Timing-anomaly free dynamic scheduling of conditional DAG tasks on multi-core systems

Chen, Peng; Liu, Weichen; Jiang, Xu; He, Qingqiang; Guan, Nan

2019

Chen, P., Liu, W., Jiang, X., He, Q., & Guan, N. (2019). Timing-anomaly free dynamic scheduling of conditional DAG tasks on multi-core systems. *ACM Transactions on Embedded Computing Systems*, 18, 1-19. doi:10.1145/3358236

<https://hdl.handle.net/10356/145468>

<https://doi.org/10.1145/3358236>

© 2019 Association for Computing Machinery (ACM). All rights reserved. This paper was published in *ACM Transactions on Embedded Computing Systems* and is made available with permission of Association for Computing Machinery (ACM).

Downloaded on 26 Apr 2023 22:11:14 SGT

Timing-Anomaly Free Dynamic Scheduling of Conditional DAG Tasks on Multi-Core Systems

PENG CHEN, The Hong Kong Polytechnic University, Nanyang Technological University
WEICHEN LIU, Nanyang Technological University, Singapore
XU JIANG, University of Electronic Science and Technology of China, China
QINGQIANG HE and NAN GUAN, The Hong Kong Polytechnic University, China

In this paper, we propose a novel approach to schedule conditional DAG parallel tasks, with which we can derive safe response time upper bounds significantly better than the state-of-the-art counterparts. The main idea is to eliminate the notorious timing anomaly in scheduling parallel tasks by enforcing certain order constraints among the vertices, and thus the response time bound can be accurately predicted off-line by somehow “simulating” the runtime scheduling. A key challenge to apply the timing-anomaly free scheduling approach to conditional DAG parallel tasks is that at runtime it may generate exponentially many instances from a conditional DAG structure. To deal with this problem, we develop effective abstractions, based on which a safe response time upper bound is computed in polynomial time. We also develop algorithms to explore the vertex orders to shorten the response time bound. The effectiveness of the proposed approach is evaluated by experiments with randomly generated DAG tasks with different parameter configurations.

CCS Concepts: • **Computer systems organization** → **Embedded hardware; Embedded software;**

Additional Key Words and Phrases: Timing anomaly, dynamic scheduling, conditional DAG, response time analysis

1 INTRODUCTION

Multi-/Many-core architecture receives more and more attention in embedded real-time systems [4, 7], since it can provide high-performance through the high degree of parallelism. Although

such a platform is considerably powerful to execute embedded applications, the software based on that is not fully supported commensurately. To cater to real-time requirements, task models, scheduling algorithms, and corresponding response time bound analysis are proposed.

To exploit the internal parallel structure of an application, several classic task models are proposed, including the fork-join model [2, 13], synchronous-parallel model [8, 20], and directed acyclic graph (DAG) model [4, 6]. Recently, due to the existence of conditional statements (i.e., if-then-else statement), a generalized conditional DAG task model [3, 5, 16, 17] of the previous three models is proposed. Conditional constructs are allowed as a part of such a conditional DAG task model. All of aforementioned models divide the DAG task into a set of jobs that can be executed in parallel on multi-core platform, where each job is denoted by a worst-case execution time (WCET). To ensure the performance of real-time applications, it is required that all of the precedence and time constraints of the conditional DAG task should be guaranteed at design time.

In scheduling the jobs of DAG task [3–5, 16, 17], there are *static scheduling* [1, 12] and *dynamic scheduling* [1, 11]. In *static scheduling*, decisions such as assigning jobs to fixed cores are made at design time. Since the runtime behavior is known beforehand, the static scheduling shows high predictability of response time; and meanwhile, it also shows low resource utilization due to workload imbalance. *Dynamic scheduling*, on the other hand, can improve the degree of resource utilization significantly, where the scheduling decisions are flexibly made at runtime. However, dynamic scheduling suffers *timing anomalies* [10, 15, 19, 21], meaning that the actual response time (where the execution time of some jobs may be less than its WCET) can be longer than the off-line estimated bound that is derived by assuming the jobs are executed for their WCETs. Hence, the off-line bound may be unsafe if such bound is estimated by simply simulating the execution of the DAG task. In addition, finding the exact estimated bound of conditional DAG task is very challenging, since one would have to analyze all exponential instances from conditional DAG structure, resulting in prohibitive computational complexity. Therefore, facing the problems, it is not easy to compute a safe and shortest response time upper bound of dynamic scheduling conditional DAG task.

To derive a safe response time bound, the authors in [3, 16, 17] proposed an analytical method for dynamic scheduling DAG task in the presence of timing anomaly. They pessimistically assume that the jobs in the critical path (namely the longest chain) are always interfered by all the jobs that are not part of the critical path. However, during the process of actual execution, these jobs in the critical path and those jobs that are not in the critical path can be executed in parallel on a multi-core platform. Hence, such a pessimistic assumption will lead to the pessimism of the off-line estimated bound. Distinct from analytical bound analysis [3, 16–18], the authors in [21] proposed a non-preemptive and non-greedy dynamic scheduler Lazy, and by simulating such scheduling algorithm at design time, a safe and tighter response time bound is computed. Despite the guarantee of timing-anomaly free execution, Lazy scheduler indeed shows laziness whether at the off-line estimation phase or on-line execution phase. That is, Lazy always idles the core when there exist ready jobs waiting to be executed. Thus, there is still room for response time reduction. Moreover, Lazy cannot directly simulate the behavior of conditional DAG task in polynomial time due to the existence of the conditional branch.

In this paper, we aim to address the drawbacks of analytical bound [3, 16, 17], improve the simulation-based bound [21], and derive a much shorter response time bound by simulating our proposed work-conserving and parallelism maximized scheduling algorithm at design time. Besides, we extend the response time bound analysis from the non-conditional DAG task model to the generalized conditional DAG task model. The main contributions of this paper are as follows:

- We formally prove that considering any given dynamic scheduler, as long as these schedulers are subject to our proposed scheduling order constraint at the on-line execution phase,

timing-anomaly free execution can be guaranteed. Thus, a safe response time bound can be computed at the off-line estimation phase.

- We develop, for any given scheduling order, effective abstractions to compute the response time bound by somehow “simulating” the execution of conditional DAG task. Moreover, we also propose an improved and generalized upper bound including an adjustable parameter for precision that can be scalable for various size and precision requirements of conditional DAG task.
- We provide a heuristic scheduling algorithm of deriving a work-conserving and parallelism maximized scheduling order at the off-line estimation phase. With such scheduling order constraint used at the on-line execution phase, a safe and much shorter response time upper bound is achieved compared with the state-of-the-art counterparts.

In the rest of this paper: Section 2 introduces the system model and discusses the runtime behavior of conditional DAG task; Section 3 illustrates our motivation of this paper; Timing-anomaly free scheduling under our proposed scheduling order constraint is formally proved, and the corresponding response time upper bound for conditional DAG task is computed in Section 4; A work-conserving and parallelism maximized scheduling order is derived using the proposed heuristic scheduling algorithm in Section 5. Experimental results are given in Section 6, and Section 7 concludes this paper.

