-time scheduling policies (both federated and global) and makes them energy-aware. The contributions of this paper thus include the first energy-aware online federated scheduling and also the first energy-aware global scheduling of DAGs. Evaluation using synthetic workload through simulation shows that our energy-aware real-time scheduling policies can achieve up to 68% energy-saving compared to classical (energy-unaware) policies. We have also performed a proof of concept system evaluation using physical hardware demonstrating the energy efficiency through our proposed approach.

2012 ACM Subject Classification Computer systems organization → Real-time system specification

Keywords and phrases Real-time scheduling, multicore, energy-efficiency, embedded systems.

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2020.2

Acknowledgements This work was supported by National Science Foundation through grants CAREER-1846126, CNS-1618185, IIS-1724227, and CNS-1850851, by Wayne State University through Rumble Fellowship, and by University of Central Florida through a startup grant.

1 Introduction

Energy-efficiency is an important requirement in embedded systems (e.g., mobile phones, tablets, cars, robots, and computerized numerical controls) as they rely on limited or unreliable sources of energy such as batteries or energy harvesters. Embedded systems are



© A. Saifullah, S. Fahmida, V. Modekurthy, N. Fisher and Z. Guo;
licensed under Creative Commons License CC-BY

32nd Euromicro Conference on Real-Time Systems (ECRTS 2020).

Editor: Marcus Völp; Article No. 2; pp. 2:1–2:24



Leibniz International Proceedings in Informatics

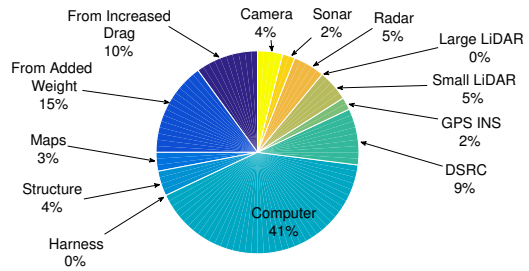
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

used in all aspects of human life and industries and are now being sparked by billions of devices through the evolution of Internet of Things. Energy consumption by these billions of devices can be significant. It is projected that computers and cell phones will consume 14% of worldwide power by 2020 [2]. A recent study using the Ford Fusion autonomy system has revealed that 41% energy is consumed by the computing platform of a self-driving car (Figure 1) [1, 3, 31]. It is estimated that up to 80% of the total energy consumption of an embedded system is due to software-related activities [52]. In such systems, a processor consumes a significant share of their power consumption. For example, in spacecraft, the RAD750 processor draws almost 33%-50% share of the power consumption [4]. Real-time performance is another critical requirement in many embedded systems applications such as self-driving car, advanced robotic system, disaster response, and surveillance systems. Many involve mission critical applications that require a predictable real-time system behavior but battery re-charging during the mission may not be possible.

With the evolution of various computation-intensive systems (e.g., cloud computing, self-driving car), today's real-time systems evolve in the form of many parallel tasks. For example, a self-driving car [41] entails a myriad of real-time tasks such as motion planning, sensor fusion, computer vision, and decision making system that exhibit *intra-task parallelism*, where each task itself can utilize multiple computing units simultaneously. For example, the decision making system collects massive amounts of data from various types of sensors and processes them in parallel. Driven by the increasing demand for parallel tasks, multicore embedded processors are inevitably evolving to many-core (e.g., Intel's 48-core SCC chip [5], TILERA's 100-core TILE-Gx100 [9], 248-core PC205 of picoChip [7]). Multicore offers opportunities for energy minimization. For example, the energy consumption of executing a certain workload equally distributed in two cores is significantly less than that of executing it in one core at double speed [56]. As the energy-related benefits resulting from Moore's law are leveling off, software-level techniques need to be exploited to reduce the power consumption on multicores. While real-time scheduling for parallel tasks on multicores has been widely studied recently, existing work mostly focused on scheduling *without* addressing energy consumption [15, 16, 62, 30, 64, 63, 14, 46, 45, 21, 40, 10, 42, 54].

In this paper, we address hard real-time scheduling of parallel tasks while minimizing their CPU energy consumption on multicore embedded systems. We consider periodic parallel tasks with deadlines, where each task is represented by a directed acyclic graph (DAG). In a DAG, the *nodes* stand for different threads of execution while the *edges* represent their dependencies. DAG is the most general model of deterministic parallel tasks [63]. Although our proposed techniques can be applied to any multicore platform for reduced energy consumption, we focus on embedded systems as their energy efficiency is of more general need. Energy-aware real-time scheduling is challenging in general due to the complicated (nonlinear) relationship between frequency, energy consumption, and execution time of a task.

Although much work exists on energy-aware real-time scheduling on multiprocessor [57, 56, 53, 20, 39, 24, 22, 61, 12, 72, 23, 26, 48, 49, 44, 66, 68], it has considered only sequential task models. There is a recent study on energy-aware parallel real-time scheduling that considered overly simplified federated scheduling of DAGs where at any time *only one*



■ **Figure 1** Energy consumption of the autonomy system in a self-driving car (Ford Fusion)[1].

task can run and the number of cores *cannot be pre-fixed* as an input [36, 19]. It adopted a table-driven scheduling where the entire schedule (up to the hyper-period which is of *exponential size* in task periods) needs to be created in-advance. In contrast, we consider scheduling many recurrent tasks exploiting both intra- and inter-task parallelism on a finite set of cores. We adopt *online* scheduling so that tasks can be scheduled as they arrive. We consider both *global* and *federated* scheduling. For global scheduling, we consider *both* dynamic priority and fixed priority scheduling.

In this paper, we focus on minimizing CPU energy consumption by means of *dynamic voltage and frequency scaling (DVFS)*. DVFS is a commonly-used power-management technique where the clock frequency of a processor is decreased to allow a reduction in the supply voltage. This reduces power consumption, which leads to reduction in the energy required for a computation. Many AMD and Intel processors support per-core DVFS for flexible power control (e.g., AMD family 10h processors, Intel Haswell processors) [33]. Our approach is to regulate the frequencies across the cores and across different execution parts of the same task for minimizing overall energy consumption. It incorporates an optimization engine and DVFS into the traditional real-time scheduling policies (e.g., earliest deadline first, deadline monotonic, and federated scheduling) and makes them energy-aware. Specifically, once the optimal execution speeds of the nodes of all DAGs are determined, we adopt these classical real-time scheduling policies. Note that these classical policies are online, highly efficient in real-time performance, but are not energy-aware. In our approach, when the processor frequencies are adjusted based on the nodes' speed assignments, the adopted real-time scheduling policy leads to energy minimization.

Specifically, we make the following new contributions.

- For federated scheduling, the objective becomes to assign a certain number of cores to each task and execution speeds to different nodes so that overall energy consumption is minimized. We first formulate this as a constrained non-linear optimization problem whose solution becomes challenging due to the discrete nature of solution space. We propose a continuous convexification approach through a new core allocation scheme in federated scheduling while retaining its theoretical real-time performance bound. This is the **first** energy-aware federated scheduling of multiple tasks which does not rely on infinite number of cores or table-driven scheduling.
- We formulate energy-aware real-time global scheduling of DAGs as a convex optimization problem for both dynamic and fixed priority. The objective is to determine the execution speeds of the DAG nodes to minimize overall energy consumption while meeting all deadlines. An optimal solution is achieved in polynomial time. Upon adjusting the processor frequencies online, the overall energy consumption is minimized under classical real-time scheduling policies. This is the **first** energy-aware global scheduling of DAGs.
- We have evaluated our energy-aware real-time scheduling policies using synthetic workloads through simulations. The results show that our approach can achieve up to 68% energy-saving compared to classical (energy-unaware) policies. We have also performed a proof of concept system evaluation using multiple ODROID XU-4 boards [55] demonstrating the energy efficiency through our proposed approach.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes our parallel tasks, multicore, and energy consumption model and some background. Section 4 presents our energy-aware federated scheduling. Section 5 presents our energy-aware global scheduling. Section 6 presents simulation results. Section 7 presents the proof of concept system evaluation. Section 8 concludes our paper with future research directions.

2 Related Work

While the works proposed in [57, 56, 53, 20, 39, 24, 22, 61, 12, 72, 23, 26, 48, 49, 44, 66, 68] studied energy-aware real-time scheduling on multiprocessor, they considered sequential tasks. A detailed review of these works can be found in a recent survey in [13]. While a parallel task can execute on multiple cores simultaneously, a *sequential task* can execute only on one core at a time. Therefore, parallel tasks scheduling is significantly different from sequential tasks. Parallel scheduling of DAGs is highly challenging as the dependencies among the nodes need to be considered while these are absent in sequential tasks.

Existing work on parallel real-time scheduling concentrates mostly on scheduling policies or analysis and has not focused on energy-consumption [15, 16, 62, 30, 64, 63, 14, 46, 45, 21, 40, 10, 42, 54, 67, 18]. Energy-aware scheduling of parallel tasks on multicore was studied in [47] for a very simplified model where a task has a fixed number of parallel threads, and the tasks are not recurrent or real-time [47]. The energy benefit of multicore scheduling was studied in [59, 43, 71, 34, 59] by running parallel threads. Energy-aware gang scheduling was studied in [58]. Gang scheduling involves a very special form of parallelism where all parallel threads of the same task use processors in the same window (i.e., they start and stop using the processors at the exact same time). A slack stealing based scheduling was studied in [74, 75] for energy minimization of an application consisting of inter-dependent sequential tasks. While those dependencies among the tasks were represented by a DAG, the model consists of a single DAG and does not consider recurrent tasks. A similar model was studied in [70, 69] for non-real-time power aware cluster computing. We consider energy-aware real-time scheduling of multiple/many recurrent DAGs.

