# On Computing Exact WCRT for DAG Tasks [†]

Jinghao Sun[1,3], Feng Li[3], Nan Guan[2*], Wentao Zhu[3], Minjie Xiang[3], Zhishan Guo[4], and Wang Yi[5]
1. Dalian University of Technology, China; 2. The Hong Kong Polytechnic University, Hong Kong, China
3. Northeastern University, China; 4. University of Central Florida, USA; 5. Uppsala University, Sweden

***Abstract*—Most current real-time parallel applications can be modeled as a directed acyclic graph (DAG) task. Existing worst-case response time (WCRT) bounds (e.g., Graham's bound) derived for DAGs may be very pessimistic. No one precisely knows the gap between the WCRT bound and the actual WCRT. In this paper, we aim to derive the exact WCRT of a DAG task under the list scheduling upon multi-core platforms. We encode the WCRT analysis problem into a satisfaction modular theoretical (SMT) formulation based on insights into the list scheduling algorithm, and prove that our SMT program can solve the WCRT precisely, providing an accurate baseline to measure the tightness of the existing WCRT bounds. Experiments show that our method significantly improves the tightness of the WCRT bound, and is practically quite efficient, e.g., it can analyze DAGs with more than 40 vertices in a few seconds.**

## I. Introduction

Nowadays multi-cores are becoming mainstream hardware platforms for embedded and real-time systems. To fully utilize the processing capacity of multi-cores, software should be fully parallelized, i.e., not only inter-task parallelism, but also intra-task parallelism needs to be explored, such that an individual task (abstraction of a parallel program) is able to potentially utilize more than one core at the same time during its execution. Parallel tasks are commonly supported by almost all modern parallel programming languages, e.g., Cilk family, OpenMP and Intel's Thread Building Blocks. The primitives in these languages and libraries, such as `parallel for`, `omp task` and `spawn/sync`, result in intra-task parallelism structures that can be well represented via *Directed Acyclic Graph* (DAG) task models, which have gained much attention in the past few years [1]–[27].

In the DAG task model, each vertex represents a sequence of instructions and each edge represents the interdependency constraints among the vertices. The *response time* of a DAG task is the time taken to execute all vertices in the DAG under the scheduling algorithm applied upon multi-core platforms. In the real-time community, researchers focus on the *worst-case response time* (WCRT). Graham [1] proposes a famous WCRT bound $R_{\text{GRA}}$ for a DAG task $G$ as follows.

$$R_{\text{GRA}} = len(G) + \frac{vol(G) - len(G)}{m} \tag{1}$$

where $len(G)$ is the length of the longest path in $G$; $vol(G)$ is the total execution time of all vertices in $G$; and $m$ is the number of cores. Graham's bound in (1) has been widely used in the real-time analysis of DAG tasks (See in Sec. II for details). However, Graham's bound which is an upper bound of the WCRT may be pessimistic in practice. To the best of our knowledge, there have been no reports of solving DAG task's exact WCRT, and thus, no one can precisely know the gap between Graham's bound and the actual WCRT.

To eliminate the pessimism brought by Graham's bound, in this paper, we propose a novel method for analyzing the response time of a DAG task. The basic idea of our method is borrowed from the "traditional" scheduling theory in the operational research (OR) domain, which aims to formulate the response time analysis problem into an optimization problem. The main difference between our method and OR approaches is that we do not intend to design the scheduling algorithm which constructs the optimal schedule of the DAG task for the purpose of minimizing the response time, but instead, we solve the maximum response time among all possible schedules under a certain given scheduling algorithm. Encoding the scheduling strategies into the constrains of the optimization problem is the main challenge of this paper.

We implement our method by using Satisfaction Modular Techniques (SMT), and formulate the WCRT analysis problem for a DAG task under the list scheduling algorithm upon the multi-core platform into a SMT program. We give a deep insight into the list scheduling algorithm, and encode the list scheduling strategies into the SMT program's constraints. Moreover, as we know that the WCRT analysis problem for a DAG task is inherently $\mathcal{NP}$-hard [1], the computation complexity heavily depends on the number of variables in the SMT program. We involve into the SMT program as few variables as possible. Experiment results show that our method can significantly improve the tightness of the WCRT bound, and is practically quite efficient, e.g., it is capable to analyze the DAG task consisting of 40 vertices within a few seconds. Most importantly, we prove that our SMT model can solve the WCRT of DAGs precisely, which provides an accurate baseline to estimate the tightness of the existing WCRT bounds.

**Motivations.** This paper focuses on a single DAG task that is scheduled by FIFO (list scheduling) policy in a non-preemptive manner, which is motivated by OpenMP applications. In OpenMP, the workload in a `parallel` region can be formulated as a single DAG task [21]. There are two types of scheduling policies in OpenMP, called breath first scheduling (BFS) and work first scheduling (WFS) respectively, and both of them schedule DAG's vertices in FIFO way [22]. Moreover, the vertex-level preemption are not allowed in OpenMP [21]. The specific problem considered in this paper captures these characteristics of OpenMP systems, and thus, our approach is capable to analyze the response time of an OpenMP program.

