Characterizations of Parallel Real-Time Workloads

Xu Jiang[1(✉)], Jinghao Sun[2], and Wang Yi[1,3]

[1] Northeastern University, Shenyang, China
jiangxu@cse.neu.edu.cn
[2] Dalian University of Technology, Dalian, China
[3] Uppsala University, Uppsala, Sweden

Abstract. The work function, originally proposed by Bonifaci et al. [4], plays an important role in timing analysis of sporadic DAG parallel tasks. Later, Baruah [1] and Li et al. [10] provide different characterizations for Bonifaci's notion of work function. The consistency and correctness of these characterizations and Bonifaci's original result is so far a pending question. In this paper, we revisit the notion of work function based analysis techniques to answer the above pending question. We show that Baruah's characterization is equivalent to Bonifaci's original formulation, while Li's characterization is strictly stronger.

1 Introduction

Multi-cores are becoming mainstream platforms for real-time embedded systems to meet the rapidly increasing performance requirements and low power consumption [12,20,23]. To fully utilize the capacity of multi-cores, not only inter-task parallelism, but also intra-task parallelism need to be explored in the design and analysis of modern real-time systems, where individual tasks are parallel programs and can potentially utilize more than one core at the same time during their executions. This enables tasks with higher execution demands and tighter deadlines, such as those used in autonomous vehicles [8], video surveillance, computer vision, radar tracking and real-time hybrid testing [6]. Nowadays parallel programming languages (and software), such as Cilk family [3], OpenMP [14] and Intel's Thread Building Blocks [17], commonly support parallel task sets with intra-task parallelism (in addition to inter-task parallelism).

The origins of this work can be traced back to 1986 when Wang, one of the authors, first met Jifeng at the Marktoberdorf Summer School on Theoretical Computer Science. The encounter marked the beginning of their professional collaboration and personal friendship, which has lasted for nearly four decades. During the school, Wang, a Ph.D. student at the time, got to know Jifeng's work with Antony Hoare of algebraic theory on programming, which has deeply influenced on Wang's subsequent research in process algebras, formal verification, and real-time computing. As one of the leading teams in the country in the field of embedded and real-time systems, the authors wish to take this opportunity to thank Jifeng for his unwavering support, inspiration, and friendship. Happy Birthday, Jifeng!.

J. P. Bowen et al. (Eds.): *Theories of Programming and Formal Methods*, LNCS 14080, pp. 235–256, 2023.
https://doi.org/10.1007/978-3-031-40436-8_9

A common way to model parallel real-time software systems is using recurrent directed acyclic graph (DAG) models. This motivates many recent work in the area of real-time scheduling for recurrent DAG task models [1,2,4,5,7,9–11,13,15,16,18,19,21]. Real-time scheduling algorithms for DAG tasks can be classified into three paradigms: 1) decomposition-based scheduling [7,15,16,18]; 2) global scheduling (without decomposition) [1,4,9,13,21]; and 3) federated scheduling [2,10], which is the trade-off between decomposition-based scheduling and global scheduling. In this paper, we focus on the global EDF scheduling algorithm, which gives the maximum flexibility, e.g., it does not decompose DAG tasks, nor restrict a DAG task to dedicated cores. It schedules vertices in the DAG until either all cores are busy or no more vertices are ready.

Although the global EDF algorithm keeps the best flexibility, its schedulability analysis is a challenging problem. In the literature, a large part of the theoretical work on schedulability analysis of recurrent DAG tasks under global EDF uses a kernel notation, called the *work function*. Intuitively, for any given recurrent DAG task set $\mathcal{T}$, the positive integer t and α, and assuming that $\mathcal{T}$ is executed on infinite number of α-speed cores, the work function $work(\mathcal{T}, t, \alpha)$ defines the maximum workload released from $\mathcal{T}$ that must be executed during an interval of duration equal to t. (See in Sect. 4 for more details).

By applying the work-function-based methodology, researchers mainly derive the following three classical theoretical results.

- Bonifaci et al. [4] first propose the notation of the work function, which is originally used to derive the *speedup bound* of the DAG tasks. The speedup bound is a comparative metric with respect to some other (optimal) scheduler. A scheduling algorithm A provides a speedup bound of α if it can successfully schedule any task set $\mathcal{T}$ on m cores of speed α as long as the compared scheduler can schedule $\mathcal{T}$ on m cores of speed 1. The speedup bound shows how close the performance of a scheduler is to the compared one, but it cannot be directly used as a schedulability test.
- Li et al. [10] reformulate Bonifaci's main result, and derive a *capacity augmentation bound* of the DAG tasks. The capacity augmentation bound is an absolute metric that can be directly used for schedulability test. A scheduling algorithm A has a capacity augmentation bound of α if it can schedule any task set $\mathcal{T}$ (on m cores of speed 1) satisfying the following two conditions:
  - the total utilization of $\mathcal{T}$ is at most m/α, and
  - the worst-case critical path length of each task is at most $1/\alpha$ of its deadline.

  Capacity augmentation bounds are stronger than speedup bounds in the sense that if a scheduler has a capacity augmentation bound of α, it is also guaranteed to have a resource augmentation bound of α. Based on the capacity augmentation bound, Li et al. [10] propose a simple linear-time schedulability test for scheduling recurrent DAG tasks under global EDF. Most importantly, Li et al. [10] prove that their capacity augmentation bound is the tightest one for the DAG task under global EDF algorithm.

– Baruah [1] also reformulates Bonifaci's main result, and proposes a pseudo-polynomial time schedulability analysis method for the DAG tasks under global EDF algorithm.

Our key observation is that although Li et al. [10] and Baruah [1] both state that they reformulate the same theorem of [4], their reformulations are totally different. Only one of them should be equivalent to the original theorem of [4]. In this paper, we devote to clarify which reformulation is the equivalent theorem to the original one of [4]. Through a deep insight into Bonifaci's main theorem, we find that Baruah's theorem is an equivalent state of Bonifaci's theorem in [4], and Li's theorem overwhelms Bonifaci's theorem, indicating that Li's theorem cannot be directly derived from Bonifaci's theorem. The correctness of Li's work needs a careful analysis. To this end, we reveal interesting properties of the work function, and try to provide a rigorous proof for Li's theorem. We extend Bonifaci's techniques to discuss the correctness of Li's theorem, but we only prove that Li's theorem is conditionally correct.

The rest of this paper is organized as follows. Section 2 discusses related work. In Sect. 3 we formally define the sporadic DAG task model and the global EDF algorithm. In Sect. 4 we revisit the notation of work function. In Sect. 5 we give a brief overview of the main theorem in [4], and revisit the existing reformulations of Bonifaci's theorem, and moreover, we discuss whether they are equivalent to the main theorem in [4]. The last section gives the conclusion.

2 Related Work

Bonifaci et al. [4] first introduce the notation of work function, and by using the work function based methodology, they propose the speedup bounds $2 - \frac{1}{m}$ and $3 - \frac{1}{m}$ for the DAG tasks under global EDF and global DM algorithms respectively. Baruah [1] reformulates the main theorem of [4], and improves the global schedulability analysis of [4]. Li et al. [9] analyze the global schedulability of DAG tasks via a methodology that is different from Bonifaci's work function method, and they propose the capacity augmentation bound of $4 - \frac{1}{m}$ for the implicit deadline DAG tasks under global EDF. Moreover, Li et al. [9] also prove that the capacity augmentation bound for the implicit deadline DAG tasks under global EDF is at least $\frac{3+\sqrt{5}}{2} \approx 2.618$. Sun et al. [21] propose the first constant capacity augmentation bound for the constraint deadline DAG tasks under global EDF, and for the implicit deadline DAG tasks, they exhibit the capacity augmentation bound of $3.82 - \frac{1}{m}$, which is better than the one proposed in [9]. The work function based methodology significantly promotes the theoretical work on capacity augmentation bound. Li et al. [10] reformulate the main theorem of [4], and propose the tightest capacity augmentation bound of DAG tasks under global EDF, i.e., they prove that the upper bound of the capacity augmentation bound achieves $\frac{2+\sqrt{5}}{2}$.