An energy-efficient clustering of parallel tasks was studied in [35, 17]. It is only suitable for clustered multicore platform where all tasks assigned to the same cluster of cores have to run at the same speed (as all cores in the same cluster run at the same speed). Recently, a simplified model of energy-aware federated real-time scheduling of DAGs was studied where at any time **only one task can run** and the number of cores **cannot be pre-fixed** as an input [36, 19]. It adopted a **table-driven scheduling**, where the entire schedule (up to the hyper-period which is of **exponential size** in task periods) is created in-advance. *In contrast, this paper (1) considers scheduling multiple/many recurrent DAGs exploiting both intra- and inter-task parallelism on a given finite set of cores; (2) adopts online scheduling so that tasks can be scheduled as they arrive; (3) considers both global and federated scheduling with guaranteed performance bounds of the algorithms; (4) considers both dynamic priority and fixed priority under global scheduling.*

3 System Model and Background

3.1 Parallel Task Model

We consider a task set $\tau = \{\tau_1, \tau_2, \dots, \tau_n\}$ of n periodic parallel tasks to be scheduled on a multicore platform consisting of m cores. Each task $\tau_i, 1 \leq i \leq n$, is represented as a DAG, where a *node* is a thread of execution (execution requirement), and the *edges* between the nodes indicate the dependencies between the associated threads of execution.

A node in τ_i is denoted by $W_i^j, 1 \leq j \leq n_i$, where n_i is the total number of nodes in τ_i . The *worst-case execution requirement (WCER)* (i.e. *the total number of CPU cycles needed in the worst-case*) of node W_i^j is denoted by c_i^j . This means that the node has *worst-case execution time (WCET)* of c_i^j on a unit-speed processor. Note that the actual execution requirement (actual number of execution cycles) of a node W_i^j can vary in practice due to

many complex factors such as data inputs, hardware features, and execution contexts but it will never exceed its WCER c_i^j . A directed edge from node W_i^j to node W_i^k , denoted as $W_i^j \rightarrow W_i^k$, implies that, for each DAG job released by task τ_i , the execution of W_i^k cannot start until W_i^j finishes. W_i^j , in this case, is called a *parent* of W_i^k , while W_i^k is its *child*. A node may have 0 or more parents or children, and can start execution only after all of its parents have finished execution. A node is *ready* to be executed as soon as all of its parents have been executed. A node having no parent is called a *source* while a node having no child is a *sink*. A DAG may have multiple sources or sinks.

The WCER (i.e., *worst-case work*) C_i of task τ_i is the sum of the WCERs of all nodes in τ_i ; that is, $C_i = \sum_{j=1}^{n_i} c_i^j$. Thus, C_i is the WCET of τ_i if it was executing on a single processor of speed 1. For task τ_i , the *critical path length*, denoted by L_i , is the sum of execution times of the nodes on a critical path. A *critical path* is a directed path that has the maximum execution time among all other paths in DAG τ_i . Thus, L_i is the *minimum execution time* of τ_i even when it is assigned an infinite number of unit-speed cores. Figure 2 illustrates a parallel task τ_i represented as a DAG with $n_i = 10$ nodes.

The *period* of task τ_i is denoted by T_i . Every instance of a task is called a *job*. Each task τ_i thus releases an infinite sequence of jobs, with T_i being the time separation between two consecutive jobs. The (relative) deadline D_i of each task τ_i is considered *implicit*, i.e., $D_i = T_i$. That is, each job of a task must finish before its next job releases. Since L_i is the minimum execution time of task τ_i even on a machine with an infinite number of cores, the condition $T_i \geq L_i$ must hold for τ_i to be schedulable (i.e. to meet its deadline). A task set is *schedulable* when all tasks in the set meet their deadlines.

The *utilization* u_i of a task τ_i , and the *total utilization* $u_{\text{sum}}(\tau)$ for task set τ of n tasks are defined as

$$u_i = \frac{C_i}{T_i}; \quad u_{\text{sum}}(\tau) = \sum_{i=1}^n u_i = \sum_{i=1}^n \frac{C_i}{T_i}$$

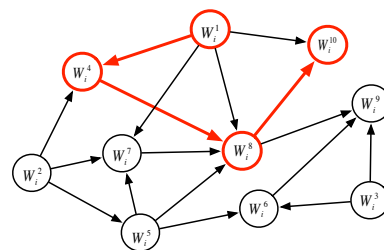
If the total utilization u_{sum} is greater than m , then no algorithm can schedule τ on m identical unit-speed processor cores.

3.2 Power/Energy Model

We consider the following widely used energy consumption model of processor [12, 73, 72, 29, 56, 57, 36]. Assuming continuous frequency scheme, let $s(t)$ denote the main frequency (speed) of a processor at time t . Then its power consumption $P(s)$ can be modeled as:

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

where P_s denotes the *static power consumption* which is introduced due to the leakage current and $P_d(s)$ is the *active power consumption*. $P_d(s)$ is introduced due to capacitor charging and discharging during processor activity, and it depends on the processor frequency.



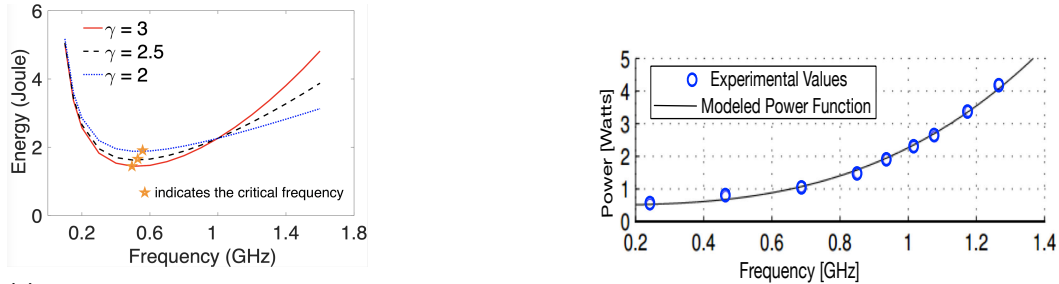
■ **Figure 2** A parallel task τ_i represented as a DAG of 10 nodes: nodes W_i^1 , W_i^2 , and W_i^3 are the sources while W_i^9 and W_i^{10} are the sinks. Suppose for example $c_i^1 = 4$, $c_i^2 = 2$, $c_i^3 = 4$, $c_i^4 = 5$, $c_i^5 = 3$, $c_i^6 = 4$, $c_i^7 = 2$, $c_i^8 = 4$, $c_i^9 = 1$, $c_i^{10} = 1$. Then, $C_i = 30$. $L_i = 14$, and the $W_i^1 \rightarrow W_i^4 \rightarrow W_i^8 \rightarrow W_i^{10}$ (thick red-colored) is a critical path on unit-speed cores.

$P_d(s)$ can be represented as αs^γ where the constant $\alpha > 0$ depends on the effective switching capacitance [56], $\gamma \in [2, 3]$ is a fixed parameter determined by the hardware, and $\beta > 0$ represents the leakage power (i.e., the static part of power consumption whenever a processor remains on or idle). Power consumption is a convex-increasing function of the processor frequency. It is possible to reduce $P_d(s)$ by reducing the processor frequency through DVFS.

The energy consumption in any given period $[t_1, t_2]$ can be calculated as $E = \int_{t_1}^{t_2} P(s) dt$, which is almost close to the actual CPU energy consumption of many known systems. Specifically, given a fixed amount of workload C to be executed on a speed- s processor, the total energy consumption is the integral of power over the period of length C/s ; i.e.,

$$E(C, s) = (\beta + \alpha s^\gamma)(C/s) = \beta C/s + \alpha C s^{\gamma-1} \quad (2)$$

Figure 3a illustrates how different values of γ and processor speed s may affect the total energy consumption to complete a certain amount of computation. In most modern processors, execution at a frequency much lower than the *critical frequencies* [56] (the highlighted most energy efficient speed in the figure) is energy inefficient as leakage power becomes the major “contribution”. This model was shown to be quite realistic [57]. Figure 3b compares the original power consumption and the power model in Equation (1) as studied in [37].



(a) Energy consumption for executing a job with 10^9 computation cycles for various γ , where $\alpha = 1.76 \text{ Watts/GHz}^\gamma$, $\beta = 0.5 \text{ Watts}$.

(b) Comparison of the power model (Eq. (2)) with experimental results in [37]. Here, $\alpha = 1.76 \text{ Watts/GHz}^3$, $\gamma = 3$, and $\beta = 0.5 \text{ Watts}$.

■ **Figure 3** CPU energy consumption model

While we consider a continuous frequency scaling of the processors, our approach is applicable to the systems with discrete frequency levels as well. Specifically, any frequency in our continuous frequency scheme can be rounded up to achieve the discrete frequency level. In fact, most current processors have quite fine-grained steps to scale frequency. For example, the ODROID XU board (used in our system experiment) has a frequency range of 0.2 – 1.4GHz for LITTLE cores and 0.2 – 2GHz for big cores, with a scale step of 0.1GHz. With such fine-grained steps, the energy consumption modeled for continuous frequency scheme becomes very close to actual scenario for modern discrete frequency processors.

3.3 Global and Federated Scheduling Overview

We consider *preemptive scheduling*, where a task with higher priority can preempt an executing task that has lower priority. Non-preemptive scheduling is less preferred for time critical systems (partially) because of its poor responsiveness due to blocking and will not be explored in this paper. We shall develop both federated and global preemptive real-time scheduling of parallel tasks for energy minimization. Under global scheduling, we shall consider classical real-time scheduling policies such as *Earliest Deadline First (EDF)* and *Deadline Monotonic (DM)*, and make them energy-aware. For federated scheduling, we shall consider the policy introduced in [46] which is described below.