[†]This work is supported by NSFC (61972076, 61772123, 61532007, 61602104), GRF (15213818, 15204917) and NSF(CNS-1850851).

[*] Corresponding author: Nan Guan, nan.guan@polyu.edu.hk

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.

## II. Related Work

Many existing work studied the response time analysis for *multiple recurrent* DAG tasks. [2]–[12] focused on DAG tasks under global scheduling. [13], [14] studied DAG tasks under partitioned (fixed-priority) scheduling. [15]–[17] focused on the DAG tasks under (semi-)federated scheduling. All the work above focus on sufficient response time analysis in which the WCRT bound of a single DAG (e.g., Graham's bound) is leveraged to bound the inner-task interference. [18]–[20] targeted the exact analysis for multiple DAG tasks. [18] proposed an exact schedulability analysis for global fixed-priority (G-FP) scheduling of recurring segmented self-suspending tasks, and hence, it can evaluate DAG tasks which have only one path. [20] proposed an exact schedulability test for multiple DAG tasks under G-FP by using state space pruning techniques. [19] supported DAG tasks that are scheduled by G-FP on a heterogeneous platform. The WCRT analysis of a single (non-recurrent) DAG task is recently studied, which respectively extends Graham's bound [1] into a DAG task with OpenMP semantics (e.g., `tied` constraints) [21]–[24], heterogeneous multi-core platform [25], [26] and conditional branches [27].

## III. System Model

In this paper, we study the single non-recurrent directed acyclic graph (DAG) task executed on a multi-core platform under the list scheduling. In the following, we first introduce the DAG task model (See in Sec. III-A), and then briefly describe the model of the multi-core platform and the list scheduling algorithm (See in Sec. III-B).

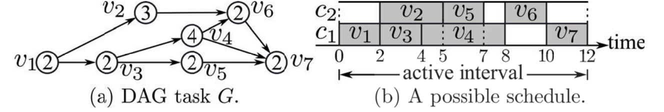
### A. Task Model

We consider a DAG task $G = (V, E)$, where $V$ is the set of vertices, and $E$ is the set of edges. Each vertex $v_i$ of $V$ represents a continuous piece of executing code, and has an execution time $e_i$, which is bounded by a worst-case execution time (WCET) $\mathrm{W}_i$, i.e., $e_i \leq \mathrm{W}_i$. Each edge $(v_i, v_j) \in E$ represents the precedence relation between vertices $v_i$ and $v_j$. Here $v_i$ is a *predecessor* of $v_j$, and $v_j$ is a *successor* of $v_i$. This indicates that $v_j$ cannot be executed until the completion of $v_i$. We denote by $\mathrm{PRED}(v_i)$ the set of predecessors of $v_i$, and $\mathrm{SUCC}(v_i)$ the set of successors of $v_i$. Moreover, $v_i$ is an *ancestor* of $v_j$ if $v_i$ is the predecessor of $v_j$ or is the predecessor of the ancestor of $v_j$. In this case, $v_j$ is a *descendant* of $v_i$. We denote by $\mathrm{ANST}(v_i)$ the set of ancestors of $v_i$, and $\mathrm{DESC}(v_i)$ the set of descendants of $v_i$. Furthermore, we denote by $\mathrm{PARA}(v_i)$ the set of vertices that can be executed in parallel with $v_i$, i.e.,

$$\mathrm{PARA}(v_i) = V - \mathrm{ANST}(v_i) - \mathrm{DESC}(v_i) - \{v_i\} \tag{2}$$

The vertex that has no ancestor is called the *source* vertex. The vertex that has no descendant is called the *sink* vertex. Without loss of generality, we assume that $G$ has exactly one source vertex $v_{src}$ and one sink vertex $v_{snk}$. In case $G$ has multiple source/sink vertices, a dummy source/sink vertex with zero execution time can be added to comply with our assumption.

An example DAG $G$ with 7 vertices and 9 edges is given in Fig. 1(a), where the worst-case execution time $\mathrm{W}_i$ of each vertex $v_i$ is labeled inside the corresponding vertex. As shown in Fig. 1(a), the source vertex of $G$ is $v_1$. The sink vertex of $G$ is $v_7$. The vertex $v_4$ has one predecessor $v_3$, and two successors $v_6$ and $v_7$. Moreover, the set of ancestors of $v_4$ is $\mathrm{ANST}(v_4) = \{v_1, v_3\}$. The set of descendants of $v_4$ is $\mathrm{DESC}(v_4) = \{v_6, v_7\}$. The set of parallel vertices of $v_4$ is $\mathrm{PARA}(v_4) = \{v_2, v_5\}$.



(a) DAG task $G$. (b) A possible schedule.

Fig. 1: An example DAG task $G$ and its possible schedule.

### B. Scheduling Model

We schedule the vertices of $G$ upon the multi-core platform $\mathcal{C}$ consisting of $m$ cores, i.e., $\mathcal{C} = \{c_1, \cdots, c_m\}$. During the scheduling process, we assign each vertex $v_i$ of $G$ to a core $c_k$, and determine the time at which $c_k$ starts to execute $v_i$. A feasible schedule must satisfy the following constraints.