2 SYSTEM MODEL AND DEFINITIONS

2.1 System Model

This paper considers the directed acyclic graph (DAG) model [5, 6] to represent the tasks of a parallel application. Formally, a DAG task τ is represented by DAG $G = (V, E)$, where $V = \{v_1, v_2, \dots, v_N\}$ is a set of vertexes denoted by rectangles, and $E \subseteq V \times V$ is a set of directed edges, shown in Figure 1. Each vertex v_i of the DAG represents a segment of a computer program (called *job*), and it has a worst-case execution time (WCET) denoted by C_i . As for each edge $e_{i,j} = (v_i, v_j) \in E$, $e_{i,j}$ represents the precedence constraint between jobs v_i and v_j . That is, job v_j only starts execution after the completion of v_i . v_i is a predecessor job of v_j , and v_j is a successor job of v_i . The job that has no incoming edges and the job that has no outgoing edges are called *source* and *sink* job, respectively. Since multiple source/sink jobs can be connected by one source/sink job using dummy job with execution time zero, any DAG can be trivially transformed to a DAG exactly with one source (denoted by v_{src}) and one sink (denoted by v_{sink}), like v_1 and v_9 in Figure 1. The properties of the deadline and period are not discussed, since we take no account of the inter-task interference.

Conditional DAG task model [3, 16, 17] is a generalized model from fork-join, synchronous-parallel, and DAG-based models. In a conditional DAG task model, conditional vertexes (i.e., if-then-else, switch statements) appear in pairs. As shown in Figure 1, (v_2, v_8) is such a *conditional pair*. v_2 is the *conditional starting job* denoted by diamond, while v_8 is the *conditional joining job* denoted by ellipse. Without loss of generality, for a conditional pair $\mathcal{P}_i$ shown in Figure 2, assume there are k ($k > 1$) outgoing edges from the *conditional starting job* α_j , and there are also exactly k incoming edges into *conditional joining job* β_i . Also assume that each branch has no nested conditional pair, since a nested conditional pair can be trivially transformed into a non-nested pair in polynomial time. The conditional pair $\mathcal{P}_i$ has k conditional branches excluding conditional starting and joining jobs, where each of these branches is a non-conditional DAG denoted by $G'_{i,1}, G'_{i,2}, \dots, G'_{i,k}$. For any two conditional branches $G'_{i,\mu}$ and $G'_{i,\nu}$ ($\mu, \nu \in \{1, 2, \dots, k\}$) belonging to conditional pair $\mathcal{P}_i$, there are no edges between $G'_{i,\mu}$ and $G'_{i,\nu}$. During the process of

Fig. 1. An example of conditional DAG task, where the number within each vertex denotes WCET of its corresponding job.

Fig. 2. An example of conditional pair $\mathcal{P}_i$ consisting of k conditional branches.

on-line execution, one of these branches will become eligible to be executed, depending upon the outcome of the conditional starting job.

As for the runtime platform of the conditional DAG task τ , we consider a multi-core platform consisting of M identical cores. Or, DAG task τ is allocated a cluster of M dedicated cores by using *federated scheduling* [3, 14]. The minimum number of dedicated cores are chosen from the multi-core platform until the deadline can be guaranteed.

2.2 Runtime Behavior

When conditional DAG task τ is executed on a multi-core platform, only one branch of each conditional pair is eligible to be executed depending upon the outcome of the conditional starting job. The collection of jobs generated by τ at runtime constitutes a particular execution instance. Let $\mathcal{I}_\xi$ be such an on-line execution instance. It is not easy to see that $\mathcal{I}_\xi$ is a non-conditional DAG instance. Hence, according to the system model, the DAG task τ has k^n execution instances.

For each job v_i of τ , let s_i and f_i be the time points when v_i starts and finishes execution, respectively. Any job can be dispatched to be executed only if there exist idle cores, namely *non-preemptive scheduling*. The *response time* of τ is the finish time of the *sink* job, denoted by $R = f_{\text{sink}}$, starting from 0 of the *source* job. In this paper, we aim to derive a safe upper bound R with the relationship $R^{\text{off}}(\mathcal{I}^*) \leq R$. $R^{\text{off}}(\mathcal{I}^*) = \max(R^{\text{off}}(\mathcal{I}_1), R^{\text{off}}(\mathcal{I}_2), \dots, R^{\text{off}}(\mathcal{I}_{k^n}))$, where $\mathcal{I}^*$ is the instance with maximum response time upper bound. We also define $R^{\text{on}}(\mathcal{I}_\xi)$ as the actual response time of execution instance $\mathcal{I}_\xi$ at runtime. Similar to [21], the response time bound $R^{\text{off}}(\mathcal{I}_\xi)$ can be computed by simulating the scheduling algorithm at the design time.

The notations used throughout this paper are summarized in Table 1. To illustrate the aforementioned definitions, we continue to discuss the example of Figure 1. As shown in Figure 3, there are two execution instances $\mathcal{I}_1$ and $\mathcal{I}_2$ of original conditional DAG task. Each instance is executed on a multi-core platform consisting of two allocated dedicated cores P_0 and P_1 , using the non-preemptive breadth-first scheduling. Assume all the jobs are executed for their designed WCETs, then the response time upper bound $R^{\text{off}}(\mathcal{I}_1)$ for instance $\mathcal{I}_1$ can be derived by simulating breadth-first scheduling at the off-line phase. Specifically, in Figure 3(c), $s_2 = 1$, $f_2 = 2$, $s_9 = 9$, $f_{\text{sink}} = f_9 = 10$, and the upper bound $R^{\text{off}}(\mathcal{I}_1)$ of instance $\mathcal{I}_1$ is 10 time units. Similarly,

Table 1. Notations Used Throughout the Paper

Notation	Definition
τ	A conditional DAG task
v_i	i^{th} job of τ
$\mathcal{N}$	Number of jobs of τ
s_i	Actual execution start time of v_i
s'_i	Virtual execution start time of v_i
f_i	Execution finish time of v_i
$\mathcal{P}_i$	i^{th} conditional pair of τ
α_i	Conditional starting job of $\mathcal{P}_i$
β_i	Conditional joining job of $\mathcal{P}_i$
n	Number of conditional pairs of τ
k	Number of branches of each pair
$\mathcal{I}_\xi$	ξ^{th} execution instance of τ
$R^{on}(\mathcal{I}_\xi)$	On-line response time of $\mathcal{I}_\xi$
$R^{off}(\mathcal{I}_\xi)$	Off-line estimated upper bound of $\mathcal{I}_\xi$
$\mathcal{I}^*$	The instance with maximum bound
M	Number of dedicated cores

 Fig. 3. Execution instances of conditional DAG task in Figure 1 and off-line simulation of instance $\mathcal{I}_1$ on two cores.

$R^{off}(\mathcal{I}_2) = 10$. Hence, for the conditional DAG task τ , the response time upper bound $R = R^{off}(\mathcal{I}^*) = \max(R^{off}(\mathcal{I}_1), R^{off}(\mathcal{I}_2)) = 10$. When executed in actual hardware, the actual execution time of each job is less than or equals to its WCET. If using the same scheduling algorithm at the off-line estimation and on-line execution phases, the off-line simulation-based upper bound $R^{off}(\mathcal{I}_\xi)$ intuitively seems safe.