We observe that Li et al. [10] and Baruach [1] reformulate the same theorem of [4], and however, their reformulations are totally different. There must be one

of them is not equivalent to the original theorem of [4]. If Li's theorem is not equivalent to Bonifaci's theorem, and, even worse, their theorem is not correct, then the capacity augmentation bound of $3.82 - \frac{1}{m}$ proposed by Sun et al. [21] should be the best known capacity augmentation bound for DAG tasks under global EDF.

3 System Model

This section presents a sporadic DAG task model for recurrent parallel tasks, and formally defines the runtime model by considering the global EDF scheduling algorithm.

3.1 Task Model

This section presents a model for recurrent DAG tasks. We consider a set of n independent sporadic DAG tasks: $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$. Each task τ_i is specified as a 3-tuple (G_i, D_i, T_i), where G_i is a directed acyclic graph (DAG), and D_i and T_i are positive integers, called the *deadline* and the *period* respectively.

The task τ_i repeatedly releases dag-jobs, and each dag-job of τ_i has a DAG-structure specified as $G_i = (V_i, E_i)$, where V_i is a set of vertices, and E_i is a set of directed edges between these vertices. Each vertex $v_i^x \in V_i$ denotes a sequential operation, and is characterized by a worst-case execution time (WCET) $c(v_i^x)$. The edges represent dependencies between the vertices: if $(v_i^x, v_i^y) \in E_i$ then vertex v_i^x must complete execution before vertex v_i^y can begin execution. A vertex v_i^x is the *predecessor* of the vertex v_i^y if there is an edge from v_i^x to (a predecessor of) v_i^y, and in this case, the vertex v_i^y is called the *successor* of v_i^x. A vertex v_i^x is called the *source* vertex of G_i if it has no predecessor. A vertex v_i^x is called the *sink* vertex of G_i if it has no successor. Multiple source vertices and sink vertices are allowed in the DAG G_i, and the DAG G_i are not required to be fully connected. Figure 1 shows two example tasks τ_1 and τ_2, each of which consists of 7 vertices in the DAG structure.

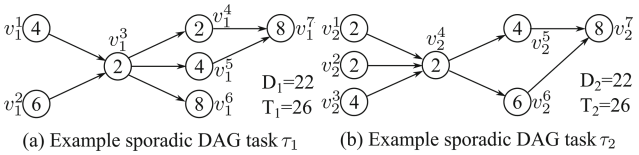

Fig. 1. An example task set consisting of two DAG tasks. Vertices are labeled with WECTs.

A release of a dag-job of τ_i at time-instant t means that all $|V_i|$ vertices $v_i^x \in V_i$ are released at time-instant t. The *period* T_i denotes the minimum duration of time that must elapse between the release of successive dag-jobs of

τ_i. Once a dag-job of τ_i is released at time-instant t, then all $|V_i|$ vertices that were released at time-instant t must complete execution by time-instant $t + D_i$. Recall that D_i is the *deadline* of τ_i.

We now introduce some useful notations related to a DAG task.

- **Volume**. The sum of the worst-case execution time of all vertices in G_i (the graph structure of τ_i's dag-job) is the volume vol_i of τ_i, i.e.,

$$vol_i = \sum_{v_i^x \in V_i} c(v_i^x) \tag{1}$$

  For example, the volume of τ_1 in Fig. 1 is $vol_1 = 34$.
- **Length**. The length of the longest path in G_i (the graph structure of τ_i's dag-job) is the length len_i of τ_i, i.e.,

$$len_i = \max_{\pi \in G_i} \sum_{v_i^x \in \pi} c(v_i^x) \tag{2}$$

  where π is the path of G_i. For example, the length of τ_1 is $len_1 = 6+2+4+8 = 20$.
- **Utilization**. For any task τ_i, we define its utilization u_i as follows.

$$u_i = \frac{vol_i}{T_i} \tag{3}$$

  For example, the task τ_1 in Fig. 1 has an utilization as $u_1 = \frac{17}{13}$. Moreover, the total utilization of the task system $\mathcal{T}$ is denoted as follows.

$$U_{\sum} = \sum_{\tau_i \in \mathcal{T}} u_i \tag{4}$$

In the literatures, these parameters above are used in schedulability analysis, e.g., capacity augmentation bounds, see in Sect. 3.3 for more details.

3.2 Global EDF Algorithm

We consider a platform $\mathcal{P}$ that consists of m identical processing cores $p_1, p_2, \cdots, p_m$, and each of them has a speed $\alpha \geq 1$. We schedule the task set $\mathcal{T}$ on m cores of $\mathcal{P}$. More specifically, at any time instant t, if a core is executing a vertex of some task, then it is called the *busy* core, and otherwise, it is called the idle core. A vertex is *ready* for execution if all its predecessors are finished. A schedule is to assign ready vertices to idle cores until all the released vertices are finished.

In this paper, we schedule tasks by using global EDF (GEDF) algorithm. Under GEDF, at each time instant the scheduler selects the highest-priority ready vertices (at most m) for execution. Vertices of the same task share the same priority (ties are broken arbitrarily) and a vertex of a task with an earlier absolute deadline has a higher priority than a vertex of a task with a later absolute deadline. In particular, vertex-level preemption and migration are both permitted in GEDF. Without loss of generality, we assume the scheduling of the task set $\mathcal{T}$ starts at time 0 (i.e., the first dag-job of the task set is released at time 0).

3.3 Schedulability

A task set $\mathcal{T}$ is schedulable on m α-speed cores if a valid schedule exits on m α-speed cores such that all dag-jobs released by $\mathcal{T}$ meet their deadlines. In particular, when scheduled on m unit-speed cores, a schedulable task set must satisfy the following conditions.

Theorem 1 (Necessary Conditions for schedulability [9]). *A task set $\mathcal{T}$ is not schedulable (by any scheduler on m unit-speed cores) unless the following conditions hold.*

- *The length of each task τ_i is less than its deadline D_i, i.e.,*

$$len_i \leq D_i, \quad \forall \tau_i \in \mathcal{T} \tag{5}$$

- *The total utilization $U_{\sum}$ is smaller than the number of cores, i.e.,*

$$\sum_{\tau_i \in \mathcal{T}} u_i \leq m \tag{6}$$

Clearly, if (5) is violated for some task, then its deadline is doomed to be violated in the worst case, even if it is executed exclusively on sufficiently many cores. If (6) is violated, then in the long term the worst-case workload of the system exceeds the processing capacity provided by the platform, and thus the backlog will increase infinitely which leads to deadline misses. We assume that *all task sets discussed in the reminder of this paper satisfy (5) and (6).*

Given a scheduling algorithm A, a task set $\mathcal{T}$ is A-schedulable on m α-speed cores if A meets all deadlines when scheduling any collection of dag-jobs that may be generated by the task set $\mathcal{T}$ on m α-speed cores. To verify whether a task set is A-schedulable is highly intractable (e.g., NP-hard in the strong sense [22]) even when there is a single DAG task. In the following we introduce two approximation metrics for A-schedulability analysis.

Definition 1 (Speedup Bound). *A scheduling algorithm A has a speedup bound α if any task set $\mathcal{T}$ that is schedulable on m unit-speed cores is A-schedulable on m α-speed cores.*

From Definition 1, we know that for any scheduling algorithm A with a speedup bound α, if a task set $\mathcal{T}$ is not A-schedulable on m α-speed cores, then all scheduling algorithms fail to schedule $\mathcal{T}$ on m unit-speed cores. Moreover, there are some task sets $\mathcal{T}$ such that they are not schedulable on m unit-speed cores, but they are A-schedulable on m α-speed cores. In this sense, the speedup bound α is a metric for approximately quantifying the quality of scheduling algorithms.