1) At any time, a vertex can only be executed by one core, and the core cannot execute more than one vertex simultaneously.
2) Each vertex $v_i$ is *eligible* to be executed only if the vertices of $\mathrm{PRED}(v_i)$ are all finished. More specifically, for each vertex $v_i$, we denote by $b_i$ the *beginning time* of $v_i$, and denote by $f_i$ the *finishing time* of $v_i$. The condition an eligible vertex $v_i$ must satisfy is the following.
$$b_i \geq f_j, \quad \forall v_j \in \mathrm{PRED}(v_i) \tag{3}$$
3) The execution of a vertex $v_i$ is non-preemptive, i.e.,
$$f_i = b_i + e_i \tag{4}$$
Recall that $e_i$ is the execution time of $v_i$.

**List Scheduling.** Given a list $L$ of the vertices in the DAG $G$, the list scheduling algorithm constructs a schedule of $G$ upon the $m$ cores as follows. Initially, at time 0, all the cores (instantaneously) scan the list $L$, from the beginning, searching for vertices, which are eligible to be executed. The first eligible vertex $v_i$ in $L$ which the core $c_k$ comes to is started by $c_k$; $c_k$ continues to execute $v_i$ for the $e_i$ units of time required to complete $v_i$. In general, at any time a core $c_k$ completes a vertex, it immediately scans $L$ for the first eligible vertex to execute. If there are currently no such vertices, then $c_k$ becomes idle. The core $c_k$ remains idle until some other core $c_{k'}$ completes a vertex $v_j$, at which time $c_k$ (and, of course, $c_{k'}$) immediately scans $L$ for eligible vertices (which may now exist because of the completion of $v_j$). If two (or more) cores both attempt to start executing a vertex, it will be our convention to assign the vertex to the core with the smaller index.

Under the list scheduling algorithm, a possible schedule of $G$ in Fig. 1(a) is given in Fig. 1(b). In this schedule, every vertex (except $v_5$) is executed as soon as it becomes eligible. At time $t = 4$, the vertex $v_5$ is eligible, but $v_5$ does not start its

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.

execution at that time. Instead, the execution of $v_5$ is delayed to $t = 5$. This is because that none of the cores is idle during the time interval $[4, 5]$.

The list scheduling algorithm schedules the DAG task in a work-conserving way, i.e., cores cannot be idle if there are eligible vertices. The work-conserving property is very important for the response time analysis. More formally, we summarize the work-conserving property of the list scheduling algorithm in Lem. 1. Before going into details, we first introduce some useful notations as follows.

**Definition 1** (Critical Predecessor)**.** *The vertex $v_i$ is the critical predecessor of vertex $v_j$ if the following condition holds.*

$$v_i = \arg\max\{f_l | v_l \in \text{PRED}(v_j)\} \tag{5}$$

Intuitively, a predecessor $v_i$ of $v_j$ is *critical* if $v_i$ is latest completed among all the predecessors of $v_j$. For example, in Fig. 1(b), vertex $v_4$ is the critical predecessor of $v_6$. Vertex $v_3$ is the critical predecessor of $v_4$.

**Lemma 1.** *The DAG G is scheduled under the listing scheduling algorithm upon $m$ cores. For any vertex $v_j$ and its critical predecessor $v_i$, the workload executed during the time interval $[f_i, b_j]$ equals $(b_j - f_i) \times m$.*

*Proof.* Since $v_i$ is the critical predecessor of $v_j$, and according to Def. 1, we know that $v_j$ becomes eligible to execute at time $f_i$. There are two cases.

- If $v_j$ starts its execution at time $f_i$, i.e., $b_j - f_i = 0$, the time interval $[f_i, b_j]$ has a zero length. The workload executed during a zero-length time interval is clearly zero, and thus Lem. 1 clearly holds.
- Otherwise, $v_i$ delays its execution to a time point later than $f_i$, i.e., $b_j - f_i > 0$. In this case, we assume that the workload executed during $[f_i, b_j]$ is strictly less than $(b_j - f_i) \times m$, indicating that some core $c_k$ must be idle during the interval $[f_i, b_j]$. Suppose that $c_k$ is idle at time $b_{j'}$ (where $b_{j'} \in [f_i, b_j]$), and according to the list scheduling algorithm, $c_k$ searches the unexecuted eligible vertex (e.g., $v_j$), and begins to execute $v_j$ at $b_{j'}$. This contradicts to the fact that $v_j$ executed at $b_j$.

In sum, under the list scheduling algorithm, the workload executed during $[f_i, b_j]$ is no less than $(b_j - f_i)m$. On the other hand, since the computation resource the multi-core platform can provide during $[f_i, b_j]$ is at most $(b_j - f_i)m$, the workload can be executed during $[f_i, b_j]$ is at most $(b_j - f_i)m$. Therefore, the workload executed during $[f_i, b_j]$ equals $(b_j - f_i)m$. □

**Worst-Case Response Time.** We denote by $b_{src}$ the beginning time of the source vertex $v_{src}$ of $G$, and $f_{snk}$ the finishing time of the sink vertex $v_{snk}$ of $G$. We call the time interval $[b_{src}, f_{snk}]$ as the *active interval* of $G$. Then, the response time $R(G)$ of $G$ equals the length of the active interval, i.e.,

$$R(G) = f_{snk} - b_{src} \tag{6}$$

In this paper, we focus on the worst-case response time (WCRT) of $G$. More specifically, we aim to find a schedule of $G$ under the list scheduling algorithm upon the platform $\mathcal{C}$ such that it has the maximum $R(G)$.

