On the Degree of Parallelism in Real-Time Scheduling of DAG Tasks

Qingqiang He[1], Nan Guan[2], Mingsong Lv[1,3], Zonghua Gu[4]

[1]The Hong Kong Polytechnic University, China
[2]City University of Hong Kong, China
[3]Northeastern University, China
[4]Umea University, Sweden

***Abstract*—Real-time scheduling and analysis of parallel tasks modeled as directed acyclic graphs (DAG) have been intensively studied in recent years. The degree of parallelism of DAG tasks is an important characterization in scheduling. This paper revisits the definition and the computing algorithms for the degree of parallelism of DAG tasks, and clarifies some misunderstandings regarding the degree of parallelism which exist in real-time literature. Based on the degree of the parallelism, we propose a real-time scheduling approach for DAG tasks, which is quite simple but rather effective and outperforms the state-of-the-art by a considerable margin.**

I. Introduction

Multi-cores are becoming the mainstream of real-time systems for performance and energy efficiency. Real-time applications must be parallelized to fully utilize the power of multi-cores. Many parallel applications can be represented as directed acyclic graph (DAG) tasks to model the dependency and parallelism within these applications [1], [2]. Real-time analysis of DAG tasks has gained much attention in recent years [3]–[7].

One of the important characterizations of a DAG task is the degree of parallelism which, intuitively, measures how much the task can execute concurrently. In this paper, we want to employ the degree of parallelism to improve the system schedulability in real-time scheduling of DAG tasks. However, the exact definition and the computing algorithm of the degree of parallelism have been long eluding the researchers in real-time community. Some works (e.g, [8]) successfully identify the definition of the degree of parallelism for simple parallel task graphs (e.g., the gang task model [8] or the fork-join task model [9]). Some works (e.g, [10]) use the degree of parallelism of DAG tasks vaguely without defining and providing algorithms to compute it. Some works (e.g, [11]) give a primitive definition of the degree of parallelism but mistakenly believe that this problem cannot be computed in polynomial time. See Section II for a detailed discussion of the status quo concerning the degree of parallelism in real-time area. This paper revisits the definition of the degree of parallelism for a DAG task by relating it to the width of a partially ordered set (poset) [12] (Section IV-A), and clarifies some misunderstandings in real-time literature (Section IV-B).

Based on the degree of the parallelism, this paper proposes a real-time scheduling approach for DAG tasks (Section V). First, we present a response time bound for a DAG task under work-conserving scheduling by minimally decomposing the DAG into disjoint parallel chains. Different from the traditional critical-path based analysis (such as [4], [6], [13]) where vertices not in the longest path cannot execute in parallel with the longest path, in our analysis, these disjoint chains can execute in parallel with each other, thus reducing the response time bound and improving system schedulability. Second, we propose an algorithm to schedule multiple DAG tasks under the federated scheduling paradigm. Third, to further reduce the proposed response time bound and enhance system schedulability, we improve an existing algorithm to compute the minimum chain decomposition by jointly utilizing the information of topology and the WCET of vertices. Our approach is simple but rather effective, outperforming the state-of-the-art by 18.6% on average and 83.1% at the maximum.

This work is partially supported by Research Grants Council of Hong Kong (GRF 11208522, 15206221).

II. Related Work

For the degree of parallelism in real-time scheduling of parallel task graphs, the definition of the degree of parallelism for the gang task model is given as "the number of concurrent processors needed to execute any job released by the gang task" [8]. In the fork-join task model [9], the "parallelism" is defined as the number of threads in each parallel segment [14]. Since the structure of the gang or fork-join task model is simple, the definition and computation of the degree of parallelism are straightforward. For the DAG task model, [15] mentioned the degree of parallelism, but did not illustrate its meaning. [16] defined the degree of parallelism by an equation to facilitate the design of their scheduling algorithm, where the meaning of the degree of parallelism is the ratio between the total workload and the length of the longest path and adding a parameter. [17] used the degree of parallelism to refer to parameters for generating DAG tasks, where the degree of parallelism means the number of successors for a vertex in the DAG task. In [18], the degree of parallelism means the number of parallel threads. [19] mentioned that "the maximal number of eligible vertices is bounded by the parallelism of the task graph", but it is not clear what "the parallelism of the task graph" exactly means and how it can be computed. [10] discussed the "parallelism" or the "parallel degree" of DAG tasks to design real-time scheduling algorithms, the meaning of which vaguely refers to the number of concurrently executed vertices at different times during the execution of the DAG task. [11] pointed out that computing the degree of parallelism is "identifying the largest set of subtasks that can execute concurrently". However, they deemed that this problem is NP-hard and turned to study a restricted version of