Definition 2 (Capacity Augmentation Bound). *A scheduling algorithm A has a capacity augmentation bound α if it can always schedule DAG task set $\mathcal{T}$ on m α-speed cores as long as $\mathcal{T}$ satisfies the above necessary conditions in (5) and (6).*

From Definition 2, for any scheduling algorithm A that has a capacity augmentation bound α, we can derive the sufficient conditions for A-schedulability analysis, i.e., a task set $\mathcal{T}$ is A-schedulable on m unit-speed cores if the following conditions both hold.

$$len_i \leq \frac{D_i}{\alpha}, \quad \forall \tau_i \in \mathcal{T}$$
$$\sum_{\tau_i \in \mathcal{T}} u_i \leq \frac{m}{\alpha}$$

A scheduling algorithm with a smaller speedup bound (as well as a smaller capacity augmentation bound) α is preferable. In particular, when the capacity augmentation bound $\alpha = 1$, the scheduling algorithm is optimal.

In the literature, researchers use the notation of the *work function* to derive the speedup bound and the capacity augmentation bound. In the next section, we introduce such an important notation.

4 Work Function

Bonifaci et al. [4] first introduce the notation of the *work function* and use it to originally characterize the amount of workload that could be generated by a sporadic DAG task when scheduled on unit-speed cores. Li et al. [10] and Baruah [1] further extend the notation of the work function to the scenarios with cores of speed α (larger than 1). In this section, we describe the work function defined in [1,10], which is in a manner consisting with the terminology introduced in Sect. 3.

We first define an ideal scheduling algorithm A_∞ as follows.

Definition 3 (Ideal Scheduling Algorithm A_∞). *The algorithm A_∞ schedules a task set $\mathcal{T}$ on infinite number of cores, and it allocates a core to each vertex v_i^x released by the tasks in $\mathcal{T}$ at the time-instant the vertex v_i^x is ready to execute, and executes the vertex v_i^x upon the allocated core until v_i^x completes its execution.*

We denote by J the collection of dag-jobs that may be released by the tasks in $\mathcal{T}$, written as $J \vdash \mathcal{T}$, and we say J is *feasible* if there is a valid schedule of J such that all dag-jobs of J meet their deadlines. We let $S_\infty(J, \alpha)$ be the schedule of J under the ideal algorithm A_∞ on the cores of speed α. We observe that the schedule $S_\infty(J, \alpha)$ executes each vertex as soon as it becomes ready to execute, thereby leaving as little work to be done later as possible.

Figure 2 shows the schedule of task set in Fig. 1 under A_∞ on unit-speed cores, where tasks τ_1 and τ_2 both successively release their dag-jobs with the period $T_1 = T_2 = 26$.

For any task τ_i of $\mathcal{T}$, we denote by J_i the collection of the dag-jobs that may be released by τ_i, written as $J_i \vdash \tau_i$, and which is also contained in the schedule $S_\infty(J, \alpha)$, i.e., $J_i \subseteq J$. For any interval I, we denote by $work(J_i, I, \alpha)$ the amount

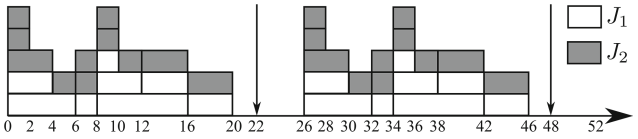

Fig. 2. An example schedule of task set in Fig. 1 under A_∞.

of execution occurring within the interval I in the schedule $S_\infty(J, \alpha)$ of dag-jobs in J_i *with deadlines that fall within* I. For example, in Fig. 2, $work(J_1, I, 1) = 4$ for the interval $I = [16, 22]$.

For any positive integer t, let $work(J_i, t, \alpha)$ be the maximum value that $work(J_i, I, \alpha)$ can take, over any interval I of duration equal to t, i.e.,

$$work(J_i, t, \alpha) = \max_{|I|=t} work(J_i, I, \alpha), \qquad \forall \tau_i \in \mathcal{T} \tag{7}$$

Finally, we define the *work function* $work(\tau_i, t, \alpha)$ of the task τ_i as the maximum value of $work(J_i, t, \alpha)$, over all collection J_i of dag-jobs that may be released by the sporadic DAG task τ_i, i.e.,

$$work(\tau_i, t, \alpha) = \max_{J_i \vdash \tau_i} work(J_i, t, \alpha), \qquad \forall \tau_i \in \mathcal{T} \tag{8}$$

We further extend the notation of the work function from individual tasks to task sets as follows. For any task set $\mathcal{T}$, the work function $work(\mathcal{T}, t, \alpha)$ of $\mathcal{T}$ is defined as the summation of the work functions of all tasks τ_i of $\mathcal{T}$, i.e.,

$$work(\mathcal{T}, t, \alpha) = \sum_{\tau_i \in \mathcal{T}} work(\tau_i, t, \alpha) \tag{9}$$

Figure 3 exhibits the work functions of schedule in Fig. 2.

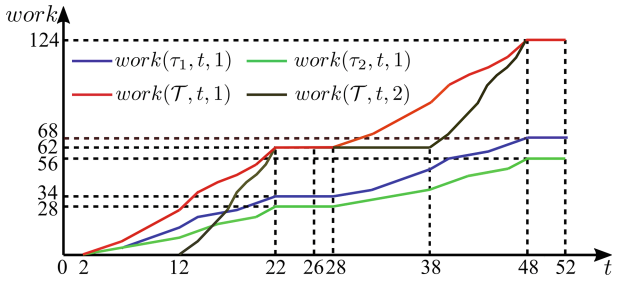

Fig. 3. The work functions of the schedule in Fig. 2.

In the following, we reveal some insights into the work function, which play the important role to support our observations in Sect. 5.1.

4.1 Monotonicity of the Work Function

We discuss whether the work function is a monotonic function with the time t and the speed α.

Lemma 1. *For any task τ_i, any speed $\alpha \geq 1$ and any time $t_1, t_2 \geq 0$, the following inequality holds.*

$$work(\tau_i, t_1, \alpha) \leq work(\tau_i, t_2, \alpha), \qquad if\ t_1 < t_2 \tag{10}$$

Proof. Suppose not, and we have

$$work(\tau_i, t_1, \alpha) > work(\tau_i, t_2, \alpha) \tag{11}$$

We let J_i be a collection of dag-jobs released by τ_i, and let $I_1 = [a, b]$ be an interval of duration equal to t_1, where a is the left boundary of I_1 and b is the right boundary of I_1. Without loss of generality, we assume that

$$work(\tau_i, t_1, \alpha) = work(J_i, I_1, \alpha) \tag{12}$$

We enlarge the interval I_1 into a larger interval I_2 by letting the left boundary of I_2 be $a - \Delta$ (where $\Delta = t_2 - t_1 > 0$), i.e., $I_2 = [a - \Delta, b]$. Since $I_1 \subset I_2$ and the larger interval I_2 may involve more work of J_i that must be done during this interval, we know that

$$work(J_i, I_1, \alpha) \leq work(J_i, I_2, \alpha) \tag{13}$$

By combining (11), (12) and (13), we have

$$work(J_i, I_2, \alpha) > work(\tau_i, t_2, \alpha)$$

and by (7) and (8), we know that $work(J_i, I_2, \alpha) \leq work(\tau_i, t_2, \alpha)$. This leads to a contradiction. □

From Lemma 1, it is easy to derive the following corollary.

Corollary 1. *For any task set $\mathcal{T}$, any speed $\alpha \geq 1$ and any time $t_1, t_2 > 0$, the following inequality holds.*

$$work(\mathcal{T}, t_1, \alpha) \leq work(\mathcal{T}, t_2, \alpha), \quad if\ t_1 < t_2 \tag{14}$$

Proof. It is directly proved by (9) and according to Lemma 1. □

Corollary 1 shows that the work function $work(\mathcal{T}, t, \alpha)$ is a non-decreasing function with time t. For example, the work functions in Fig. 3 all keep the non-decreasing properties.