## IV. SMT Formulation

In this section, we apply satisfiability modulo techniques (SMT) to formulate the WCRT analysis problem of the DAG task $G$ under list scheduling upon $m$ cores. As we know that the response time analysis problem is inherently $\mathcal{NP}$-hard [1], the computation complexity heavily depends on the number of variables in the SMT program. To this end, we give a deep insight into the work-conserving property, based on which we involve into the SMT model as few variables as possible. The constants and variables used in our SMT model are summarized in Table I and II.

TABLE I: Constants involved in the SMT formulation

| constant | description |
|---|---|
| $m$ | the number of cores on the platform $\mathcal{C}$ |
| $V$ | the set of vertices of $G$ |
| $E$ | the set of edges of $G$ |
| $\text{W}_i$ | the worst-case execution time of the vertex $v_i \in V$ |
| $\text{PRED}(v_i)$ | the set of predecessors of the vertex $v_i \in V$ |
| $\text{SUCC}(v_i)$ | the set of successors of the vertex $v_i \in V$ |
| $\text{PARA}(v_i)$ | the set of parallel vertices of the vertex $v_i \in V$ |

TABLE II: Variables involved in the SMT formulation

| variable | type | description |
|---|---|---|
| $b_i$ | real | the beginning time of the vertex $v_i$ |
| $f_i$ | real | the finishing time of the vertex $v_i$ |
| $e_i$ | real | the execution time of the vertex $v_i$ |
| $\text{B}_{ij}^x$ | boolean | the variable indicating whether vertex $v_x$ is involved in the interval $[f_i, b_j]$ |
| $\text{L}_{ij}^x$ | boolean | the variable indicating whether the vertex involved in $[f_i, b_j]$ starts before $f_i$ |
| $\text{R}_{ij}^x$ | boolean | the variable indicating whether the vertex involved in $[f_i, b_j]$ finishes after $b_j$ |

**Proposition 1.** *The number of variables involved in the SMT model is bounded by $O(|V| \cdot |E|)$.*

### § Objective Function

From (6), we derive the objective function of the SMT formulation as follows.

$$\max\left(f_{snk} - b_{src}\right) \tag{7}$$

recalling that $f_{snk}$ is the finishing time of $v_{snk}$, and $b_{src}$ is the beginning time of $v_{src}$.

### § Constraints

In the following, we design the constraints to formulate the feasible schedule of the DAG $G$ under the list scheduling algorithms upon $m$ cores.

**WCET bounds.** For any vertex $v_i$ of $V$, the execution time $e_i$ of $v_i$ should be bounded by the WCET. This is ensured by the following constraints.

$$e_i \leq \text{W}_i, \quad \forall v_i \in V \tag{8}$$

**Precedence Constraints.** For any edge $(v_i, v_j)$ of $E$, we use the following constraints to formulate the precedence

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.

constraints such that the vertex $v_j$ cannot starts its execution unless the predecessor $v_i$ of $v_j$ finishes.

$$f_i \leq b_j, \quad \forall (v_i, v_j) \in E \tag{9}$$

**Non-preemptive.** For any vertex $v_i$ of $V$, the following constraints indicate that the vertex $v_i$ is executed in a non-preemptive way.

$$f_i = b_i + e_i, \quad \forall v_i \in V \tag{10}$$

**Work-Conserving.** The list scheduling algorithm schedules the DAG $G$ in a work-conserving way, i.e., at any time, there is no idle core in $\mathcal{C}$ if some vertex $v_j$ is eligible. There are two cases.

CASE 1. If the eligible vertex $v_j$ has no predecessor, i.e., $v_j$ is the source vertex of $G$: $v_j = v_{src}$, according to the work-conserving property, $v_{src}$ must be executed on a core of $\mathcal{C}$ at time 0. This is ensured by Constraint (11).

$$b_{src} \quad = \quad 0 \tag{11}$$

CASE 2. If the eligible vertex $v_j$ has at least one predecessor. In this case, we let $v_i$ be the critical predecessor of $v_j$, and without loss of generality, we assume that the time interval $[f_i, b_j]$ has a non-zero length. According to Lem. 1, the workload executed during the interval $[f_i, b_j]$ should be no less than $(b_j - f_i) \times m$. In the following, we provide the constraints to formulate the work-conserving property stated in Lem. 1. Before going into details, we first give some useful notations.

For any vertex $v_j$ and its critical predecessor $v_i$, the logic expression $\Pi_1$ in (12) indicates whether the length of the time interval $[f_i, b_j]$ is strictly larger than 0.

$$\Pi_1: \quad \left( \bigwedge_{v_x \in \text{PRED}(v_j)} f_i \geq f_x \right) \wedge f_i < b_j \tag{12}$$

**Lemma 2.** *For any vertex $v_j$ and its critical predecessor $v_i$, $\Pi_1 = 1$ indicates a non-zero length of the time interval $[f_i, b_j]$.*

*Proof.* From (12), we know that if $\Pi_1 = 1$, the following conditions must both be satisfied.