Federated scheduling was first introduced in [46] as a generalized approach of partitioned scheduling for parallel tasks which has later been widely used in the literature for real-time parallel scheduling. In **federated scheduling**, a task is classified as a **high-utilization task** if its utilization ≥ 1 , or as a **low-utilization task** otherwise. Each high-utilization task is allocated a dedicated cluster (set) of cores. A multiprocessor scheduling algorithm is used to schedule all low-utilization tasks (each having utilization < 1), each of which is run sequentially, on a shared cluster composed of the remaining cores. Given a task set τ , the federated scheduling algorithm works as follows: First, tasks are divided into two disjoint sets: τ_{high} contains all high-utilization tasks – tasks with worst-case utilization at least one ($u_i \geq 1$), and τ_{low} contains all the remaining low-utilization tasks. Consider a high-utilization task τ_i with WCER C_i , worst-case critical-path length L_i , and deadline D_i ($= T_i$). We assign m_i dedicated cores to τ_i where

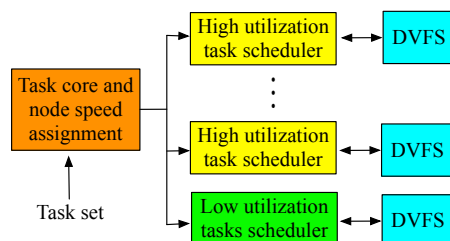
$$m_i = \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil.$$

We use $m_{\text{high}} = \sum_{\tau_i \in \tau_{\text{high}}} m_i$ to denote the total number of cores assigned to high-utilization tasks τ_{high} . We assign the remaining cores to all low-utilization tasks τ_{low} , denoted as $m_{\text{low}} = m - m_{\text{high}}$. The scheduling algorithm admits the task set τ , if $m_{\text{low}} \geq 2 \sum_{\tau_i \in \tau_{\text{low}}} u_i$.

After a valid core allocation, runtime scheduling proceeds as follows: (i) Any greedy (work-conserving) parallel scheduler can be used to schedule a high-utilization task τ_i on its assigned m_i cores. Informally, a **greedy scheduler** is one that never keeps a core idle if some node is ready to execute. (ii) Low-utilization tasks are treated and executed as though they are sequential tasks and any multiprocessor scheduling algorithm (such as partitioned EDF [50], or various rate-monotonic schedulers [11]) with a utilization bound of at most $\frac{1}{2}$ can be used to schedule them on m_{low} cores.

4 Energy-Aware Federated Scheduling

We first describe our approach for federated scheduling of real-time parallel tasks for minimizing their energy consumption on multicore. Since energy consumption of the cores is a function of their frequencies (speeds), our approach is to regulate the frequencies (speed) across the cores and across different nodes of the same task for minimizing overall energy consumption. For federated scheduling, we also have to determine a cluster (set) of cores on which a task will execute. We design our algorithms considering WCER of the nodes considering hard real-time systems that need both real-time guarantee and energy efficiency. Considering WCER, a node is executed at a uniform speed but different nodes can run at different speeds in our approach. Our approach incorporates an optimization engine and DVFS into the federated scheduling. Specifically, we first determine a cluster (set) of cores on which a task will execute and assign execution speeds to different nodes of all tasks. We then incorporate DVFS into each cluster that will adjust the processor frequency during scheduling according to the assigned speeds of the nodes. Once the execution speeds of the nodes and the number of cores of all DAGs are determined, and the processor frequencies are adjusted based on the nodes' speed assignments, the adopted scheduling policy leads to energy minimization. Our federated scheduling uses the policy presented in Section 3.3 and makes it energy-aware. The scheduling



■ **Figure 4** Energy-aware federated scheduling framework

is done after assigning the node execution speeds (i.e. the frequencies at which the cores will run when executing the nodes) and the number of cores of each high utilization task and for the low-utilization ones. Figure 4 shows our energy-aware federated scheduling framework.

4.1 Frequency and Core Assignment Formulation

Our objective is to assign cores to each DAG and determine execution speeds for different nodes of the DAG to minimize overall energy consumption. Such a speed and core assignment must guarantee that all DAGs remain schedulable when the federated policy is applied for their scheduling. Hence, we apply existing highly efficient schedulability conditions as constraints based on processor **capacity augmentation** analysis [46] for DAGs on multicore. Considering overall energy consumption as the objective and using existing schedulability conditions based on processor capacity augmentation analysis, we formulate energy-aware real-time federated scheduling of DAGs as a constrained non-linear optimization problem.

► **Definition 1.** [46] *A scheduling algorithm \mathbb{S} with **capacity augmentation bound** b , $b \geq 1$, can always schedule a task set τ with total utilization of u_{sum} on m cores of speed b as long as τ satisfies two conditions on unit speed cores: (1) $u_{sum} \leq m$; and (2) $L_i \leq D_i, \forall \tau_i$.*

To formulate the problem, we first derive an expression to represent our objective (overall energy consumption) based on the energy model. By Equation (2), the total energy consumption for running node W_i^j at speed s_i^j becomes

$$E(c_i^j, s_i^j) = \left(\beta + \alpha (s_i^j)^\gamma \right) (c_i^j / s_i^j) = \beta c_i^j / s_i^j + \alpha c_i^j (s_i^j)^{\gamma-1}$$

Thus total energy consumption by one job of task τ_i becomes $\sum_{j=1}^{n_i} E(c_i^j, s_i^j)$. To consider overall energy-consumption, we consider a complete schedule up to the hyper-period, denoted by H , of the tasks. Thus the overall energy consumption by task set τ is given by

$$\sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j), \quad \text{where } w_i = \frac{H}{T_i}$$

which becomes our objective (4). The actual energy consumption is also affected by a number of other issues such as frequency switching, turn on/off overhead (see Section 4.4) that are not considered in the above objective. As shown before in Figure 3b, this objective can still represent a close approximation of the actual energy consumption of CPU. We want to minimize this objective while ensuring the schedulability under federated scheduling policy. Now we determine the constraints that must guarantee the schedulability under federated scheduling. The following remark plays a key role in establishing our constraints.

► **Remark 2.** A scheduler \mathbb{S} has a **capacity augmentation bound** of b if it can schedule any task set τ on m cores which satisfies the following two conditions [46]:

- **Condition 1:** The total utilization of τ is at most $\frac{m}{b}$.
- **Condition 2:** For each task $\tau_i \in \tau$ the worst-case critical-path length L_i (execution time of τ_i on an infinite number of cores) is at most $\frac{1}{b}$ fraction of its deadline.

The federated scheduling that we consider (described in Section 3.3) has a capacity augmentation bound of 2 [46]. Using this bound, we have to incorporate the two conditions of Remark (2) as constraints. When node W_i^j of task τ_i is assigned speed s_i^j , where $1 \leq j \leq n_i, 1 \leq i \leq n$, the total utilization can be expressed as $\sum_{i=1}^n (\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}) / T_i$. Thus, the above **Condition 1** for schedulability is expressed as constraint (5) in our formulation.

Assigning different speeds to different nodes may change a task's critical path. Let $\Phi_i(s_i)$ be the set of nodes on a critical path if the nodes of τ_i are assigned speeds $s_i = \{s_i^1, s_i^2, \dots, s_i^j, \dots, s_i^{n_i}\}$. The critical path length $L_i(s_i)$ under this speed is given by

$$L_i(s_i) = \sum_{W_i^j \in \Phi_i(s_i)} \frac{c_i^j}{s_i^j} \quad (3)$$

Hence, **Condition 2** for schedulability is expressed as constraint (6) in our formulation.

Since assigning various speeds affects WCET and L_i , it affects the number of cores m_i assigned to each high utilization task τ_i and the total cores m_{low} assigned to all low utilization tasks. Hence, in addition to the constraints defined above, we have to maintain the following constraint for each speed assignment in federated scheduling as given in Section 3.3.

$$\begin{aligned} m_{\text{low}} = m - m_{\text{high}} &\geq 2 \sum_{\tau_i \in \tau_{\text{low}}} u_i \\ \Leftrightarrow m_{\text{high}} + 2 \sum_{\tau_i \in \tau_{\text{low}}} u_i &\leq m \Leftrightarrow \sum_{\tau_i \in \tau_{\text{high}}} m_i + 2 \sum_{\tau_i \in \tau_{\text{low}}} u_i \leq m \\ \Leftrightarrow \sum_{\tau_i \in \tau_{\text{high}}} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil + 2 \sum_{\tau_i \in \tau_{\text{low}}} \frac{C_i}{T_i} &\leq m \\ \Leftrightarrow \sum_{\forall \tau_i \text{ with } \frac{C_i}{T_i} \geq 1} \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil + 2 \sum_{\forall \tau_i \text{ with } \frac{C_i}{T_i} < 1} \frac{C_i}{T_i} &\leq m \end{aligned}$$

Considering node W_i^j of task τ_i is assigned speed s_i^j , where $1 \leq j \leq n_i, 1 \leq i \leq n$, we write the above condition as constraint (7) in our formulation. Thus our objective is to determine speeds $s_i = \{s_i^1, s_i^2, \dots, s_i^j, \dots, s_i^{n_i}\}, \forall \tau_i$ to

$$\text{Minimize } \sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j) \quad (4)$$

$$\text{subject to } \sum_{i=1}^n \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}}{T_i} \leq \frac{m}{2} \quad (5)$$

$$L_i(s_i) \leq \frac{T_i}{2}, \forall \tau_i \quad (6)$$

$$\begin{aligned} &\sum_{\forall \tau_i \text{ with } \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} \geq 1} \left\lceil \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j} - L_i(s_i)}{T_i - L_i(s_i)} \right\rceil + \\ &2 * \sum_{\forall \tau_i \text{ with } \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} < 1} \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} \leq m \quad (7) \end{aligned}$$

It is worth noting that an optimal solution of the above formulation will not select any speed lower than critical speed as the solution then is non-optimal (a scenario where the leakage power dominates). As can be seen from Figure 3a, an optimization technique may choose a node's speed above the critical speed of the core to meet deadline i.e. to meet constraints (5), (6), and (7). But it will never choose a speed lower than the critical speed as it will neither help in meeting deadline nor will help in decreasing energy consumption.