3 MOTIVATION

For conditional DAG task τ , we can compute the response time bound off-line of each instance by simulating its scheduling algorithm (See Figure 3(c)), provided that the conditional DAG and the number of dedicated cores are given. During the process of on-line execution, the actual execution time of each job may be less than its designed WCET. Intuitively it seems that the actual response time of instance $\mathcal{I}_\xi$ is shorter compared with off-line estimated bound, formally $R^{on}(\mathcal{I}_\xi) \leq R^{off}(\mathcal{I}_\xi)$. However, if τ employs dynamic scheduler with non-preemptive behavior, counter intuitively, we find that the actual response time of $\mathcal{I}_\xi$ may exceed its off-line estimated bound, namely $R^{on}(\mathcal{I}_\xi) > R^{off}(\mathcal{I}_\xi)$. Such behavior is called *scheduling timing anomaly* and identified in [10, 15, 19]. Thus the off-line estimated bound may be not safe for dynamic scheduling conditional DAG task. Dynamic scheduling policies are commonly used (e.g., breadth-first and depth-first scheduling). However, such schedulers are usually locally greedy and unable to prevent timing anomalies.

Fig. 4. Timing-anomaly execution of instance $\mathcal{I}_\xi$ under non-preemptive breadth-first scheduling.

We take an example in Figure 4 to illustrate this scheduling timing anomaly [19] phenomenon. The DAG task of the example is assumed to be executed upon two processors P_0 and P_1 based on non-preemptive breadth-first scheduling algorithm. Assume the actual execution time of job v_2 is 1, instead of the designed WCET 3, and all other jobs are executed for their WCETs at runtime, shown in Figure 4.

If without a certain order constraint for the jobs or any other feasible way, the scheduler may choose the job v_5 instead of v_6 at scheduling point 3. The actual response time $R^{on}(\mathcal{I}_\xi)$ (See Figure 4(b)) of the instance $\mathcal{I}_\xi$ will be 9 time units when the actual execution time of v_2 is 1 time unit. Compared with the off-line estimated bound $R^{off}(\mathcal{I}_\xi) = 8$ time units (See Figure 4(a)), the actual response time is counter-intuitively increased from 8 to 9 time units, formally $R^{on}(\mathcal{I}_\xi) > R^{off}(\mathcal{I}_\xi)$. This is because, the starting execution time of the heavy job v_6 with long worst case execution time is latter than that of off-line estimation phase since the available processors are occupied in advance, thus leading to actual response time increase of DAG task τ . Therefore, due to the existence of scheduling timing anomaly [19], if not addressed, the off-line estimated bound may not be safe.

Another challenge with conditional DAG task τ is that it is not easy to compute the exact response time bound $R^{off}(\mathcal{I}^*)$ off-line by using the simulation-based technique. This is due to the existence of conditional branches, to find the exact bound $R^{off}(\mathcal{I}^*)$, we have to exhaustively enumerate all the execution instances of τ . Specifically, since τ has n conditional pairs, and at most k branches of each pair, it has at most k^n execution instances that should be considered separately, $R^{off}(\mathcal{I}^*) = \max(R^{off}(\mathcal{I}_1), R^{off}(\mathcal{I}_2), \dots, R^{off}(\mathcal{I}_{k^n}))$. However, considering each of these instances to derive $R^{off}(\mathcal{I}^*)$ is computationally infeasible, since the number of instances k^n can be exponential in the number of conditional pairs n .

4 RESPONSE TIME ANALYSIS

In this section, we first present a certain order constraint among jobs for timing-anomaly free execution to ensure the off-line estimated upper bound is safe. Second, under the order constraint, we then compute the safe response time upper bound of conditional DAG task τ within polynomial time with an effective abstraction.

Fig. 5. Timing-anomaly free execution using the same execution-order as that of off-line estimation phase under non-preemptive breadth-first scheduling.

4.1 Timing-anomaly Free Guarantee

Before proposing the order constraint among jobs, we need to know the *execution order* at both off-line estimation and on-line execution phases.

Definition 1 (Execution Order). For non-conditional DAG instance I_ξ with N jobs, the execution order O_E of I_ξ is the job sequence with the starting time points, where jobs start execution, in ascending order.

$$O_E = (v_1 \geq v_2 \geq v_3 \geq \dots \geq v_N),$$

where $s_1 \leq s_2 \leq s_3 \leq \dots \leq s_N$.

Example 1. For the instance I_ξ of Figure 4, we can derive the execution order O_E^{off} at the off-line estimation phase and O_E^{on} at the on-line execution phase, respectively.

$$\begin{aligned} O_E^{off} &= (v_1 \geq v_2 \geq v_3 \geq v_6 \geq v_4 \geq v_5 \geq v_7); \\ O_E^{on} &= (v_1 \geq v_2 \geq v_3 \geq v_4 \geq v_5 \geq v_6 \geq v_7); \end{aligned}$$

When job v_2 takes less execution time during on-line execution than that of off-line estimation phase, the actual response time $R^{on}(I_\xi)$ of instance I_ξ is increased up to 9 time units, thus leading to unsafety of estimated bound. We can observe that the jobs have different execution order at the off-line estimation and on-line execution phases. Specifically, the sub-execution order $(v_6 \geq v_4 \geq v_5)$ ($\subset O_E^{off}$) is changed to $(v_4 \geq v_5 \geq v_6)$ ($\subset O_E^{on}$) of Figure 4. This is because, due to non-preemption and local greed of the dynamic scheduler (e.g., breadth-first scheduling), the predecessor job v_2 of v_4 finishes completion earlier than that of off-line estimated phase and then the successor job v_4 starts execution immediately on the idle core P_1 before v_6 , breaking the original order $(v_6 \geq v_4 \geq v_5)$. The job with longer WCET (i.e., v_6) is executed later and the degree of parallelism is reduced (i.e., the difference between the minimum and maximum values of the finish time of the cores is increased from 1 to 5 time units).

Inspired by this observation, we assume the jobs are executed on-line as the same execution order of off-line estimation phase, shown in Figure 5. When v_2 finishes completion at time point 2 to keep the same execution order as the execution order O_E^{off} of off-line estimation phase, v_4 is delayed by 1 time unit until v_6 starts execution at time point 3. All the other jobs remain unchanged. By doing so, the off-line estimated upper bound (See Figure 4(a)) covers the actual response time (See Figure 5) of on-line execution phase, formally $R^{on}(I_\xi) \leq R^{off}(I_\xi)$. In view of this, we can conclude that the timing-anomaly behavior can be eliminated if the off-line estimation and on-line execution phases are conducted by the same execution order. That is, $O_E^{off} = O_E^{on}$.

In the following Theorem 1, we will formally prove that if the execution order generated by any dynamic scheduler is consistent at both off-line estimation and on-line execution phases, the timing-anomaly execution behavior can be eliminated and thus the off-line estimated bound would be safe.