- The first item of (12) is true, i.e., $v_i$ has the maximum finishing time among all predecessors of $v_j$. According to Def. 1, $v_i$ is the critical predecessor of $v_j$.
- The second item of (12) is true. In this case, the time interval $[f_i, b_j]$ has a non-zero length.

In sum, $\Pi_1 = 1$ indicates that $v_i$ is the critical predecessor of $v_j$, and the length of the time interval $[f_i, b_j]$ is larger than 0. This completes the proof. □

**Definition 2** (Involved Vertices)**.** *For any vertex $v_j$ and its critical predecessor $v_i$, a vertex $v_x$ is involved in the time interval $[f_i, b_j]$ if the following condition holds.*

$$\Pi_2: \quad b_x < b_j \wedge f_x > f_i \tag{13}$$

*We denote by $\text{I}_{ij}$ the set consisting of all the vertices that are involved in the interval $[f_i, b_j]$.*

For example, in Fig. 1, as we know that vertex $v_3$ is the critical predecessor of $v_5$, the involved vertices of the time interval $[f_3, b_5]$ (with length 1) include $v_2$ and $v_4$.

**Lemma 3.** *For any vertex $v_j$ and its critical predecessor $v_i$, $v_x \in \text{PARA}(v_j)$ if $v_x \in \text{I}_{ij}$.*

*Proof.* Suppose not. For any vertex $v_x$ involved in the interval $[f_i, b_j]$, i.e., $v_x \in \text{I}_{ij}$, we assume that $v_x \notin \text{PARA}(v_j)$, and there are two possible cases as follows.

- If $v_x$ is an ancestor of $v_j$, from $\Pi_2$ of (13), we know that $f_x > f_i$, i.e., $v_x$ finishes later than $v_i$. This contradicts to the fact that $v_i$ is the critical predecessor of $v_j$.
- If $v_x$ is a descendant of $v_j$, from $\Pi_2$ of (13), we know that $b_x < b_j$, i.e., $v_x$ begins earlier than $v_j$. This contradicts to the fact that a vertex can begin its execution only when its predecessors (and its ancestors) all finish the execution.

In sum, and from (2), we know that $v_x$ must belong to $\text{PARA}(v_j)$. This completes the proof. □

For any edge $(v_i, v_j) \in E$, and for any vertex $v_x \in \text{PARA}(v_j)$, we denote the boolean variable $\text{B}_{ij}^x$ as follows.

$$\text{B}_{ij}^x = \begin{cases} 1: v_i \text{ is the critical predecessor of } v_j, \text{ and} \\ \quad\; v_x \text{ is the involved vertex of } [f_i, b_j] \\ 0: \qquad\qquad else \end{cases} \tag{14}$$

With the definition of $\text{B}_{ij}^x$ in (14), and according to Lem. 3, the involved vertex set $\text{I}_{ij}$ of the interval $[f_i, b_j]$ can be defined as follows.

$$\text{I}_{ij} = \{v_x | \text{B}_{ij}^x = 1, \forall v_x \in \text{PARA}(v_j)\} \tag{15}$$

To implement (14), we propose the following logic expression.

$$\Pi_1 \wedge \Pi_2 \leftrightarrow \text{B}_{ij}^x = 1, \quad \forall (v_i, v_j) \in E, \forall v_x \in \text{PARA}(v_j) \tag{16}$$

According to Lem. 2 and Def. 2, the satisfaction of the first item $\Pi_1 \wedge \Pi_2$ implies that $v_i$ is the critical predecessor of $v_j$, and $v_x$ is involved in the time interval $[f_i, b_j]$ (with a non-zero length). This is the condition under which $\text{B}_{ij}^x = 1$ as defined in (14). By rewriting (16), we eventually derive the constraints as follows.

$$\neg\Pi_1 \vee \neg\Pi_2 \vee (\text{B}_{ij}^x = 1), \; \forall (v_i, v_j) \in E, \forall v_x \in \text{PARA}(v_j) \tag{17}$$
$$(\text{B}_{ij}^x = 0) \vee \Pi_1, \qquad \forall (v_i, v_j) \in E, \forall v_x \in \text{PARA}(v_j) \tag{18}$$
$$(\text{B}_{ij}^x = 0) \vee \Pi_2, \qquad \forall (v_i, v_j) \in E, \forall v_x \in \text{PARA}(v_j) \tag{19}$$

Constraints (17) to (19) together enforce the boolean variable $\text{B}_{ij}^x$ to submit to (14).

For any involved vertex set $\text{I}_{ij}$ (for the edge $(v_i, v_j) \in E$), we further define two types of subsets of $\text{I}_{ij}$ as follows:

- **Left Set $\text{I}_\text{L}$:** each vertex $v_x$ of $\text{I}_{ij}$ satisfying the following condition belongs to the left subset $\text{I}_\text{L}$.

$$\Pi_3: \qquad b_x < f_i \tag{20}$$

- **Right Set $\text{I}_\text{R}$:** each vertex $v_x$ of $\text{I}_{ij}$ satisfying the following condition belongs to the right subset $\text{I}_\text{R}$.

$$\Pi_4: \qquad f_x > b_j \tag{21}$$

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.

