Energy-Efficient Real-Time Scheduling of DAG Tasks

ASHIKAHMED BHUIYAN and ZHISHAN GUO*, University of Central Florida, USA
ABUSAYEED SAIFULLAH, Wayne State University, USA
NAN GUAN, Hong Kong Polytechnic University, Hong Kong
HAOYI XIONG, Big Data Lab, Baidu Inc., China

This work studies energy-aware real-time scheduling of a set of sporadic Directed Acyclic Graph (DAG) tasks with implicit deadlines. While meeting all real-time constraints, we try to identify the best task allocation and execution pattern such that the average power consumption of the whole platform is minimized. To our knowledge, this is the first work that addresses the power consumption issue in scheduling multiple DAG tasks on multi-cores and allows intra-task processor sharing. First, we adapt the decomposition-based framework for federated scheduling and propose an energy-sub-optimal scheduler. Then, we derive an approximation algorithm to identify processors to be merged together for further improvements in energy-efficiency. The effectiveness of the proposed approach is evaluated both theoretically via approximation ratio bounds and also experimentally through simulation study. Experimental results on randomly generated workloads show that our algorithms achieve an energy saving of 60% to 68% compared to existing DAG task schedulers.

CCS Concepts: • **Computer systems organization** → *Real-time system architecture*;

Additional Key Words and Phrases: Parallel task, real-time scheduling, energy minimization, convex optimization

ACM Reference format:
Ashikahmed Bhuiyan, Zhishan Guo*, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. 2018. Energy-Efficient Real-Time Scheduling of DAG Tasks. *ACM Trans. Embed. Comput. Syst.* 17, 5, Article 84 (September 2018), 25 pages.
https://doi.org/10.1145/3241049

1 INTRODUCTION

Due to size and weight limitations in embedded systems, energy consumption is a cornerstone in their design, especially for battery-operated ones. Energy-efficient and power-aware computing therefore are gaining increasing attention in the embedded systems research. It is important due to the market demand of increased battery life for portable devices. Moreover, reducing energy consumption could lead to smaller power bills. Being motivated by this goal, there has been a trend in embedded system design and development towards multi-core platforms. To better utilize

Authors' addresses: A. Bhuiyan and Z. Guo*, Department of Electric and Computer Engineering, University of Central Florida, 4328 Scorpius Street, Orlando, FL 32816; emails: zhishan.guo@ucf.edu, zhishan.guo@ucf.edu; A. Saifullah, Department of Computer Science, Wayne State University, 5057 Woodward Ave, Detroit, MI 48202; email: saifullah@wayne.edu; N. Guan, Department of Computing, Hong Kong Polytechnic University, Hung Hom, Hong Kong; email: nan.guan@polyu.edu.hk; H. Xiong, Big Data Lab, Baidu Inc., Building No. 2, Xibeiwang East Road, Haidian District, Beijing, China; email: xionghaoyi@baidu.com.

1539-9087/2018/09-ART84 $15.00
https://doi.org/10.1145/3241049

the capacity of multi-core platforms, parallel computation (where an individual task executes in multiple processors simultaneously) needs to be considered. For example, a recent study (Pagani and Chen 2013) has shown that the energy consumption of executing certain workloads perfectly distributed in two cores is significantly less than that of executing the same workload in one core at double frequency.

In this article, we deal with workloads that are represented as DAGs—that are considered to be one of the most representative models of deterministic parallel tasks (Baruah et al. 2012). There are many existing works that focused on real-time scheduling of parallel tasks or their schedulability analysis (Baruah et al. 2012; Bonifaci et al. 2013; Baruah 2014; Li et al. 2013, 2014). However, none of them addressed the energy consumption issue.

Energy-Aware Real-Time Scheduling. In the design of embedded systems, energy minimization is a prime requirement. Much work has been done on minimizing the energy cost with respect to sequential tasks for multi-core systems (Fisher et al. 2009; Pagani and Chen 2013, 2014; Narayana et al. 2016). Specifically, Pagani and Chen (2013) and Pagani and Chen (2014) present an energy efficient task partitioning scheme, where the cores are grouped in frequency islands. The authors in Aydin and Yang (2003) considers both feasibility and energy-awareness while partitioning periodic real-time tasks on a multi-core platform. For EDF scheduling, they show that if the workload is balanced evenly among the processors, deriving optimal energy consumption and finding a feasible partition is NP-Hard. To date, only a little work has been done for energy-aware real-time scheduling of parallel tasks. In general, minimizing energy/power consumption of a real-time system is challenging due to the complex (non-linear) relationship between frequency, energy consumption, and execution time of each task.

In this article, we study the scheduling of a set of sporadic DAG tasks with implicit deadlines on a multi-core platform. To the best of our knowledge, this is the first work that addresses the power consumption issue in scheduling multiple DAG tasks on multi-core. We assume that all the cores that are assigned to a DAG task will always remain active, which leads to a non-negligible power consumption. To reduce this effect, we allow intra- and inter-task processor sharing to remove or reduce the number of lightly loaded cores. After merging, the cores that are not required can be shut off completely. When the average case execution times are typically small compared to the worst-case execution time (WCET), the cores will remain idle (in that case the active power consumption will be minimized, please see the Power/Energy model described at Section 2). Specifically, we make the following key contributions:

(i) We propose a power-consumption-aware scheduling algorithm for sporadic DAG tasks with implicit deadlines.
(ii) Under the federated scheduling and task decomposition framework, our table-driven scheduler is shown to be optimal in the sense of average power consumption (i.e., named sub-optimal due to extra constraints included).
(iii) We propose an efficient processor merging technique that is widely applicable for energy-efficiency improvements to most of the existing work on federated DAG task scheduling. We formally prove the NP-completeness of the problem, propose an approximation algorithm, and prove the upper bound of its approximation ratio.
(iv) Based on randomly generated workload, simulations are conducted to verify the theoretical results and demonstrate the effectiveness of our algorithm.

The rest of this article is organized as follows. Section 2 presents the system model and formally defines our problem. Section 3 adapts the task decomposition scheme and proposes an (sub-)optimal federated scheduler based on segment extension and problem transformation (into a convex optimization with linear inequality constraints). Section 4 relaxes the federated limitation by presenting and analyzing techniques for intra-DAG and inter-DAG processor merging, so

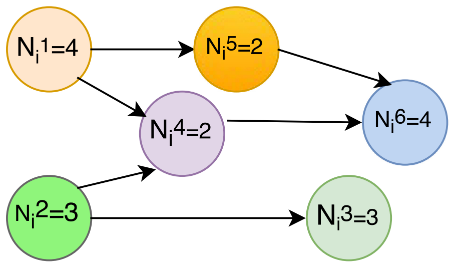


Fig. 1. A DAG τ_i with total execution time $C_i = 18$ and minimum inter-arrival separation $T_i = 12$. It is a heavy task since $C_i > T_i$. The path $\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^4 \rightarrow \mathcal{N}_i^6$ is the critical path with minimum makespan of $L_i = 10 \leq T_i$. Hence, τ_i may meet its deadline provided enough processors.

that energy consumption is further reduced. Section 5 implements gradient-based solvers and compares the proposed method with state-of-the-art schedulers. Section 6 discusses related work and Section 7 concludes the article.

2 BACKGROUND AND SYSTEM MODEL

We consider a multi-core platform of m identical cores to schedule a set of sporadic DAGs. The task set is denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task $\tau_i (1 \leq i \leq n)$ is represented as a DAG with a minimum inter-arrival separation of T_i time units (often referred as the *period*). The *nodes* in a DAG stand for different execution requirements while the *edges* represent the dependencies among the corresponding execution requirements. A parallel task τ_i contains a total of n_i nodes, each denoted by $\mathcal{N}_i^j (1 \leq j \leq n_i)$. The *execution requirement* of node $\mathcal{N}_i^j$ is denoted by c_i^j. A directed edge from node $\mathcal{N}_i^j$ to node $\mathcal{N}_i^k (\mathcal{N}_i^j \rightarrow \mathcal{N}_i^k)$ implies that the execution of $\mathcal{N}_i^k$ cannot start until $\mathcal{N}_i^j$ finishes for every instance (*precedence constraints*). $\mathcal{N}_i^j$, in this case, is called a *parent* of $\mathcal{N}_i^k$, while $\mathcal{N}_i^k$ is a *child* of $\mathcal{N}_i^j$. The *degree of parallelism* M_i of a DAG task τ_i is the number of nodes that can be simultaneously executed.

Each DAG τ_i has an *execution requirement* (i.e., *work*) of C_i which is the sum of the execution requirements of all of its nodes; i.e., $C_i = \sum_{j=1}^{n_i} c_i^j$. A *critical path* of a DAG task is a directed path with the maximum total execution requirements among all other paths in the DAG. For τ_i, the *critical path length*, denoted by L_i, is the sum of execution requirements of the nodes on a critical path. Thus, L_i is the *minimum makespan* of τ_i, meaning that it needs at least L_i time units even when the number of cores m is unlimited.

Any two consecutive instances of task τ_i are separated by at least T_i time units—T_i is also the relative deadline of the task as we only consider *implicit deadlines*. 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. A DAG is *heavy* if its execution requirement is greater than its period (i.e., $C_i > T_i$). A schedule is said to be *feasible* when all sub-tasks (nodes) receive enough execution (up to their execution requirements) within T_i time units from their arrivals, while all precedence constraints are satisfied. The aforementioned terms are illustrated in Figure 1.

Motivating Example: Now we present a simple but effective example that maps an actual application to a DAG which is shown in Figure 2. In this figure, we present the partial work-flow of an *unmanned aerial vehicle (UAV)* which is adopted from Siebert and Teizer (2014). Here, each rectangular box denotes a sub-task (or node in the DAG) with their execution requirements (not shown in the figure) and edges represent data dependencies among them. As described above, a node cannot start execution until all of its parents are completed and some of the nodes can be executed in parallel, e.g., *Satellite* and *Remote Control*.

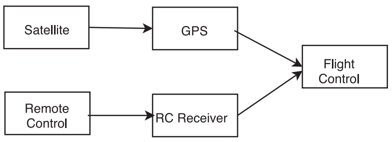


Fig. 2. Partial work-flow of a UAV system represented as a DAG.