THEOREM 1. For the non-conditional DAG instance I_ξ , given any dynamic and non-preemptive scheduler, if

$$O_E^{on} = O_E^{off}$$

then

$$R^{on}(I_\xi) \leq R^{off}(I_\xi)$$

PROOF. Let s_i^{on} and s_i^{off} be the start time points, f_i^{on} and f_i^{off} be the finish time points, and C'_i and C_i ($C'_i \leq C_i$) be the execution time of job v_i at on-line execution phase and off-line estimation phase, respectively. Since the scheduler is non-preemptive to schedule jobs, the following equations for any job v_i ($i = 1, 2, \dots, N$) are derived:

$$f_i^{on} = s_i^{on} + C'_i \leq s_i^{on} + C_i$$

$$f_i^{off} = s_i^{off} + C_i$$

Since

$$R^{on}(I_\xi) = \max_{i \in \{1, 2, \dots, sink\}} f_i^{on} = f_{sink}^{on}$$

$$R^{off}(I_\xi) = \max_{i \in \{1, 2, \dots, sink\}} f_i^{off} = f_{sink}^{off}$$

Thus, we only need to prove:

$$f_{sink}^{on} \leq f_{sink}^{off}$$

Since $f_i^{on} \leq s_i^{on} + C_i$ and $f_i^{off} = s_i^{off} + C_i$, if we derive $s_{sink}^{on} \leq s_{sink}^{off}$, the inequation $f_{sink}^{on} \leq f_{sink}^{off}$ is derived. Assume $O_E = O_E^{off} = O_E^{on} = (v_1 \geq v_2 \geq \dots \geq v_i \geq \dots \geq v_{sink})$, we have $s_\omega^{on} \leq s_\mu^{on}$ and $s_\omega^{off} \leq s_\mu^{off}$ for $\forall (v_\omega \geq v_\mu) \subset O_E$. In the following, we use mathematical induction to prove $s_i^{on} \leq s_i^{off}$ for $\forall i \in \{1, 2, \dots, sink\}$.

Base Case: When $i = 1$, since the source job v_1 starts execution at time 0, $s_1^{on} = s_1^{off} = 0$. Therefore, $s_i^{on} \leq s_i^{off}$ for $i = 1$.

Inductive Step: Assume that $s_h^{on} \leq s_h^{off}$ for $\forall h \in \{1, 2, \dots, i-1\}$. We need to prove $s_i^{on} \leq s_i^{off}$.

Since off-line estimation and on-line execution conduct the same execution order, $s_{i-1}^{on} \leq s_{i-1}^{on}$ and $s_{i-1}^{off} \leq s_{i-1}^{off}$. At scheduling time point s_i^{off} at the off-line estimation phase, assume v_j just finishes execution and it triggers the execution start of v_i . Let $\mathcal{S}_f$ represent the set of jobs including v_j that have finished execution at s_i^{off} of off-line estimation phase. Apart from the running job v_i , there are at most $M-1$ running jobs over the multi-core platform consisting of M identical cores. For $\forall v_\mu \in \mathcal{S}_f$, $s_\mu^{on} \leq s_\mu^{off}$, then $f_\mu^{on} = s_\mu^{on} + C'_\mu \leq s_\mu^{off} + C_\mu = f_\mu^{off}$. That is, all the jobs of $\mathcal{S}_f$ including the predecessor jobs of v_i have also finished execution before s_i^{off} at the on-line execution phase. There exist at least one idle core at time s_i^{off} at on-line execution phase. Therefore, considering the greed of dynamic scheduler, job v_i actually starts execution before and not latter than s_i^{off} . That is, $s_i^{on} \leq s_i^{off}$. When $i = sink$, we have $s_{sink}^{on} \leq s_{sink}^{off}$. The theorem is proved. $\square$

Following the Theorem 1, by using in-order execution, formally $O_E^{on} = O_E^{off}$, timing-anomaly free execution is indeed guaranteed and thus the off-line estimated bound is safe, e.g., Figure 5. Take a panoramic view of the scheduling with in-order execution constraint for the multi-core platform, however, such scheduling scheme is limited to flexibility and exhibits low utilization of processors. Let us revisit the illustrative example of Figure 4(a). When the number of available cores increases from 2 to 3, the timeline of the instance I_ξ is shown in Figure 6. As we can see,

Fig. 6. In-order execution when allocated 3 cores.

cores P_0 and P_1 remain idle in the middle in order to keep in-order execution. Specifically, v_4 and v_5 will start execution after the start time of v_6 in O_E , thus P_0 and P_1 have to keep idle at the time interval $[2, 3]$ until v_6 starts execution. In the extreme case, when the target DAG task τ is allocated with enough dedicated cores, there are a large number of idle cores waiting to be used (e.g., up to $M-1$ cores upon the multi-core platform consisting of M identical cores) in the middle time interval, thus leading to low utilization of processors. Besides, the on-line actual execution order is inflexible, which is only determined off-line by the given scheduling algorithm.

To improve processor utilization and increase scheduling flexibility, we find the in-order execution constraint can be relaxed slightly without timing anomaly. Without loss of generality, given a DAG task τ and its corresponding execution order O_E , assume $(v_i \geq v_j \geq v_k) \subseteq O_E$. v_i is a heavy job with long worst case execution time. τ is executed on a multi-core platform. With such in-order execution, the latter jobs v_j and v_k cannot start execution until all the previous jobs (e.g., v_i) have started execution. Assume at scheduling point t at runtime, job v_i has not been created, while jobs v_j and v_k have been created and put into the ready queue exactly. This is because, the actual execution time of the predecessor jobs of v_j and v_k is shorter than their WCETs. v_j and v_k thus can be created earlier than that of off-line estimation phase. When using in-order execution at scheduling point t , v_j and v_k have to wait until v_i starts execution, even if enough processors are available. Actually, to eliminate timing anomaly, we only need to conduct processor reservation for job v_i at scheduling point t . It seems that v_i is *virtually* executed, since it has not been *actually* created. While v_j and v_k can be dispatched for actual execution upon other available processors. Because the heavy job v_i can be executed immediately upon the reserved core once v_i is ready for execution, there would be no waiting for the release of available processors. Thus the starting execution time of v_i cannot be latter than that of off-line estimation phase.

By using such approach, its benefits are twofold. First, the on-line actual response time (when v_i is the *sink* job) of DAG task τ will still be within the off-line estimated upper bound; that is, timing anomaly will be eliminated. Second, processor utilization will be improved due to more flexible scheduling. Based on these observations, we will now formally introduce the following definition.

Definition 2 (Virtual Execution Order). For non-conditional DAG instance I_ξ with N jobs, the virtual execution order O_{VE} of I_ξ is the job sequence with the virtual starting time points, where jobs virtually start execution, in ascending order.

$$O_{VE} = (v_1 > v_2 > v_3 > \dots > v_N),$$

where $s'_1 \leq s'_2 \leq s'_3 \leq \dots \leq s'_N$; s'_i is the virtual start time where the job v_i is conducted processor reservation, rather than actual start time s_i .

Note that there is a slight difference between virtual start time s'_i and actual start time s_i for job v_i . To eliminate timing anomaly, if any job v_k ($v_k > v_i$) ($\subseteq O_{VE}$) has started actual execution or been conducted processor reservation, v_i would start virtual execution and be also correspondingly conducted processor reservation at scheduling point. Once all the predecessor jobs of v_i finish

Fig. 7. An example of virtual execution order.

execution, v_i is created, and hence starts actual execution on its reserved core. For example, in Figure 7, $s'_6 = 2$, $s_6 = 3$ of job v_6 .