Lemma 2. *For any task τ_i, any time $t \geq 0$ and any speeds $\alpha_1, \alpha_2 \geq 1$, the following inequality holds.*

$$work(\tau_i, t, \alpha_1) \geq work(\tau_i, t, \alpha_2), \qquad if\ \alpha_1 < \alpha_2 \tag{15}$$

Proof. For any collection J_i of the dag-jobs released by τ_i, and for any interval $I = [a, b]$ of duration equal to t, where a is the left boundary of I, we know that the ideal algorithm A_∞ on the cores of speed α_2 executes more work of J_i during the interval $[0, a]$, and therefore it leaves less work of J_i that to be done during I. Consequently, we have,

$$work(J_i, I, \alpha_1) \geq work(J_i, I, \alpha_2), \quad \forall J_i \vdash \tau_i,\ \alpha_1 < \alpha_2 \tag{16}$$

and by (7), we have

$$work(J_i, t, \alpha_1) \geq work(J_i, I, \alpha_2), \quad \forall J_i \vdash \tau_i,\ \alpha_1 < \alpha_2$$

and by (8), we have

$$work(\tau_i, t, \alpha_1) \geq work(\tau_i, t, \alpha_2)$$

This completes the proof. □

Lemma 2 shows that the work function $work(\tau_i, t, \alpha)$ is a decreasing function of speed α. In the following corollary, we extend Lemma 2 from an individual task to the task set.

Corollary 2. *For any task set $\mathcal{T}$, any time $t \geq 0$ and any speeds $\alpha_1, \alpha_2 \geq 1$, the following inequality holds.*

$$work(\mathcal{T}, t, \alpha_1) \geq work(\mathcal{T}, t, \alpha_2), \quad \text{if } \alpha_1 < \alpha_2 \tag{17}$$

Proof. It is directly proved by (9) and according to Lemma 2. □

For example, in Fig. 3, the curve of work function $work(\mathcal{T}, t, 2)$ is always below the curve of work function $work(\mathcal{T}, t, 1)$.

4.2 Critical Points of the Work Function

In this section, we introduce some critical points of the work function $work(\mathcal{T}, t, \alpha)$. We first define two types of *critical time points* of the work function as follows.

Definition 4 (Left Critical Time Point). *The left critical time point of the work function $work(\mathcal{T}, t, \alpha)$ is the time-instant t^* that satisfies the following conditions.*

- $\forall t < t^*$, $work(\mathcal{T}, t, \alpha) < work(\mathcal{T}, t^*, \alpha)$, *and*
- $\forall \epsilon > 0$, $work(\mathcal{T}, t^*, \alpha) = work(\mathcal{T}, t^* + \epsilon, \alpha)$.

For example, in Fig. 3, $t = 22$ and $t = 48$ are both left critical time points of $work(\mathcal{T}, t, 1)$.

Definition 5 (Right Critical Time Point). *The right critical time point of the work function $work(\mathcal{T}, t, \alpha)$ is the time-instant t^+ that satisfies the following conditions.*

– $\forall t > t^+$, $work(\mathcal{T}, t, \alpha) > work(\mathcal{T}, t^+, \alpha)$, *and*
– $\forall \epsilon > 0$, $work(\mathcal{T}, t^+, \alpha) = work(\mathcal{T}, t^+ - \epsilon, \alpha)$.

For example, in Fig. 3, $t = 28$ is the right critical time point of $work(\mathcal{T}, t, 1)$, and $t = 38$ is the right critical point of $work(\mathcal{T}, t, 2)$.

Definition 6 (Flat Interval). *For any successive critical time points t^* and t^+, where t^* is the left critical time point, and t^+ is the right critical time point, the interval $F = [t^*, t^+]$ is called the flat interval.*

Clearly, for any flat interval $I = [t^*, t^+]$, and for any time-instant $t \in I$, we know that

$$work(\mathcal{T}, t^*, \alpha) = work(\mathcal{T}, t, \alpha) = work(\mathcal{T}, t^+, \alpha) \tag{18}$$

For example, in Fig. 3, the interval $I = [22, 38]$ is the flat interval of $work(\mathcal{T}, t, 2)$.

Definition 7 (Slope Interval). *For any successive critical time points t^+ and t^*, where t^+ is the right critical time point, and t^* is the left critical time point, the interval $S = [t^+, t^*]$ is called the slope interval.*

For example, in Fig. 3, the interval $S = [38, 48]$ is the slope interval of $work(\mathcal{T}, t, 2)$.

Definition 8 (Non-Convex Slope Interval). *A slope interval S of the work function $work(\mathcal{T}, t, \alpha)$ is non-convex if the following inequality holds for any $t_1, t_2 \in S$, and any $\lambda \in (0, 1)$,*

$$\lambda work(\mathcal{T}, t_1, \alpha) + (1-\lambda) work(\mathcal{T}, t_2, \alpha) \geq work(\mathcal{T}, \lambda t_1 + (1-\lambda) t_2, \alpha).$$

Moreover, a work function is non-convex if it contains no convex slope interval.

Definition 9 (Encounter Point). *The encounter point of the work function $work(\mathcal{T}, t, \alpha)$ is the time point t^* such that for any speeds $\alpha_1, \alpha_2 \geq 1$,*

$$work(\mathcal{T}, t^*, \alpha_1) = work(\mathcal{T}, t^*, \alpha_2) \tag{19}$$

For example, $t^* = 22$ and $t^* = 48$ are both the encounter points of $work(\mathcal{T}, t, \alpha)$.

In the following, we show how to identify an encounter point. Before going into details, we first give the following lemma.

Lemma 3. *For any time $t \geq 0$, if there is a collection J_i of dag-jobs released by the task τ_i and an interval $I = [a, b]$ of duration equal to t such that*

$$work(\tau_i, t, \alpha) = work(J_i, I, \alpha),$$

then the right boundary b of I must equal to $r_i + D_i$, where r_i is the release time of a dag-job of J_i.

Proof. Suppose not. There is an interval $I' = [a+\delta, b+\delta]$ of duration equal to t (where $\delta < T_i$), such that $work(J_i, I', \alpha) > work(J_i, I, \alpha)$. As illustrated in Fig. 4, although there may be a dag-job of τ_i released between the interval $[b, b+\delta]$, the work function $work(J_i, I', \alpha)$ does not involve the workload of such dag-job since its deadline does not fall in the interval I'. Therefore, $work(J_i, I', t)$ will not bring more workload than $work(J_i, I, t)$. More precisely, let $W[a, a+\delta]$ be the work done by A_∞ within the interval $[a, a+\delta]$, and we know that $W[a, a+\delta] \geq 0$. Moreover, since $work(J_i, I', \alpha) = work(J_i, I, \alpha) - W[a, a+\delta]$ (See in Fig. 4), we have: $work(J_i, I', \alpha) \leq work(J_i, I, \alpha)$. This contradicts the assumption. □

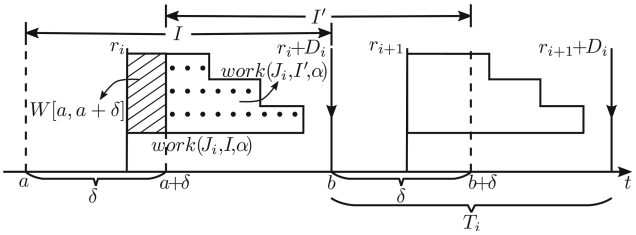

Fig. 4. Illustration for the proof of Lemma 3.

Lemma 4 reveals a sufficient condition for the encounter points of the work function $work(\tau_i, t, \alpha)$.