Power/Energy Model. Let $s(t)$ (we are assuming continuous frequency scheme) denote the main frequency (speed) of a processor at a certain time t. Then its power consumption $P(s)$ can be modeled as:

$$P(s) = P_{sta} + P_{dyn}(s) = \beta + \alpha s^{\gamma}, \tag{1}$$

where P_{sta} denotes the static power consumption that is introduced in the system due to the leakage current and $P_{dyn}(s)$ denotes the active power consumption. $P_{dyn}(s)$ is introduced due to the switching activities and it depends on the processor frequency. $P_{dyn}(s)$ can be represented as αs^{γ} where the constant $\alpha > 0$ depends on the effective switching capacitance (Pagani and Chen 2013), $\gamma \in [2, 3]$ 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). Clearly, the power consumption function is a convex-increasing function of the processor frequency. By means of dynamic voltage and frequency scaling (DVFS), it is possible to reduce $P_{dyn}(s)$ by reducing the processor frequency. In this article, we focus on minimizing the active energy consumption (due to $P_{dyn}(s)$) by means of DVFS. We also target to minimize the static power consumption (due to P_{sta}) by reducing the number of processors by allowing intra-task processor sharing.

Note that static power consumption is partially related to frequency. It is mainly because highest frequency (that can be used) can be margined by the voltage level that is inversely proportional to the circuit delay. Such relationship is counted towards the dynamic part of the power consumption by adopting a slightly larger γ value. Various platforms may take different values, yet the proposed approach should work in general. Besides, we assume that each core can execute with any frequency ranged between f_{min} to f_{max}, where, $f_{min} \geq$ critical frequencies[1] (Pagani and Chen 2013). Within this allowable frequency range $[f_{min} - f_{max}]$ we consider the upper bound of static energy consumption. Hence we can consider that static energy consumption is fixed w.r.t. the frequency.

Assumptions behind this model: (i) Continuous frequency scheme is considered in this work. However, compared to the system with discrete frequency scheme, this assumption does not affect the applicability of our approach. This is because, at any time, a computed frequency can be rounded up to the next discrete level. Specifically, if the step between the discrete levels is fine-grained, a computed frequency can be rounded up to the next discrete level. State of the art processors provide this feature. For example, ODROID XU-3 (ODROID XU-3 2013) board has a frequency range of 200−1400MHz for the LITTLE cores and 200−2000MHz for the big cores with scale steps of 100MHz. So our approach will provide almost the same energy efficiency in a platform with discrete frequency scheme.

(ii) We assume *per-core speed scaling*, that is each core is able to set its operating frequency independently. Many of the existing platforms (e.g., ODROID XU-3 (2013)) group the cores into a cluster where all cores in the same cluster execute at the same speed. Considering such cluster-based platform makes the scheduling problem significantly complex as it makes the objective function (see Equation (9)) *non-convex*. In future, we aim to address the energy minimization problem considering a cluster-based platform that is more realistic but comes with a tradeoff (complexity). As the objective function will become non-convex in such a platform, it needs to be tackled in

[1]Optimal frequency to reduce the energy consumption (considering a negligible sleeping overhead).

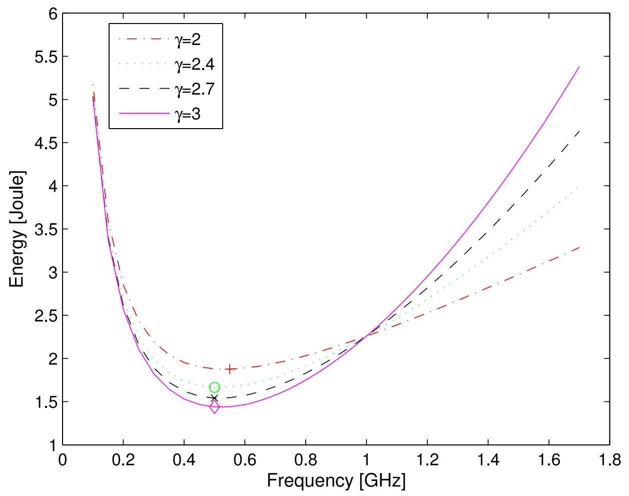


Fig. 3. Energy consumption for executing a job with 10^9 computation cycles under various γ values, where $\alpha = 1.76\text{W}/\text{GHz}^\gamma$ and $\beta = 0.5\text{W}$, according to Equation (2). The lowest points represent the most energy efficient execution frequency (i.e., critical frequency).

an approach that differs significantly from this work. In this work, we have considered a model without such correlation between the cores. However, it still leads to a significant contribution towards energy optimization of scheduling parallel real-time task systems in a multi-core platform. The energy consumption of any given period $[b, f]$ can be calculated as $E(b, f) = \int_b^f P(s(t))dt$, which is known as a nice approximation to the actual energy consumption of many known systems. Specifically, given a unit-amount of workload to be executed on a speed-s processor, the total energy consumption is the integral of power over the period of length $1/s$; i.e.,

$$E(s) = (\beta + \alpha s^\gamma)/s = \beta/s + \alpha s^{\gamma-1}. \tag{2}$$

Figure 3 shows how different values of γ (varying from 2 to 3) and processor speed s may affect the total energy consumption to complete a certain (fixed) amount of computation. It is obvious that execution under a speed much lower than the *critical frequencies* is extremely energy inefficient (as leakage power becomes the major "contribution"). The power model we adapted complies with much existing (and recent) work in the community, e.g., Aydin and Yang (2003), Xu et al. (2005), Devadas and Aydin (2010), Pagani and Chen (2013, 2014), and Guo et al. (2017).

Problem Statement. Given a set of implicit-deadline sporadic parallel tasks to be scheduled on a multi-core platform consisting of enough number of identical cores, we want to determine a *feasible* scheduling strategy, while minimizing the overall power consumption for the assigned processors. Here, by enough, we mean the number of available processors is no smaller than the sum of the max degree of parallelism of the DAG tasks. Energy-optimal scheduling of parallel tasks on multi-cores is NP-hard in the strong sense (Li 2012). Thus we do not expect to solve this energy optimization problem optimally in this article. Instead, we will tackle this problem in the following two steps:

- First, we restrict ourselves under the framework of federated scheduling and follow the existing task decomposition framework (Saifullah et al. 2014) (Section 3), such that the NP-hardness no longer holds. We identify an energy-sub-optimal table-driven scheduler under those additional conditions.

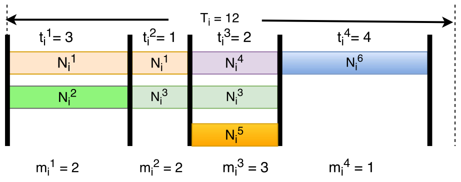


Fig. 4. Scheduling window constraints for all nodes in τ_i (in Figure 1) after task decomposition. Here m_i^k denotes the degrees of parallelism at kth segment and the node-core mapping is $\bar{\mathcal{M}}_i = \{1, 2, 2, 1, 3, 1\}$. Scheduling windows for the nodes are $[1, 2], [1, 1], [2, 3], [3, 3], [3, 3], [4, 4]$ respectively. The average power consumption under such settings is 3.33 Watts after extending each segment by a common factor of $T_i/L_i = 1.2$.

- Then, based on the "sub-optimal" solution, we propose heuristics for merging the assigned processors (Section 4) to further improve the overall energy efficiency when the unnecessary restrictions are removed.

Remark 1. Note that, after task decomposition is performed on any task τ_i, its maximum degree of parallelism M_i can be known (refer to Figure 4). While scheduling τ_i, rather than considering any fixed number of processors, we use M_i as the upper bound of its processor requirement. This upper bound can be further improved by allowing intra-task merging discussed in Section 4.1.

3 ENERGY-SUB-OPTIMAL FEDERATED SCHEDULING FOR DAG TASKS

In this section, we restrict our focus on the federated scheduling of DAG tasks. Under the federated approach to multi-core scheduling, each individual task is either restricted to execute on a single processor (as in partitioned scheduling), or has exclusive access to all the processors on which it may execute. Since each processor is dedicated to one DAG task, we can consider each task individually, and try to minimize the energy consumption for a single DAG task (which is the goal of this section).

Given a DAG task, we first apply the existing task decomposition (Saifullah et al. 2014) technique to transform a parallel task into a set of sequential tasks with scheduling window (for a specific node it denotes the time frame from its release offset to its deadline) constraints for each node (Section 3.1)—they are further relaxed into necessary conditions by segment extension (Section 3.2). By variable substitution, we then transform the energy minimization problem into a convex optimization problem with linear inequality constraints, which can be solved *optimally* with gradient-based methods (Section 3.3).

3.1 Task Decomposition

Task decomposition is a well-known technique that simplifies the scheduling analysis of parallel real-time tasks (Saifullah et al. 2014). In our approach, we adopt task decomposition as the first step that converts each node $\mathcal{N}_i^l$ of the DAG task τ_i to an individual sub-task τ_i^l with a release offset (b_i^l), deadline (f_i^l), and execution requirement (c_i^l). The release time and deadlines are assigned in a way that all dependencies (represented by edges in the DAG) are respected. Thus the decomposition ensures that if all the sub-tasks are schedulable then the DAG is also schedulable. For the sake of completeness, we briefly describe how task decomposition works in this subsection with an example (please refer to Section 4 of Saifullah et al. (2014) for more details).

We adapt a slightly modified version of the approach used in Saifullah et al. (2014). First, we perform the task decomposition using the techniques in Saifullah et al. (2014) as described below. Assuming the execution of the task is on an unlimited number of cores, we draw a vertical line at every time instant where a node starts or ends for each node starting from the beginning. These vertical lines split the DAG into segments, and each segment consists of an equal amount of execution by the nodes that lie in the segment. Parts of different nodes in the same segment can now be considered as threads of execution that run in parallel, and the threads in a segment can start only after those in the preceding segment have finished their executions. Now we will say that the resulting segmented structure of the task is converted into synchronous form and will denote it as τ_i^{syn}. We first allot time to the segments and then add all times assigned to different segments of a node to calculate its allocated time.

Since the minimum makespan, $L_i \leq T_i$, at the end of each period, there may be a slack where all processors are idle (which is typically energy inefficient). We allocate such idle period *uniformly* by multiplying each segment by a common factor of T_i/L_i for task τ_i. Task decomposition provides its processor assignment $\mathcal{M}_i^l$ (i.e., a node-to-processor mapping) and a scheduling window $[b_i^l, f_i^l)$ on top of it, in which each node $\mathcal{N}_i^l$ of a task τ_i will be scheduled. The following example demonstrates how task decomposition works.

Example 3.1. Consider task τ_i shown in Figure 1. First, we assign all the nodes with no parent ($\mathcal{N}_i^1$ and $\mathcal{N}_i^2$) to separate processors. Then we continue to consider nodes only when all its parent node(s) are assigned. As a result, the beginning of the node will be the latest finishing time of its parent(s)—these are boundaries of the segments, denoted by vertical lines in Figure 4. Specifically, if a node has a single parent, we can start to consider the node right after the finishing time of its parent. For example, when $\mathcal{N}_i^2$ is completed, $\mathcal{N}_i^3$ is immediately assigned to the same processor (as $\mathcal{N}_i^2$ is the only parent).