Example 2. We can derive the virtual execution order O_{VE}^{off} of the instance of Figure 4(a) if executed on 3-core platform at the off-line estimation phase. When such virtual execution order is employed at on-line execution phase, shown in Figure 7, the actual execution order O_E^{on} can be also derived.

$$\begin{aligned} O_{VE}^{off} &= O_{VE}^{on} = (v_1 > v_2 > v_3 > v_6 > v_4 > v_5 > v_7); \\ O_E^{on} &= (v_1 \geq v_2 \geq v_3 \geq v_4 \geq v_5 \geq v_6 \geq v_7). \end{aligned}$$

As we can see in Figure 7, job v_6 virtually starts execution at scheduling point 2 (in fact, only conducting processor reservation for v_6), and it starts actual execution at time 3 until its predecessor jobs (e.g., v_3) all finish execution. The observations are threefold. First, distinct from the original actual execution order ($v_1 \geq v_2 \geq v_3 \geq v_6 \geq v_4 \geq v_5 \geq v_7$), even v_4 starts actual execution before the start time of v_6 , the timing-anomaly free execution is still guaranteed (proved in Theorem 2), which also does for 2-core platform. Second, the total necessary idle time, which is used to avoid timing-anomaly execution and cannot be utilized in the middle, is reduced from 2 to 1 time unit in this example (See Figure 6 and Figure 7), thus improving processor utilization. Third, the off-line virtual execution order is the same as the off-line normal execution order, formally $O_{VE}^{off} = O_E^{off}$, which is also determined by the given scheduling algorithm. This is because at the off-line estimation phase, as the jobs are assumed to be executed for their WCETs, there is a time coincidence between actual execution and processor reservation. Then the on-line virtual execution order follows the corresponding off-line one, formally $O_{VE}^{on} = O_{VE}^{off}$. Such the on-line virtual execution order O_{VE}^{on} is fixed, which is determined by the given scheduling algorithm. While the on-line actual execution order is relatively flexible, instead of fixed, depending on the actual execution time of the jobs and being subject to the virtual execution order constraint.

THEOREM 2. *For the non-conditional DAG instance I_ξ , given any dynamic and non-preemptive scheduler, if*

$$O_{VE}^{on} = O_{VE}^{off}$$

then

$$R^{on}(I_\xi) \leq R^{off}(I_\xi)$$

PROOF. When employing in-order virtual execution at the on-line execution phase, the scheduler will reserve a core before and not latter than that of off-line estimation phase at each scheduling point. Similar to Theorem 1, each job at the on-line execution phase thus will start actual execution before and not latter than the starting execution time of off-line estimation phase. The theorem is proved. $\square$

Based on in-order virtual execution at off-line estimation and on-line execution phases, the timing-anomaly behavior is eliminated and the total necessary idle time in the middle is reduced, compared with in-order actual execution. Contrary to the intuitive approach with dynamic scheduling, not the natural execution order policy that greatly flexibly chooses the ready job (e.g., Figure 4(b)) but the virtual execution order policy that slightly restricts the actual execution order (e.g., Figure 5) guarantees the timing-anomaly free execution. In fact, the timing anomaly can be eliminated when using any dynamic scheduler, as long as these schedulers are subject to our proposed virtual execution order constraint.

4.2 Computing Response Time Bound of Conditional DAG Task

In Section 4.1, we proved that, for any dynamic scheduler, as long as the in-order virtual execution constraint is followed at the off-line estimation and on-line execution phases, the timing-anomaly execution behavior is eliminated, and then the off-line estimated bound will be safe. In this section, for a given scheduling order O_{VE} of the conditional DAG task τ , we will compute the response time upper bound by simulating its execution off-line, assuming each job is executed for its WCET.

Without loss of generality, assume the non-nested conditional DAG task τ has n conditional pairs, where each pair has at most k conditional branches. As pointed out in Section 3, there are at most k^n different execution instances at runtime, which is exponential in the number of conditional pairs. Thus, the exact optimal response time bound of τ is $R^{off}(\mathcal{I}^*) = \max(R^{off}(\mathcal{I}_1), R^{off}(\mathcal{I}_2), \dots, R^{off}(\mathcal{I}_{k^n}))$. To find $R^{off}(\mathcal{I}^*)$, we need to exhaustively enumerate all these k^n execution instances, which is computationally infeasible. We instead aim to obtain a relatively bigger value of the upper bound within polynomial time.

Before computing the polynomial-time bound, we first define some useful notations and relationships. For an execution instance $\mathcal{I}_\xi$ of conditional DAG task τ , let a tuple $\mathcal{T}_{\xi, v_i} = \langle t_{\xi,1}, t_{\xi,2}, t_{\xi,3}, \dots, t_{\xi,M} \rangle$, where $t_{\xi,1} \leq t_{\xi,2} \leq t_{\xi,3} \leq \dots \leq t_{\xi,M}$, denote a set of finish times of the M cores when job v_i and any job v_k that has the relationship $(v_k > v_i) (\subset O_{VE})$ have finished execution, and the remaining jobs have not started execution. Such as in Figure 7, $\mathcal{T}_{\xi, v_6} = \langle t_{\xi,1}, t_{\xi,2}, t_{\xi,3} \rangle = \langle 4, 4, 7 \rangle$. Given two tuples $\mathcal{T}_{1, v_i}$ and $\mathcal{T}_{2, v_i}$, we say $\mathcal{T}_{2, v_i}$ covers $\mathcal{T}_{1, v_i}$ when v_i finishes execution, if

$$t_{1,1} \leq t_{2,1}, t_{1,2} \leq t_{2,2}, t_{1,3} \leq t_{2,3}, \dots, t_{1,M} \leq t_{2,M}$$

such the *covers* relationship is denoted by

$$\mathcal{T}_{1, v_i} \preceq \mathcal{T}_{2, v_i}$$

To store the virtual execution order O_{VE} of conditional DAG task, it can be achieved by some data structures (e.g., linked list). In this paper, the conditional pair is replaced with a regular and non-conditional job, hence the conditional DAG task can be transformed into an *equivalent* non-conditional DAG task. The job order of such equivalent non-conditional DAG task is denoted by O_{main} . While for each conditional pair, it consists of k sub-orders; such as, the sub-order of j^{th} branch of pair $\mathcal{P}_i$ is denoted by $O_{i,j}$. Therefore, O_{VE} includes a main order O_{main} and at most $n \times k$ sub-orders.

To effectively find the response time bound off-line, we first calculate a value denoted by R_{apr} within polynomial time. Next, we will formally prove the exact response time bound $R^{off}(\mathcal{I}^*)$ is upper-bounded by such derived value.

Algorithm 1 presents the detailed description of the algorithm for deriving a value R_{apr} . The value of the elements in $\mathcal{T}_{max, v_0}$ is initially set as 0, where v_0 is dummy job with execution time 0. When encountering each conditional job, we respectively “execute” its k conditional branches off-line until any job v_ω , which has the relationship $v_\omega > \beta_i$ (namely *conditional joining job*) in the virtual execution order O_{VE} , finishes completion. Then we obtain the finish time of each core

Fig. 8. An example of computation of R_{apr} .