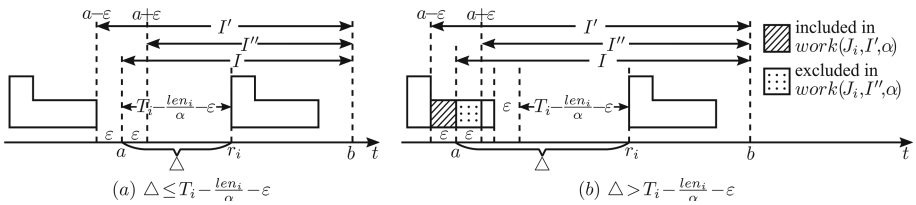

Fig. 5. Illustration for the proof of Lemma 5.

Lemma 4. *For any task τ_i, any speed $\alpha \geq 1$ and any time $t = kT_i + D_i$, $work(\tau_i, t, \alpha) = (k+1)vol_i$.*

Proof. There must exist a collection J_i of dag-jobs released by the task τ_i and an interval $I = [a, b]$ of the duration equal to t, such that $work(J_i, I, \alpha) = work(\tau_i, t, \alpha)$. Since the length of the interval I equals to $KT_i + D_i$ and according to Lemma 3, the left boundary a of I equals to the release time r_i of a dag-job of J_i, and the right boundary b of I equals to the deadline $r_i + kT_i + D_i$ of the other dag-job of J_i. It indicates that $work(J_i, I, \alpha) = (k+1)vol_i$, and therefore, $work(\tau_i, t, \alpha) = (k+1)vol_i$. □

From Lemma 4, we directly derive the following corollary.

Corollary 3. *The time-instant t^* is an encounter point of the work function $work(\mathcal{T}, t, \alpha)$ for any $\alpha \geq 1$, if it satisfies $T_i|(t^* - D_i)$, $\forall \tau_i \in \mathcal{T}$.*

Proof. This is proved by Lemma 4 and according to Definition 9.

Lemma 5. *For any task τ_i, any speed $\alpha > 1$, and any time $t = kT_i + D_i + \Delta$ (where $\Delta < D_i$), the following conditions contradict with each other.*

$$work(\tau_i, t, \alpha) = work(\tau_i, t + \epsilon, \alpha), \quad \forall \epsilon > 0 \tag{20}$$

$$work(\tau_i, t, \alpha) > work(\tau_i, t - \epsilon, \alpha), \quad \forall \epsilon > 0 \tag{21}$$

Proof. There must exist a collection J_i of dag-jobs released by the task τ_i and an interval $I = [a, b]$ of duration equal to t such that $work(\tau_i, t, \alpha) = work(J_i, I, \alpha)$. According to Lemma 3, the right boundary b of I equals to $r_i + kT_i + D_i$, and the left boundary a of I equals to $r_i - \Delta$, where r_i is the release time of a dag-job of J_i, as illustrated in Fig. 5. There are two possible cases.

- If $\Delta \leq T_i - \frac{len_i}{\alpha} - \epsilon$ (See in Fig. 5(a)), we know that the work done by A_∞ within the interval $[a - \epsilon, a]$ equals to 0. Moreover, since $\Delta > 0$, we know that the work done by A_∞ within the interval $[a, a + \epsilon]$ equals to 0. Therefore, $work(J_i, I, \alpha) = work(J_i, I', \alpha)$ and $work(J_i, I, \alpha) = work(J_i, I'', \alpha)$, where $I' = [a - \epsilon, b]$ is the interval of duration equal to $t + \epsilon$, and $I'' = [a + \epsilon, b]$ is the interval of duration equal to $t - \epsilon$. According to Lemma 3, we have $work(\tau_i, t + \epsilon, \alpha) = work(\tau_i, t, \alpha)$ and $work(\tau_i, t - \epsilon, \alpha) = work(\tau_i, t, \alpha)$.
- If $\Delta > T_i - \frac{len_i}{\alpha} - \epsilon$ (See in Fig. 5(b)), we know that the work done by A_∞ within the interval $[a - \epsilon, a]$ must be larger than 0. Moreover, since $\Delta < D_i$, we know that the work done by A_∞ within the interval $[a, a + \epsilon]$ must be larger than 0. Therefore, $work(J_i, I, \alpha) < work(J_i, I', \alpha)$ and $work(J_i, I, \alpha) > work(J_i, I'', \alpha)$, where $I' = [a - \epsilon, b]$ is the interval of duration equal to $t + \epsilon$, and $I'' = [a + \epsilon, b]$ is the interval of duration equal to $t - \epsilon$. According to Lemma 3, we have $work(\tau_i, t + \epsilon, \alpha) > work(\tau_i, t, \alpha)$ and $work(\tau_i, t - \epsilon, \alpha) < work(\tau_i, t, \alpha)$.

In sum, we know that the conditions of Lemma 5 contradict with each other.

The following lemma ties the encounter point and the critical time point together, which plays a very important role to derive the main result in Sect. 5.1.

Lemma 6. *Any left critical time point t^* of the work function $work(\mathcal{T}, t, \alpha)$ must be an encounter point of $work(\mathcal{T}, t, \alpha)$.*

Proof. According to Corollary 3, for some time t^*, if $\forall \tau_i \in \mathcal{T}$, $T_i|(t^* - D_i)$, then t^* is an encounter point. Therefore, it is sufficient to prove this lemma by showing that the left critical time point t^* satisfies $T_i|(t^* - D_i)$, $\forall \tau_i \in \mathcal{T}$.

Suppose that there is a task τ_i such that $t^* = kT_i + D_i + \Delta$ with a positive integer $\Delta < D_i$, we aim to show that t^* is not a left critical time point. According to Lemma 5, we know that for any $\epsilon > 0$, there are two possible cases.

- If $work(\tau_i, t, \alpha) = work(\tau_i, t + \epsilon, \alpha)$ and $work(\tau_i, t, \alpha) \leq work(\tau_i, t - \epsilon, \alpha)$, then we know that $work(\mathcal{T}, t, \alpha) = work(\mathcal{T}, t + \epsilon, \alpha)$, and $work(\mathcal{T}, t, \alpha) \leq work(\mathcal{T}, t - \epsilon, \alpha)$, indicating that the first condition of Definition 4 does not hold.
- If $work(\tau_i, t, \alpha) \neq work(\tau_i, t + \epsilon, \alpha)$ and $work(\tau_i, t, \alpha) > work(\tau_i, t - \epsilon, \alpha)$, then we know that $work(\mathcal{T}, t, \alpha) \neq work(\mathcal{T}, t + \epsilon, \alpha)$, and $work(\mathcal{T}, t, \alpha) > work(\mathcal{T}, t - \epsilon, \alpha)$, indicating that the second condition of Definition 4 does not hold.

In sum, we know that t^* is not a left critical time point. □

5 A Review of the Main Result of [4]

Bonifaci et al. [4] first use the work function to derive a sufficient condition for the schedulability test. We now describe their main result in a manner consistent with the terminology introduced in above sections.

Theorem 2 (Lem. 3 of [4]). *Consider a collection J of dag-jobs released by the tasks in $\mathcal{T}$, and let $\alpha \geq 1$. Then at least one of the following holds:*

i all dag-jobs in J are completed within their deadline under global EDF on m cores of speed α, or
ii J is not feasible under A_∞ on unit-speed cores, or
iii there is an interval I such that any feasible schedule for J must finish more than $(\alpha m - m + 1)|I|$ units of work within I.

Proof Sketch of Theorem 2

It is sufficient to prove this theorem by assuming that both (i) and (ii) do not hold, and showing that (iii) satisfies. More specifically, J can be successfully scheduled by A_∞ on unit-speed cores, but fails to be scheduled by global EDF on m cores of speed α. In the following, the key point is to construct an interval I such that any feasible schedule of J must execute more than $(\alpha m - m + 1)|I|$ units of work within I.