When a node has multiple parents, we consider the parent that has the latest finishing time. The child node may be assigned to the same processor assigned to its parent with the latest finishing time. For example, $\mathcal{N}_i^4$ has two parents $\mathcal{N}_i^1$ and $\mathcal{N}_i^2$ where $\mathcal{N}_i^1$ completes execution later. So $\mathcal{N}_i^4$ is assigned to the same processor of $\mathcal{N}_i^1$. Please note that a node may have multiple siblings such that it may not always share the same processor with its latest finished parent node—under such scenario, a new processor is allocated to the node. For example, the only parent of $\mathcal{N}_i^5$ is $\mathcal{N}_i^1$ which completes execution at t_i^2. So $\mathcal{N}_i^5$ would be able to execute in the same processor starting from the third segment. But $\mathcal{N}_i^5$ is assigned to a different processor as that specific processor at t_i^3 is already "taken" by its sibling $\mathcal{N}_i^4$.

3.2 Segment Extension

For a DAG task τ_i, the aforementioned task decomposition results in a mapping between a node ($\mathcal{N}_i^l$) and a processor ($\mathcal{M}_i^l$). One of the key issues with the task decomposition process is that the identified scheduling window constraints for the nodes may not be necessary. Take the task described in Figure 4 as an example, where Node $\mathcal{N}_i^3$ may execute in the 4th segment. However, task decomposition requires that Node $\mathcal{N}_i^3$ must finish by the end of Segment 3, which is unnecessary. In this subsection, we describe a systematic way of eliminating such unnecessities so that the boundary constraints for all nodes (b_i^l's and f_i^l's) are both *necessary* and *sufficient.*

Each DAG τ_i is first converted to a synchronous form denoted by τ_i^{syn} with techniques described in Section 3.1. We use m_i^k to denote the number of parallel threads in the kth segment of τ_i^{syn}. We then apply Algorithm 1 to greedily extend the deadlines f_i^l of each node $\mathcal{N}_i^l$, following any topological order. Note that while performing task decomposition, a node starts execution immediately when all of its predecessors finish execution. Thus the starting time b_i^l cannot be moved earlier—only f_i's have room to be relaxed.

Task decomposition technique determines scheduling window constraints for the nodes which are *sufficient* but may not be *necessary.* If segment extension is performed (using Algorithm 1) after applying task decomposition, then scheduling window constraints become *necessary* and *sufficient* (see Lemma 1). It may happen that for some particular DAG structure, Algorithm 1 fails to change

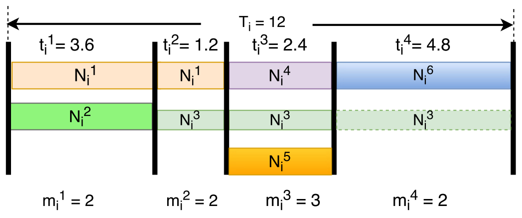


Fig. 5. The segment-node mapping for τ_i (in Figure 1) after segment extension. Scheduling windows for the nodes become [1, 2], [1, 1], [2, 4], [3, 3], [3, 3], [4, 4] respectively, which results in an average power consumption of 3.08W. The height of each block represents the speed of the processor during each segment.

(i.e., expand) scheduling window for any node. However, it does not impact the schedulability of the DAG. From this discussion, it should be clear that performing segment is not mandatory but it is done to further reduce the energy consumption.

Please note that we have considered table-driven schedulers which usually pre-compute which task would run when. This schedule is stored in a table at the time the system is designed or configured. So what would be the size of the scheduling table for a given set of real-time tasks to be run on a system? The answer is when a set of n tasks is to be scheduled, then the entries in the table will replicate themselves after LCM $(T_1, T_2 \ldots T_n)$, where LCM $(T_1, T_2 \ldots T_n)$ is the hyperperiod for the tasks. However, while considering energy consumption we did not consider the space complexity of the scheduling solutions.

Example 3.2. Consider again the DAG task τ_i shown in Figure 1. Our algorithm greedily extends the ending segment f_i^l of the nodes as much as possible in the topological order (i.e., increasing l). Using this approach, Node $\mathcal{N}_i^3$ can now execute in Segment 4 (dashed rectangle at Figure 5) and the execution window for all the other nodes remain unchanged. Please note that the processor assignment $\mathcal{M}_i^l$ for any node $\mathcal{N}_i^l$ of a task τ_i remains unchanged in the segment extension process.

Lemma 1. *Under the task decomposition and scheduling framework, after running Algorithm 1, the timing constraints we set for each node in a DAG become* necessary *and* sufficient.

Proof. First, we show that if task decomposition (done by Saifullah et al. (2014)) is considered the timing constraints set for each node in a DAG is only sufficient. Then we prove that if the segment extension is applied after employing the task decomposition, it makes the timing constraints necessary and sufficient.

On task decomposition, calculated scheduling window for each node satisfies all predecessor constraints, without changing the deadline for the DAG and it completes the proof of sufficient part. Now we prove that the timing constraints are unnecessary by proving a simple example. Consider the task described in Figure 4 again. Task decomposition requires that Node $\mathcal{N}_i^3$ must finish by the end of Segment 3, which is unnecessary, because it may execute in the fourth segment.

Now consider both task decomposition and segment extension are applied to a DAG. In that case, the sufficient part is trivial. The scheduling window satisfies all predecessor constraints, while the deadline for the DAG task does not change.

Assume the window after modification $[b_i^l, f_i^l]$ for some node $\mathcal{N}_i^l$ is not necessary; i.e., it can be further extended. Then it must be one of the following two cases:

- An earlier b_i^l still satisfies all predecessor constraints, which is impossible since it is the time all parents are finished.
- A later f_i^l is possible, which contradicts with Lines 5–7 of Algorithm 1 as it is already the starting point of its child, or the deadline of the whole DAG. □

ALGORITHM 1: Segment Extension

```
Input: A DAG task $\tau_i$, scheduling windows after decomposition $[b_i^l, f_i^l]$ for any node $\mathcal{N}_i^l \in \tau_i$.
Output: Extended segment window $[b_i^l, f_i^l)$ for each node $\mathcal{N}_i^l \in \tau_i$.
Assume that all nodes $\mathcal{N}_i^l$ are ordered topologically, such that predecessor constraint may only occur
    between $\mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}$ when $l < l'$;
for each node $\mathcal{N}_i^l \in \tau_i$ do
    if node $\mathcal{N}_i^l$ has successor node(s); i.e., $\exists l', \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}$ then
        $f_i^l \leftarrow \min_{l' | \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}} \{b_i^{l'}\} - 1$;
    end
    else
        $f_i^l \leftarrow$ last segment of $\tau_i^{syn}$;
    end
end
```

3.3 Problem Transformation

After task decomposition and segment extension, we have identified the scheduling window $[b_i^l, f_i^l]$ for each node $\mathcal{N}_i^l$, and there is no overlap for any two windows (for different nodes) on the same processor. A natural question arises: *Given a specific node (job) with a pre-determined scheduling window on a dedicated processor, what is the most energy-efficient execution (speed) pattern?*

Theorem 2. *The total energy consumption (assuming processor remains on) $\int_a^{a+\Delta} s(t)^\gamma dt$ is minimized in any scheduling window $[a, a+\Delta]$ of length Δ when execution speed remains the same; i.e., $s(t) \equiv C/\Delta$, where $C = \int_a^{a+\Delta} s(t)dt$ is the (given) task demand in the window.*

Proof. We define $p(t) = s(t)/C$, then $p(x)$ is a *probability density function (PDF)* over $[a, a+\Delta]$; i.e.,

$$\int_a^{a+\Delta} p(t)dt = 1. \tag{3}$$

As a result,

$$\begin{aligned}
\int_a^{a+\Delta} s(t)^\gamma dt &= \int_a^{a+\Delta} (C \cdot p(t))^\gamma dt \\
&\quad \{\text{re-arranging}\} \\
&= \frac{C^\gamma}{\Delta^{\gamma-1}} \cdot \left(\frac{1}{\Delta} \int_a^{a+\Delta} (\Delta \cdot p(t))^\gamma \, dt \right) \\
&\quad \{\text{Jensen's Inequality (Chandler 1987), the convexity of function } x^\gamma \\
&\quad \text{when } 2 \le \gamma \le 3 \text{ and } x \ge 0, \text{and } p(x) \text{ being a PDF}\} \\
&\ge \frac{C^\gamma}{\Delta^{\gamma-1}} \left(\int_a^{a+\Delta} p(t)dt \right)^\gamma \\
&\quad \{\text{From Equation (3)}\} \\
&= \frac{C^\gamma}{\Delta^{\gamma-1}} \\
&\quad \{\text{Definition of integrating a constant function}\} \\
&= \int_a^{a+\Delta} \left(\frac{C}{\Delta} \right)^\gamma dt.
\end{aligned}$$

Thus, the minimal total energy consumption in the specified interval $\int_a^{a+\Delta} s(t)^\gamma dt$ can be achieved when speed $s(t)$ remains constant (C/Δ) throughout the interval $[a, a+\Delta]$. □

According to Theorem 2, executing all segments with a uniform speed yields minimum possible power consumption under such framework. Hence we can assume that *the speed of any processor does not change within a segment.* Let S_j^k denote the speed of processor j in the kth segment (executing node $\mathcal{N}_i^l$), and t_i^k denote the length of the segment. The objective is to determine the length of each segment $t_i^k (\geq 0)$ and its execution speed $S_j^k (\geq 0)$ such that total power consumption is minimized.

The first set of constraints guarantees the real-time correctness that each node $\mathcal{N}_i^l$ receives enough execution within its designated window $[b_i^l, f_i^l)$ on its assigned processor $\mathcal{M}_i^l$; i.e.,

$$\forall l, \mathcal{N}_i^l \in \tau_i : \sum_{k=b_i^l}^{f_i^l} t_i^k S_{\mathcal{M}_i^l}^k \geq c_{i,l}. \tag{5}$$

We need one more set of inequalities to bound the total length for all segments of each DAG by its period:

$$\forall i, \sum_k t_i^k \leq T_i. \tag{6}$$

Any non-negative speed assignment and segment length setting that satisfy the constraints described in Equations (5) and (6) yield a *correct* schedule that all nodes receive enough execution in their specified scheduling windows (that satisfy all predecessor constraints). Based on these constraints, we would like to add our objective for minimizing average energy consumption per period:

$$\textbf{Minimize}_{\{t_i^k, S_j^k\}} M_i \beta T_i + \sum_{l=1}^{\zeta_i} \sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (S_{\mathcal{M}_i^l}^k)^\gamma,$$

where M_i is the degree of parallelism (and also the number of processors assigned to the task) and ζ_i is the total number of segments assigned to DAG task τ_i (determined in the previous step). Since the constraints represented in Equation (5) are non-convex quadratic inequalities, it is in general computationally intractable to solve in polynomial time. We transform this problem into a convex optimization by substituting some variables.