978-3-9819263-7-8/DATE23/© 2023 EDAA

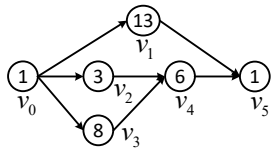


Fig. 1. Illustration of the DAG task model.

DAG called "nested fork-join DAG". All of the above-mentioned literature tried to use the term "degree of parallelism" to measure how the task can execute concurrently. But different of them refer to similar but different things.

For real-time scheduling of DAG tasks, existing approaches can be categorized into three major paradigms: decomposition-based scheduling [20], global scheduling [3], [21], federated scheduling [4]. The closely related approach to this work is federated scheduling where each DAG task is scheduled independently on a set of dedicated processors. Federated scheduling was generalized to constrained deadline tasks [5], arbitrary deadline tasks [22], and conditional DAG tasks [23]. To address the resource-wasting problem in federated scheduling, a series of federated-based scheduling algorithms [6], [7], [13], [24] were proposed.

III. Task Model

A parallel real-time task is modeled as a DAG $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. Each vertex $v \in V$ represents a piece of sequentially executed workload with worst-case execution time (WCET) $c(v)$. An edge $(v_i, v_j) \in E$ represents the precedence relation between v_i and v_j, i.e., v_j can start execution only after vertex v_i finishes its execution. A *path* $\lambda = (\pi_0, \cdots, \pi_k)$ is a set of vertices such that $\forall i \in [0, k-1] : (\pi_i, \pi_{i+1}) \in E$. The length of a path λ is defined as $len(\lambda) \coloneqq \sum_{\pi_i \in \lambda} c(\pi_i)$. The *longest path* is a path with largest $len(\lambda)$ among all paths in G, and we use $len(G)$ to denote the length of the longest path. For any vertex set $V' \subset V$, $vol(V') \coloneqq \sum_{v \in V'} c(v)$. The volume of G is the total workload in the DAG task, defined as $vol(G) \coloneqq vol(V) = \sum_{v \in V} c(v)$. If there is an edge $(u, v) \in E$, u is a *predecessor* of v, and v is a *successor* of u. If there is a path in G from u to v, u is an *ancestor* of v and v is a *descendant* of u. We use $pre(v)$, $suc(v)$, $anc(v)$ and $des(v)$ to denote the set of predecessors, successors, ancestors and descendants of v, respectively. Fig. 1 shows an example DAG G where the number inside vertices is the WCET. For this DAG, $len(G) = 16$, $vol(G) = 32$.

IV. Degree of Parallelism

This section introduces the degree of parallelism for DAG tasks. The concept of degree of parallelism has long existed in the literature concerning real-time scheduling [8]–[11], [14]–[19]. However, these works only mention it in a primitive or vague manner. Intuitively, the degree of parallelism should mean the maximum number of vertices that can run in parallel or the minimum number of cores (or processors) such that no vertices can be blocked. A DAG is a poset (partially ordered set) and the degree of parallelism for a DAG is actually the width of a poset, which corresponds to the maximum number of mutually incomparable elements in the poset. In this section, we illustrate this concept under the context of DAG tasks.

A. Definition

This subsection presents the precise definition of the degree of parallelism for DAG tasks. For two distinct vertices u, v, we say that u is *parallel* to v if and only if $u \notin anc(v) \wedge v \notin anc(u)$. For example, in Fig. 1, v_1 is parallel to v_2.

Definition 1 (Antichain [25])**.** *An antichain is a set of vertices such that any two distinct vertices in this antichain are parallel to each other. In particular, a vertex set containing only one vertex is an antichain.*

For example, in Fig. 1, $\{v_1, v_2, v_3\}$ is an antichain. The *size* of an antichain is the number of vertices in this antichain.