Abstracting away Non-Uniformity. We use the result of speed-up bound (capacity augmentation bound) derived in [46]. We change processor speeds while maintaining the speed-up bound. While the bound in [46] was derived considering uniform-speed, in using it, as long as no core violates the speed-up bound it does not matter if the cores are running at different speeds or not. We can think this in terms of choosing an execution time for each node. In terms of scheduling it is not important how the execution speed-ups are obtained. Once our optimization finds the “best” feasible assignment of execution times for each node that satisfy the bounds of [46], we can use these execution times to do the scheduling upon (logically) identical multiprocessor platform. Thus, we abstracted away any non-uniformity in speed of the processors by considering that each node can change its execution time.

4.2 Steps for Solving Frequency and Core Assignment Problem

In the problem formulated in (4)–(7), the objective (4) and constraint (5) are convex. As we described before, selecting a different speed for a node can change the critical path in a DAG. That is, whenever we choose a new value of our decision variables (speeds), we have to run an algorithm to detect a critical path in the DAG. Every change in decision variable may invoke that algorithm. Thus constraint (6) raises a key challenge as the invocation of critical path finding algorithm may affect the characteristics of the optimization problem. Another key challenge in the above problem is that it is not differentiable as Constraint (7) is non-differentiable. Non-differentiability raises a significant challenge in optimization problem and restrict the applicability of many efficient approaches for solving. Our techniques to handle these challenges are described as follows.

4.2.1 Critical Path Formulation Using Convex Constraints

To address the challenge stemmed from constraint (6), we determine critical path of a DAG through a topological sorting of the nodes and represent the method through a set of convex constraints. For this, a DAG has to have a unique source and a unique sink. If a DAG has multiple sources, a new dummy node with WCER of zero is created as the parent of all source nodes, and this dummy node is considered as the unique source. Similarly, if a DAG has multiple sinks, a new dummy node with WCER of zero is created as the child of all sink nodes, and this dummy node becomes the unique sink. Hence, without loss of generality, from now onward we consider that each τ_i is a DAG with one source W_i^1 and one sink $W_i^{n_i}$, and that $W_i^1, W_i^2, \dots, W_i^{n_i}$ is a topological ordering of the nodes of τ_i .

It is well-known that the shortest path problem for graphs with non-negative edge weights can be formulated as a linear programming maximization problem (e.g., see Chapter 29 in Cormen et al. [28]). Therefore, we can reverse the constraints of the shortest path formulation and use node weights to obtain quantification of the longest path from the source node W_i^1 to any other node W_i^k . Thus, based on topological ordering, we can express constraint (6) of the above formulation for critical path function in terms of convex constraints (8), (9), and (10) using a new variable d as follows.

$$d_i[W_i^1] = \frac{c_i^1}{s_i^1}, \forall \tau_i \quad (8)$$

$$d_i[W_i^\ell] \geq d_i[W_i^k] + \frac{c_i^\ell}{s_i^\ell}, \quad \forall \tau_i, (W_i^k \rightarrow W_i^\ell) \quad (9)$$

$$d_i[W_i^{n_i}] \leq \frac{T_i}{2}, \forall \tau_i \quad (10)$$

Constraint (8) fixes the weight of the source node. Constraint (9) enforces that the longest path from source to node W_i^ℓ must be no less than the longest path to any adjacent predecessor node plus the weight of W_i^ℓ . Finally, constraint (10) checks to ensure that the longest path to the sink node $W_i^{m_i}$ (i.e., the critical path) is bounded according to constraint (6).

4.2.2 Handling Non-Differentiability by Refining Core Allocation

Upon expressing constraint (6) of the problem given in (4) – (7) in terms of convex constraints (8), (9), and (10), the problem still remains non-differentiable due to constraint (7). A potential approach to solving the problem is to adopt a penalty based simulated annealing. Simulated annealing is a global optimization framework that employs stochastic global exploration to escape from local minima and is suitable for problems where gradient information is not available. A penalty-based approach (such as ℓ_1 – penalty method [25, 65]) makes it adoptable to constrained problem. While such a method can achieve global optimality or high-quality solution under certain theoretical conditions and parameter setups, its running time can be very very long, making it impractical for real-time task scheduling. Instead, we propose to formulate a convex optimization problem by modifying the federated scheduling policy while retaining the theoretical performance bound of the scheduling technique compared to the original one. Therefore, an optimal solution of the proposed convex problem should be quite close to that of the original problem defined in (4) – (7).

Since non-differentiability exists in constraint (7), we aim to make it continuous and convex. Its non-differentiability stems from the assignment of integer number of processor cores to the tasks. Hence, we slightly modify the federated scheduling policy by assigning a continuous value to each task based on which the task is actually scheduled on an integer number of cores (as the number of cores is always integer). This provably retains the theoretical schedulability performance of the federated scheduling. Instead of assigning $\lceil \frac{C_i - L_i}{T_i - L_i} \rceil$ cores to a high utilization task τ_i , we assign the value $(\frac{C_i - L_i}{T_i - L_i} + 1)$ to it (in our optimization) while the task is scheduled on $\lfloor \frac{C_i - L_i}{T_i - L_i} + 1 \rfloor$ cores. The remaining cores are assigned to low-utilization tasks. Theorem 3 proves that the federated scheduling retains its capacity augmentation bound of 2 upon the above modification.

► **Theorem 3.** *Assigning the value $(\frac{C_i - L_i}{T_i - L_i} + 1)$ to each high utilization task τ_i and scheduling it on $\lfloor \frac{C_i - L_i}{T_i - L_i} + 1 \rfloor$ cores, and assigning the remaining cores to all low-utilization tasks does not change the capacity augmentation bound of 2 of the federated scheduling.*

Proof. We consider a task set τ that satisfies Conditions 1 and 2 from Definition 1 for $b = 2$. As proved in [46], the capacity augmentation bound is 2 if every high-utilization task gets at least $\lceil \frac{C_i - L_i}{T_i - L_i} \rceil$ cores and all low-utilizations tasks together get at least $2 \sum_{\tau_i \in \tau_{\text{low}}} u_i$ cores. Hence, it is sufficient to prove that these two conditions hold upon our modification.

Our assigned value to a high utilization task τ_i ,

$$\frac{C_i - L_i}{T_i - L_i} + 1 > \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil.$$

The actual number of cores on which task τ_i is scheduled in the modified federated scheduling is

$$m_i = \left\lfloor \frac{C_i - L_i}{T_i - L_i} + 1 \right\rfloor \geq \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil.$$

Now we have to prove that the total number of cores assigned to low utilization tasks is at least $2 \sum_{\tau_i \in \tau_{\text{low}}} u_i$. For simplicity let $\sigma_i = \frac{T_i}{L_i}$. Hence, $T_i = \sigma_i L_i$, $C_i = u_i T_i = \sigma_i u_i L_i$.

Since each task satisfies Condition 2 from Definition 1 for $b = 2$, $L_i \leq \frac{T_i}{b} \Rightarrow \sigma_i \geq b = 2$. (Note $T_i = D_i$). By the definition of high-utilization task τ_i , $u_i \geq 1$. Together with $\sigma_i \geq 2$, we can state $\frac{(u_i-1)(\sigma_i-2)}{\sigma_i-1} \geq 0$. Now

$$\begin{aligned} \frac{C_i - L_i}{T_i - L_i} + 1 &= \frac{\sigma_i u_i L_i - L_i}{\sigma_i L_i - L_i} + 1 = \frac{\sigma_i u_i + \sigma_i - 2}{\sigma_i - 1} \\ &\leq \frac{\sigma_i u_i + \sigma_i - 2}{\sigma_i - 1} + \frac{(u_i - 1)(\sigma_i - 2)}{\sigma_i - 1} = \frac{2u_i(\sigma_i - 1)}{\sigma_i - 1} = 2u_i. \end{aligned}$$

Now total cores assigned to low-utilization tasks,

$$\begin{aligned} m_{\text{low}} &= m - \sum_{\tau_i \in \tau_{\text{high}}} \left(\frac{C_i - L_i}{T_i - L_i} + 1 \right) \geq m - \sum_{\tau_i \in \tau_{\text{high}}} 2u_i \\ &\geq 2u_{\text{sum}} - \sum_{\tau_i \in \tau_{\text{high}}} 2u_i = 2 \sum_{\tau_i \in \tau_{\text{low}}} u_i \end{aligned}$$

◀

The above modification in federated scheduling makes constraint (7) continuous and convex.

4.3 Energy-Aware Scheduling upon Convex Optimization

By incorporating the results from Sections 4.2.1 and 4.2.2, we can express the new formulation for determining node speeds as follows.

