6-1-2017

Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks

Zhishan Guo
Missouri University of Science and Technology, guozh@mst.edu

Ashikahmed Bhuiyan

Abusayeed Saifullah
Missouri University of Science and Technology, saifullaha@mst.edu

Nan Guan

et. al. For a complete list of authors, see http://scholarsmine.mst.edu/comsci_facwork/404

Recommended Citation

Z. Guo et al., "Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks," *Leibniz International Proceedings in Informatics, LIPIcs*, vol. 76, pp. 221-2221, Dagstuhl Research Online Publication Server, Jun 2017.
The definitive version is available at http://dx.doi.org/10.4230/LIPIcs.ECRTS.2017.22

Energy-Efficient Multi-Core Scheduling for Real-Time DAG Tasks

Zhishan Guo[1], Ashikahmed Bhuiyan[2], Abusayeed Saifullah[3], Nan Guan[4], and Haoyi Xiong[5]

1 **Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA**
guozh@mst.edu
2 **Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA**
abvn2@mst.edu
3 **Department of Computer Science, Wayne State University, Detroit, MI, USA**
saifullah@wayne.edu
4 **Department of Computing, Hong Kong Polytechnic University, Hong Kong**
csguannan@comp.polyu.edu.hk
5 **Department of Computer Science, Missouri University of Science and Technology, Rolla, MO, USA**
xiongha@mst.edu

Abstract

In this work, we study 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 the best of 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. We first 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 and to prove the bound of the approximation ratio. We perform a simulation study to demonstrate the effectiveness and efficiency of the proposed scheduling. The simulation results show that our algorithms achieve an energy saving of 27% to 41% compared to existing DAG task schedulers.

1998 ACM Subject Classification D.4.7 Real-Time Systems and Embedded Systems

Keywords and phrases parallel task, real-time scheduling, energy minimization, convex optimization

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.22

1 Introduction

Energy consumption remains the cornerstone in designing embedded systems which are mostly battery-operated. 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. In order to better utilize 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 [25] has shown that

the energy consumption of executing certain workload perfectly distributed in two cores is significantly less than that of executing the same workload in one core at double frequency.

In this paper, we deal with tasks that are represented as DAGs – that are considered to be the most generalized model of deterministic parallel tasks. For such task models, several results are obtained on schedulability tests under various scheduling policies in [4] [7] [2]. Bonifaci et al. [7] 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 [2] with an improved pseudo-polynomial time sufficient schedulability test. Analysis of federated and global EDF scheduling is performed in [21] [22]. Processor-speed augmentation bounds for both preemptive and non-preemptive real-time scheduling on multi-core processors is derived in [30]. The work in [3] 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 of those nice preliminary work on the schedulability analysis of parallel tasks, none of them addresses the energy/power 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 [14] [26] [25] [24]. Specifically, [25] and [26] present an energy efficient task partitioning scheme, where the cores are grouped in frequency islands. The authors in [1] 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. Till date, only little work has been done for energy-aware real-time scheduling of parallel task models. 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 paper, 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 may lead to a non-negligible power consumption. In order to reduce this effect, we also allow intra-task processor sharing if any core is lightly loaded. First, it will balance the load among the cores and reduce idle time. Second, the required number of cores to schedule a task can be reduced. 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* in this paper.

- We propose a multi-processor scheduling algorithm along with the power consumption issues for sporadic DAG tasks with implicit deadlines.
- 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).

- We further allow merging of processors that have been assigned to the same DAG task. We also 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.
- Simulations are conducted to verify the theoretical results and demonstrate the effectiveness of our algorithm.

The rest of this paper is organized as follows. Section 2 presents the system model and formally defines our problem. Section 3 adapts the task decomposition scheme that transfers parallel DAG tasks into sequential ones and describes our (sub-)optimal federated scheduler based on segment extension and problem transformation (into a convex optimization with linear inequality constraints). Section 4 presents and analyzes the techniques for intra-DAG processor sharing. Section 5 implements gradient based solvers and compares the proposed method with other state-of-the-art schedulers. Section 6 discusses related work and Section 7 concludes the paper.

2 Background and System Model

We consider a multi-core platform of m identical cores to schedule a set of sporadic parallel tasks. The task set is denoted by $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$, 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 is 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 task is *heavy* if it will miss its deadline when all nodes are run sequentially on a processor. 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.