Definition 2 (Chain [25])**.** *A chain $\gamma = (\pi_0, \cdots, \pi_k)$ is a set of vertices such that $\forall i \in [0, k-1]$, π_i is an ancestor of π_{i+1}. In particular, a vertex set containing only one vertex is a chain.*

For example, in Fig. 1, (v_0, v_2, v_5) is a chain. In contrast to antichain, any two distinct vertices in a chain are *not* parallel to each other.

Definition 3 (Chain Decomposition)**.** *A chain decomposition σ is a partition of the vertex set V of the DAG task G into disjoint chains.*

The *size* of a chain decomposition σ is the number of chains in this chain decomposition. A *minimum chain decomposition* is a chain decomposition with the minimum size. For example, in Fig. 1, $\{(v_0, v_3), (v_2, v_4), (v_1, v_5)\}$ is a chain decomposition and the size of this chain decomposition is 3.

Theorem 1 (Dilworth's Theorem [26])**.** *For a DAG task G, the maximum size of antichains of G is the same as the minimum size of chain decompositions of G.*

Since the maximum size of antichains and the minimum size of chain decompositions are the same by Dilworth's theorem, the degree of parallelism for a DAG task is given as follows.

Definition 4 (Degree of Parallelism [12])**.** *For a DAG task G, the degree of parallelism is defined as the maximum size of antichains of G or the minimum size of chain decompositions of G.*

For example, for the DAG task in Fig. 1, the degree of parallelism is 3. The definition of the degree of parallelism for the gang task model in [8] and the definition for the fork-join task model in [14] are consistent with Definition 4 (note that gang tasks and fork-join tasks are simplified versions of DAG tasks). A DAG is a poset and a poset can also be viewed as a DAG. So the concepts introduced in this subsection are also applicable to poset. Actually Definition 4 is exactly the definition of the width for a poset [12].

B. Clarification

Concerning the concept of the degree of parallelism, this subsection clarifies some misunderstandings that exist in real-time literature.

Clarification on Complexity. [11] claims that computing the degree of parallelism for a DAG is NP-hard. [11] states that computing the degree of parallelism consists in "identifying the largest set of subtasks that can execute concurrently", which is the same as Definition 4. [11] continues observing that the complexity of computing the degree of parallelism for a DAG is "therefore equivalent to the problem known as the maximum independent set problem in graph theory", which is not true. Even if the degree of parallelism for a DAG can be reduced to the maximum independent set problem in a graph, since DAG is a restricted case of graph, the fact that the maximum independent set problem in a graph is NP-hard does not imply that computing the degree of parallelism for a DAG is NP-hard.

In fact, by Definition 4, the degree of parallelism for a DAG is the width of a poset. And the width of a poset can be computed in polynomial time [27]–[29]. We summarize this fact into Theorem 2.

Theorem 2. *The problem of computing the degree of parallelism for a DAG task is in P.*

Clarification on Algorithm. [10] claims that the workload distribution function in [11] can describe the "parallel degree" of a DAG task. First, it is not clear in [10] what it means by "parallel degree". Second, the degree of parallelism can not be computed by using the workload distribution function. The workload distribution function in [11] is defined by (1) and (2).

$$f(v) \coloneqq c(v) + \max_{u \in pre(v)} \{f(u)\} \tag{1}$$

$$wd(t) \coloneqq \sum_{v \in V} \begin{cases} 1, & \text{if } t \in [f(v) - c(v), f(v)) \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

In (1), the finish time $f(v)$ of each vertex is computed assuming the DAG executes on an infinite number of cores. Then for a time t, (2) computes the number of vertices that execute in parallel at time t. The maximum value of a workload distribution $wd(t)$ is defined to be $\max\{wd(t)\}$. For example, the maximum value of the workload distribution for Fig. 1 is 3.