Among all feasible schedule of J, we focus on the schedule $S_\infty(J, 1)$ (Recall that $S_\infty(J, 1)$ is obtained by scheduling J under A_∞ on unit-speed cores, and according to the assumption that A_∞ successfully schedules J, $S_\infty(J, 1)$ is feasible). For any feasible schedule, Bonifaci et al. [4] give the following observation.

Observation 1 *For any feasible schedule $S_f(J, 1)$ under scheduling algorithm A_f on m unit-speed cores, and for any interval I, the work of $S_f(J, 1)$ that must be done (by A_f) within I is larger than the work of $S_\infty(J, 1)$ that must be done (by A_∞) within I.*

Proof For any interval $I = [a, b]$, we denote by I' the interval before I, i.e., $I' = [0, a)$. We know that the following statement holds.

(∗) The work done by A_f (on m unit-speed cores) within I' is no more than the work done by A_∞ (on infinite number of unit-speed cores) within I'.

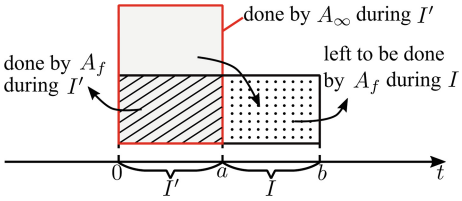

Fig. 6. Illustration for the proof of Observation 1.

By (*), we know that A_f leaves more work that must be done within I than the one that must be done by A_∞ within I, as illustrated in Fig. 6. This completes the proof.

Here we should note that the above observation cannot be extended to the feasible schedule $S_f(J, \alpha)$ such that J is scheduled by A_f on α-speed cores. This is because the key statement (*) cannot be satisfied on α-speed cores as shown in Example 1.

Example 1. Figure 7(a) gives a task τ_1, and we schedule it by A_∞ on unit-speed cores as shown in Fig. 7(b), and schedule it by the global EDF on 2 cores of speed 4 as shown in Fig. 7(c). Clearly, during the interval $[0, 2]$, all workload of a dag-job released by τ_1 is finished by A_f on 2 cores of speed 4, but only half of them is done by A_∞ on unit-speed cores, i.e., the workload done by A_∞ (on unit-speed cores) is less than the workload done by the global EDF (on 2 cores of speed 4).

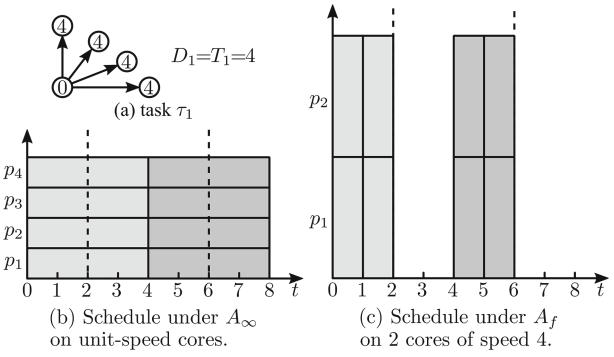

Fig. 7. The task τ_1 and its schedules discussed in Example 1.

Recall that $work(J, I, 1)$ denotes the work of J that must be done by A_∞ (on infinite number of unit-speed cores) within I, and denote by $work_f(J, I, 1)$

the work of J that must be done by A_f (on m unit-speed cores) within I, and according to Observation 1, we know that for any interval I,

$$work(J, I, 1) \leq work_f(J, I, 1) \tag{22}$$

Bonifaci et al. [4] construct an interval I^*, and prove that the work (denoted as $W_{edf}(J, I^*, \alpha)$) done by EDF on m α-speed cores within I^* is smaller than the work of J that must be done by A_∞ (on infinite number of unit-speed cores) within I^*, i.e.,

$$W_{edf}(J, I^*, \alpha) \leq work(J, I^*, 1)$$

and by (22), we know that

$$W_{edf}(J, I^*, \alpha) \leq work_f(J, I^*, 1)$$

and moreover, Bonifaci et al. [4] also show that

$$W_{edf}(J, I^*, \alpha) \geq (\alpha m - m + 1)|I^*|$$

and thus, we have

$$work_f(J, I^*, 1) \geq (\alpha m - m + 1)|I^*|$$

This completes the proof of Theorem 2.

It should be emphasized that Observation 1 is very important in the proof, which indicates that the feasible schedule mentioned in (iii) of Theorem 2 must be restricted on the unit-speed cores by default, even though it is not explicitly stated in the original theorem.

5.1 Existing Reformulations of Theorem 2

Baruah [1] and Li et al. [10] respectively reformulate Theorem 2 as follows.

Theorem 3 (Thm. 1 of [1]). *Sporadic DAG task set $\mathcal{T}$ is global EDF schedulable on m α-speed cores if the following conditions both hold.*

i For any task $\tau_i \in \mathcal{T}$, $len_i \leq D_i$, and
ii For any time $t \geq 0$,

$$work(\mathcal{T}, t, 1) \leq (\alpha m - m + 1) \times t \tag{23}$$

Theorem 4 (Lem. 8 of [10]). *Sporadic DAG task set $\mathcal{T}$ is global EDF schedulable on m α-speed cores if*

$$work(\mathcal{T}, t, \alpha) \leq (\alpha m - m + 1) \times t, \quad \forall t \geq 0 \tag{24}$$

Clearly, Theorem 3 and Theorem 4 are totally different, and only one of them is equivalent to Theorem 2. The following lemmas reveal which one is the equivalent reformulation.

Lemma 7. *Theorem 3 is equivalent to Theorem 2.*

Proof. On the one hand, (i) of Theorem 3 equivalently indicates that any collection J released by the tasks in $\mathcal{T}$ is feasible under A_∞ on unit-speed cores, i.e., $S_\infty(J, 1)$ is feasible. Therefore, (ii) of Theorem 2 does not hold.

On the other hand, (ii) of Theorem 3 equivalently indicates that there is a feasible schedule of J, e.g. $S_\infty(J, 1)$ under A_∞, such that the work of J that must be done by A_∞ within any interval I is no more than $(\alpha m - m + 1)|I|$, i.e., (iii) of Theorem 2 does not hold.

According to Theorem 2, (i) of Theorem 2 must hold, i.e., $\mathcal{T}$ is global EDF schedulable on m α-speed cores.

From Lemma 7, we know that Theorem 3 is correct. Moreover, Theorem 4 seems correct due to the following reasons.

- The task set $\mathcal{T}$ is assumed to be schedulable under A_∞ on unit-speed cores by default in [10], i.e., (ii) of Theorem 2 does not hold. Moreover, it obviously indicates that $\mathcal{T}$ is A_∞-schedulable on α-speed cores.
- (24) ensures that for any collection J released by the tasks of $\mathcal{T}$ the feasible schedule $S_\infty(J, \alpha)$ under A_∞ on α-speed cores satisfies the following condition: the work of J that must be done by A_∞ on α-speed cores within any interval I is no more than $(\alpha m - m + 1)|I|$, i.e., (iii) of Theorem 2 "does not" hold.

According to Theorem 2, $\mathcal{T}$ is global EDF schedulable on m α-speed cores. This seems complete the proof of Theorem 4.

However, the proof above may be incorrect. The reason is as follows. From Observation 1 and Example 1, we know that the feasible schedule mentioned in (iii) of Theorem 2 is assumed to be on unit-speed cores by default. Although the schedule $S_\infty(J, \alpha)$ used in Theorem 4 is feasible, it is not applied on unit-speed cores. Therefore, it is not sufficient to use Theorem 2 (nor Theorem 3) to prove Theorem 4. Actually, Theorem 4 overwhelms Theorem 3 as shown in the following lemma.

Lemma 8. *Theorem 4 overwhelms Theorem 3.*

Proof. From Corollary 2, we know that $\forall t \geq 0$ and $\alpha > 1$, $work(\mathcal{T}, t, \alpha) \leq work(\mathcal{T}, t, 1)$. Therefore, if (23) holds, then (24) must hold. It indicates that Theorem 4 overwhelms Theorem 3, and according to Lemma 7, we complete the proof.