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:

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

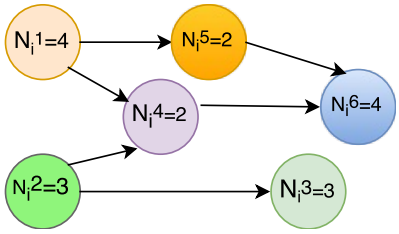

Figure 1 A DAG τ_i with total execution time $C_i = 18$ and minimum inter-arrival separation $T_i = 12$. It is a heavy task since $C_i > T_i$. The path $\mathcal{N}_i^1 \to \mathcal{N}_i^4 \to \mathcal{N}_i^6$ is the critical path with minimum makespan of $P_i = 10 \leq T_i$. As a result, this task may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

where P_s denotes the static power consumption which is introduced in the system due to the leakage current and $P_d(s)$ denotes the active power consumption. $P_d(s)$ is introduced due to the switching activities and it depends on the processor frequency. $P_d(s)$ can be represented as αs^γ where the constant $\alpha > 0$ depends on the effective switching capacitance [25], $\gamma \in [2, 3]$ 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 $P_d(s)$ by reducing the processor frequency. In this paper, we focus on minimizing the active energy consumption (due to $P_d(s)$) by means of DVFS. We also target to minimize the static power consumption (due to P_s) by reducing the number of processors by allowing intra-task processor sharing.

The energy consumption of any given period $[b, f]$ can be calculated as $E = \int_b^f P(t)\, dt$, which is known as a nice approximation to the actual 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., $E(C, s) = (\beta + \alpha s^\gamma)(C/s) = \beta C/s + \alpha C s^{\gamma-1}$. Figure 2 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 amount of computation. It is obvious that execution under a speed much lower than the *critical frequencies* [25] (the highlighted most energy efficient execution speed) 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., [1, 33, 32, 13, 25, 26].

Problem Statement. Given a set of implicit-deadline sporadic parallel tasks to be scheduled on a multi-core platform consisting of enough[1] number of identical cores, we want to determine a *feasible* scheduling strategy, while minimizing the overall power consumption for the assigned processors.

Energy-optimal scheduling of parallel tasks on multi-cores is NP-hard in the strong sense [23]. Thus we do not expect to solve this energy optimization problem optimally in this paper. Instead, we will tackle this problem in the following two steps:

- First, we put additional constraints of federated scheduling and follow the existing task decomposition framework [30] (Sec 3), such that the NP-hardness no longer holds. We identify an energy-sub-optimal table-driven scheduler under those additional conditions.

[1] By enough, we mean the number of available processors is no smaller than the sum of max degree of parallelism of the DAG tasks.

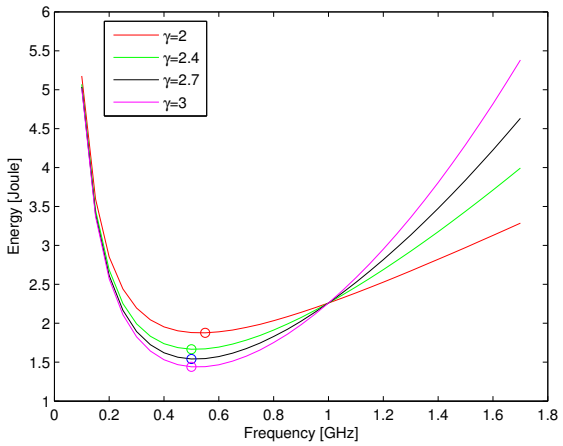

Figure 2 Energy consumption for executing a job with 10^9 computation cycles under various γ values, where $\alpha = 1.76$ W$atts/$GHz^{γ} and $\beta = 0.5$ W$atts$.

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

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 [30] technique to transform a parallel task into a set of sequential tasks with scheduling window [2] constraints for each node (Subsection 3.1) – they are further relaxed into necessary conditions by segment extension (Subsection 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 (Subsection 3.3).

3.1 Task Decomposition

Task decomposition is a well-known technique that simplifies the scheduling analysis of parallel real-time tasks [30]. 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

[2] Scheduling window for a specific node denotes the time frame from its release offset to its deadline.

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 [30] for more details).