Theorem 3. *For a DAG task, the maximum value of the workload distribution in [11] is a lower bound on the degree of parallelism.*

Proof. Suppose that the maximum value is d and is reached at time t. Therefore, at time t, the number of vertices that are executing in parallel is d. These vertices constitute an antichain of the DAG task, and the size of this antichain is d. By Definition 4, the degree of parallelism is the maximum size of antichains of the DAG task, which is equal to or larger than d. □

For the algorithms of computing the degree of parallelism for a DAG, in [27], this problem is solved by partitioning the DAG greedily into several initial chains, and reducing the number of chains by finding the alternating sequences with time complexity $O(|V|^3)$. [30] shows that this problem can be reduced to the bipartite matching problem. The bipartite matching problem [31] is solved by the Hopcroft-Karp algorithm in time $O(|E|\sqrt{|V|})$ [32]. The bipartite matching problem can be treated as the graph matching problem, thus solved by the blossom algorithm in time $O(|E||V|^2)$ [33], or be treated as the maximum flow problem, thus solved by the Ford-Fulkerson algorithm [34]. The bipartite matching problem can also be solved by the alternating path algorithm which is a simplified version of [33] and [34], and has time complexity $O(|V|^3)$ (see [35] for details)

V. Real-Time Scheduling

This section presents the real-time scheduling approach of DAG tasks based on the degree of parallelism.

A. Scheduling One Task

The DAG task is scheduled on a multi-core platform with m identical cores. A vertex v is *eligible* if all its predecessors have finished and thus v can immediately execute if there are available cores. The DAG task is scheduled by *any* algorithm that satisfies the *work-conserving* property, i.e., an eligible vertex must be executed if there are available cores. Without loss of generality, we assume the DAG task starts execution at time 0. Next, we derive an upper bound on the response time R of the DAG task G under work-conserving scheduling.

Given a minimum chain decomposition σ^* of G (the computation of σ^* will be discussed in Section V-C), we arbitrarily select n number of chains from σ^* such that $n \leq m$. The set of vertices that are in the selected n number of chains is denoted as V_{pr}. $V_{sr} \coloneqq V \setminus V_{pr}$. We partition the interval between the start time of G and the finish time of G into two types of intervals.

- I_{sr}: time interval during which $\exists v \in v_{sr}$, v is executing;
- I_{pr}: time interval during which $\forall v \in v_{sr}$, v is *not* executing.

Note that a time interval is not necessarily continuous. For a time interval I, we denote the length of this interval as $|I|$.

Lemma 1.

$$|I_{sr}| \leq vol(V_{sr}) \tag{3}$$

Proof. Since the scheduling algorithm is work-conserving, by the definition of I_{sr}, (3) holds. □

Lemma 2.

$$|I_{pr}| \leq len(G) \tag{4}$$

Proof. By the definition of I_{pr}, only vertices in V_{pr} can execute in I_{pr}. Recall that there are n number of chains in V_{pr}, which is no larger than the number of cores m. This means that during I_{pr}, $\forall v \in V$, if vertex v is eligible, v will be executed immediately on some core, since the scheduling is work-conserving. Therefore, during I_{pr}, the execution of any path of G is not delayed. If we assume $|I_{pr}| > len(G)$, then there must be a chain whose length is larger than the length of the longest path of G, which is a contradiction. The conclusion is reached. □

Theorem 4. *The response time R of the DAG G is bounded by:*

$$R \leq len(G) + vol(V_{sr}) \tag{5}$$

Proof. By the definitions of I_{sr} and I_{pr}, we have $R = |I_{pr}| + |I_{sr}|$. By Lemma 1 and Lemma 2, the conclusion follows. □

Corollary 1. *If the DAG G with the degree of parallelism p is scheduled by work-conserving scheduling on a platform with the number of cores $m \geq p$, the response time R of the DAG is bounded by the length of the longest path of the DAG, i.e.,*

$$R \leq len(G) \tag{6}$$

Proof. Let σ^* be the minimum chain decomposition of $G = (V, E)$. By Definition 4, p equals the size of σ^*. Since $p \leq m$, let V_{pr} be the set of vertices that are in the p number of chains from σ^*. Note that σ^* is a chain decomposition of G, so $V_{pr} = V$. Therefore, $V_{sr} = V \setminus V_{pr} = \varnothing$, which means $vol(V_{sr}) = 0$. By (5), we reach the conclusion. □

Corollary 1 shows that our bound in (5) perfectly degrades to the bound in (6) for the special case of $m \geq p$, which is a well-established result in literature [9], [36].