Remark 2. According to Theorem 2, executing all segments with a uniform speed yields minimum possible power consumption. If any segment of any core remains idle (scheduling window for any node does not fall at that segment), then we consider that the execution speed for that segment is 0.

Remark 3. In this article, we are assuming that the time required to finish a task is exactly equaled to their WCET. However, it may happen that some of the tasks may finish early than their WCET. In that case, some of the cores (that are assigned to that tasks) may remain idle for some time. In this work, we did not consider that the cores can switch into a deep sleep state during the idle time. Entering into the deep sleep state costs additional energy consumption and it is not beneficial if the idle time slot is less than a certain threshold. By remaining idle we mean that one or more cores are active but not executing any task. It helps to reduce the active power consumption (i.e., $P_d(s)$ in Equation (1)). It would lead to the further minimization of the total power consumption (refer to the Power/Energy model described at Section 2). So our model actually provides the upper bound of the energy consumption.

Replacing speed with period lengths and executions. Fortunately, Theorem 2 provides us the basis to get rid of part of the variables. Since all nodes are executed at *constant* speeds within their scheduling windows, given the total length of each assigned segments (i.e., scheduling window), the execution speed of any given node can be determined. As a result, the energy consumption to finish this node can also be calculated. That is, given a node $\mathcal{N}_i^l$ with total execution requirement of c_i^l, to be executed on segments between b_i^l and f_i^l, we have

$$\forall k \in [b_i^l, f_i^l], S^k_{\mathcal{M}_i^l} = c_i^l / \left(\sum_{j=b_i^l}^{f_i^l} t_i^j \right), \tag{7}$$

which means although a node may be executed in consecutive segments $\forall k \in [b_i^l, f_i^l]$, the speed remains constant throughout the scheduling window and can be represented by a function of executions c_i^l and segment lengths t_i^j. Substituting Equation (7) into the second term of the objective function, we have

$$\begin{aligned}\sum_{l=1}^{\zeta_i} \sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (S^k_{\mathcal{M}_i^l})^\gamma &= \sum_{l=1}^{\zeta_i} \left(\sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (c_i^l)^\gamma \left(\sum_{j=b_i^l}^{f_i^l} t_i^j \right)^{-\gamma} \right)\\ &\{\text{moving unrelated terms out of the summations}\}\\ &= \alpha \sum_{l=1}^{\zeta_i} \left(\left(\sum_{j=b_i^l}^{f_i^l} t_i^j \right)^{-\gamma} \left(\sum_{k=b_i^l}^{f_i^l} t_i^k \right) (c_i^l)^\gamma \right)\\ &\{\text{combining similar terms}\}\\ &= \alpha \sum_{l|\mathcal{M}_i^l=j} c_l^\gamma \left(\sum_{k=b_i^l}^{f_i^l} t_i^k \right)^{1-\gamma}.\end{aligned} \tag{8}$$

Thus, the original optimization problem can be equivalently transformed into the following one with only t_i^k as variables,

$$\begin{gathered}\textbf{Minimize}_{\{t_i^k\}}\ M_i \beta T_i + \alpha \textstyle\sum_{l|\mathcal{M}_i^l=j} c_l^\gamma (\sum_{k=b_i^l}^{f_i^l} t_i^k)^{1-\gamma}\\ \textbf{Subject to } \forall i, \textstyle\sum_k t_i^k \le T_i,\\ \forall i, t_i^k \ge 0.\end{gathered} \tag{9}$$

Lemma 3. *The objective function (according to Equation (9)) is a convex function.*

Proof. Since leakage power consumption remains constant (which is convex), we will prove that the dynamic part of the energy consumption function is convex:

$$E(\tau) = \sum_{1 \le i \le n} C_i^\gamma (<\alpha_i, \tau>)^{1-\gamma}. \tag{10}$$

Here τ refers to a k-dimension positive vector, in which each element is positive and refers to the length of a specific segment of a DAG task. α_i is a binary vector, in which each element $\alpha_{i,j} \in \{0, 1\}$ identifies if the node is selected for the segment. $|\alpha_i| \ge 1$ since at least one segment must be assigned. $<\alpha_i, \tau>$ refers to the inner-product of the two vectors, C_i refers to a non-negative constant, and $\gamma \in [2, 3]$. Thus the energy consumption is modeled as $E(\tau)$ – a function over the time-allocation $\tau \in \mathbb{R}_+^k$.

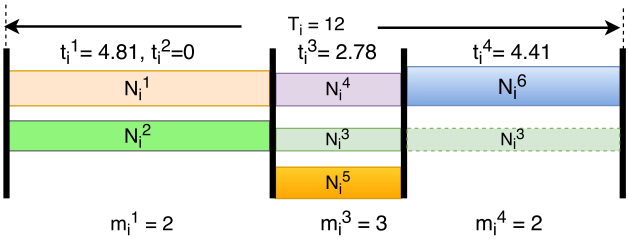


Fig. 6. The sub-optimal segment length assignment for power efficiency of the sample task τ_i (in Figure 1), with an average power consumption of 2.94W. The height of each block represents the speed of the processor during each segment.

We prove the convexity of $E(\tau)$ when $\tau \in \mathbb{R}_+^k$ with the following four steps:

(1) We name $f(\tau) = <\alpha, \tau>$ as a function of inner-product of τ with any binary vector α and $|\alpha| \geq 1$. Obviously, this function is a linear function over τ and should be both *convex* and *concave*. Further, given $\tau \in \mathbb{R}_+^k$, we have $f(\tau) > 0$. Thus we can conclude $f(\tau)$ is a *positive concave* function.
(2) According to page 3-3 of Boyd and Vandenberghe (2004), x^p is convex when $x > 0$ and $p \leq 0$. Thus, when $\gamma \in [2, 3]$ (i.e., $-2 \leq 1 - \gamma \leq -1$) and $x > 0$, the function $g(x) = x^{1-\gamma}$ should be a *non-increasing* convex function.
(3) According to pages 3–17 of Boyd and Vandenberghe (2004), if $g(x)$ is a *non-increasing convex* function and $f(\tau)$ is a *concave* function over $\forall \tau \in \mathbb{R}_+^k$, then $g(f(\tau))$ is a convex function over $\forall \tau \in \mathbb{R}_+^k$.
(4) The function $E(\tau)$ and $f_i(\tau)$ could be written as

$$E(\tau) = \sum_{1 \leq i \leq n} C_i^\gamma g(f_i(\tau)), \tag{11}$$

$$f_i(\tau) = (<\alpha_i, \tau>). \tag{12}$$

As C_i^γ is non-negative, $E(\tau)$ could be considered as the *non-negative-weighted sum* of convex functions (i.e., $g(f_i(\tau))$), and $E(\tau)$ is a *convex function.* □

THEOREM 4. *Any gradient-based method (e.g., fmincon (Fmincon 2018) in Matlab) would lead to sub-optimal power consumption under federated scheduling scheme with task decomposition.*

PROOF. The sub-optimality comes from three facts:

- The objective function is *convex* as it is a sum of several convex (including linear) functions of the variables t_i^k (detailed proof in Lemma 3).
- The linear equality constraints are necessary and sufficient (Lemma 1) for real-time schedulability and predecessor conditions from the input DAG task.
- The variables t_i^k are sufficient to represent a possible optimal scheduler regarding power consumption; i.e., it is safe to assume uniform speed during each segment (Theorem 2). □

Figure 6 shows the sub-optimal segment length assignment for the given task τ_i.

4 PROCESSOR SHARING: EFFICIENCY IMPROVEMENT

Task decomposition transforms the parallel task into a set of sequential tasks. The process tries to maximize the degree of parallelism (i.e., assigning as many processors to each DAG task as possible). However, some of these processors may be lightly loaded with poor energy efficiency as

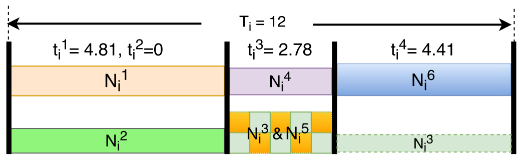


Fig. 7. The execution pattern for τ_i (in Figure 1) after merging Processors 2 and 3, where Nodes $\mathcal{N}_i^3$ and $\mathcal{N}_i^5$ will share Processor 2 (i.e., execute under EDF) within time window $[4.81, 7.59)$ at a higher execution speed. The average power consumption is further reduced to 2.80W. The height of each block represents the speed of the processor during each segment.

the leakage power consumption becomes the majority cost (as demonstrated in Figure 3). Thus the solution derived in Section 3 is only sub-optimal and can be further improved if we allow merging the lightly loaded processors into a single one, such that leakage power is reduced (see Figure 7).

In this section, we deal with this issue and further improve the overall energy efficiency of our scheduler by merging the workloads assigned to different processors onto a single one. Specifically, in Section 4.1, we merge processors that have been assigned to the same DAG task. In this step, each DAG task is handled individually and the resulting processor-node/DAG assignment remains in the federated scheduling framework. However, in this subsection, we have assumed that each processor to be merged only once. That is we only allow the combination of two processors that have never been part of any merging previously. In Section 4.2, this constraint is relaxed and we allow merging three or more processors into one. In Section 4.3, we discuss the importance (to improve the overall energy efficiency of our scheduler) of applying any gradient-based method to calculate the optimal segment length after an intra-DAG processor merging. In Section 4.4, we further extend the technique for inter-DAG processor merging.

Remark 4. Normally, the effect of task migrations and context switches is not considered while deriving schedulability test for real time tasks. We are also ignoring the effects of these phenomena.

4.1 Merging Processors Assigned to the Same DAG

Federated scheduling of DAG tasks provides isolation among tasks during execution, such that inter-task interference as well as context switch delays remain small during runtime. In this subsection, we stay in the federated scheduling framework and only consider potential merges among processors of the *same* DAG.