We adapt a slightly modified version of the approach used in [30]. First, we perform the task decomposition using the techniques in [30] 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 $P_i \leq T_i$, 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/P_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 1.** Consider task τ_i shown in Figure 1. First of all, 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 3. 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 3 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 Sec. 3.1. We use m_i^k to denote the number of parallel threads in the k-th segment

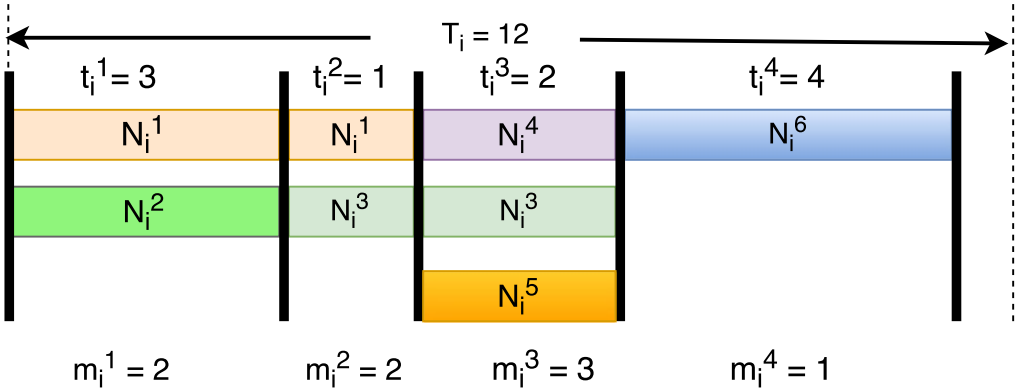

Figure 3 Scheduling window assignments to the nodes of τ_i (in Figure 1) after task decomposition, where m_i^k denotes the degrees of parallelism at k-th segment and the node-processor mapping is: $\bar{\mathcal{M}}_i = \{1, 2, 2, 1, 3, 1\}$, and scheduling windows for the nodes are $[1, 2], [1, 1],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

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.

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 $(T_1, T_2 \cdots T_n)$, where LCM $(T_1, T_2 \cdots T_n)$ is the hyper-period for the tasks. However, while considering energy consumption we did not consider the space complexity of the scheduling solutions.

Algorithm 1 Segment Extension

1: **Input:** A DAG task τ_i, scheduling windows after decomposition $[b_i^l, f_i^l]$ for any node $\mathcal{N}_i^l \in \tau_i$.
2: **Output:** Extended segment window $[b_i^l, f_i^l)$ for each node $\mathcal{N}_i^l \in \tau_i$.
3: 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 $l < l'$.
4: **for** each node $\mathcal{N}_i^l \in \tau_i$ **do**
5: **if** node $\mathcal{N}_i^l$ has successor node(s); i.e., $\exists l', \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}$
6: **then** $f_i^l \leftarrow \min_{l' | \mathcal{N}_i^l \longrightarrow \mathcal{N}_i^{l'}} \{b_i^{l'}\} - 1$;
7: **else** $f_i^l \leftarrow$ last segment of τ_i^{syn};
8: **end for**
9: **return** $[b_i^l, f_i^l]$ for each node $\mathcal{N}_i^l$.

▶ **Example 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 4) 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.

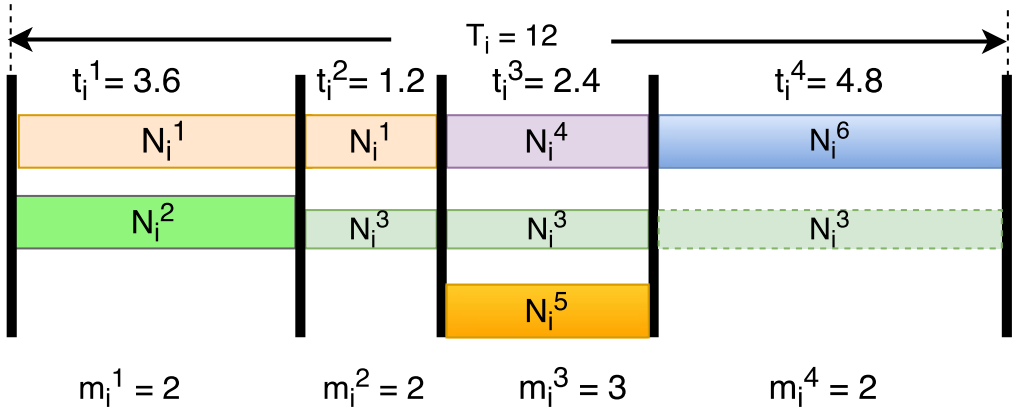

Figure 4 The segment-node mapping for τ_i (in Figure 1) after segment extension. Scheduling windows for the nodes become $[1,2],[1,1],[2,\mathbf{4}],[3,3],[3,3],[4,4]$ respectively, which results in an average power consumption of 3.08 Watts. The height of each block represents the speed of the processor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

▶ **Lemma 3.** *Under the task decomposition and scheduling framework, after running Algorithm 1 (Segment Extension), the timing constraints we set for each node in a DAG become* necessary *and* sufficient*.*