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j) \\ \text{subject to} \quad & \sum_{i=1}^n \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}}{T_i} \leq \frac{m}{2}; \\ & d_i[W_i^1] = \frac{c_i^1}{s_i^1}, \forall \tau_i; \\ & d_i[W_i^\ell] \geq d_i[W_i^k] + \frac{c_i^\ell}{s_i^\ell}, \quad \forall \tau_i, (W_i^k \rightarrow W_i^\ell); \\ & d_i[W_i^{n_i}] \leq \frac{T_i}{2}, \forall \tau_i; \\ & \sum_{\tau_i \text{ with } \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} \geq 1} \left(\frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j} - L_i(s_i)}{T_i - L_i(s_i)} + 1 \right) \\ & + 2 * \sum_{\tau_i \text{ with } \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} < 1} \sum_{j=1}^{n_i} \frac{c_i^j}{T_i s_i^j} \leq m \end{aligned}$$

The objective and all the constraints in the above problem are now convex, making it a convex optimization problem. Therefore, we can find its **optimal** solution in **polynomial time** through gradient based or Interior Point method using any convex problem solver. In practice, it is very efficient to use any standard convex optimization tool such as CVX [32], IPOPT [38], or MATLAB's `fmincon` function [6] for an optimal solution.

Note that by solving the above formulation, we jointly determine speed of nodes and the number of cores for each DAG as well as for all low-utilization tasks. After the core and speed assignment, every cluster of cores for high-utilization task boils down to scheduling a single DAG on m_i cores. Note that we need to run the above optimization only once. After determining the running speeds of all nodes, we schedule the tasks on their assigned cores. Through incorporating DVFS with federated scheduling, the speeds of the cores are adjusted according to what nodes they are executing. Such speed switching is quite feasible and common in practice. Modern microprocessors tend to change their DVFS setting rather frequently in response to rapid changes in the application behavior [60]. Note that at anytime if a core remains idle, it can be shutdown for energy saving and turned on later when needed.

Low-utilization tasks are treated and executed as if they were sequential tasks. Thus, the cluster of cores assigned to low-utilization task boils down to multiprocessor scheduling of sequential tasks. Our speed assignment technique hence considered an entire low-utilization task as a single node and there was one speed assignment for every low-utilization task. As we mentioned before, any multiprocessor scheduling algorithm such as partitioned EDF [50], or various rate-monotonic schedulers [11] with a utilization bound of at most $\frac{1}{2}$ can be used to schedule all the low-utilization tasks on the allocated m_{low} cores. Since there is much work on energy-aware real-time scheduling of sequential tasks on multiprocessor, we can also adopt one of that policy for further reducing energy consumption. One potential approach is to adopt the Adaptive DVFS technique proposed in [51] that adaptively determines the frequency of each task for better performance on both schedulability and energy consumption. It schedules the tasks with prolonged execution while ensuring that all meet deadlines.

4.4 Other Factors of Energy Consumption

In federated scheduling, there is no preemption across tasks and hence we have not considered preemption cost. Our proposed solution is optimal only for the formulated problem. Our formulation has ignored several factors of energy consumption. For example, it has ignored the time and energy overhead in speed switching as well as the overhead associated with processor turn on/off. Also, the speed assignment is based on the WCER of the tasks while the actual execution requirement of a task may vary. Our result provides a strong fundamental basis to address these issues in the future. In the future, we shall also address energy minimization considering other key components, i.e., GPU, system bus, and memory/cache.

5 Energy-Aware Global Scheduling

We now describe our proposed scheduling approach for energy-aware global real-time scheduling. Here also, our approach is to regulate the frequencies (speed) across the cores and across different nodes of the same task for minimizing overall energy consumption. Hence, our approach

incorporates an optimization engine and DVFS into the classical real-time scheduling policies such as EDF and DM and makes them energy-aware. Specifically, once the execution speeds (and hence the processor frequencies) of the nodes of all DAGs are determined through optimization, we can adopt an existing global real-time scheduling policy. Figure 5 shows our energy-aware global scheduling framework. When the processor frequencies are adjusted based on the nodes' speed assignments, the adopted scheduling policy leads to energy minimization. We describe our techniques for global EDF and DM.

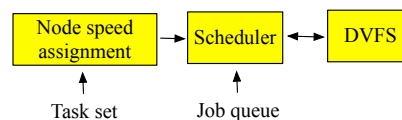


Figure 5 Energy-aware global scheduling framework

5.1 Energy-Aware Global EDF

EDF is a classic and widely adopted dynamic-priority scheduling policy where tasks are prioritized according to their absolute deadlines. In our approach, the EDF policy is adopted after assigning the node execution speeds. Once the execution speeds of the nodes of all DAGs are determined, and the processor frequencies are adjusted accordingly through DVFS, running EDF policy leads to energy minimization. A node is ready to execute when all its predecessors have executed. EDF for parallel tasks works as follows: at each time step, the scheduler first tries to schedule as many ready nodes from all jobs with the earliest deadline as it can; then it schedules ready nodes from the jobs with the next earliest deadline, and so on, until either all cores are busy or no nodes are ready.

Formulation. We use the overall energy consumption used in our formulation in Section 4.1 as our objective here also. Similar to that formulation, here also we want to minimize this objective while ensuring the schedulability under the global EDF scheduling policy. For global EDF scheduling of DAG tasks on multicore, the work in [46] derived a capacity augmentation bound of 2.618. When node W_i^j of task τ_i is assigned speed s_i^j , where $1 \leq j \leq n_i, 1 \leq i \leq n$, using a capacity augmentation bound of 2.618, we define **Condition 1** and **Condition 2** for schedulability in Remark (2) as Constraints (12) and (13), respectively, in our formulation. Thus, for global EDF scheduling for energy minimization, we first formulate our problem as follows. Our objective is to determine speeds $s_i = \{s_i^1, s_i^2, \dots, s_i^j, \dots, s_i^{n_i}\}, \forall \tau_i$ to

$$\text{Minimize } \sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j) \quad (11)$$

$$\text{subject to } \sum_{i=1}^n \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}}{T_i} \leq \frac{m}{2.618} \quad (12)$$

$$L_i(s_i) \leq \frac{T_i}{2.618}, \forall \tau_i \quad (13)$$

Convexification. The objective (11) and constraint (12) are convex. By incorporating the results from Section 4.2.1, constraint (13) can be replaced by three convex constraints using a new variable d considering $W_i^1, W_i^2, \dots, W_i^{n_i}$ as a topological ordering of the nodes of DAG τ_i with one source and one sink. Thus, we convert the above problem formulation as the following convex optimization problem where our objective is to determine speeds $s_i = \{s_i^1, s_i^2, \dots, s_i^j, \dots, s_i^{n_i}\}, \forall \tau_i$ to

$$\text{Minimize } \sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j) \quad (14)$$

$$\text{subject to } \sum_{i=1}^n \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}}{T_i} \leq \frac{m}{2.618} \quad (15)$$

$$d_i[W_i^1] = \frac{c_i^1}{s_i^1}, \forall \tau_i \quad (16)$$

$$d_i[W_i^\ell] \geq d_i[W_i^k] + \frac{c_i^\ell}{s_i^\ell}, \quad \forall \tau_i, (W_i^k \rightarrow W_i^\ell) \quad (17)$$

$$d_i[W_i^{n_i}] \leq \frac{T_i}{2.618}, \forall \tau_i \quad (18)$$

The objective and all the constraints in the above problem are now convex, making it a convex optimization problem. Therefore, we can find its **optimal** solution in **polynomial**

time through any convex problem solver. Note that we need to run the above optimization only once. After determining the running speeds of all nodes, we execute the tasks based on EDF scheduling. During execution, a node runs at its assigned speed. Through incorporating DVFS with EDF scheduling, the speeds of the cores are adjusted according to what nodes they are executing. Note that at anytime if a core remains idle, it can be shutdown for energy saving and turned on later when needed.

For the same reasons we explained for the problem in Section 4, the above said solution is optimal only for the problem formulated in (14)–(18), and is not an optimal solution for actual energy minimization for a number of issues not addressed in the problem formulation such as frequency switching overhead, preemption cost in global scheduling, task execution time uncertainty, and contributions from other components including GPU, system bus, and memory/cache. As stated before, we shall address these issues in the future.

5.2 Energy-Aware Global DM

DM is an efficient and widely used fixed-priority scheduling policy for real-time systems where tasks are assigned priorities according to their (relative) deadlines; the task with the shortest deadline being assigned the highest priority. Our approach for energy-aware DM scheduling is similar to that for energy-aware EDF scheduling described above. We first determine the execution speeds of the nodes. We then incorporate DVFS with DM so that every node runs at the assigned speed during execution.

For global DM scheduling of DAG tasks on multicore, the work in [46] derived a capacity augmentation bound of 3.732. Using this bound, we formulate the problem in the same way as follows considering $W_i^1, W_i^2, \dots, W_i^{n_i}$ as a topological ordering of the nodes of DAG τ_i . Our objective is to determine speeds $s_i = \{s_i^1, s_i^2, \dots, s_i^j, \dots, s_i^{n_i}\}, \forall \tau_i$ to

$$\begin{aligned}
 & \text{Minimize} && \sum_{i=1}^n w_i \sum_{j=1}^{n_i} E(c_i^j, s_i^j) \\
 & \text{subject to} && \sum_{i=1}^n \frac{\sum_{j=1}^{n_i} \frac{c_i^j}{s_i^j}}{T_i} \leq \frac{m}{3.732} \\
 & && d_i[W_i^1] = \frac{c_i^1}{s_i^1}, \forall \tau_i \\
 & && d_i[W_i^\ell] \geq d_i[W_i^k] + \frac{c_i^\ell}{s_i^\ell}, \quad \forall \tau_i, (W_i^k \rightarrow W_i^\ell) \\
 & && d_i[W_i^{n_i}] \leq \frac{T_i}{3.732}, \forall \tau_i
 \end{aligned}$$

The characteristics of the above problem are the same as those of the energy-aware EDF scheduling. Specifically, this problem is also convex and its optimal solution can be achieved in polynomial time through any convex problem solver.