Given a DAG task τ_i with a federated task-to-processor assignment $j = \mathcal{M}_i^k$, the processor execution speeds S_j^k for each segment, segment lengths t_i^k, and the period T_i. For any processor j assigned to this DAG, its original power consumption can be calculated as

$$P_j = \beta + \sum_k \frac{t_i^k}{T_i} (S_j^k)^\gamma. \tag{13}$$

Any pair of processors $\{j, j'\}$ share the same period and segment information as they are assigned to the same DAG task. As a result, the new execution speed for each segment (when merged together) will simply be the sum of the two old ones; i.e., $S_j^l + S_{j'}^l$, and the average power consumption for this new processor can be calculated as

$$P_{j,j'} = \beta + \sum_k \frac{t_i^k}{T_i} (S_j^k + S_{j'}^k)^\gamma. \tag{14}$$

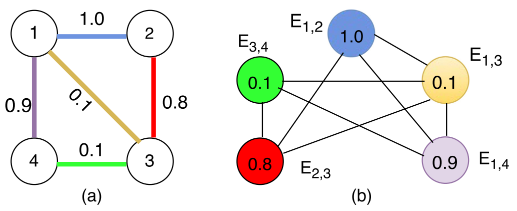


Fig. 8. The equivalence of the MPS problem and the MIS problem, where (a) shows a DAG of four processor assignments with potential power savings for merging each pair of the processors, and (b) shows its equivalent expression with vertices representing all edges in (a) and edges representing the mutual exclusive constraints.

The pairwise potential power saving can be calculated directly by

$$\mathcal{P}_{j,j'} = P_j + P_{j'} - P_{j,j'}. \tag{15}$$

With the pairwise potential power saving, the Maximization of Power Saving (MPS) problem we are dealing with in the section can be described as follows:

- Given the potential power savings ($\mathcal{P}_{M_i \times M_i}$) for merging each pair of the M_i processors, we wish to find a list of mutual exclusive processor-pairs $\{(p_1, p'_1), \ldots, (p_N, p'_N)\} (N \leq M_i/2)$, such that the total power saving $\mathcal{P}_i = \sum_{j=1}^{N} \mathcal{P}_{p_j, p'_j}$ is maximized.

Theorem 5. *The MPS problem is NP-Complete.*

Proof. MPS is in NP as it takes linear time to verify whether a given solution satisfies the mutual exclusion constraints. The NP-Hardness comes from the reduction of a well-known NP-Complete problem: *Maximum Independent Set* (MIS). An independent set is a set of (weighted) nodes in a graph that no two of which are adjacent. For each node in the graph of MIS, we can construct an edge with same weight in the graph of MPS, and adjacency of those edges (whether or not they share a node) in MPS can be determined by the adjacency of the edges in the graph of MIS; i.e., each edge in MIS corresponds to a node in MPS (see Figure 8). Since this polynomial (linear)-time mapping maintains the adjacency relationship of weighted nodes (in MIS) or edges (in MPS), a solution of MIS (a subset of n_m non-adjacent nodes with maximum total weight) will correspond to a solution of MPS (n_m non-adjacent edges with maximum total weight), and vice versa. □

Example 4.1. Take the processor assignment in Figure 8 as an example, where four processors are assigned to a DAG task. The weight $\mathcal{P}_{i,j}$ for each edge represents the potential power saving when merging processors i and j, calculated from Equation (15). The edge $\{2, 4\}$ is missing since merging these two processors will lead to higher power consumption (i.e., $\mathcal{P}_{2,4} < 0$). For each vertex in Figure 8(b), there is a corresponding edge with the same weight in Figure 8(a), and vice versa. A feasible subset of edges in Figure 8(a) (e.g., $\{1, 4\}$ and $\{2, 3\}$) corresponds to a subset of vertices in Figure 8(b) (e.g., $E_{1,4}$ and $E_{2,3}$) that none of the two are directly connected by an edge.

For this example, we could choose to merge Processors 1&2 and 3&4 (with a gain of 1.1W), 1&4 and 2&3 (with a gain of 1.7W) or 1&3 (with a gain of 0.1W). Although obviously the second option is leading to the optimal solution, we need to explore all combinations to find that out (Theorem 5 already shows the intractability). As a result, instead of seeking for the global optimal solution for merging, here we choose to greedily select (see Step 2 below) the pair with the maximum gain in each step.

Now we describe the **key steps of our proposed processor merging method**:

(1) **For** *each* pair of processors $\{j, j'\}$ of the (same) DAG, calculate the potential power savings $\mathcal{P}_{j,j'}$ for merging them together according to Equation (15).
(2) *Greedily* choose the pair $\{j, j'\}$ of processors with the maximum power saving $\mathcal{P}_{j,j'}$, and merge them together by updating $\mathcal{P}'$ value(s) of the nodes on j' to j. The merged nodes will be executed on processor j under EDF, with given per-segment (fixed) speed settings. Note that EDF is an optimal uni-processor scheduler for sporadic task systems, and thus will guarantee temporal correctness as far as cumulative capacity remains the same.
(3) *Remove* the two processors (and also the new one, see Remark 4) from the candidate pool, by updating elements in the jth row, the j'th row, the jth column, and the j'th column of the power saving matrix $\mathcal{P}$ into 0.
(4) **If** there are no positive elements in $\mathcal{P}$, then return the updated mapping $\mathcal{P}'$, **else** go to Step 2 (i.e., merging two un-touched processors may lead to further energy savings).

Although the MIS problem in general cannot be approximated to any constant factor in polynomial time (unless P = NP) (Bazgan et al. 2005), fortunately, each edge in the original figure can be joint with at most $2(M_i - 2)$ other edges, which indicates that the degree of each vertex in the graph after problem transformation is upper bounded by $2(M_i - 2)$. Thus we have the following *approximation ratio bound.*

Theorem 6. *The greedy approach has an approximation ratio no greater than* $(2M_i - 2)/3$, *where* $M_i \geq 3$ *is the total number of processors before merging of DAG task* τ_i; *i.e., the degree of parallelism of the task.*

Proof. Since we only allow a processor to be considered in *one* pair in each round, the graph resulted from the reduction in Theorem 5 is a $(2M_i - 4)$-regular graph; i.e., the degree of each vertex cannot exceed $2M_i - 4$. According to Theorem 5 in Halldórssonz and Radhakrishnan (1997), the greedy algorithm achieves an approximation ratio of $(2M_i - 2)/3$. □

Remark 5. Note that when $M_i = 2$, there are only two processors in the candidate pool, and the decision is straightforward—based on whether merging them can lead to lower power consumption.

Remark 6. As already mentioned in Section 2, we consider a continuous frequency scaling and, at any segment, the computed frequency can be rounded up to the next discrete mode. Initially, it may seem that increasing the speed (by stepping up to the next level) give more room to perform inter-core merging. However, the speed at any segment depends on the workload at this segment and its sub-optimal length which is calculated through a gradient-based method. So rounding up the frequency to its next level may break the sub-optimality and may not yield better energy efficiency.

4.2 Multiple Merging among the Processors Assigned to the Same DAG

In the previous subsection, we have described the technique of merging processors assigned to the same DAG to reduce the total number of processors (by means of merging two lightly loaded processors onto one). So far we have assumed that each processor should be merged only once. However, allowing multiple merging to any specific processor can further increase the energy saving. We are relaxing the assumption of only one merge per processor and in this section, we will describe the technique of multiple merging. Here are the key steps of our proposed intra-DAG processor merging (multiple times) method:

(1) **For** *each* pair of processors $\{j, j'\}$ of the (same) DAG, calculate the potential power savings $\mathcal{P}_{j,j'}$ for merging them together according to Equation (15).
(2) **If** there is no positive element in $\mathcal{P}$, then return with no power saving, **else** go to Step 3.
(3) *Greedily* choose the pair $\{j, j'\}$ of processors with the maximum power saving $\mathcal{P}_{j,j'}$, and merge them together by updating $\mathcal{P}'$ value(s) of the nodes on j' to j. The merged nodes will be executed on processor j under EDF, with given per-segment (fixed) speed settings. Note that EDF is an optimal uni-processor scheduler for sporadic task systems, and thus will guarantee temporal correctness as far as cumulative capacity remains the same.
(4) *Remove* the two processors (and also the new one, see Remark 4) from the candidate pool by updating elements in the jth row, the j'th row, the jth column, and the j'th column of the power saving matrix $\mathcal{P}$ into 0.
(5) **If** there is no positive elements in $\mathcal{P}$, then go to Step 6, **else** go to Step 3 (i.e., merging two un-touched processors may lead to further energy savings).
(6) Let M' be the total number of merges conducted in Steps 1 to 4, where $M' \leq M_i/2$ (M_i = the total number of processor allocated to τ_i). Update M_i as $M_i \leftarrow M_i - M'$.
(7) Repeat Steps 1 to 4.

4.3 Calculating Optimal Segment Length after the Intra-DAG Processor Merging

In Section 3.3, we have discussed the technique to optimally determine the segment length for a DAG task τ_i and in Section 4.1, we have described the processor merging technique among the same DAG task. We already know from Theorem 4 that any gradient-based method would lead to sub-optimal power consumption under federated scheduling scheme with task decomposition. Now we will show that if we re-apply any gradient-based method to calculate segment length after applying the intra-DAG processor merging it will further reduce the power consumption (more details can be found in Section 5). Once we solve the optimization problem mentioned in Section 3.3, for each node $\mathcal{N}_i^l$ we have the information regarding its execution speed within its scheduling window. As all the processors assigned to the same DAG share the same period and segment information, after merging the new execution speed for each segment will simply be the sum of the two old ones. We consider that the DAG τ_i is reduced to τ_i' after a merge. The consecutive time windows where the speed remains same will be considered under the same node (according to Theorem 2). As we have the updated information regarding the execution speed of τ_i' within its scheduling window it is easy to calculate the execution time for each node as well.

Example 4.2. Figure 9 shows how the DAG τ_i from Figure 1 is modified after intra-task processor sharing. For this specific task τ_i (see Figure 7) node $\mathcal{N}_i^3$ and $\mathcal{N}_i^5$ share processor 2. In this case, we will consider that the node $\mathcal{N}_i^3$ and the node $\mathcal{N}_i^5$ jointly form a new node $\mathcal{N}_i^{35}$. The remaining part of node $\mathcal{N}_i^3$ (which is executing within time window [7.59,12)) will be considered as another node.