Proof. The sufficient part is trivial. The scheduling window satisfies all predecessor constraints, while the deadline of 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. ◀

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 predetermined scheduling window on a dedicated processor, what is the most energy-efficient execution (speed) pattern?*

▶ **Theorem 4.** *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 $p(x) = s(t)/C$, then $p(x)$ is a *probability density function (PDF)* over $[a, a+\Delta]$; i.e.,

$$\int_a^{a+\Delta} p(t) \, dt = 1. \tag{2}$$

As a result,

$$\begin{aligned}
&\int_a^{a+\Delta} s(t)^\gamma \, dt = \int_a^{a+\Delta} (C \cdot p(t))^\gamma \, dt \\
&\quad \{\text{re-arranging}\} \\
&\quad = \frac{C^\gamma}{\Delta^{\gamma-1}} \cdot \left(\frac{1}{\Delta} \int_a^{a+\Delta} (\Delta \cdot p(t))^\gamma \, dt \right) \\
&\quad \{\text{Jensen's Inequality [9], the convexity of function } x^\gamma \\
&\quad \text{when } 2 \le \gamma \le 3 \text{ and } x \ge 0, \text{ and } p(x) \text{ being a PDF}\} \\
&\quad \ge \frac{C^\gamma}{\Delta^{\gamma-1}} \left(\int_a^{a+\Delta} p(t) \, dt \right)^\gamma \\
&\quad \{\text{From (2)}\} \\
&\quad = \frac{C^\gamma}{\Delta^{\gamma-1}} \\
&\quad \{\text{Definition of integrating a constant function}\} \\
&\quad = \int_a^{a+\Delta} \left(\frac{C}{\Delta} \right)^\gamma \, dt.
\end{aligned} \tag{3}$$

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 4, 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 k-th 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 $t_i^k (\ge 0)$ and its execution speed $S_j^k (\ge 0)$ 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.,

$$\forall l, \mathcal{N}_i^l \in \tau_i : \sum_{k=b_i^l}^{f_i^l} t_i^k S_{\mathcal{M}_i^l}^k \ge c_{i,l}. \tag{4}$$

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

$$\forall i, \sum_k t_i^k \le T_i. \tag{5}$$

Any non-negative speed assignment and segment length setting that satisfy the constraints described in (4) and (5) 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:

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

where M_i is the degree of parallelism (and also the number of processors assigned to the task) and n_i is the total number of segments assigned to DAG task τ_i (determined in the previous step).

Since the constraints represented in (4) 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. *According to Theorem 4, 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), we simply consider that the execution speed for that segment is 0.*

▶ Remark. *In this paper, we are assuming that the time required to finish a task is exactly equaled to their worst-case execution time (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. It would lead to the further minimization of the active power consumption (please see 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 4 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. I.e., 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:

$$\forall k \in [b_i^l, f_i^l], S^k_{\mathcal{M}_i^l} = c_i^l / (\sum_{j=b_i^l}^{f_i^l} t_i^j), \tag{6}$$

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 (6) into the second term of the objective function, we have:

$$\begin{aligned}
&\sum_{l=1}^{n_i} \sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (S^k_{\mathcal{M}_i^l})^\gamma = \sum_{l=1}^{n_i} \left(\sum_{k=b_i^l}^{f_i^l} t_i^k \alpha (c_i^l)^\gamma (\sum_{j=b_i^l}^{f_i^l} t_i^j)^{-\gamma} \right) \\
&\quad \{\text{moving unrelated terms out of the summations}\} \\
&\quad = \alpha \sum_{l=1}^{n_i} \left((\sum_{j=b_i^l}^{f_i^l} t_i^j)^{-\gamma} (\sum_{k=b_i^l}^{f_i^l} t_i^k)(c_i^l)^\gamma \right) \\
&\quad \{\text{combining similar terms}\} \\
&\quad = \alpha \sum_{l | \mathcal{M}_i^l = j} c_l^\gamma (\sum_{k=b_i^l}^{f_i^l} t_i^k)^{1-\gamma}.
\end{aligned} \tag{7}$$