The following example reveals that Theorem 4 strictly overwhelms Theorem 3, i.e., there is a work function that satisfies (24), but does not satisfy (23).

Example 2. We consider the task τ_1 in Fig. 8(a), and schedule it by A_∞ as shown in Fig. 8(b) and (c).

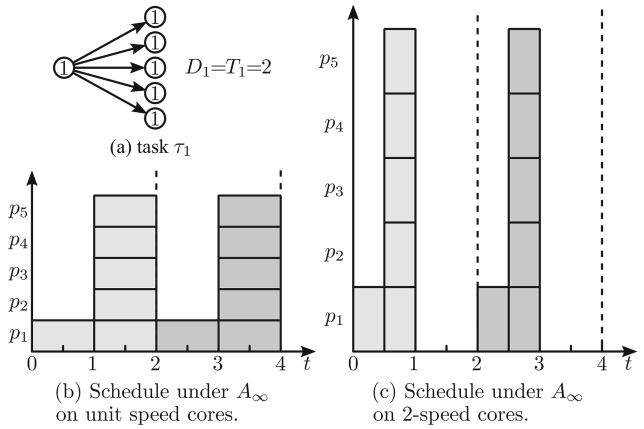

Fig. 8. An example task set and its schedule under A_∞.

The work function of τ_1 is given as follows.

$$work(\tau_1,t,\alpha)=\begin{cases} 0 & 0<t\leq 2-\frac{2}{\alpha} \\ 5\alpha t-10\alpha+10 & 2-\frac{2}{\alpha}<t\leq 2-\frac{1}{\alpha} \\ \alpha t-2\alpha+6 & 2-\frac{1}{\alpha}<t\leq 2 \\ 6\lfloor\frac{t}{2}\rfloor+work(\tau_1,t-2\lfloor\frac{t}{2}\rfloor,\alpha) & t>2 \end{cases}$$

By letting $m = 3$ and $\alpha \geq 2$, we first show the violation of (23), i.e., there is a time point t such that $work(\tau_1,t,1) > (\alpha m - m + 1)t = (3\alpha - 2)t$. Such a time point must exist, because $work(\tau_1,0,1) = 0$ and for any $t \in [0, 2 - \frac{1}{\alpha}]$, the gradient of $work(\tau_1,t,1)$ equals 5α, which is larger than $3\alpha - 2$. As illustrated by Fig. 9, the blue curve represents the work function $work(\tau_1,t,1)$, and during the interval $t \in (0,1]$, we know that $work(\tau_1,t,1) \geq (\alpha m - m + 1)t$.

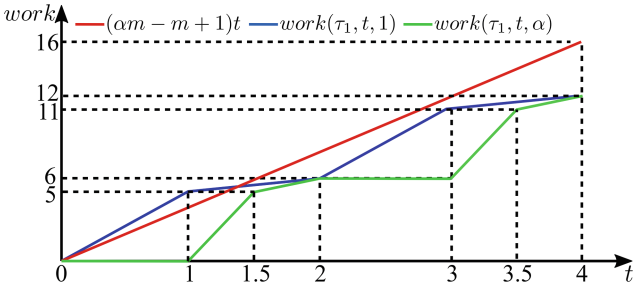

Fig. 9. The work function curves with $m = 3$ and $\alpha = 2$.

In the following, we show that (24) holds, i.e., $\forall t$, $work(\tau_1,t,\alpha) \leq (\alpha m - m + 1)t$ when $m = 3$ and $\alpha \geq 2$. As we know that the period of work function $work(\tau_1,t,\alpha)$ equals 2, we discuss the value of $\Delta = (\alpha m - m + 1)t - work(\tau_1,t,\alpha)$

(with $m = 3$ and $\alpha \geq 2$) in each period $t \in [2k, 2k+2]$ ($k = 0, 1, \cdots$). We further divide each period into three disjoint intervals as follows.

- When $t \in [2k, 2k + 2 - \frac{2}{\alpha})$, and since $\alpha \geq 2$, we have

$$\Delta \geq 6\alpha k - 10k \geq 0$$

- When $t \in [2k + 2 - \frac{2}{\alpha}, 2k + 2 - \frac{1}{\alpha})$, and since $\alpha \geq 2$, we have

$$\Delta \geq (6\alpha - 10)(k+1) + \frac{4}{\alpha} \geq 0$$

- When $t \in [2k + 2 - \frac{1}{\alpha}, 2k + 2]$, and since $\alpha \geq 2$, we have

$$\Delta \geq (6\alpha - 10)k + (6\alpha - 12) + \frac{2}{\alpha} \geq 0$$

In sum, we know that (24) holds for any time point t, as illustrated in Fig. 9, where the green curve representing the work function $work(\tau_1, t, \alpha)$ is always below the red curve representing $(\alpha m - m + 1)t$.

The following lemma shows a sufficient condition which ensures that if (24) holds, then (23) holds.

Lemma 9. *For any non-convex work function $work(\mathcal{T}, t, \alpha)$, (24) implies (23).*

Proof. Suppose not. There is a non-convex work function $work(\mathcal{T}, t, \alpha)$ that satisfies (24) holds, but violates (23), i.e., there is a time t such that

$$work(\mathcal{T}, t, 1) > (\alpha m - m + 1)t \tag{25}$$

According to Lemma 1, the work function $work(\mathcal{T}, t, 1)$ is an increasing function, and thus, there must be a time $t < t^*$ such that

$$work(\mathcal{T}, t^*, 1) = (\alpha m - m + 1)t^* \tag{26}$$

We consider two cases.

Case 1: t^* is in a flat interval $I = (a, b]$. From Definition 6, we know that

$$work(\mathcal{T}, a, 1) = work(\mathcal{T}, t^*, 1)$$

and by (27), we have

$$work(\mathcal{T}, a, 1) = (\alpha m - m + 1)t^*$$

and since $(\alpha m - m + 1)a < (\alpha m - m + 1)t^*$ (with $a < t^*$), we know that

$$work(\mathcal{T}, a, 1) > (\alpha m - m + 1)a \tag{27}$$

From Definition 4, we know that a is a left critical time point, and according to Lemma 6, a must be an encounter point. From Definition 9, we know that

$$work(\mathcal{T}, a, 1) = work(\mathcal{T}, a, \alpha)$$

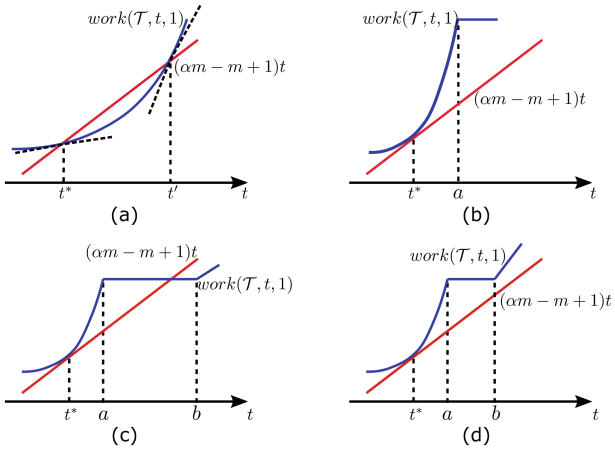

Fig. 10. Illustration for the proof of Lemma 9.

and by (27), we have

$$work(\mathcal{T}, a, \alpha) > (\alpha m - m + 1)a$$

It indicates that (24) does not hold for Case 1.