Now we have a changed DAG τ_i' where each node $\mathcal{N}_i^l$ has some execution requirements and predecessor constraints. After merging we did not change the segment length. As we are merging two different processors into one, the execution requirement at specific segment changes. So it is worth calculating the segment length of the DAG τ_i'. To determine the sub-optimal segment length t_i^k, we will use any *gradient based* (e.g., fmincon (Fmincon 2018) in Matlab) method because:

- The objective function still remains *convex* as it is the sum of multiple convex functions of the variables t_i^k; and
- The linear equality constraints are necessary and sufficient (Lemma 1) for real-time schedulability and predecessor conditions from the *modified* DAG task.

Figure 10 shows the sub-optimal segment length assignment for the modified DAG (which is achieved by merging the lightly loaded processors) τ_i'.

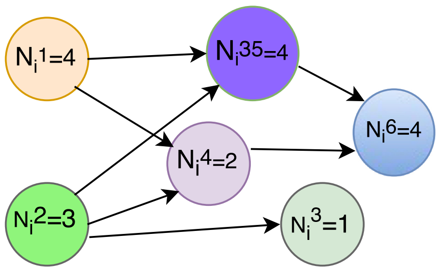


Fig. 9. The modification bought into τ_i (from Figure 1) after merging processors 2 and 3, where nodes $\mathcal{N}_i^3$ and $\mathcal{N}_i^5$ will share processor 2 (i.e., execute under EDF) within time window [4.81, 7.59) at a higher execution speed. Prior to merging, the predecessor constraint was $\mathcal{N}_i^2 \rightarrow \mathcal{N}_i^3$ and $\mathcal{N}_i^1 \rightarrow \mathcal{N}_i^5$. To respect this constraint both the nodes $\mathcal{N}_i^1$ and $\mathcal{N}_i^2$ will be considered as the parent of the newly formed node $\mathcal{N}_i^{35}$.

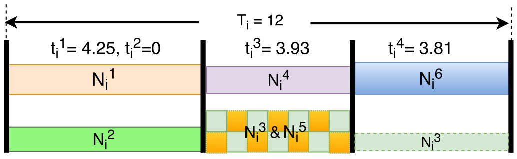


Fig. 10. The sub-optimal segment length assignment (after merging Processors) for power efficiency of the sample task τ_i. Again, the height of each block represents the speed of the processor during each segment.

4.4 Merging Processors Assigned to Different DAGs

The merging process described in the previous subsections may significantly reduce the total number of lightly loaded (energy-inefficient) processors. However, due to the federated scheduling limitation, one (or more) lightly loaded processor(s) for each DAG may still not get paired with just because it (they) cannot find a good "partner" that was assigned the same DAG task. In this subsection, we further select the lightest loaded unmerged processors of each DAG (after intra-DAG merging) as a candidate, and perform inter-DAG merging under a similar approach; i.e., calculate all pairwise energy savings and greedily merge the pairs with maximum possible power saving.

Note that different tasks may have different periods, release patterns (sporadic), and execution patterns (segment lengths after decomposition), such that we cannot simply cumulate the execution speeds with Equation (14) when calculating the new speed pattern for power consumption upon inter-DAG merging. In this section, we propose a fast algorithm to estimate the average energy consumption of the two processors from two different DAG tasks after merging them into one processor, with (potential) unsynchronized release.

With respect to the unknown phase difference between the two DAGs, we assume that all phases are equally likely to occur, and model the speed patterns of them as two random processes $\mathcal{S}_i(t)$ and $\mathcal{S}_j(t)$, where $t \in [0, +\infty)$. The power consumption of the merged processor at time instant t is:

$$e_{i,j}(t) = \beta + \alpha \cdot (\mathcal{S}_i(t) + \mathcal{S}_j(t))^\gamma. \tag{16}$$

The expectation of $e_{i,j}(t)$ over $t \in [0, +\infty)$ is

$$\begin{aligned} \mathbb{E}\left(e_{i,j}(t)\right) &= \beta + \alpha \cdot \mathbb{E}\left((\mathcal{S}_i(t) + \mathcal{S}_j(t))^\gamma\right) \\ &\quad \{\text{The values of } \mathcal{S}_1(t) \text{ and } \mathcal{S}_2(t) \text{ are finite}\} \\ &= \beta + \alpha \sum_{s \in S_{1,2}} s \cdot p(s), \end{aligned} \tag{17}$$

where $S_{1,2}$ is the (finite) set of values that $(\mathcal{S}_1(t) + \mathcal{S}_2(t))^\gamma$ can possibly take, which can be calculated as

$$S_{1,2} = \left\{(s_1^l + s_2^{l'})^\gamma | 1 \le l \le M_1, 1 \le l' \le M_2\right\},$$

and $p(s)$ refers to the probability of the value $s \in S_{1,2}$; i.e.,

$$p(s) = \sum_{1 \le l \le M_1} \sum_{1 \le l' \le M_2} \frac{1}{M_1 M_2} \cdot \delta(s, (s_1^l + s_2^{l'})^\gamma).[2]$$

The key to calculating average power consumption is to identify all the possible execution speeds (sum of a pair of speeds, each of which is selected from the set of possible execution speeds of the two processors being merged) and the likelihood (or joint distribution) of this speed to occur according to the original execution patterns. We calculate the average power consumption, because in a real-time system energy saving scheme for the average behavior (while guaranteeing the worst case requirements) seems more beneficial in terms of energy saving.

Remark 7. Federated scheduling may cause non-negligible resource waste. For example, a heavy task with utilization 1.1 needs at least 2 cores exclusively allocated to it. An approach known as Semi-Federated Scheduling proposed recently in Jiang et al. (2017), to solve the above resource waste problem. We plan to consider this approach for minimizing energy consumption in a multicore platform.

5 SIMULATION STUDY

In this section, we use experiments to evaluate the power efficiency of the proposed mechanisms, and compare them with existing algorithms for DAG task systems.

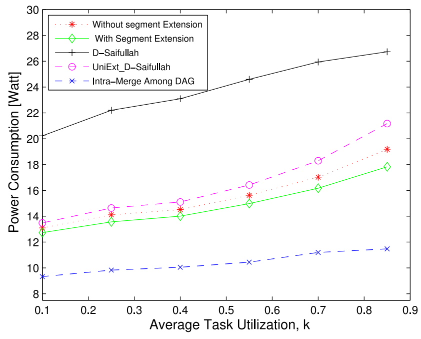
Generation of workloads. Our DAG generator follows the Erdos-Renyi method (Cordeiro et al. 2010) with a given number of nodes. For the *harmonic period* case, the periods are multiples of each other (Saifullah et al. 2014) by enforcing them to be powers of 2. Specifically, we find the smallest value χ such that $L_i \le 2^\chi$ and set T_i to be 2^χ. Regarding the *arbitrary period* case, we use *Gamma distribution* (Gamma Distribution 2018) to generate a random parameter, and set the period as $T_i = L_i + 2(c_i/m)(1 + \Gamma(2,1)/4)$ (according to Saifullah et al. (2014)). Power consumption is calculated using Equation (1) and the value of α, β *and* γ are set to 1.76, 0.5, and 3, respectively.

5.1 Experiment under Single Merging of Processors

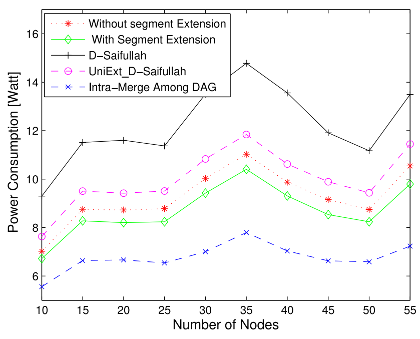
Here compare the power consumption by varying two parameters: (i) task periods (densities) (Section 5.1.1) and (ii) number of nodes in each DAG task (Section 5.1.2). Under each parameter setting, we randomly generate 100 different DAG task sets, each consisting of 5 DAG tasks. Note that, we could report the energy consumption over a hyper-period for each task set. However, reporting energy consumption over a hyper-period for each task set will not be a fair comparison as tasks are randomly generated and their periods and hyper-periods vary a lot. To ensure a fair comparison, we have divided such energy consumption (in Joule) over a hyper-period by the length of the hyper-period (in seconds), such that the reported data represents average power consumption (in watts) on the y-axis. We compare the *average power consumption* of the following scheduling algorithms:

- Federated scheduling with task decomposition, where each node is executed as soon as possible under full speed (Saifullah et al. 2014), denoted by *D-Saifullah*;
- Federated scheduling with task decomposition (length of each segment is further extended uniformly according to their loads) (Saifullah et al. 2014), denoted by *UniExt_D-Saifullah*;

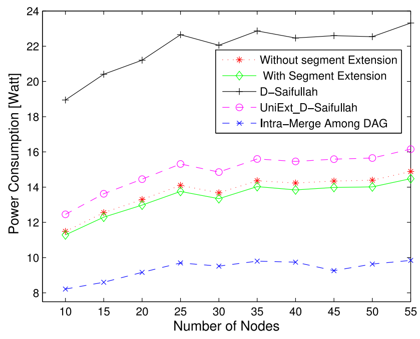
[2] $\delta(s, (s_1^l + s_2^{l'})^\gamma) = 1$ if $s = (s_1^l + s_2^{l'})^\gamma$, and $= 0$ elsewhere.



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Comparison of average power consumption per task set with different approaches for tasks with harmonic periods.



(c) Comparison of average power consumption per task set with different approaches for tasks with arbitrary periods.

Fig. 11. Power consumption comparisons for task sets for various settings under single merging.

- Federated scheduling with task decomposition, where lengths of segments are determined by the proposed convex optimization (Section 3.3);
- Energy-sub-optimal federated scheduling with task decomposition, where lengths of segments are determined by convex optimization (Section 3.3) after performing segment extension (Section 3.2);
- Federated scheduling with intra-DAG processor merging (Section 4.1);

5.1.1 Varying Task Periods (or Utilization). Here we vary the minimum inter-arrival separation for each task, such that the average utilization of a set is controlled. We vary the period in an allowable range ($L_i \leq T_i \leq C_i$) by assigning T_i as $L_i + (1-k)(C_i - L_i)$, where $k \in [0, 1]$ is named as the *utilization* of the task—note that this is different from the normal utilization definition for sequential tasks. We fix the number of nodes within each DAG task as 30, and show the average power consumption in Figure 11(a).

The first thing we notice from Figure 11(a) is that the average energy consumption increases as the average utilization of the set increases (due to decreasing of the period). This phenomenon makes sense as higher utilization would lead to tighter real-time restrictions, which lead to less room for our segment length optimization.