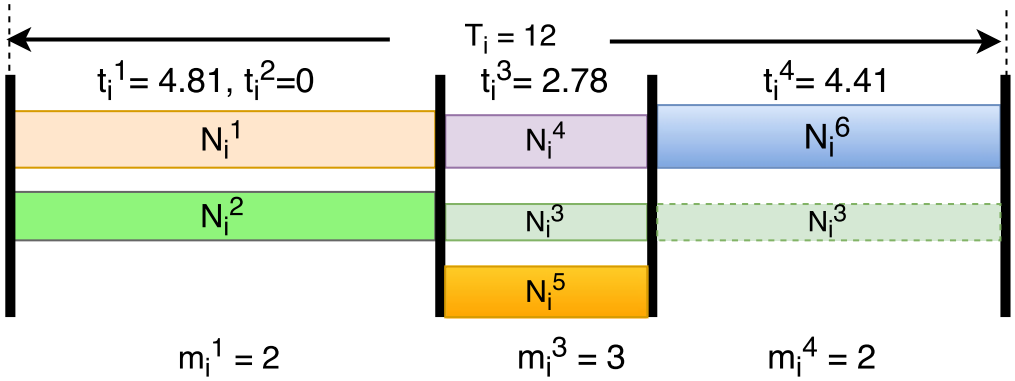

Figure 5 The sub-optimal segment length assignment for power efficiency of the sample task τ_i (in Figure 1), with an average power consumption of 2.94 Watts. The height of each block represents the speed of the processor during each segment.

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

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

▶ **Theorem 5.** *Any gradient based method (e.g., fmincon [15] 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 Appendix A).
- The linear equality constraints are necessary and sufficient (Lemma 3) 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 4). ◀

Figure 5 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 the leakage power consumption becomes the majority cost (as demonstrated in Figure 2). Thus the solution derived in Sec 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 6 as an example.

In this section, we try to 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 Subsection 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.

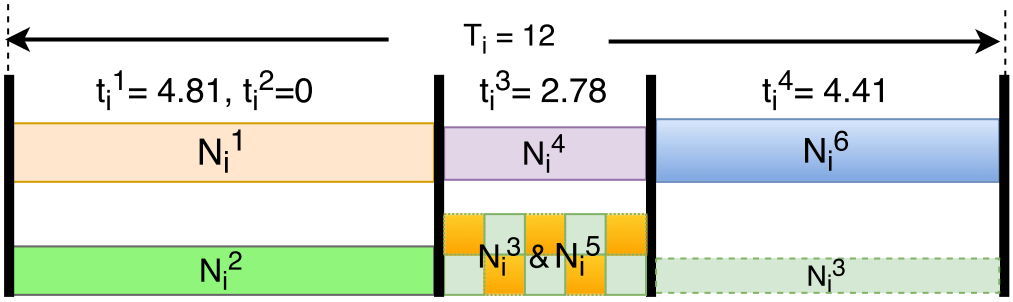

Figure 6 The execution pattern for τ_i (in Fig. 1) after merging Processors 2 and 3, where Nodes $\mathcal{N}_i^3$ and $\mathcal{N}_i^5$ will share Processor 2 (i.e., execute under EDF) within time window $[4.81, 7.59)$ at a higher execution speed. The average power consumption is further reduced to 2.80 Watts. The height of each block represents the speed of the processor during each segment.

▶ Remark. *In practice we have found that once a merge is performed, it is very likely that the new processor becomes quite heavily loaded. As a result, merging a third processor into any merged pair rarely leads to further energy saving. Thus* we only allow the combination of two processors that have never been part of any merging previously. *We plan to consider merging 3 or more processors into one in future work.*

▶ Remark. *In this paper we allow each processor to be merged only once. So the number of context switches is at most 1 per segments. If there are total n_i nodes in task τ_i then the number of context switches is at most $n_i/2$. Normally, the effect of task migrations and context switches is not considered while deriving schedulability test for real time tasks. We are also not considering 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 run-time. 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

$$P_j = \beta + \sum_k \frac{t_i^k}{T_i} (S_j^k)^\gamma. \tag{8}$$

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:

$$P_{j,j'} = \beta + \sum_k \frac{t_i^k}{T_i} (S_j^k + S_{j'}^k)^\gamma. \tag{9}$$

The pairwise potential power saving can be calculated directly by:

$$\mathcal{P}_{j,j'} = P_j + P_{j'} - P_{j,j'}. \tag{10}$$

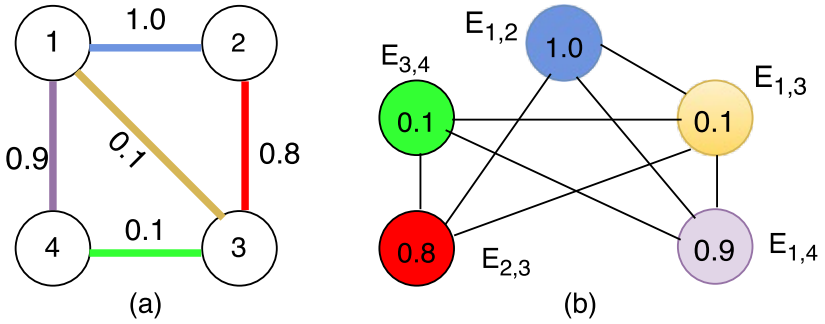

Figure 7 The equivalence of the MPS problem and the MIS problem, where (a) shows a DAG of four processor assignments with potential power savings for merging each pair of the processors, and (b) shows its alternative (equivalent) expression with vertices representing all edges in (a), and edges representing the mutual exclusive constraints.