6 Evaluation

In this section, we present the evaluation of our energy-minimization policies using synthetic workloads through simulations. We used a custom-built simulator developed in MATLAB. We conduct simulation for both federated scheduling and global scheduling.

6.1 Simulation Setup and Evaluation Metrics

We generated DAG task sets using the Erdős-Rényi method $G(n_i, p)$ where n_i is the number of nodes and p is the probability of adding edges between nodes [27]. The value of n_i was chosen randomly from range [5, 10]. If a DAG was disconnected, we added the minimum number of edges needed to make it connected. Each node's execution requirement was randomly chosen from range [5, 10]. We assign harmonic period T_i to a task τ_i by finding the smallest value x such that critical path, $L_i \leq 2^x$ and set T_i to be either 2^x or 2^{x+1} . T_i value was chosen randomly to get a fair distribution of high and low utilization tasks in a task set. Harmonic periods were used to reduce the time needed for collecting the results. We added tasks to a set until its total utilization was achieved. We show the simulation results for an average over 900 task sets using 20 cores (m). To compute the node speeds, we use MATLAB's `fmincon` which provides an optimal solution of our convex problem.

The energy consumption of a task set is computed based on our objective function. We then compute the ratio of the energy consumption and the hyper-period to get the *average power consumption* and use it as our key metric for evaluation. In our proposed approach, the optimization problem uses a schedulability test as a constraint that guarantees that a task is always schedulable (assuming no extra system overhead). Thus, evaluation in terms of schedulability is redundant and is not shown when our approaches are compared against traditional (energy-unaware) real-time scheduling policies under similar schedulability conditions. However, under federated scheduling, we compare our approach against an additional baseline proposed in [19] that trades energy consumption for schedulability but cannot pre-set the number of cores as an input. For this particular case, schedulability becomes a differentiating factor and will be used as a metric for evaluation. Specifically, our approach always considers schedulable tasks and outperforms the baseline (Section 6.2).

6.2 Results for Energy-Aware Federated Scheduler

We first compare the performance of the energy-aware federated scheduler against the energy-aware simplified federated scheduling, called *intra-DAG merging*, proposed in [19]. As discussed before in Section 2, in this intra-DAG merging approach, the limitation on number of concurrent tasks degrades the schedulability of tasks while our approach ensures schedulability of any feasible task set. To demonstrate their schedulability difference, we generate 900 random DAG task sets with utilization between $0.1m$ and m . We evaluate the performance under varying number of tasks. Figure 6 shows that intra-DAG merging performs poorly in schedulability ratio (i.e., the fraction of schedulable cases) compared to our energy-aware federated scheduler. When the number of tasks reaches 20, the schedulability ratio of the intra-DAG merging approach is 0, while our scheduler can always schedule the generated task set. Since the performance of Intra-DAG merging is very poor in terms of schedulability, we do not consider it in comparing energy-consumption.

Figure 7a shows the results for our energy-aware federated scheduler for 900 task sets under varying total utilization of each set in the range [2, 18]. All other parameters are fixed

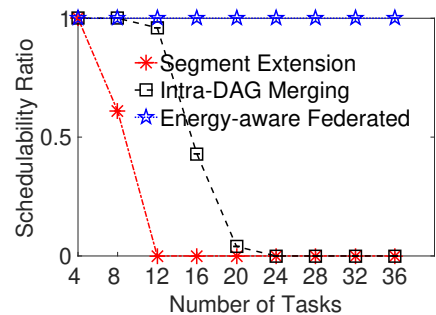
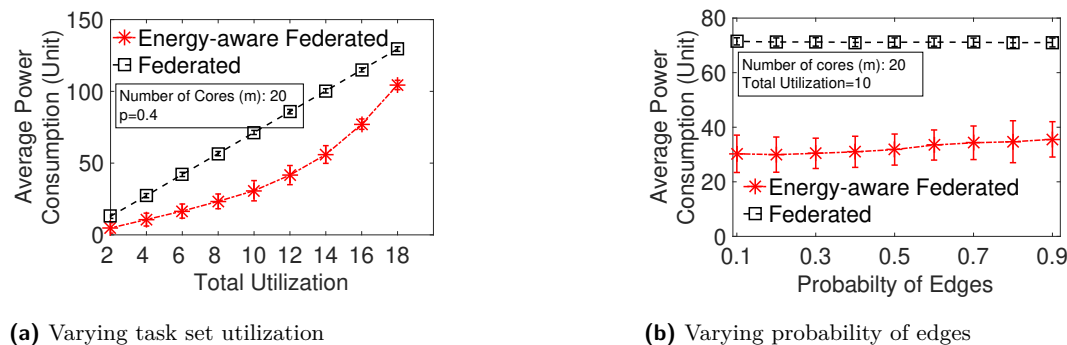


Figure 6 Schedulability under energy-aware federated and existing approach

as before. We compare the performance with (energy-unaware) federated scheduler. We considered the speed of tasks running in federated scheduler to be 2.0. The figure shows that our energy-aware federated scheduler saves up to 62.95% of the power when compared to federated scheduler. Figure 7b shows the energy performance under varying p . We see that our energy-aware federated scheduler consistently outperforms federated scheduler and can achieve up to 57.95% energy saving compared to the latter.



■ **Figure 7** Performance of Energy-Aware Federated Scheduler

6.3 Results for Energy-Aware Global Scheduling

We now present the results under global EDF and global DM scheduling.

6.3.1 Energy Savings under Energy-Aware Global EDF

In Figure 8a, we show the average power consumption over a hyper-period for 900 DAG task sets and vary the total utilization from 2 to 18. For a moderately dense DAG, we set $p = 0.4$. We compare our energy-aware global EDF policy with traditional energy-unaware global EDF (named ‘EDF’ subsequently) scheduling as, to our knowledge, no approach has yet been proposed for global energy-aware scheduling of parallel tasks for multicore. Under EDF, all tasks run at the speed-up bound, 2.618, since a speed of 2.618 always ensures schedulability of a feasible task set. Figure 8a shows that the average power consumption of our approach increases with an increase in total utilization. This is due to the reduction in available slack with the increase in total utilization. From this result, we can see that our energy-aware EDF consumes up to 68.17% less power compared to EDF. Figure 8b shows the performance under different DAG structures by varying edge probability p uniformly in range $[0.1, 0.9]$ setting total utilization at 10. Our approach is consistently better than EDF and can achieve up to 62.4% power saving compared to EDF.

6.3.2 Energy Savings under Energy-Aware Global DM

In Figure 9a, we show the average power consumption in our energy-aware DM against traditional energy-unaware DM (named ‘DM’ subsequently) under varying total utilization. Here our parameters are chosen similar to the EDF case. We consider the speeds of all task running under DM to be 3.732 which is the corresponding speed up bound for ensuring schedulability under DM. As the speed-up bound for DM is higher than EDF, we see an increase in average power consumption for both energy-aware DM and DM. However, our energy-aware DM consumes up to 64.10% less power than DM. Figure 9b shows the performance under different DAG structures by varying p uniformly in range $[0.1, 0.9]$ setting

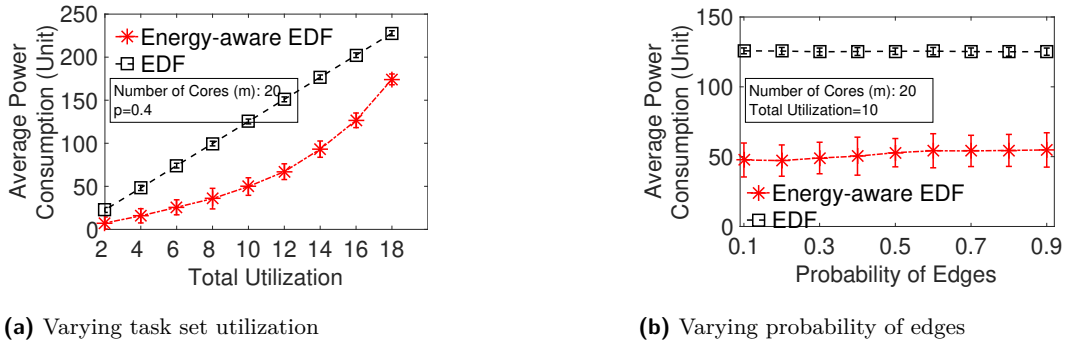


Figure 8 Performance of Energy-Aware Global EDF

total utilization at 10. As the figure shows our approach is consistently better than DM and can achieve up to 62.1% power saving compared to DM. For both schedulers, the DAG structure does not show observable effect in performance.