As shown in Figure 11(a), stretching each segment would lead to significant power savings compared to finishing them at full speed and leaving the processor idle for some portion of time (matching Theorem 2). Comparing to the existing uniform stretching for all segments of each DAG task, our convex optimization-based methods would find a better execution pattern in terms of power efficiency. We also found that segment extension is helpful in removing unnecessary constraints for finding better execution patterns.

It is easy to tell that the improvements to the average power consumption are huge when applying the processor merging techniques (see Section 4). The improvement is larger when utilization of the task is high. On average, our proposed methods (including segment extension and intra-DAG merging) are leading to a reduction of the power consumption ranging from 29.2% to 40.5%.

5.1.2 Varying Numbers of Nodes in a DAG Task. Now we vary the number of nodes within each DAG without changing the period. In this set of comparisons, we consider both harmonic (reported in Figure 11(b)) and arbitrary periods (reported in Figure 11(c)) for a set. For each setting of parameters, we randomly generate 100 task sets with various number of nodes (10 to 55, with an increment of 5) and report the average performances of the power consumption over the 100 sets.

First, we observe similar improvements in energy efficiency with the proposed techniques when the number of nodes vary, comparing to the previous set of experiments (with fixed number of nodes and varying task utilization). Specifically, the intra-DAG merging technique (refer to Section 4.1) leads to a reduction in the power consumption for at least 27.29% (34.27%) for harmonic (arbitrary) periods, (compared to the result of convex optimization with segment extension discussed in Section 3.3), while the average power savings are 28.23% and 37.80%.

Second, when comparing curves in Figures 11(b) and (c), we observe that task sets with harmonic periods typically result in lower energy consumption compared to arbitrary periods (under same task utilization and number of nodes per task).

Finally, from the reported performances, we did not observe significant dependencies between the power consumption and the number of nodes for the DAG tasks. This indicates that the proposed methods are *robust* to various settings of parameters and combination of DAG tasks.

5.2 Experiment under Multiple Merging of Processors

In this subsection, we show the improvement in power consumption using our proposed technique of multiple merging among the processors. We compare our results with a simple baseline that was studied in Zhu et al. (2004). This baseline approach was not designed for energy-efficient parallel scheduling. It studied a greedy slack stealing scheduling (GSS) approach for energy minimization for 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. We can consider that approach for scheduling one DAG. As the work in Zhu et al. (2004) did not consider parallel task and we consider this work as a baseline to compare with our work, to provide a fair comparison it is required to modify the power and system models and the graph model used in Zhu et al. (2004).

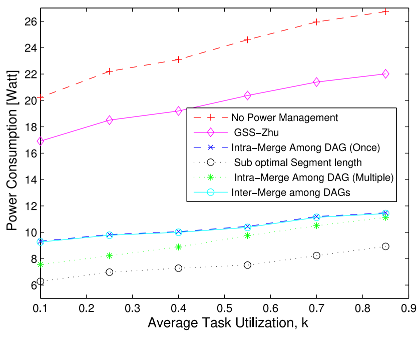
Regarding the power model, first, Zhu et al. (2004) did not consider the processor static power dissipation which is considered by our model (refer to Equation (1)). Second, Zhu et al. (2004) considered two real processor models, (a) *the Transmeta model* and the (b) *Intel XScale model* (refer to Subsection 2.3 in Zhu et al. (2004)). Both of them provide a set of voltage/speed levels. But in our model, we have considered continuous frequency scheme. So while executing the GSS algorithm proposed in Zhu et al. (2004), we use the energy model used in Equation (1). We will assume that whenever a processor is introduced in the system it always remains on. We will also consider minimum inter arrival separation (i.e. period) for a DAG. We make these assumptions to incorporate the static power consumption according to Equation (9).

Regarding the graph model, Zhu et al. (2004) considered three different kinds of vertices: computation nodes, AND nodes, and OR nodes (refer to Subsection 2.1 in Zhu et al. (2004)). Here the computation nodes are labeled by two attributes, c_i and a_i, which denotes the maximum and average computation requirement for the corresponding node. AND nodes and OR nodes do not have any such attributes. An AND node can be executed after all of its predecessors finish execution. Similarly, all of its successors can start execution after it finishes execution. But for the OR nodes, it depends on only one of its predecessors. Similarly only one of its successors depends on this node. However, in our work, we have considered only the computation nodes with only one attribute, their worst-case execution time. To provide a fair comparison, we change the graph model used in Zhu et al. (2004) according to ours. So we will consider the DAG where each node will be considered as a computation node. Instead of two, there will be only one attribute c_i^j which denotes maximum computation requirement for the node $\mathcal{N}_i^j$. We also consider the *precedence constraints* among the computation nodes. Note that we have conducted our evaluation considering the homogeneous platform. However, some nice recent works have considered energy-aware DAG scheduling in heterogeneous platform (Mei et al. 2014) and noble DAG scheduling algorithms (Pathan et al. 2018; Rho et al. 2017). We plan to compare our approach with these works in future. In this subsection, we will evaluate the performance of the technique proposed by us and the technique proposed in Zhu et al. (2004) based on power consumption. Like the previous subsection, we will vary two parameters (i) task periods (utilization) (Section 5.1.1) and (ii) number of nodes in each DAG task (Section 5.1.2) and will use the same set of DAGs. In this subsection, we compare the average power consumption considering the following schemes:

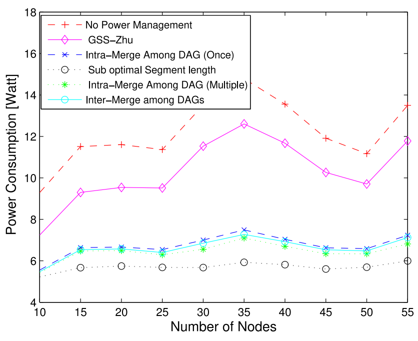
- No Power Management (where every task executes at full speed)
- Greedy Slack Stealing (GSS) algorithm, denoted by *GSS-Zhu*;
- Federated scheduling with intra-DAG processor merging (each processor to be merged only once) (Section 4.1)
- Federated scheduling with intra-DAG processor merging (each processor can be merged multiple times) (Section 4.2)
- Recalculation of the segment lengths later to the intra-DAG processor merging, where lengths of segments are determined by the proposed convex optimization (Section 4.3)
- Shared scheduling with inter-DAG processor merging (Section 4.4);

5.2.1 The Effect of Varying Task Periods (or Utilization). Here we have compared the performance of intra-DAG processor merging (only one merge), intra-DAG processor merging (multiple merges), inter-DAG processor merging, and the recalculation of the segment lengths after applying the intra-DAG processor merging by varying the minimum inter-arrival separation for each task as described in Section 4. We use the same settings, the number of nodes within each DAG task is 30. We show the average power consumption in Figure 12(a).

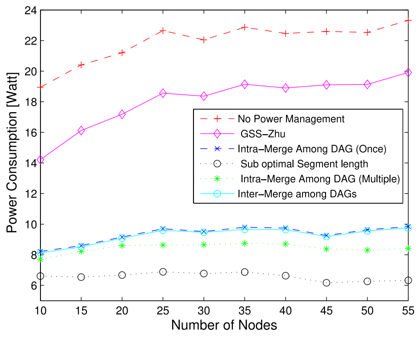
Similar to the phenomenon we have noticed before (Figure 11(a)) the average energy consumption is directly proportional to the average task utilization. The results in Figure 12(a) indicate that our scheduling algorithm is superior to the GSS. Total energy consumption by the intra-DAG processor merging (only one merge) is significantly less than the GSS, which is further reduced by allowing multiple merging and recalculating the optimal segment length after merging. In particular, the energy consumption by the intra-DAG processor merging (only one merge) is at least 44.85% less than the GSS. It is easily observable that the improvements to the average power consumption are huge when our convex optimization-based methods find a better execution pattern after intra-DAG processor merging (only one merge) and the improvement is larger when the utilization of the task is high. This phenomenon makes sense as the objective function described in Equation (9) does not lose its convexity after intra-DAG processor merging. So, re-applying a gradient-based method for determining sub-optimal segment length assignment (after merging



(a) Comparison of power consumption with different approaches for DAGs with a fixed number of nodes as 30.



(b) Comparison of average power consumption per task set with different approaches for tasks with harmonic periods.



(c) Comparison of average power consumption per task set with different approaches for tasks with arbitrary periods.

Fig. 12. Power consumption comparisons for task sets for various settings under multiple merging.

Processors) yields a better result in further reducing the power consumption. On average, it leads to a reduction of the power consumption ranging from 59.41% 63.09%.

5.2.2 Varying Numbers of Nodes in a DAG Task. Now we will show the comparison of the above-mentioned algorithms by varying the number of nodes (from 10 to 55, with an increment of 5). We randomly generate 100 task sets and report the average power consumption over the 100 sets. Again we will consider both harmonic (reported in Figure 12(b)) and arbitrary periods (reported in Figure 12(c)).

Again, the improvements observed in energy efficiency with our proposed techniques under varying number of nodes are similar to that in the previous set of experiments (varying task utilization with a fixed number of nodes). Figures 12(b) and (c) suggest that both versions (single and multiple) of intra-DAG processor merging performs significantly better than the GSS algorithm. Intra-DAG processor merging results in at least 23.3% (single merge) and 25.05% (multiple merges) lower energy consumption compared to the GSS for the harmonic periods. When considering the arbitrary periods, this reduction becomes 42.22% (single merge) and 45.83% (multiple merges). If the gradient-based methods is considered to find a better execution pattern after intra-DAG processor merging (single merge), then it brings further improvements in energy consumption. On average, for the harmonic task periods, it leads to a reduction of the power consumption ranging from 27.8 to 52.94% and for the arbitrary task periods the range is 53.55 to 68.22%. All these results indicate that our proposed techniques outperform the GSS algorithm in terms of energy efficiency.

Finally, Figures 12(b) and (c) report that the task sets with harmonic periods result in lower energy consumption compared to the task sets with arbitrary periods. Also, for the DAG task sets, we have observed that there are no significant dependencies between the energy consumption and the number of nodes.

6 RELATED WORK

The work that deals with schedulability tests for various scheduling policies on parallel task model is already mentioned in Section 1. None of them has considered power/energy consumption issues. In addition, much work has been done in energy/power consumption minimization for sequential tasks. Bini et al. discuss the problem of finding an optimal solution for a system with discrete speed levels for a set of periodic/sporadic tasks (Bini et al. 2009). They have considered both EDF and Fixed-Priority (FP) scheduling policies. Jejurikar has considered non-preemptive tasks to deal with shared resources (Jejurikar 2005). Chen et al. and Liu et al. presents an energy-efficient design for heterogeneous multiprocessor platform in Chen et al. (2009) and Liu et al. (2012), respectively. No previous work considers parallel task model.