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 $\{(p_1, p'_1), ..., (p_N, p'_N)\}(N \leq M_i/2)$, such that the total power saving $\mathcal{P}_i = \sum_{j=1}^{N} \mathcal{P}_{p_j, p'_j}$ is maximized.

▶ **Theorem 6.** *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 from a well known NP-Complete problem: *Maximum Independent Set* (MIS). An independent set is a set of (weighted) vertices in a graph that no two of which are adjacent. For each vertex in the graph of MIS, we can construct an edge with the same weight in the graph of MPS, and the adjacency of those edges (whether or not they share a common vertex) 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 vertex in MPS (see Figure 7 for an illustration). Since this polynomial (linear)-time mapping maintains the adjacency relationship of weighted vertices (in MIS) or edges (in MPS), a solution of MIS (a subset of n_m non-adjacent vertices with maximum total weight) will correspond to a solution of MPS (n_m non-adjacent edges with maximum total weight), and vice versa. ◀

▶ **Example 7.** Take the processor assignment in Figure 7 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 (10). 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 7 (b), there is a corresponding edge with the same weight in Figure 7 (a), and vice versa. A feasible subset of edges in Figure 7 (a) (e.g., $\{1, 4\}$ and $\{2, 3\}$) corresponds to a subset of vertices in Figure 7 (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 1.1 Watts), 1&4 and 2&3 (with a gain of 1.7 Watts), or 1&3 (with a gain of 0.1 Watts). Although obviously the second option is leading to the optimal solution, we need to explore all combinations to find that out (Theorem 6 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 (10).
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 is no positive elements in $\mathcal{P}$, 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) [5], 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 8.** *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*[3] *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 6 is a $(2M_i - 4)$-regular graph; i.e., the degree of each vertex cannot exceed $2M_i - 4$. According to Theorem 5 in [18], the greedy algorithm achieves an approximation ratio of $(2M_i - 2)/3$. ◀

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 [12] with a given number of nodes. For the *harmonic period* case, the periods are multiples of each other [30] by enforcing them to be powers of 2. Specifically, we find the smallest value α such that $L_i \leq 2^\alpha$ and set T_i to be 2^α. Regarding the *arbitrary period* case, we use *Gamma distribution* [16] 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 [30]).

We compare the power consumption by varying two parameters: (i) task periods (densities) (Sec. 5.1) and (ii) number of nodes in each DAG task (Sec. 5.2). Under each parameter setting, we randomly generate 100 different DAG task sets, each consisting of 5 DAG tasks, and 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 [30];

[3] 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.

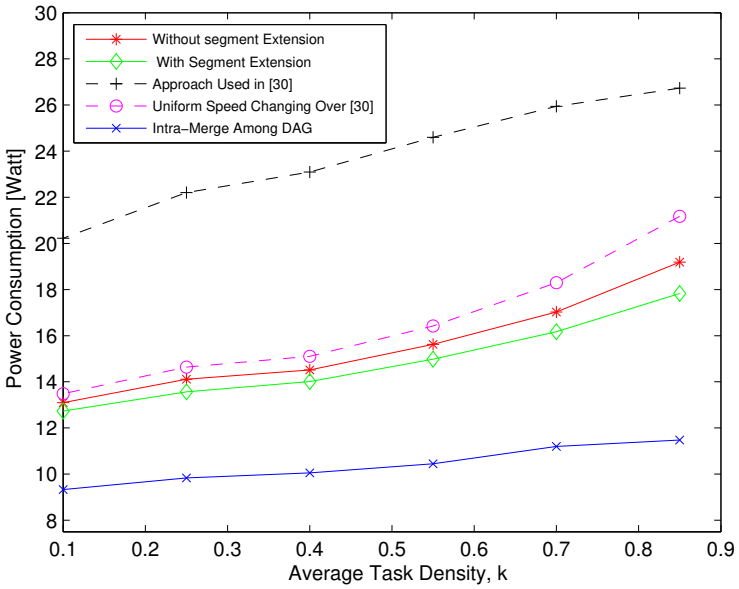

Figure 8 Comparison of power consumption with different approaches for DAG tasks with a fixed number of nodes as 30.