For example, in Fig. 1, for the time interval $[f_3, b_5]$ of the edge $(v_3, v_5)$, $v_2$ belongs to the left subset $\text{I}_\text{L}$ of $\text{I}_{35}$, and $v_4$ belongs to the right subset $\text{I}_\text{R}$ of $\text{I}_{35}$.

For any edge $(v_i, v_j) \in E$, and any vertex $v_x \in \text{PARA}(v_j)$, we use the boolean variables $\text{L}_{ij}^x$ (and $\text{R}_{ij}^x$) to index whether $v_x$ is in $\text{I}_\text{L}$ (and $\text{I}_\text{R}$) or not. More formally,

$$\text{L}_{ij}^x = \begin{cases} 1: v_x \text{ is involved in the left subset } \text{I}_\text{L} \text{ of } \text{I}_{ij} \\ 0: \quad\quad\quad\quad\quad else \end{cases} \tag{22}$$

$$\text{R}_{ij}^x = \begin{cases} 1: v_x \text{ is involved in the right subset } \text{I}_\text{R} \text{ of } \text{I}_{ij} \\ 0: \quad\quad\quad\quad\quad else \end{cases} \tag{23}$$

To implement (22) and (23), we derive the following two inequalities. For each edge $(v_i, v_j) \in E$ and for any vertex $v_x \in \text{PARA}(v_j)$,

$$\text{B}_{ij}^x = 0 \vee \neg\Pi_3 \vee \text{L}_{ij}^x = 1 \tag{24}$$
$$\text{B}_{ij}^x = 0 \vee \neg\Pi_4 \vee \text{R}_{ij}^x = 1 \tag{25}$$

Constraints (24) ensure that for any vertex $v_x$ that is involved in the interval $[f_i, b_j]$ (e.g., $\text{B}_{ij}^x = 1$), if $v_x$ is in the left subset $\text{I}_\text{L}$ ($\Pi_3 = 1$), then $\text{L}_{ij}^x$ must equal 1. Similarly, Constraints (25) ensure that if $v_x$ belongs to the right set $\text{I}_\text{R}$, then the corresponding variable $\text{R}_{ij}^x = 1$.

**Lemma 4.** *For any vertex $v_j$ and its critical predecessor $v_i$, the workload $W_{ij}$ executed during the interval $[f_i, b_j]$ can be calculated as*

$$W_{ij} = \sum_{v_x \in \text{PARA}(v_j)} (e_x \text{B}_{ij}^x - \max\{f_i - b_x, 0\}\text{L}_{ij}^x - \max\{f_x - b_j, 0\}\text{R}_{ij}^x) \tag{26}$$

*Proof.* According to Lem. 3, the vertices that are executed during the time interval $[f_i, b_j]$ belong to $\text{PARA}(v_j)$. For each vertex $v_x \in \text{PARA}(v_j)$, there are three possible cases.

- If $v_x$ belongs to $\text{I}_\text{L}$, then the workload of $v_x$ executed during $[f_i, b_j]$ equals $e_x - (f_i - b_x)$.
- If $v_x$ belongs to $\text{I}_\text{R}$, then the workload of $v_x$ executed during $[f_i, b_j]$ equals $e_x - (f_x - b_j)$.
- If $v_x$ does not belong to $\text{I}_\text{L} \cup \text{I}_\text{R}$, then the workload of $v_x$ executed during $[f_i, b_j]$ equals $e_x$.

In sum, and by (14) (22) and (23), the workload $W_{ij}$ executed during $[f_i, b_j]$ can be calculated by (26). □

By using the variables above, the constraint to formulate the work-conserving property is given as follows. For each edge $(v_i, v_j) \in E$, and each vertex $v_x \in \text{PARA}(v_j)$,

$$\neg\Pi_1 \quad \vee \sum_{v_x \in \text{PARA}(v_j)} e_x \text{B}_{ij}^x - \sum_{v_x \in \text{PARA}(v_j)} \max\{f_i - b_x, 0\}\text{L}_{ij}^x - \sum_{v_x \in \text{PARA}(v_j)} \max\{f_x - b_j, 0\}\text{R}_{ij}^x = (b_j - f_i)m \tag{27}$$

Constraints (27) ensure that for any vertex $v_j$ and its critical predecessor $v_i$, if the time interval $[f_i, b_j]$ has a non-zero length (e.g., $\Pi_1 = 1$), then the workload executed during the interval $[f_i, b_j]$ equals the computation resource that can be provided during the interval $[f_i, b_j]$. More formally, the correctness of (27) is proved in the following lemma.