B. Scheduling Multiple Tasks

In a task set, a sporadic parallel real-time task is represented as (G, D, T) where $G = (V, E)$ is the DAG; D is the relative deadline; T is the period. We consider constrained deadline, i.e., $D \leq T$. We schedule the task set by the widely-used federated scheduling [4], which is simple to implement and has good guaranteed real-time performance. In the original federated scheduling [4], each heavy task (tasks with $vol(G) \geq D$) is assigned and executed exclusively on m cores under a work-conserving scheduler, where m is computed by (7).

$$m = \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \tag{7}$$

The correctness of (7) is proved in [4]. The light tasks (tasks with $vol(G) < D$) are treated as sequential sporadic tasks and are scheduled on the remaining cores by sequential scheduling algorithms such as global EDF [37] or partitioned EDF [38].

In our scheduling approach, the number of cores m assigned to each heavy task is computed by (8).

$$m = \min \left\{ m', \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \right\} \tag{8}$$

where m' is computed by Alg. 1.

In Alg. 1, the computation of the minimum chain decomposition $(\gamma_i)_1^p$ will be discussed in Section V-C. $(\gamma_i)_1^p$ is the compact representation of $\gamma_1, \gamma_2, \cdots, \gamma_p$ and p is the degree of parallelism of the DAG task. In the for-loop, we use Theorem 4 to search for the minimum number of core m' such that the computed response time bound (Line 4) is no larger than the deadline. The for-loop in Alg. 1 runs no more than p times, which is bounded by $|V|$. If the number of cores assigned to a heavy task is computed using (8), then the task will not miss its deadline, which is a result of (7) and Theorem 4. The method of scheduling light tasks is the same as [4].

C. Computing Minimum Chain Decomposition

This subsection discusses how to compute the minimum chain decomposition required by Alg. 1. As stated in Section IV-B, a minimum chain decomposition can be computed by algorithms for the degree of parallelism or by solving the bipartite matching problem. However, these existing algorithms such as [27], [32]–[34] only consider the topology of the DAG, not taking into account the WCET of vertices. For purpose of solely computing the degree of parallelism (i.e., minimizing the number of chains in a chain decomposition), the information of topology is sufficient. However, in Alg. 1, we want that not only the number of chains in this chain decomposition is minimized, but also the volume of V_{sr} is minimized, such that the response time bound in Theorem 4 is minimized and the number of cores returned by Alg. 1 is minimized. Next, by jointly utilizing the information of topology and the WCET of vertices, we propose an approach that computes a minimum chain decomposition such that $vol(V_{sr})$ is reduced as much as possible.

Algorithm 1: Computing the Number of Cores

Input : $G = (V, E)$; the deadline D; the minimum chain decomposition $(\gamma_i)_1^p$ of G
Output: the number of cores m'

```
1 foreach n ← p, p − 1, ··· , 1 do
2 |   V_pr ← the set of vertices that are in (γ_i)_1^n
3 |   V_sr ← V \ V_pr
4 |   if len(G) + vol(V_sr) ≤ D then
5 |   |   m' ← n
6 |   else
7 |   |   break
8 |   end
9 end
```

Our approach is presented in Alg. 3, which first reduces this problem to bipartite matching by [30], second employs the alternating path algorithm (see Section IV-B) to solve the bipartite matching problem. [30] presents a reduction from the problem of degree of parallelism to the bipartite matching problem. [30] also describes the correspondence between a DAG and a bipartite graph, and the correspondence between a chain decomposition and a matching. A *matching* φ of a bipartite graph BG is a set of edges such that no edge in φ shares a vertex with any other edge in φ. A *maximum matching* is a matching with the maximum cardinality. A maximum matching corresponds to a minimum chain decomposition [30]. All *Correspond* procedures in Alg. 3 can be found in [30]. Line 1 first converts the DAG G into its corresponding bipartite graph BG. Line 2 calls Alg. 2 to compute a simple greedy chain decomposition σ. Alg. 2 iteratively computes a longest path of the DAG (Line 3), then deducts this longest path from the DAG (Line 5) until there are no vertices in the DAG (Line 2), thus yielding a chain decomposition of the DAG. Line 4 of Alg. 2 ensures that there are no common vertices among different chains of the chain decomposition. Line 3 of Alg. 3 computes the corresponding matching φ of σ. In Line 4, the alternating path algorithm takes an initial matching φ as input, and outputs a maximum matching φ^*. Line 5 computes the corresponding minimum chain decomposition σ^* of φ^*. The while-loop in Alg. 2 can execute no more than $|V|$ times. Since the alternating path algorithm runs in time $O(|V|^3)$, the time complexity of Alg. 3 is also $O(|V|^3)$.