Case 2: t^* is in a slope interval I, and without loss of generality, we assume that the gradient of the work function $work(\mathcal{T}, t, 1)$ at t^* is no less than $\alpha m - m + 1$. Suppose not. The gradient of the work function $work(\mathcal{T}, t, 1)$ at t^* is no more than $\alpha m - m + 1$. According to the assumption (25) and since the slope interval I is non-convex, we know that there must be a time t' such that $work(\mathcal{T}, t', 1) = (\alpha m - m + 1)t'$, and where the gradient of the work function $work(\mathcal{T}, t, 1)$ at t' is larger than $\alpha m - m + 1$ as illustrated in Fig. 10(a).

In the following, we consider two cases.

– If there is no flat interval follows the slope interval I. Since the gradient of the work function $work(\mathcal{T}, t, 1)$ at t^* is no less than $\alpha m - m + 1$, and the slope interval I is non-convex, we know that for any $t > t^*$,

$$work(\mathcal{T}, t, 1) > (\alpha m - m + 1)t \tag{28}$$

Let a be the nearest encounter point after t^*, and according to Lemma 6 and Definition 9, $work(\mathcal{T}, a, 1) = work(\mathcal{T}, a, \alpha)$. By (28), we have the following inequality as shown in Fig. 10(b).

$$work(\mathcal{T}, a, \alpha) > (\alpha m - m + 1)a$$

– Otherwise, denote by $I' = [a, b]$ be the flat interval that follows the slope interval I. There are three cases.
  - $work(\mathcal{T}, a, 1) < (\alpha m - m + 1)a$. In this case, since the slope interval I is non-convex, and the gradient of the work function $work(\mathcal{T}, t, 1)$ at t^* is larger than $\alpha m - m + 1$, for any time $t \in (t^*, a]$, $work(\mathcal{T}, t, 1) > (\alpha m - m + 1)t$. This leads to a contradiction.

- $work(\mathcal{T}, a, 1) > (\alpha m - m + 1)a$ and $work(\mathcal{T}, b, 1) < (\alpha m - m + 1)b$ as illustrated in Fig. 10(c). In this cases, we know that there must be a time $t \in I$ such that $work(\mathcal{T}, t, 1) = (\alpha m - m + 1)t$, and we have discussed this in Case 1.
- $work(\mathcal{T}, b, 1) > (\alpha m - m + 1)b$ as illustrated in Fig. 10(d). In this case, according to Definition 6, we know that $work(\mathcal{T}, a, 1) = work(\mathcal{T}, b, 1)$, and thus, $work(\mathcal{T}, a, 1) > (\alpha m - m + 1)a$. According to Lemma 6, we have $work(\mathcal{T}, a, 1) = work(\mathcal{T}, a, \alpha)$. Therefore, we know that $work(\mathcal{T}, a, \alpha) > (\alpha m - m + 1)a$.

In sum, (24) does not hold for Case 2. This completes the proof.

6 Conclusion

Since Bonifaci first proposed the work function in [4], the work function plays a very important role in schedulability analysis of the sporadic DAG tasks. Especially, Li et al. [10] derive the best capacity augmentation bound for global EDF algorithm by using the work function methodology. This paper revisits the work function methodology, and shows that Lem. 8 of [10] which is said to be a reformulation of Lem. 3 of [4] is not equivalently reformulated from Lem. 3 of [4], and we prove that Lem.8 of [10] strictly overwhelms Lem. 3 of [4]. Thus, the main result of [10] should be carefully discussed.

References

1. Baruah, S.: Improved multiprocessor global schedulability analysis of sporadic DAG task systems. In: 26th Euromicro Conference on Real-Time Systems, pp. 97–105. IEEE (2014)
2. Baruah, S.: The federated scheduling of systems of conditional sporadic DAG tasks. In: International Conference on Embedded Software (EMSOFT), pp. 1–10. IEEE (2015)
3. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: an efficient multithreaded runtime system. ACM SigPlan Notices **30**(8), 207–216 (1995)
4. Bonifaci, V., Marchetti-Spaccamela, A., Stiller, S., Wiese, A.: Feasibility analysis in the sporadic DAG task model. In: 2013 25th Euromicro Conference on Real-Time Systems, pp. 225–233. IEEE (2013)
5. Ferry, D., Li, J., Mahadevan, M., Agrawal, K., Gill, C., Lu, C.: A real-time scheduling service for parallel tasks. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 261–272. IEEE (2013)
6. Huang, H.M., Tidwell, T., Gill, C., Lu, C., Gao, X., Dyke, S.: Cyber-physical systems for real-time hybrid structural testing: a case study. In: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, pp. 69–78 (2010)
7. Jiang, X., Long, X., Guan, N., Wan, H.: On the decomposition-based global EDF scheduling of parallel real-time tasks. In: 2016 IEEE Real-Time Systems Symposium (RTSS), pp. 237–246. IEEE (2016)

8. Kim, J., Kim, H., Lakshmanan, K., Rajkumar, R.: Parallel scheduling for cyber-physical systems: analysis and case study on a self-driving car. In: Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems, pp. 31–40 (2013)
9. Li, J., Agrawal, K., Lu, C., Gill, C.: Analysis of global EDF for parallel tasks. In: 25th Euromicro Conference on Real-Time Systems, pp. 3–13. IEEE (2013)
10. Li, J., Chen, J.J., Agrawal, K., Lu, C., Gill, C., Saifullah, A.: Analysis of federated and global scheduling for parallel real-time tasks. In: 26th Euromicro Conference on Real-Time Systems, pp. 85–96. IEEE (2014)
11. Li, J., Luo, Z., Ferry, D., Agrawal, K., Gill, C., Lu, C.: Global EDF scheduling for parallel real-time tasks. Real-Time Syst. **51**, 395–439 (2015)
12. Marongiu, A., Capotondi, A., Tagliavini, G., Benini, L.: Improving the programmability of STHORM-based heterogeneous systems with offload-enabled OpenMP. In: Proceedings of the First International Workshop on Many-core Embedded Systems, pp. 1–8 (2013)
13. Melani, A., Bertogna, M., Bonifaci, V., Marchetti-Spaccamela, A., Buttazzo, G.C.: Response-time analysis of conditional DAG tasks in multiprocessor systems. In: 27th Euromicro Conference on Real-Time Systems, pp. 211–221. IEEE (2015)
14. OpenMP Forum: OpenMP Application Program Interface, Version 3.0. OpenMP Architecture Review Board, May 2008. https://www.openmp.org/wp-content/uploads/spec30.pdf
15. Qamhieh, M., Fauberteau, F., George, L., Midonnet, S.: Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In: Proceedings of the 21st International conference on Real-Time Networks and Systems, pp. 287–296 (2013)
16. Qamhieh, M., George, L., Midonnet, S.: A stretching algorithm for parallel real-time DAG tasks on multiprocessor systems. In: Proceedings of the 22nd International Conference on Real-Time Networks and Systems, pp. 13–22 (2014)
17. Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism. O'Reilly Media, Sebastopol (2007)
18. Saifullah, A., Ferry, D., Li, J., Agrawal, K., Lu, C., Gill, C.D.: Parallel real-time scheduling of DAGs. IEEE Trans. Parallel Distrib. Syst. **25**(12), 3242–3252 (2014)
19. Saifullah, A., Li, J., Agrawal, K., Lu, C., Gill, C.: Multi-core real-time scheduling for generalized parallel task models. Real-Time Syst. **49**, 404–435 (2013)
20. Stotzer, E., et al.: OpenMP on the low-power TI keystone II ARM/DSP system-on-chip. In: Rendell, A.P., Chapman, B.M., Müller, M.S. (eds.) IWOMP 2013. LNCS, vol. 8122, pp. 114–127. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40698-0_9
21. Sun, J., et al.: A capacity augmentation bound for real-time constrained-deadline parallel tasks under GEDF. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2200–2211 (2018)
22. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
23. Wang, C., Chandrasekaran, S., Chapman, B., Holt, J.: libEOMP: a portable OpenMP runtime library based on MCA APIs for embedded systems. In: Proceedings of the 2013 International Workshop on Programming Models and Applications for Multicores and Manycores, pp. 83–92 (2013)