**Lemma 5.** *A schedule satisfies the work-conserving property if and only if Constraints (27) hold.*

*Proof.* By (26), we can rewrite (27) as follows.

$$\Pi_1 \rightarrow W_{ij} = (b_j - f_i)m \tag{28}$$

**Necessity.** We suppose that the schedule satisfies the work-conserving property, and prove the satisfaction of (27) by showing that when $\Pi_1 = 1$, the equality $W_{ij} = (b_j - f_i)m$ holds. Since $\Pi_1 = 1$, and according to Lem. 2, $v_i$ is the critical predecessor of $v_j$, and the length of the time interval $[f_i, b_j]$ is larger than 0. According to Lem. 1, we know that under the list scheduling, the workload $W_{ij}$ executed during $[f_i, b_j]$ equals $(b_j - f_i)m$.

**Sufficiency.** It is sufficient to prove that any non-work-conserving schedule must violate (27). Since the schedule is non-work-conserving, we know that there is a vertex $v_j$ and its critical predecessor $v_i$ such that the workload $W_{ij}$ executed during the interval $[f_i, b_j]$ is less than $(b_j - f_i)m$. It indicates that the equality in (28) does not hold, and thus, the constraint (27) is violated. This completes the proof. □

**Summarizing SMT models.** In this paper, we solve the SMT model **MODEL SMT** involving the objective function (7) and the constraints (8) to (11), (17) to (19), (24), (25) and (27).

**Proposition 2.** *The number of constraints involved in* MODEL SMT *is bounded by $O(|V| \cdot |E|)$.*

**Theorem 1.** MODEL SMT *can precisely solve the WCRT of DAGs under the list scheduling upon $m$ cores.*

*Proof.* It is sufficient to prove this theorem by showing that the solution of MODEL SMT is no less than the WCRT, and meanwhile corresponds to a feasible schedule.

First, since the constraints of MODEL SMT do not formulate all scheduling constraints described in Sec. III-B, the solution space of MODEL SMT involves all feasible schedules. Therefore, the solution X of MODEL SMT derives the upper bound of the WCRT of $G$ under list scheduling upon $m$ cores.

In the following, we show that the solution X corresponds to a feasible schedule. Suppose not, i.e., the schedule of X violates some scheduling constraints described in Sec. III-B. From (8) to (11), (17) to (19), (24), (25) and (27), and according to Lem. 1, the schedule of X satisfies the scheduling constraints 2) to 3) of Sec. III-B and the work-conserving property. Therefore, the scheduling constraint 1) of Sec. III-B must be violated by the schedule of X, i.e., at some time points $t$, there are more than $m$ cores executing vertices simultaneously. We sort the vertices of $G$ in the increasing order $\delta$ of their starting time in the schedule of X. We use the list scheduling algorithm to schedule the vertices in the order $\delta$ upon $m$ cores, and eventually obtain a new schedule. Since each vertex $v_j$ in the new schedule will not be executed earlier than $v_j$ itself in the schedule of X, the new obtained schedule may have a larger response time. On the other hand, since the objective function (7) maximizes the response time of X. This leads to the contradiction. □

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.

## V. EVALUATION

This section evaluates our methods using synthetically generated task graphs. For each task instance, we compare Graham's bound $R_{\text{GRA}}$ in (1) and the exact WCRT $R_{\text{EXA}}$ solved by MODEL SMT by using the gap defined as follows.

$$\text{GAP} = \frac{R_{\text{GRA}} - R_{\text{EXA}}}{R_{\text{EXA}}} \tag{29}$$

We implement our SMT models by using Python 3.7, and solve them by Z3 solver. The code runs on a PC with Intel Core i5-6300U CPU at 2.4GHz with 8G RAM. In our experiments, we evaluate the computation time $t$ for solving MODEL SMT.

We randomly generate DAGs using TGFF tool [28], a DAG generator developed to facilitate standardized random benchmarks for scheduling research. More precisely, we construct the DAG $G$ by invoking the highly configurable graph generation algorithm supported by TGFF 3.0 and above. In the graph generation algorithm, we can configure the number $n$ of vertices. The out degree of a vertex is in the range $[\frac{\delta}{2}, \delta]$. The execution time of a vertex is in the range $[\frac{e}{2}, e]$.

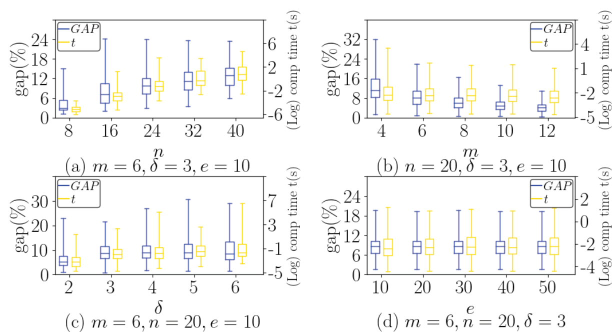



Fig. 2: Evaluation results for different configurations.

We conduct experiments with different combinations of parameters in Fig. 2. The values of configurations are written in the figure caption. For each data point, 1000 random experiments have been run[1]. We observe that MODEL SMT significantly improve the WCRT bound, i.e., the gap GAP between $R_{\text{GRA}}$ and $R_{\text{EXA}}$ achieves 30% in some cases. The compuation time is $\log_{10}$ transformed to adjust for the wide range of data. For most instances, MODEL SMT is capable of analyzing DAGs in a few seconds especially when DAG contains less than 40 vertices, e.g., the computation time $t$ of MODEL SMT is no more than 3s on average.