Algorithm 2: GreedyDecomposition(G)

Input : DAG $G = (V, E)$
Output: a chain decomposition $(\gamma_i)_0^k$

1 $G' \leftarrow G$; $i \leftarrow 0$
2 **while** $vol(G') \neq 0$ **do**
3 $\quad \gamma_i \leftarrow$ the longest path of G'
4 $\quad \gamma_i \leftarrow \gamma_i \setminus \{v \in \gamma_i | c(v)$ of G' is $0\}$
5 $\quad$ for each vertex $v \in \gamma_i$, let $c(v)$ of G' be 0
6 $\quad i \leftarrow i + 1$
7 **end**

Algorithm 3: Our Approach

Input : DAG $G = (V, E)$
Output: a minimum chain decomposition σ^*

1 $BG \leftarrow$ CorrespondBipartite(G)
2 $\sigma \leftarrow$ GreedyDecomposition(G)
3 $\varphi \leftarrow$ CorrespondMatching(σ)
4 $\varphi^* \leftarrow$ BipartiteMatching(BG, φ)
5 $\sigma^* \leftarrow$ CorrespondDecomposition(φ^*)

For the input of the alternating path algorithm in Line 4, the initial matching φ can trivially be $\varnothing$ or any other matchings. Since the target of existing algorithms [27], [32]–[34] is to compute the degree of parallelism, none of them uses an initial matching as the input that takes the information of WCETs of vertices into account. We observe that the initial matching has a big influence on the computed minimum chain decomposition and thus the computed number of cores in Alg. 1. In our approach of Alg. 3, the chain decomposition σ and its corresponding matching φ carry the information of the WCET of vertices (Line 2, 3); then the alternating path algorithm (Line 4) iteratively adjusts this initial matching φ into a maximum matching φ^* (thus a corresponding minimum chain decomposition σ^*) using the information of the topology of the DAG.

Example 1. *For the DAG in Fig. 1, suppose the deadline D is 20. The number of cores in the original federated scheduling computed by (7) is 4. Existing algorithms for computing the degree of parallelism may output minimum chain decompositions such as $\{(v_0, v_3), (v_2, v_4), (v_1, v_5)\}$, or $\{(v_0, v_1, v_5), (v_2, v_4), (v_3)\}$. Using these chain decompositions for Alg. 1, the computed number of cores is 3, which is the degree of parallelism. In our approach, the initial chain decomposition computed by Alg. 2 is $\{(v_0, v_3, v_4, v_5), (v_1), (v_2)\}$, which is already a minimum chain decomposition and is the output of Alg. 3. And the number of cores computed by Alg. 1 is 2.*

VI. Evaluation

This section evaluates the proposed scheduling approach in Section V. The following methods are compared.

- OUR. Our method presented in Section V.
- FED. The original federated scheduling proposed in [4].
- VFED. The virtually-federated scheduling in [7], by adding servers on top of federated scheduling to reclaim unused computing capacity.

As shown in [7], VFED has the best performance among all existing multi-DAG scheduling algorithms of different paradigms (federated, global and partitioned), so we only include VFED in our comparison.

Task Generation. The DAG tasks are generated using the Erdös-Rényi method [39], where the number of vertices $|V|$ is randomly chosen in a specified range. For each pair of vertices, it generates a random value in $[0, 1]$ and adds an edge to the graph if the generated value is less than a predefined *parallelism factor* pf. The larger pf, the more sequential the graph is. The period T (which equals D in the experiment) is computed by $len(G) + \alpha(vol(G) - len(G))$, where α is a parameter. By (7), the number of cores required by a task is at most $\lceil \frac{1}{\alpha} \rceil$. We consider α in [0, 0.5] to let heavy tasks require at least two cores. The default settings are as follows. The WCETs of vertices $c(v)$, the parallelism factor pf, the vertex number $|V|$ and α are randomly and uniformly drawn in $[50, 100]$, $[0.1, 0.9]$, $[50, 250]$ and $[0, 0.5]$, respectively. The number of cores m is set to be 32 (but changing in Fig. 2) and the normalized utilization nu of task sets is randomly and uniformly drawn in [0, 0.8]. For each configuration (i.e., each data point in the figures), we randomly generate 1000 task sets to compute the average acceptance ratio.