- Federated scheduling with task decomposition, where length of each segment is further extended uniformly (according to their loads) [30];
- Federated scheduling with task decomposition, where lengths of segments are determined by the proposed convex optimization (Sec. 3.3);
- Energy-sub-optimal federated scheduling with task decomposition, where lengths of segments are determined by convex optimization (Sec. 3.3) after performing segment extension (Sec. 3.2);
- Federated scheduling with intra-DAG processor merging (Sec. 4.1);

5.1 Varying Task Periods (Densities)

Here we vary the minimum inter-arrival separation for each task, such that the average density of a set is controlled. We vary the period in an allowable range ($P_i \leq T_i \leq C_i$) by assigning T_i as $P_i + (1-k)(C_i - P_i)$, where $k \in [0,1]$ is named as the *density* of the task – note that this is different from the normal density definition for sequential tasks. We fix the number of nodes within each DAG task as 30, and show the average power consumption in Figure 8.

The first thing we notice from Figure 8 is that the average energy consumption increases as the average density of the set increases (due to decreasing of the period). This phenomenon makes sense as higher density would lead to tighter real-time restrictions, which lead to less room for our segment length optimization.

As shown in Figure 8, 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.

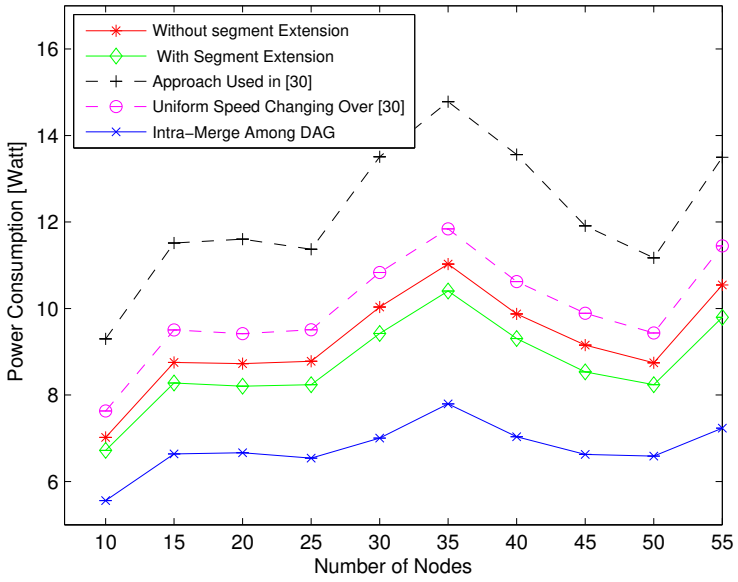

Figure 9 Comparison of average power consumption per task set with different approaches for tasks with harmonic periods.

It is easy to tell that the improvements to the average power consumption are huge when applying the processor merging techniques described in Sec. 4. The improvement is larger when density 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%.

5.2 Different Numbers of Nodes in a DAG Task

Now we vary the number of nodes within each DAG task without changing the period T_i. In this set of comparisons, we consider both harmonic (reported in Figure 9) and arbitrary periods (reported in Figure 10) for a set. For each setting of parameters, we randomly generate 100 task sets with various number of nodes (from 10 to 55, with an increment of 5) and report the average performances of the power consumption over the 100 sets for each case.

First of all, 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 density). Specifically, the intra-DAG merging techniques discussed in Subsections 4.1 lead to a reduction in the power consumption for at least 27.29% and 34.27% for harmonic and arbitrary periods, respectively (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%.

Secondly, when comparing curves in Figures 9 and 10, we observe that task sets with harmonic periods typically result in lower energy consumption compared to arbitrary periods (under same task density 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