ALGORITHM 1: Computing a value R_{apr}

Input: Non-nested conditional DAG $G = (V, E)$; virtual execution order O_{VE} ; M cores;

Output: A value R_{apr} .

```

1  $\mathcal{T}_{max, v_0} = \langle 0, 0, 0, \dots, 0 \rangle$ ;
2 for ( $v_i \in O_{main}$  from  $v_{src}$  to  $v_{sink}$ ) do
3   if ( $v_i$  is a conditional job) then
4     for ( $j = 1 \rightarrow k$ ) do
5        $\mathcal{T}_{j, \beta_i} = \langle t_{j,1}, t_{j,2}, t_{j,3}, \dots, t_{j,M} \rangle$ ;
6        $AscendingSort(\mathcal{T}_{j, \beta_i})$ ;
7     for ( $m = 1 \rightarrow M$ ) do
8        $t_{max, m} = \max_{j \in \{1, 2, \dots, k\}} t_{j, m}$ ;
9      $\mathcal{T}_{max, \beta_i} = \langle t_{max,1}, t_{max,2}, t_{max,3}, \dots, t_{max,M} \rangle$ ;
10  else
11     $\mathcal{T}_{max, v_i} = \langle t_{max,1}, t_{max,2}, t_{max,3}, \dots, t_{max,M} \rangle$ ;
12  $R_{apr} = \max_{m \in \{1, 2, \dots, M\}} t_{max, m}$ ;
13 Return  $R_{apr}$ ;

```

denoted by a tuple $\mathcal{T}_{j, \beta_i} = \langle t_{j,1}, t_{j,2}, t_{j,3}, \dots, t_{j,M} \rangle$ in Line 5. To facilitate comparison, after that, these time values are sorted by ascending order, that is $t_{j,1} \leq t_{j,2} \leq t_{j,3} \leq \dots \leq t_{j,M}$. For these k time tuples, the proposed algorithm finds the maximum finish time of each core in Line 9, namely $\mathcal{T}_{j, \beta_i} \preceq \mathcal{T}_{max, \beta_i}$ for any instance $\mathcal{I}_j$. After the completion of all the n conditional jobs and the *sink* job, the final finish time $\mathcal{T}_{max, v_{sink}}$ of each core is derived. Let R_{apr} be the maximum value among the M time values in $\mathcal{T}_{max, v_{sink}}$, namely $R_{apr} = t_{max, M}$ if $\mathcal{T}_{max, v_{sink}}$ is sorted in ascending order.

More specifically, as the example shown in Figure 8, assume the conditional DAG task τ has 1 conditional pair and the pair has 2 conditional branches, namely two execution instances (denoted by $\mathcal{I}_1$ and $\mathcal{I}_2$) for τ . We first respectively “execute” these 2 instances, and derive their sorted bounds

of the cores starting from 0. Therefore, for instance I_1 , $\mathcal{T}_{1,v_{\text{sink}}} = (3, 4, 5)$; for instance I_2 , $\mathcal{T}_{2,v_{\text{sink}}} = (2, 3, 7)$. Then, according to Algorithm 1, $\mathcal{T}_{\max,v_{\text{sink}}} = (3, 4, 7)$ that has the relationships $\mathcal{T}_{1,v_{\text{sink}}} \preceq \mathcal{T}_{\max,v_{\text{sink}}}$ and $\mathcal{T}_{2,v_{\text{sink}}} \preceq \mathcal{T}_{\max,v_{\text{sink}}}$. Hence, $R_{\text{apr}} = \max(3, 4, 7) = 7$.

The computation complexity of Algorithm 1 is $O(n \cdot k \cdot M \log M)$ ($M \log M$ is the time complexity of sorting algorithm), where n is the number of conditional pairs of the conditional DAG, k is the maximum number of conditional branches of each pair, and M is the number of allocated dedicated cores. We can observe that the proposed algorithm to derive R_{apr} is a polynomial-time algorithm.

THEOREM 3. *For the conditional DAG task τ , the exact optimal but exponential-time response time bound $R^{\text{off}}(\mathcal{I}^*)$ is upper-bounded by*

$$R^{\text{off}}(\mathcal{I}^*) \leq R_{\text{apr}}$$

where $R^{\text{off}}(\mathcal{I}^*) = \max(R^{\text{off}}(\mathcal{I}_1), R^{\text{off}}(\mathcal{I}_2), \dots, R^{\text{off}}(\mathcal{I}_{k^n}))$; R_{apr} is computed by Algorithm 1.

PROOF. To prove $R^{\text{off}}(\mathcal{I}^*) \leq R_{\text{apr}}$, we need to prove $R^{\text{off}}(\mathcal{I}_\xi) \leq R_{\text{apr}}$ for any execution instance $\mathcal{I}_\xi$. Let two tuples $\mathcal{T}_{\xi,\beta_i}$ and $\mathcal{T}_{\max,\beta_i}$ be the finish times of instance $\mathcal{I}_\xi$ and that obtained by Algorithm 1 lines 3-9 when the conditional joining job β_i of conditional pair $\mathcal{P}_i$ finishes execution, respectively. According to Algorithm 1, for any conditional joining job β_i , $\mathcal{T}_{\xi,\beta_i} \preceq \mathcal{T}_{\max,\beta_i}$. After the completion of the conditional joining job β_n and sink job, we can get

$$\mathcal{T}_{\xi,v_{\text{sink}}} \preceq \mathcal{T}_{\max,v_{\text{sink}}}$$

where the time values of $\mathcal{T}_{\xi,v_{\text{sink}}}$ and $\mathcal{T}_{\max,v_{\text{sink}}}$ are respectively sorted in ascending order, meaning that $t_{\xi,M} \leq t_{\max,M}$.

Since

$$R^{\text{off}}(\mathcal{I}_\xi) = \max_{m \in \{1, 2, \dots, M\}} t_{\xi,m} = t_{\xi,M}$$

and

$$R_{\text{apr}} = \max_{m \in \{1, 2, \dots, M\}} t_{\max,m} = t_{\max,M}$$

$R^{\text{off}}(\mathcal{I}_\xi) \leq R_{\text{apr}}$, thus the theorem is proved. $\square$

Compared with the exact optimal response time bound $R^{\text{off}}(\mathcal{I}^*)$ with exponential-time computational complexity, even in the absence of tightness, such approximate bound R_{apr} can be still safe and computed within polynomial time.

4.3 Improved Response Time Bound

Our obtained response time bound R_{apr} in Section 4.2 is safe and computed within polynomial time, but still might be pessimistic. As an example in Figure 8, the covered graph indeed covers the time graphs that are obtained respectively by simulating the two instances. For instance 1 and 2, their total utilized time of the cores is less than that of covered graph with 2 time units, respectively. Whether instance $\mathcal{I}_1$ or $\mathcal{I}_2$ is executed at on-line execution phase, the response time bound R_{apr} would be pessimistic. Since the covered graph must cover the time graphs of all the instances with different shapes (formally $\mathcal{T}_{\xi,\beta_i} \preceq \mathcal{T}_{\max,\beta_i}$ when encountering any conditional pair $\mathcal{P}_i$ of any instance $\mathcal{I}_\xi$), with the increasing of the number of conditional pairs, the obtained response time bound R_{apr} would be more pessimistic. This is mainly due to the fact that Algorithm 1 overestimates the bound of cores when encountering each conditional pair. However, the considered worst-case scenario will never occur.