We evaluate the schedulability of task sets using the acceptance ratio as the metric. The larger acceptance ratio, the better the performance. Fig. 2 shows the results by changing the number of cores on which the task set is scheduled. As mentioned before, other parameters (i.e., nu, α, pf and $|V|$) are randomly selected in their intervals. Compared to VFED, the maximum performance improvement of our method is 21.6% among all numbers of cores. With the number of cores increasing, the performances of FED and VFED decrease. This is because most existing multi-core scheduling analysis techniques (for example, those of FED and VFED) are inherently unable to utilize the resources enabled by multi-core computing in an efficient manner, thus their performances cannot scale proportionally to the increase of the number of cores. Our method by using the degree of parallelism of tasks can exploit the resources more efficiently, almost scaling proportionally to the increase of the number of cores. In the following experiments, we use the core number $m = 32$ as a representative for evaluation.

In Fig. 3a, compared to VFED, the improvement of acceptance ratio is up to 83.1% with $nu = 0.35$. Fig. 3b presents the result by changing α. Different α means different deadlines. When α approaches 0, the deadline D approaches $len(G)$; the core number computed in (7) approaches infinite; the acceptance ratio of FED and VFED approaches 0. Our method for computing the core number in (8) does not have this limitation. Even if the deadline equals the length of the longest path, our method can still achieve good performance (our acceptance ratio is 57.7%, while the acceptance ratio of the other two methods is 0). When α increases, the computed core numbers approach 1, and the performances of FED and

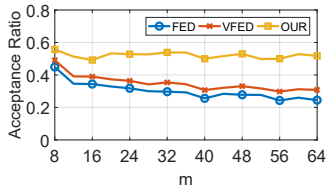
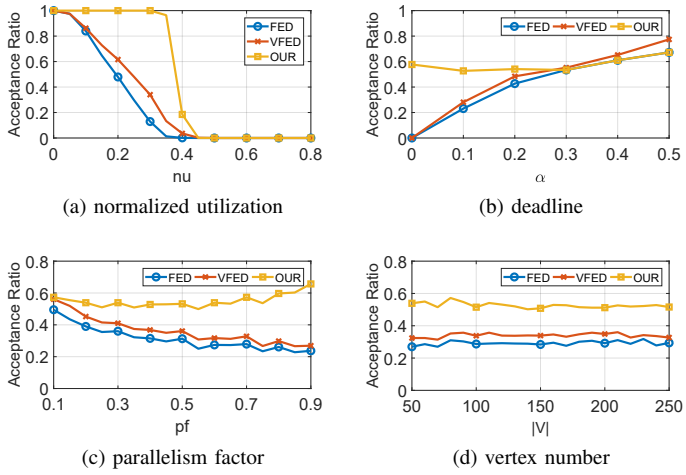


Fig. 2. Evaluation of different numbers of cores.



(a) normalized utilization (b) deadline

(c) parallelism factor (d) vertex number

Fig. 3. Evaluation results ($m = 32$).

OUR become indistinguishable. When α increases, VFED outperforms FED and OUR slightly. This is because VFED can potentially reclaim the computing resources which are wasted by the analysis technique behind (7). Fig. 3c shows the result by changing the parallelism factor pf. When pf increases, the DAG becomes more sequential which means that the volume $vol(G)$ becomes close to the length of the longest path $len(G)$. Therefore, by the task generation method, the deadline D becomes close to $len(G)$, which means that the core number computed by (7) becomes large. Subsequently, the performances of FED and VFED decrease. The results with changing vertex number are reported in Fig. 3d, which shows that the evaluated methods are insensitive to the vertex number of the graph. Compared to VFED, our method can improve the acceptance ratio by 18.6% on average.

VII. Conclusion

In this paper, based on the degree of parallelism, we propose a real-time scheduling approach for DAG tasks, which is simple and effective. Experiments show that compared to the state-of-the-art, our method can improve the system schedulability by 18.6% on average and 83.1% at the maximum.

References

[1] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *RTCSA*, 2016.