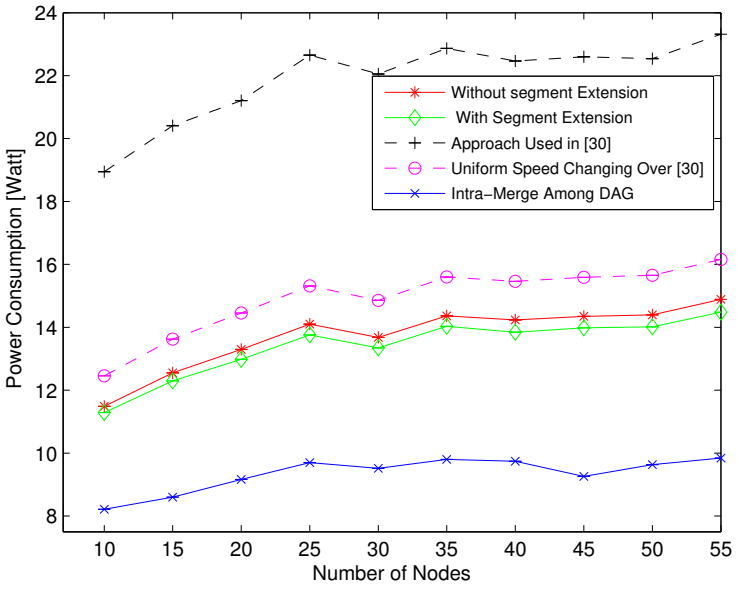

Figure 10 Comparison of average power consumption per task set with different approaches for tasks with arbitrary periods.

that the proposed methods are *robust* to various settings of parameters and combination of DAG tasks.

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 [6]. They have considered both EDF and Fixed-Priority (FP) scheduling policies. Jejurikar has considered non-preemptive tasks in order to deal with shared resources [19]. Chen et al. presents an energy-efficient design for heterogeneous multiprocessor platform [11]. No previous work considers parallel task model.

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 [34]. For dependent tasks, [10] 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 [29]. The authors in [28] consider power-aware policy for scheduling parallel hard real-time systems, where the multi-thread processing is used. [27] considers dealing with parallel tasks under Gang scheduling policy, where all parallel instances of a task use a processor in the same window. Based on level-packing, an efficient scheduling algorithm is proposed [20] [31]. The authors in [31] 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 [17], where precedence constraints can be specified among the tasks. As mentioned previously, no existing work allows intra-task processor sharing, and considers the (more general) DAG task workload model.

7 Conclusion

This paper studies the scheduling of a set of sporadic DAG tasks with implicit deadlines. Upon guaranteeing real-time correctness, we try to minimize the overall power consumption of the whole 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 (i) considers power issues for scheduling recurrent DAG tasks and (ii) allows intra-Task processor sharing. Still, our work has its restrictions: (i) during the processor merging process, we allow each processor to be merged only once – two or more merging may further reduce the power consumption; (ii) we only considered implicit deadlines, and the extension to constrained deadline case is not trivial; (iii) we have shown the evaluation through simulation. In the future, we plan to validate our algorithm in modern-generation processor to show how much the predicted energy savings correlate to the measurements on a real-life system.

References

1 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.

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

3 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.

4 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.

5 Cristina Bazgan, Bruno Escoffier, and Vangelis Th. Paschos. Completeness in standard and differential approximation classes: Poly-apx- and ptas-completeness. *Theoretical Computer Science*, 339(2-3):272–292, 2005.

6 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.

7 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.

8 Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

9 David Chandler. Introduction to modern statistical mechanics. *pp. 288. Foreword by David Chandler. Oxford University Press, Sep 1987. ISBN-10: 0195042778. ISBN-13: 9780195042771*, page 288, 1987.

10 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.

11 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.

12 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.

13 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.

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

15 `https://www.mathworks.com/help/optim/ug/fmincon.html`.

16 `http://en.wikipedia.org/wiki/Gammadistribution`.

17 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.

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

19 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.

20 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.

21 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.

22 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.

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

24 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.

25 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.

26 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.

27 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.

28 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. ACM, 2016.

29 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.

30 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.

31 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.

32 Ruibin Xu, Dakai Zhu, Cosmin Rusu, Rami Melhem, and Daniel Mossé. Energy-efficient policies for embedded clusters. *ACM SIGPLAN Notices*, 40(7):1–10, 2005.

33 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.

34 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.

A The Convexity of the Dynamic Energy Consumption

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

$$E(\tau) = \sum_{1 \le i \le n} C_i^{\gamma} (< \alpha_i, \tau >)^{1-\gamma}. \tag{11}$$

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. $|\alpha_i| \geq 1$ 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 $\tau \in \mathbb{R}_+^k$.

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 [8], 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 page 3-17 of [8], if $g(x)$ is a *non-increasing convex* function and $f(\tau)$ is a *concave* function over $\forall \tau \in \mathbb{R}_+^k$, then $g(f(\tau))$ should be a convex function over $\forall \tau \in \mathbb{R}_+^k$.

4. The function $E(\tau)$ and $f_i(\tau)$ could be written as:

$$E(\tau) = \sum_{1 \leq i \leq n} C_i^{\gamma} g(f_i(\tau)) \tag{12}$$

$$f_i(\tau) = (< \alpha_i, \tau >) \tag{13}$$

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 $E(\tau)$ should be a *convex function*.