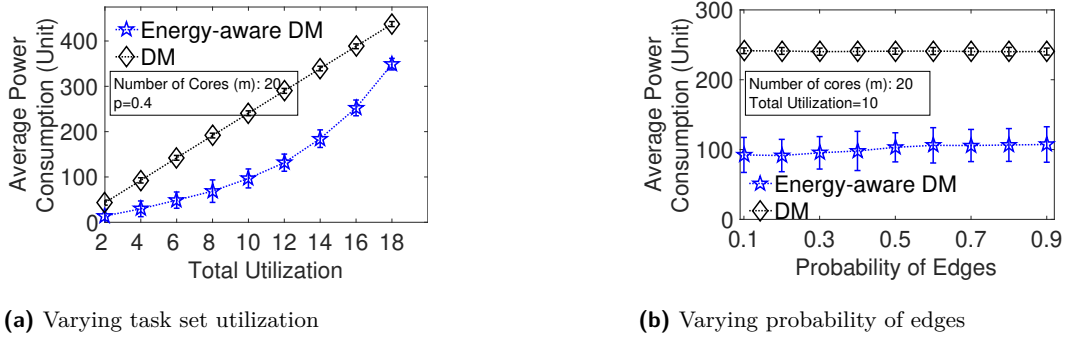


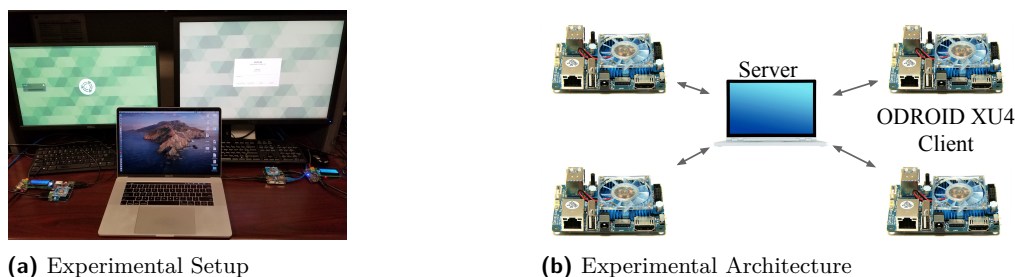
Figure 9 Performance of Energy-Aware Global DM

7 Proof of Concept

Although many AMD and Intel processors support per-core DVFS, we could not use those in PC/Laptop for our experiment as we found it quite challenging to accurately measure the CPU power consumption due to interference from numerous sources such as peripheral devices and other programs. Hence, we have developed a proof of concept system for our results using multiple ODROID boards [55] where this issue is less severe.

7.1 Proof of Concept System Setup

ODROID board equips an Exynos5 Octa 5422 SoC consisting of two quad-core clusters – a ‘LITTLE’ cluster with four ARM Cortex-A7 and a ‘big’ cluster with four ARM Cortex-A15. We have used Ubuntu 16.04.1 LTS MATE operating system with Linux kernel 4.14. Cores of a cluster operate at the same frequency and have the same voltage. Since a single ODROID board does not support per-core DVFS, we created our proof of concept system as a multiprocessor platform by connecting four boards where we used one core from each board. Thus our setup uses four cores from four different boards to enable per-core DVFS. Such a proof of concept platform was made up only to enable per-core DVFS-based experiment.



■ **Figure 10** Experiment Setup and Architecture

Figure 10a shows our experimental setup and Figure 10b shows its architecture as a multi-processor environment. We connected each ODROID board to a central server that resides on a MacBook Pro via a wired LAN. The central server schedules nodes from all jobs, based on the proposed energy-aware federated scheduler, on the clients. Each node is emulated by a C program that is executing an empty `for` loop for a specified time. Since our focus is to measure energy consumption, we rely only on an empty `for` loop to expend processor cycles which consumes energy similar to a regular instruction execution while avoiding any cache/memory overheads. The scheduler located at the server maintains the dependencies among nodes. Thus, a node was executed only when all of its predecessor nodes had finished execution. Upon receiving a request from the server, the ODROID client executes the C program as a process on a dedicated core (i.e., without interference from OS or other programs). In our implementation, we used `CPUFREQ-SET` program from `CPUFREQUTILS` package for setting the frequency of each CPU. The overhead from the client program and driver is very low and can be included in the context switch cost.

We set the frequency of both the big and LITTLE cluster at 600Mhz by default. Upon receiving a request from the server, the client changes the frequency of the LITTLE cluster to the requested value and executes the program for a specified duration. Upon its completion, the frequency is scaled back to 600Mhz . The choice of 600Mhz stems from the observations that below 600Mhz the operating system struggles to schedule processes on time. Note that we do not put the core on the ODROID board that is executing the client to sleep as it incurs additional time overhead which can affect the schedulability of a task set.

We used ODROID’s smartpower device [8] for measuring power consumption during our experiments. The smartpower device uses a discrete sampling of voltage and current to compute the instantaneous power and total energy consumption for the entire board. It transmits these values over WiFi, which is collected manually for the hyper-period of the task set. Note that the energy measurement over the hyper-period for one board includes the total energy consumed by context switches, OS interferences, and peripheral chips/devices connected to the ODROID board. Since we use four ODROID boards, the number of peripheral devices and their static energy consumption scales accordingly.

7.2 Implementation and Experimental Methodology

We generated 4 DAG task sets using the approach described in Section 6.1. The worst-case execution time of each node was chosen randomly from $[27s, 54s, 81s, 108s, 135s]$, which corresponds to $[2, 4, 6, 8, 10] \times 10^9$ empty for loop iterations on 0.7Ghz . The task sets for experiments had total utilization of $0.3m$, $0.4m$, $0.5m$ and $.6m$ ($m = 4$). The value of p was fixed at 0.6. All other parameters were kept the same as simulation. The speed of each node in a task was calculated using MATLAB’s `fmincon`. Here also we evaluate in terms of

average power consumption which is the average power in Watts consumed by the platform during the execution of one task set up to its hyper-period. We measured the total energy consumed over the hyper-period and used it to compute the average power.

7.3 Proof of Concept Results

Figure 11 shows that the average power consumption of the energy-aware federated scheduler is lower than the existing energy-unaware federated scheduler (named ‘federated scheduler’ subsequently) at all values of total utilization. There is up to 0.159 Watts of power savings in our approach. Due to the randomness of the task set generation policy, the task set with utilization value 2 has tasks with smaller critical path length compared to the task set with utilization 2.4. (This is due to our task generation policy that connected two nodes in the graph based on whether a randomly generated number is less than 0.6 or not. For task at utilization 2.4, the random task generator did not add as many edges as the task at utilization 2 which resulted in smaller critical path length.) This resulted in higher frequency assignment for all nodes in the former. Thus, the average power consumption for task set with utilization 2.4 was lower than that of task set with utilization 2.

Figure 11 shows the average power consumption for both the baseline and the proposed approach is in the order of 6 – 7 Watts. This high power consumption is due to the static power consumed by input devices (like keyboard and mouse), communication chips, RAM/memory modules, and the big cluster operating 600Mhz on each ODROID board. In a processor with per-core DVFS, the static power consumption would be significantly lower than that exhibited by the proof of concept platform, and hence the average power savings as a percentage of the baseline power consumption will be significantly higher. Due to the high static power consumption of our experimental platform, the proof of concept results only show energy savings but are not indicative of the percentage of energy savings that can be obtained through the proposed approach.

8 Conclusion and Future Directions

In this paper, we have proposed energy-aware real-time scheduling for parallel tasks. We have demonstrated its performance through simulations as well as a proof of concept system evaluation on a physical platform. The adopted resource augmentation bounds only provide a sufficient condition for schedulability. Any improvement over the resource augmentation bounds in the future would lead to more energy efficient scheduler under our framework. We have limited the scope of our results to CPU energy consumption ignoring several practical factors. For example, we have ignored the overhead in processor speed switching as well as turn on/off. Also, the speed assignment is based on the worst-case execution requirement of the tasks while the actual execution requirement of a task may vary across runs/instances. Yet, our results provide a strong fundamental basis to incorporate these considerations into future research. In the future, we shall also take into account all key components, i.e., CPU, GPU, system bus, and memory/cache, that contribute to energy consumption.

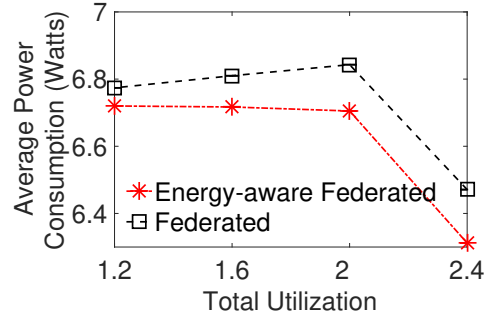


Figure 11 Performance of energy-aware federated scheduling

References

- 1 <https://www.therobotreport.com/self-driving-cars-power-consumption/>.
- 2 <https://www.greentechmedia.com>.
- 3 <https://www.wired.com/story/self-driving-cars-power-consumption-nvidia-chip/>.
- 4 https://web.archive.org/web/20090326020946/http://www.aero.org/conferences/mrqw/2002-papers/A_Burcin.pdf.
- 5 Intel scc. <https://www.intel.cn/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-single-chip-cloud-overview-paper.pdf>.
- 6 Matlab fmincon. <https://www.mathworks.com/help/optim/ug/fmincon.html>.
- 7 Pico 205. <https://en.wikipedia.org/wiki/PicoChip>.
- 8 Smartpower. <http://hardkernel.com/shop/smartpower2-with-15v-4a/>.
- 9 TILE-Gx™. http://www.tilera.com/products/processors/TILE-Gx_Family.
- 10 Björn Andersson and Dionisio de Niz. Analyzing global-edf for multiprocessor scheduling of parallel tasks. In *OPODIS*, pages 16–30. Springer, 2012.
- 11 Björn Andersson and Jan Jonsson. The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *15th Euromicro Conference on Real-Time Systems, 2003. Proceedings.*, pages 33–40, 2003.
- 12 Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Parallel and Distributed Processing Symposium. Proceedings. International*, pages 9–pp. IEEE, 2003.
- 13 Mario Bambagini, Mauro Marinoni, Hakan Aydin, and Giorgio Buttazzo. Energy-aware scheduling for real-time systems: A survey. *ACM Trans. Embed. Comput. Syst.*, 15(1):7:1–7:34, January 2016.
- 14 Sanjoy Baruah. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *26th Euromicro Conference on Real-Time Systems*, pages 97–105, 2014.
- 15 Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *27th Euromicro Conference on Real-Time Systems*, pages 222–231. IEEE, 2015.
- 16 Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *Real-Time Systems Symposium (RTSS), IEEE 33rd*, pages 63–72. IEEE, 2012.
- 17 A. Bhuiyan, D. Liu, A. Khan, A. Saifullah, N. Guan, and Z. Guo. Energy-efficient parallel real-time scheduling on clustered multi-core. *IEEE Transactions on Parallel and Distributed Systems*, 31(9):2097–2111, 2020.
- 18 A. A. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo. Mixed-criticality multicore scheduling of real-time gang task systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 469–480, 2019.
- 19 Ashikahmed Bhuiyan, Zhishan Guo, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient real-time scheduling of dag tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 17(5):84, 2018.
- 20 Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Transactions on Embedded Computing Systems (TECS)*, 8(4):31, 2009.
- 21 Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. Feasibility analysis in the sporadic DAG task model. In *25th Euromicro Conference on Real-Time Systems*, pages 225–233. IEEE, 2013.
- 22 Gang Chen, Kai Huang, and Alois Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):111, 2014.
- 23 Jian-Jia Chen, Heng-Ruey Hsu, and Tei-Wei Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 408–417. IEEE, 2006.