[2] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *RTAS*, 2020.

[3] J. Li, K. Agrawal, C. Lu, and C. Gill, "Outstanding paper award: Analysis of global edf for parallel tasks," in *ECRTS*, 2013.

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

[5] S. Baruah, "The federated scheduling of constrained-deadline sporadic dag task systems," in *DATE*, 2015.

[6] N. Ueter, G. Von Der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *RTSS*, 2018.

[7] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and W. Yi, "Virtually-federated scheduling of parallel real-time tasks," in *RTSS*, 2021.

[8] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," *Real-Time Systems*, 2019.

[9] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling parallel real-time tasks on multi-core processors," in *RTSS*, 2010.

[10] S. Zhao, X. Dai, and I. Bate, "DAG scheduling and analysis on multi-core systems by modelling parallelism and dependency," *TPDS*, 2022.

[11] J. Fonseca, G. Nelissen, and V. Nélis, "Improved response time analysis of sporadic DAG tasks for global FP scheduling," in *RTNS*, 2017.

[12] G. Grätzer, *General lattice theory*. Springer Science & Business Media, 2002.

[13] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *RTSS*, 2017.

[14] Y. Tang, N. Guan, and W. Yi, "Real-time task models," *Handbook of Real-Time Computing*, 2022.

[15] W. Wu, A. Bouteiller, G. Bosilca, M. Faverge, and J. Dongarra, "Hierarchical dag scheduling for hybrid distributed systems," in *IPDPS*, 2015.

[16] K. Nikolova, A. Maeda, and M. Sowa, "Parallelism-independent scheduling method," *IEICE FOECC*, 2000.

[17] L. He and S. Jarvis et al., "Mapping dag-based applications to multiclusters with background workload," in *CCGrid*, 2005.

[18] Z. Houssam-Eddine, N. Capodieci, R. Cavicchioli, and G. Lipari et al., "The hpc-dag task model for heterogeneous real-time systems," *TC*, 2020.

[19] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of DAG tasks on multi-cores," *TPDS*, 2019.

[20] X. Jiang, X. Long, N. Guan, and H. Wan, "On the decomposition-based global edf scheduling of parallel real-time tasks," in *RTSS*, 2016.

[21] Q. He, M. Lv, and N. Guan, "Response time bounds for dag tasks with arbitrary intra-task priority assignment," in *ECRTS*, 2021.

[22] S. Baruah, "Federated scheduling of sporadic dag task systems," in *IPDPS*, 2015.

[23] ——, "The federated scheduling of systems of conditional sporadic dag tasks," in *ICESS*, 2015.

[24] X. Jiang, N. Guan, X. Long, Y. Tang, and Q. He, "Real-time scheduling of parallel tasks with tight deadlines," *JSA*, 2020.

[25] A. Frank, "On chain and antichain families of a partially ordered set," *Journal of Combinatorial Theory, Series B*, 1980.

[26] R. Dilworth, "A decomposition theorem for partially ordered sets," *Annals of Mathematics*, 1950.

[27] K. P. Bogart, *Introductory combinatorics*. Saunders College Publishing, 1989.

[28] S. Felsner, V. Raghavan, and J. Spinrad, "Recognition algorithms for orders of small width and graphs of small dilworth number," *Order*, 2003.

[29] S. Ikiz and V. K. Garg, "Online algorithms for dilworth's chain partition," *University of Texas at Austin, Tech. Rep*, 2004.

[30] D. R. Fulkerson, "Note on dilworth's decomposition theorem for partially ordered sets," in *Proc. Amer. Math. Soc*, 1956.

[31] M. R. Garey and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *WH Freeman and Company*, 1979.

[32] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on computing*, 1973.

[33] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, 1965.

[34] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, 1956.

[35] "lec4.pdf," http://www.columbia.edu/%7ecs2035/courses/ieor8100.F12/lec4.pdf.

[36] A. I. Tomlinson and V. K. Garg, "Monitoring functions on global states of distributed programs," *JPDC*, 1997.

[37] S. Baruah, "Techniques for multiprocessor global schedulability analysis," in *RTSS*, 2007.

[38] S. Baruah and N. Fisher, "The partitioned multiprocessor scheduling of sporadic task systems," in *RTSS*, 2005.

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