The computation time $t$ and the gap GAP become larger when $n$ and $\delta$ increase, and become smaller when $m$ increases. When the execution time $e$ increases, the computation time $t$ slightly increases, and the gap GAP does not change significantly. From Fig. 2(a), there is an exponential pattern in the growth of the runtime when the number of vertices increases. If MODEL SMT cannot solve the WCRT within 5 minutes, it is considered to be 'time out'. Fig. 3 shows that the time-out instance ratio increases when the vertex number increases.

[1]In a box plot, the top and the bottom of the box respectively represent the first and third quartiles of data. The middle line of the box represents the median of data. The whiskers extended from the box show the range of data.

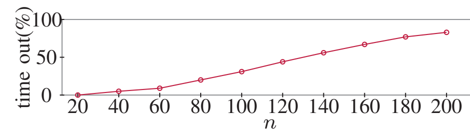


Fig. 3: Time-out instance ratio.

## VI. CONCLUSION

Response time analysis (for DAG tasks) is one of the most important problem in the real-time community. Currently, Graham's bound is widely used in response time analysis, but it is pessimistic. In this paper, we solve the exact WCRT by using SMT techniques. Experimental work shows that our method can significantly tighten the WCRT bound. In the future, we plan to propose more efficient techniques to accelerate the solving of our SMT model.

## REFERENCES

[1] R. Graham, "Bounds on multiprocessing timing anomalies," *SIAP*, 1969.
[2] J. Li et.al, "Outstanding paper award: Analysis of global edf for parallel tasks," *ECRTS*, 2013.
[3] V. Bonifaci et.al, "Feasibility analysis in the sporadic dag task model," *ECRTS*, 2013.
[4] J. Li et.al, "Analysis of federated and global scheduling for parallel real-time tasks," *ECRTS*, 2014.
[5] A. Saifullah et.al, "Parallel real-time scheduling of dags," *TPDS*, 2014.
[6] M. Qamhieh et.al, "A stretching algorithm for parallel real-time dag tasks on multiprocessor systems," *RTNS*, 2014.
[7] J. Sun et.al, "A capacity augmentation bound for real-time constrained-deadline parallel tasks under gedf," *TCAD*, 2018.
[8] S. Baruah et.al, "A generalized parallel task model for recurrent real-time processes," *RTSS*, 2012.
[9] M. Qamhieh et.al, "Global edf scheduling of directed acyclic graphs on multiprocessor systems," *RTNS*, 2013.
[10] A. Parri et.al, "Response time analysis for g-edf and g-dm scheduling of sporadic dag-tasks with arbitrary deadline," *RTNS*, 2015.
[11] M. Serrano et.al, "Response-time analysis of dag tasks under fixed priority scheduling with limited preemptions," *DATE*, 2016.
[12] M. Nasri et.al, "Response-time analysis of limited-preemptive parallel dag tasks under global scheduling," *ECRTS*, 2019.
[13] J. Fonseca et.al, "Response time analysis of sporadic dag tasks under partitioned scheduling," in *SIES*, 2016.
[14] D. Casini et.al, "Partitioned fixed-priority scheduling of parallel tasks without preemptions," in *RTSS*, 2018.
[15] S. Baruah, "The federated scheduling of constrained-deadline sporadic dag task systems," *DATE*, 2015.
[16] X. Jiang et.al, "Semi-federated scheduling of parallel real-time tasks on multiprocessors," in *RTSS*, 2017.
[17] N. Ueter et.al, "Reservation-based federated scheduling for parallel real-time tasks," in *RTSS*, 2018.
[18] A. Burmyakov et.al, "An exact schedulability test for global fp using state space pruning," in *RTNS*, 2015.
[19] W. Wang et.al, "Schedulability analysis and symbolic verification method for heterogeneous multicore real-time systems," *IJPE*, 2017.
[20] B. Yalcinkaya et.al, "An exact schedulability test for non-preemptive self-suspending real-time tasks," in *DATE*, 2019.
[21] R. Vargas et.al, "Openmp and timing predictability: a possible union?" *DATE*, 2015.
[22] M. Serrano et.al, "Timing characterization of openmp4 tasking model," *CASES*, 2015.
[23] J. Sun et.al, "Real-time scheduling and analysis of openmp task systems with tied tasks," *RTSS*, 2017.
[24] ——, "Real-time scheduling and analysis of synchronous openmp task systems with tied tasks," *DAC*, 2019.
[25] M. Serrano et.al, "Response-time analysis of dag tasks supporting heterogeneous computing," *DAC*, 2018.
[26] M. Han et.al, "Response time bounds for typed dag parallel tasks on heterogeneous multi-cores," *TPDS*, 2019.
[27] J. Sun et.al, "Calculating response-time bounds for openmp task systems with conditional branches," *RTAS*, 2019.
[28] K. Vallerio, "Task graphs for free (tgff v3. 0)," 2008.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on April 28,2023 at 17:29:37 UTC from IEEE Xplore. Restrictions apply.