For parallel task models, several results are obtained on schedulability tests under various scheduling policies in Baruah et al. (2012), Bonifaci et al. (2013), and Baruah (2014). Bonifaci et al. (2013) prove a speedup bound of $(2 - 1/m)$ for Earliest Deadline First (EDF) and $(3 - 1/m)$ for Deadline Monotonic (DM), respectively, where m is the number of processors. For global EDF scheduling, these techniques are further generalized (Baruah 2014) with an improved pseudo-polynomial time sufficient schedulability test. Analysis of federated and global EDF scheduling is performed in Li et al. (2013, 2014). Processor-speed augmentation bounds for both preemptive and non-preemptive real-time scheduling on multi-core processors are derived in Saifullah et al. (2014). The work in Baruah et al. (2015) studies global EDF scheduling for conditional sporadic DAG tasks, which is an extension to the normal sporadic DAG task model. Certain conditional control-flow constructs (such as *if-then-else* constructs) can be modeled using the conditional sporadic DAG task model. Despite those nice preliminary work on the schedulability analysis of parallel tasks, none of them addresses the energy/power consumption issue.

Actually, intra-task parallelization and power consumption issues have not yet received sufficient attention. Zhu et al. have considered power-aware scheduling for graph-tasks (Zhu et al. 2002). The work in Zhu et al. (2004) proposed the greedy slack stealing algorithm that is able to deal with the task represented by AND/OR graphs. It proves the correctness of the proposed algorithm in terms of meeting the applications time constraint considering it is executing on an N-processor system. Through simulation, it has also analyzed the performance of the algorithm in terms of processor energy saving and showed that the GSS is able to achieve some energy efficiency. However, that work considered the scheduling of only a single DAG and the DAG was not periodic/recurrent. For dependent tasks, Chen et al. (2014) provides techniques that combine dynamic voltage and frequency scaling (DVFS) and dynamic power management, where each core in the platform can be switched on and off individually. For block-partitioned multi-core processors (where cores are grouped into blocks and each block has a common power supply scaled by DVFS), energy efficiency is investigated in Qi and Zhu (2011). The authors in Paolillo et al. (2016) consider power-aware policy for scheduling parallel hard real-time systems, where the multi-thread processing is used. Paolillo et al. (2014) considers dealing with parallel tasks under Gang scheduling policy, where all parallel instances of a task use a processor in the same window. The authors in Xu et al. (2012) have considered energy minimization for frame-based tasks (i.e., same arrival time and a common deadline for all the tasks) with implicit deadlines. Similar frame-based model is considered in Guo et al. (2011), where precedence constraints can be specified among the tasks.

As mentioned previously, no existing work allows intra and inter-task processor sharing when considering the (more general) sporadic DAG task workload model.

7 CONCLUSION

This article studies the scheduling of a set of sporadic DAGs with implicit deadlines. Upon guaranteeing real-time correctness, we try to minimize the overall power consumption of the platform. A power-sub-optimal scheduler is proposed under the condition of federated scheduling and task decomposition. Achieving the optimal solution for the more general (non-federated) case is shown to be NP-Complete. Based on the solution under federated scheduling, a greedy heuristic is proposed to further improve the power efficiency, with proved upper bound of the approximation ratio. To our knowledge, this is the first work in the real-time systems community that considers power issues for scheduling recurrent DAG tasks. This work only considers implicit deadline tasks, while in future, we plan to extend the study for the constrained deadline. Such extension is not trivial, as the processor may have to remain idle due to the gap between the deadline of a job and the release of its successor, leading to energy inefficiency. Also, we plan to validate our algorithmic findings on an actual platform, so that we could have a better understanding of the relationship between the predicted energy savings and the actual measurements on a state-of-the-art platform.

REFERENCES

Hakan Aydin and Qi Yang. 2003. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*. IEEE, 9–pp.

Sanjoy Baruah. 2014. Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems*. IEEE, 97–105.

Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. 2015. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*. IEEE, 222–231.

Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS'12)*. IEEE, 63–72.

Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos. 2005. Completeness in standard and differential approximation classes: Poly-APX- and PTAS-completeness. *Theor. Comput. Sci.* 339, 2–3 (2005), 272–292.

Enrico Bini, Giorgio Buttazzo, and Giuseppe Lipari. 2009. Minimizing CPU energy in real-time systems with discrete speed management. *ACM Trans. Embed. Comput. Syst.* 8, 4 (2009), 31.

Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. 2013. Feasibility analysis in the sporadic DAG task model. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*. IEEE, 225–233.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

David Chandler. 1987. *Introduction to Modern Statistical Mechanics*. Oxford University Press, Oxford, 288.

Gang Chen, Kai Huang, and Alois Knoll. 2014. Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination. *ACM Trans. Embed. Comput. Syst.* 13, 3s (2014), 111.

Jian-Jia Chen, Andreas Schranzhofer, and Lothar Thiele. 2009. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09)*. IEEE, 1–12.

Daniel Cordeiro, Gregory Mouni, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frederic Wagner. 2010. Random graph generation for scheduling simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, 60.

Vinay Devadas and Hakan Aydin. 2010. Coordinated power management of periodic real-time tasks on chip multiprocessors. In *Proceedings of the 2010 International Green Computing Conference*. IEEE, 61–72.

Nathan Fisher, Jian-Jia Chen, Shengquan Wang, and Lothar Thiele. 2009. Thermal-aware global real-time scheduling on multicore systems. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium, (RTAS'09)*. IEEE, 131–140.

Fmincon. 2018. Retrieved from https://www.mathworks.com/help/optim/ug/fmincon.html.

Gamma Distribution. 2018. Retrieved from https://en.wikipedia.org/wiki/Gamma_distribution.

Yifeng Guo, Dakai Zhu, and Hakan Aydin. 2011. Reliability-aware power management for parallel real-time applications with precedence constraints. In *Proceedings of the 2011 International Green Computing Conference and Workshops (IGCC'11)*. IEEE, 1–8.

Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. 2017. Energy-efficient multi-core scheduling for real-time DAG tasks. In *Proceedings of the LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Magnús Halldórssonz and Jaikumar Radhakrishnan. 1997. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica* 18, 1 (1997), 145–163.

Ravindra Jejurikar. 2005. Energy aware non-preemptive scheduling for hard real-time systems. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems (ECRTS'05)*. IEEE, 21–30.

Xu Jiang, Nan Guan, Xiang Long, and Wang Yi. 2017. Semi-federated scheduling of parallel real-time tasks on multiprocessors. *arXiv Preprint arXiv:1705.03245* (2017).

Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2013. Outstanding paper award: Analysis of global EDF for parallel tasks. In *Proceedings of the 25th Euromicro Conference on Real-Time Systems*. IEEE, 3–13.

Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. 2014. Analysis of federated and global scheduling for parallel real-time tasks. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems*. IEEE, 85–96.

Keqin Li. 2012. Energy efficient scheduling of parallel tasks on multiprocessor computers. *J. Supercomput.* 60, 2 (2012), 223–247.

Cong Liu, Jian Li, Wei Huang, Juan Rubio, Evan Speight, and Xiaozhu Lin. 2012. Power-efficient time-sensitive mapping in heterogeneous systems. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 23–32.

Jing Mei, Kenli Li, and Keqin Li. 2014. Energy-aware task scheduling in heterogeneous computing environments. *Cluster Comput.* 17, 2 (2014), 537–550.

Sujay Narayana, Pengcheng Huang, Georgia Giannopoulou, Lothar Thiele, and R. Venkatesha Prasad. 2016. Exploring energy saving for mixed-criticality systems on multi-cores. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'16)*. IEEE, 1–12.

ODROID XU-3. 2013. Retrieved from http://www.hardkernel.com/main/main.php.

Santiago Pagani and Jian-Jia Chen. 2013. Energy efficient task partitioning based on the single frequency approximation scheme. In *Proceedings of the IEEE 34th Real-Time Systems Symposium (RTSS'13)*. IEEE, 308–318.

Santiago Pagani and Jian-Jia Chen. 2014. Energy efficiency analysis for the single frequency approximation (SFA) scheme. *ACM Trans. Embed. Comput. Syst.* 13, 5s (2014), 158.

Antonio Paolillo, Joël Goossens, Pradeep M. Hettiarachchi, and Nathan Fisher. 2014. Power minimization for parallel real-time systems with malleable jobs and homogeneous frequencies. In *Proceedings of the IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 1–10.

Antonio Paolillo, Paul Rodriguez, Nikita Veshchikov, Joël Goossens, and Ben Rodriguez. 2016. 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*. ACM, 329–338.

Risat Pathan, Petros Voudouris, and Per Stenström. 2018. Scheduling parallel real-time recurrent tasks on multicore platforms. *IEEE Trans. Parallel Distrib. Syst.* 29, 4 (2018), 915–928.

Xuan Qi and Dakai Zhu. 2011. Energy efficient block-partitioned multicore processors for parallel applications. *J. Comput. Sci. Technol.* 26, 3 (2011), 418–433.

Jaeyong Rho, Takuya Azumi, Mayo Nakagawa, Kenya Sato, and Nobuhiko Nishio. 2017. Scheduling parallel and distributed processing for automotive data stream management system. *J. Parallel Distrib. Comput.* 109 (2017), 286–300.

Abusayeed Saifullah, David Ferry, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. 2014. Parallel real-time scheduling of DAGs. *IEEE Trans. Parallel Distrib. Syst.* 25, 12 (2014), 3242–3252.

Sebastian Siebert and Jochen Teizer. 2014. Mobile 3D mapping for surveying earthwork projects using an unmanned aerial vehicle (UAV) system. *Autom. Construct.* 41 (2014), 1–14.

Huiting Xu, Fanxin Kong, and Qingxu Deng. 2012. Energy minimizing for parallel real-time tasks based on level-packing. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 98–103.

Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. 2005. Energy-efficient policies for embedded clusters. In *ACM SIGPLAN Notices*, Vol. 40. ACM, 1–10.

Dakai Zhu, Nevine AbouGhazaleh, Daniel Mossé, and Rami Melhem. 2002. Power aware scheduling for AND/OR graphs in multiprocessor real-time systems. In *Proceedings of the International Conference on Parallel Processing, 2002*. IEEE, 593–601.

Dakai Zhu, Daniel Mosse, and Rami Melhem. 2004. Power-aware scheduling for AND/OR graphs in real-time systems. *IEEE Trans. Parallel Distrib. Syst.* 15, 9 (2004), 849–864.

Received December 2017; revised May 2018; accepted July 2018