Without loss of generality, assume a conditional DAG with n conditional pairs, where each pair has at most k conditional branches. To reduce the pessimism of the response time bound R_{apr} , we first introduce a parameter δ ($0 \leq \delta \leq n$) that denotes the number of conditional pairs. For a subgraph of conditional DAG with the first previous δ conditional pairs, there are k^δ different

execution instances at the on-line execution phase. In fact, for a given constant δ , it is practical to enumerate all these k^δ instances and find the maximum bounds of the M cores, whose time complexity is $O(k^\delta)$. Moreover, for these previous δ conditional pairs, the obtained time values of cores (namely the tuple $\mathcal{T}_{I^*, \beta_\delta}$) are more accurate than that of Algorithm 1 (namely the tuple $\mathcal{T}_{\max, \beta_\delta}$), formally $\mathcal{T}_{I^*, \beta_\delta} \preceq \mathcal{T}_{\max, \beta_\delta}$, since it considers all the k^δ actual instances and choose the exact one. To reduce computation overhead, while for the remaining $n - \delta$ conditional pairs, we still employ Algorithm 1, whose complexity is $O(|n - \delta| \cdot k \cdot M \log M)$. In this way, assume the tighter and generalized response time bound is denoted by R_δ , whose total computation complexity is $O(k^\delta) + O(|n - \delta| \cdot k \cdot M \log M)$.

Then we have

$$R^{\text{off}}(I^*) \leq R_\delta \leq R_{\text{apr}}$$

We can observe that R_δ is tighter than R_{apr} and safe without the magnitudely increasing of time complexity. The higher of the variable δ , the tighter of response time bound, but is computationally more expensive. As for the value setting of δ , it can be specified according to the requirement of the users, making a tradeoff between the tightness of response time bound and the computation overhead. Hence, such generalized R_δ can compute a safe and tight response time bound within polynomial time, and can be scalable for various size and precision requirements of conditional DAG task.

5 FINDING VIRTUAL EXECUTION ORDER $\mathcal{O}_{VE}$

In Section 4, we proposed the general method of timing-anomaly free execution, and based on that, computed a safe, tight and generalized bound R_δ . However, before computing the response time bound, the virtual execution order $\mathcal{O}_{VE}$ must be given at the off-line estimation phase. To minimize response time, in this section, we propose a heuristic algorithm to obtain a proper virtual execution order $\mathcal{O}_{VE}$.

We first consider the non-conditional DAG instance, our proposed heuristic algorithm is mainly based on the following two properties.

- **Work-conserving.** At off-line estimation phase, the scheduler never idles the cores if there exists a job waiting to be executed. *Work-conserving scheduling* is widely used in many literatures, such as [5, 14, 17].
- **Maximized parallelism.** The scheduling algorithm aims to improve the degree of parallelism, making the difference between the minimum and maximum time values of the cores minimized after the completion of the *sink* job.

In this paper, we consider the single-source single-sink DAG. The virtual execution order is obtained when simulating the execution of the instance at the off-line estimation phase, and each job is assumed to be executed for its WCET. In Algorithm 2, the order set $\mathcal{O}_{VE}$ is initialized to include *source* job, while the ready queue $\mathcal{Q}_R$ initially includes the successor jobs of *source* job.

During off-line simulation, we use the work-conserving scheduling strategy. That is, as long as there exist idle cores, the jobs from ready queue $\mathcal{Q}_R$ will be dispatched, shown in Line 4. To improve the degree of parallelism, at the scheduling point, the heavy job with longest WCET of the ready queue is scheduled in highest priority. Otherwise, the longest job is impossible to be executed on multiple cores in parallel since it can not be divided into sub-jobs. In contrast, the light jobs whose WCET sum equals to the heaviest job can be executed on multiple cores, apparently improving the degree of parallelism. Especially, if the job with longest WCET is executed just before the *sink* job but after the completion of all the parallel light jobs, the degree of parallelism is reduced significantly. Note that in Line 7, the *completion* of a job will trigger the scheduling operation.

ALGORITHM 2: Finding O_{VE} for non-conditional instance

Input: Non-conditional instance $G = (V, E)$, M cores;

Output: Virtual execution order O_{VE} ;

```
1  $O_{VE} = \{v_{src}\};$ 
2 Add the successor jobs of  $v_{src}$  to  $Q_R$ ;
3 while ( $O_{VE}.size() \neq |V|$ ) do
4   while ( $Q_R \neq \emptyset$  and  $hasIdleCore()$ ) do
5      $v_i = MaxWcet(Q_R);$ 
6      $O_{VE}.Add(v_i);$ 
7      $Q_R.Delete(v_i);$ 
8   Obtain the next scheduling point;
9   for ( $v_i \in V \setminus Q_R \setminus O_{VE}$ ) do
10    if ( $NoIndegree(v_i)$ ) then
11       $Q_R.Add(v_i);$ 
12 Return  $O_{VE};$ 
```

After that, Algorithm 2 adds the current jobs with no incoming jobs into the ready queue Q_R , since their predecessor jobs have finished completion. After all the completion of $|V|$ jobs, the virtual execution order O_{VE} is returned.

The proposed heuristic algorithm is work-conserving at off-line estimation phase, since a job from the ready queue Q_R is always dispatched to be *executed* if there exists an idle core. To guarantee timing-anomaly free execution, it should be noted that such scheduling algorithm may be not work-conserving at on-line execution phase when the execution of some jobs is less than its WCET. The proposed heuristic algorithm is also parallelism maximized, since the longest job from the ready queue Q_R is always dispatched first, making the difference between the minimum and maximum time values of the cores minimized.

As for the conditional DAG with n conditional pairs and with k branches of each pair, it is hard to simulate its execution off-line since there are k^n exponential execution instances. To facilitate analysis, we in this paper consider the conditional pair as a whole, which is replaced with a regular and non-conditional job, also stating in Section 4.2. The WCET of conditional job is the largest WCET sum among its k branches, so that it can cover the worst-case scenario. Hence, the proposed Algorithm 2 can be still applicable for this *equivalent* non-conditional DAG, where the position of the conditional job is fixed in the virtual execution order O_{main} . Note even we regard a conditional pair as a single conditional job, but actually it consists of k non-conditional branches. Thus, the sub-order $O_{i,j}$ of j^{th} branch of pair $\mathcal{P}_i$ is derived by repeatedly using the Algorithm 2 according to current time states of the cores at the off-line estimation phase.

6 PERFORMANCE EVALUATION

6.1 Experimental Setup

In this section, we conduct a set of experiments to evaluate the effectiveness of the proposed virtual execution order to derive the response time upper bound for non-conditional and conditional DAG task. Our approach is evaluated in comparison to 3 related counterparts - dynamical scheduler Lazy proposed in [21], and analytical bounds in [10, 16]. Our performance evaluation is conducted using randomly generated DAG task with different parameter configurations. By using the Erdos-Renyi method $G(n, p)$ [9], the parameters of each DAG task are generated as follows:

Fig. 9. Normalized bound with varying the number of processors for conditional DAG task.