- 24 Jian-Jia Chen, Andreas Schranzhofer, and Lothar Thiele. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *Parallel & Distributed Processing. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.
- 25 Yixin Chen and Minmin Chen. Extended duality for nonlinear programming. *Comput. Optim. Appl.*, 47:33–59, 2010.
- 26 Alexei Colin, Arvind Kandhalu, and Ragunathan Rajkumar. Energy-efficient allocation of real-time applications onto heterogeneous processors. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10. IEEE, 2014.
- 27 Daniel Cordeiro, Gregory Mouni, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frederic Wagner. Random graph generation for scheduling simulations. In *Proceedings of the 3rd international ICST conference on simulation tools and techniques*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.
- 28 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- 29 Vinay Devadas and Hakan Aydin. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Green Computing Conference, 2010 International*, pages 61–72. IEEE, 2010.
- 30 David Ferry, Jing Li, Mahesh Mahadevan, Kunal Agrawal, Christopher Gill, and Chenyang Lu. A real-time scheduling service for parallel tasks. In *RTAS'13*.
- 31 James H. Gawron, Gregory A. Keoleian, Robert D. De Kleine, Timothy J. Wallington, and Hyung Chul Kim. Life cycle assessment of connected and automated vehicles: Sensing and computing subsystem and vehicle level effects. *Environmental Science & Technology*, 52(5):3249–3256, 2018.
- 32 Michael Grant and Stephen Boyd. CVX: MATLAB software for disciplined convex programming, 2012. <http://cvxr.com/cvx/>.
- 33 Akhil Guliani and Michael M. Swift. Per-application power delivery. In *Proceedings of the Fourteenth EuroSys Conference 2019, EuroSys '19*, pages 5:1–5:16, 2019.
- 34 Yifeng Guo, Dakai Zhu, and Hakan Aydin. Reliability-aware power management for parallel real-time applications with precedence constraints. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, 2011.
- 35 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 *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 156–168, 2019.
- 36 Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient multi-core scheduling for real-time DAG tasks. In *LIPICs-Leibniz International Proceedings in Informatics*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- 37 Jason Howard, Saurabh Dighe, Sriram R Vangal, Gregory Ruhl, Nitin Borkar, Shailendra Jain, Vasantha Erraguntla, Michael Konow, Michael Riepen, Matthias 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*, 46(1):173–183, 2011.
- 38 IPOPT. Interior point optimizer, 2011. <https://projects.coin-or.org/Ipopt>.
- 39 Ravindra Jejurikar. Energy aware non-preemptive scheduling for hard real-time systems. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 21–30. IEEE, 2005.
- 40 Xu Jiang, Nan Guan, Xiang Long, and Wang Yi. Semi-federated scheduling of parallel real-time tasks on multiprocessors. *arXiv preprint arXiv:1705.03245*, 2017.
- 41 J. Kim, H. Kim, K. Lakshmanan, and R. Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 31–40, 2013.
- 42 Junsung Kim, Hyoseung Kim, Karthik Lakshmanan, and Ragunathan Raj Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In

- Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, pages 31–40. ACM, 2013.
- 43 Fanxin Kong, Nan Guan, Qingxu Deng, and Wang Yi. Energy-efficient scheduling for parallel real-time tasks based on level-packing. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 635–640. ACM, 2011.
 - 44 Wan Yeon Lee. Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):530–537, 2012.
 - 45 Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Outstanding paper award: Analysis of global EDF for parallel tasks. In *25th Euromicro Conference on Real-Time Systems*, pages 3–13. IEEE, 2013.
 - 46 Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *26th Euromicro Conference on Real-Time Systems*, pages 85–96. IEEE, 2014.
 - 47 Keqin Li. Energy efficient scheduling of parallel tasks on multiprocessor computers. *J Supercomput.*, 60:223–247, 2012.
 - 48 Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 23–32. ACM, 2012.
 - 49 Di Liu, Jelena Spasic, Gang Chen, and Todor Stefanov. Energy-efficient mapping of real-time streaming applications on cluster heterogeneous mpsocs. In *Embedded Systems For Real-time Multimedia (ESTIMedia), 2015 13th IEEE Symposium on*, pages 1–10. IEEE, 2015.
 - 50 José María López, José Luis Díaz, and Daniel F García. Utilization bounds for edf scheduling on real-time multiprocessor systems. *Real-Time Syst.*, 28(1):39–68, October 2004.
 - 51 Junyang Lu and Yao Guo. Energy-aware fixed-priority multi-core scheduling for real-time systems. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 277–281, 2011.
 - 52 G. Luo, B. Guo, Y. Shen, H. Liao, and L. Ren. Analysis and optimization of embedded software energy consumption on the source code and algorithm level. In *2009 Fourth International Conference on Embedded and Multimedia Computing*, pages 1–5, 2009.
 - 53 Sujay Narayana, Pengcheng Huang, Georgia Giannopoulou, Lothar Thiele, and R Venkatesha Prasad. Exploring energy saving for mixed-criticality systems on multi-cores. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, 2016.
 - 54 Geoffrey Nelissen, Vandy Bertin, Joël Goossens, and Dragomir Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 321–330. IEEE, 2012.
 - 55 ODROID XU-4. <http://www.hardkernel.com/>.
 - 56 Santiago Pagani and Jian-Jia Chen. Energy efficient task partitioning based on the single frequency approximation scheme. In *Real-Time Systems Symposium (RTSS), IEEE 34th*, pages 308–318. IEEE, 2013.
 - 57 Santiago Pagani and Jian-Jia Chen. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(5s):158, 2014.
 - 58 Antonio Paolillo, Joël Goossens, Pradeep M Hettiarachchi, and Nathan Fisher. Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies. In *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2014.
 - 59 Antonio Paolillo, Paul Rodriguez, Nikita Veshchikov, Joël Goossens, and Ben Rodriguez. Quantifying energy consumption for practical fork-join parallelism on an embedded real-time operating system. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 329–338, 2016.

- 60 Sangyoung Park, Jaehyun Park, Donghwa Shin, Yanzhi Wang, Qing Xie, Massoud Pedram, and Naehyuck Chang. Accurate modeling of the delay and energy overhead of dynamic voltage and frequency scaling in modern microprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):695–708, 2013.
- 61 Xuan Qi and Dakai Zhu. Energy efficient block-partitioned multicore processors for parallel applications. *Journal of Computer Science and Technology*, 26(3):418–433, 2011.
- 62 Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. In *RTSS '11*.
- 63 Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D Gill. Parallel real-time scheduling of DAGs. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3242–3252, 2014.
- 64 Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49(4):404–435, 2013.
- 65 Abusayeed Saifullah, Chengjie Wu, Paras Tiwari, You Xu, Yong Fu, Chenyang Lu, and Yixin Chen. Near optimal rate selection for wireless control systems. *ACM Transactions on Embedded Computing Systems*, 13(4s):128:1–128:25, 2013. Special Issue on Real-Time and Embedded Systems.
- 66 Euseong Seo, Jinkyu Jeong, Seonyeong Park, and Joonwon Lee. Energy efficient scheduling of real-time tasks on multicore processors. *IEEE transactions on parallel and distributed systems*, 19(11):1540–1552, 2008.
- 67 Corey Tessler, Venkata Modekurthy, Nathan Fisher, and Abusayeed Saifullah. Bringing inter-thread cache benefits to federated scheduling. In *The 26th IEEE/USENIX Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.
- 68 Leping Wang and Ying Lu. Efficient power management of heterogeneous soft real-time clusters. In *Real-Time Systems Symposium, 2008*, pages 323–332. IEEE, 2008.
- 69 Lizhe Wang, Samee U. Khan, Dan Chen, Joanna Kolodziej, Rajiv Ranjan, Cheng-Zhong Xu, and Albert Zomaya. Energy-aware parallel task scheduling in a cluster. *Future Gener. Comput. Syst.*, 29(7):1661–1670, 2013.
- 70 Lizhe Wang, Gregor Von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 368–377. IEEE Computer Society, 2010.
- 71 Huiting Xu, Fanxin Kong, and Qingxu Deng. Energy minimizing for parallel real-time tasks based on level-packing. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 98–103. IEEE, 2012.
- 72 Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN Notices*, volume 40, pages 1–10. ACM, 2005.
- 73 Chuan-Yue Yang, Jian-Jia Chen, and Tei-Wei Kuo. An approximation algorithm for energy-efficient scheduling on a chip multiprocessor. In *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, pages 468–473. IEEE Computer Society, 2005.
- 74 Dakai Zhu, Nevine AbouGhazaleh, Daniel Mossé, and Rami Melhem. Power aware scheduling for and/or graphs in multiprocessor real-time systems. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 593–601. IEEE, 2002.
- 75 Dakai Zhu, Daniel Mosse, and Rami Melhem. Power-aware scheduling for and/or graphs in real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 15(9):849–864, 2004.