- $\mathcal{N}$. The number of vertexes of a DAG task is uniformly chosen from the range $(4, \mathcal{N}_{max})$.
- C_i . The worst-case execution time (WCET) C_i of each vertex v_i is uniformly chosen from the range (C_{min}, C_{max}) .
- $e_{i,j}$. A connectivity probability threshold p is predefined. If the randomly generated probability from $(0, 1)$ is less than p , the edge $e_{i,j}$ between the two jobs v_i and v_j is added. The former vertexes are only allowed to point to the later vertexes, so that the acyclicity is guaranteed.
- $\mathcal{P}_i$. Conditional pair $\mathcal{P}_i$ is derived by combining the multiple independent non-conditional DAGs with a pair of conditional starting and joining jobs.

Similar to $e_{i,j}$, each regular vertex in a generated non-conditional DAG is replaced with a $\mathcal{P}_i$ when the generated probability from $(0,1)$ is less than the predefined probability threshold p_{con} . Then, a conditional DAG can be generated from a non-conditional DAG. We repeatedly ran hundreds of experiments using different parameter configurations, and averaged them as the final results.

6.2 Experimental Results

In this part, we aim to present the experimental results of the improvement of the response time bound for non-conditional/conditional DAG task. We first conduct a set of experiments on conditional DAG task. The state-of-the-art related work [16] is proposed by Melani et al. to obtain the response time upper bound of conditional DAG task. Assume a conditional DAG task is assigned M dedicated identical cores, the corresponding bound is derived by [16] (here we take no account of inter-task interference):

$$R_{Anal} = Len + \frac{Vol - Len}{M} \quad (1)$$

Where Vol is the maximum WCET sum from non-conditional instances of the conditional DAG task, and Len is the length of the critical path (namely the longest chain) of the instance that has maximum Vol .

Our response time upper bound $R(R_{\delta=0}, R_{\delta=5})$ is derived by setting the value of parameter δ as 0 and 5, where δ denotes the number of previous conditional pairs that are used to obtain the exact bounds of the cores (See Section 4.3). Figure 9 and Figure 10 show the normalized response time bound by our method, with the baseline of that obtained by Equation (1). With changing the number of processors and the connectivity probability threshold p , our derived bound is always less than the bound obtained by the analytical formula in Equation (1) under the same conditional DAG task. The formula in Equation (1) assumes that these jobs of the critical path and those jobs that are not in the critical path are not executed in parallel. In fact, the jobs of the critical path are allowed to be executed in parallel with those jobs that are not part of the critical path. Therefore, with the overly pessimistic assumption of the analytical method [16], the response time upper

Fig. 10. Normalized bound with varying connectivity probability threshold p for conditional DAG task.

Fig. 11. Normalized bound with varying connectivity probability threshold p for non-conditional DAG.

bound estimation of the formula Equation (1) is pessimistic than that of our work. Especially, with shifting the adjustable parameter δ from 0 to 5, the response time upper bound is tighter, formally $R_{\delta=5} \leq R_{\delta=0}$.

We also conduct experiments for the non-conditional DAG task. To derive the response time bound of non-conditional DAG task, there are two related works Lazy scheduler [21], and the classic bound [10] that is widely used in many references, such as [3, 5, 16, 17]. Similar to our work, the authors in [21] perform bound analysis by simulating the execution of DAG task using Lazy scheduler at design time. The response time bound obtained by the work [21] is denoted as R_{Lazy} . R_{Anal} denotes the bound obtained by the analytical method [10]. While R_{Our} denotes the bound obtained by our proposed scheduling order. All of the results are averaged and normalized to the analytical bound R_{Anal} obtained in [10]. That is, the final bounds obtained by these three methods are R_{Our}/R_{Anal} , R_{Lazy}/R_{Anal} and 1, respectively.

As we can see in Figure 11, the response time upper bound obtained by our proposed method is always the smallest one. Our method at least has the same performance as Lazy in the worst case. This is because we employ that work-conserving and parallelism maximized scheduling at the off-line estimation phase, assuming the jobs are executed for their WCETs. Note that, to guarantee anomaly-free execution at on-line execution phase, the ready jobs may also not be executed even if the idle cores exist in our work. In contrast, the Lazy scheduler [21] may idle the cores even if some jobs are ready whether at off-line estimation phase or on-line execution phase. Hence, the performance of our method outperforms the other two methods regarding the bound reduction. Specifically, when $M=2$, as the connectivity probability p increases, the normalized bound of our work firstly decreases and then increases, achieving the bound minimization when $p = 0.3$. When $M=4$ and $M=6$, the normalized bound R_{Our}/R_{Anal} increases when p increases, since each job has more possible precedence constraints. The parallelism is reduced when p increases; in the extreme case when $p = 1$, each job is executed almost sequentially. Thus, the improvement by changing the execution order of jobs has less room when p becomes larger.

7 CONCLUSION

In this paper, we present a suite of dynamic scheduling techniques for conditional DAG task, including a scheduling order constraint rule for timing-anomaly free execution, an effective abstraction of deriving the response time upper bound within polynomial time and a vertex order algorithm obtaining a shorter upper bound, that are necessarily designed to boost the real-time performance of multi-core systems. By using the proposed techniques for conditional DAG task, not only the timing-anomaly free execution is guaranteed, but also a shorter response time bound is derived within polynomial time compared with the state-of-the-art counterparts. To the best of our knowledge, this is the first work that contributes to conditional DAG task, by simulating its execution off-line, to compute a safe and tighter estimated response time upper bound.

ACKNOWLEDGMENT

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818) and National Natural Science Foundation of China (Grant No. 61672140), and Nanyang Assistant Professorship (NAP) M4082282 and Start-Up Grant (SUG) M4082087 from Nanyang Technological University, Singapore.

REFERENCES

- [1] Hamid Arabnejad and Jorge G. Barbosa. 2014. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE Transactions on Parallel and Distributed Systems* 25, 3 (2014), 682–694.
- [2] Philip Axer, Sophie Quinton, Moritz Neukirchner, Rolf Ernst, Björn Döbel, and Hermann Härtig. 2013. Response-time analysis of parallel fork-join workloads with real-time constraints. In *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 215–224.
- [3] Sanjoy Baruah. 2015. The federated scheduling of systems of conditional sporadic DAG tasks. In *Proceedings of the 12th International Conference on Embedded Software*. IEEE Press, 1–10.
- [4] Sanjoy Baruah, Marko Bertogna, and Giorgio Buttazzo. 2015. *Multiprocessor Scheduling for Real-time Systems*. Springer.
- [5] Sanjoy Baruah, Vincenzo Bonifaci, and Alberto Marchetti-Spaccamela. 2015. The global EDF scheduling of systems of conditional sporadic DAG tasks. In *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 222–231.
- [6] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*. IEEE, 63–72.
- [7] Jean Paul Calvez, Alan Wyche, and Charles Edmundson. 1993. *Embedded Real-time Systems*. Vol. 23. Wiley New York.
- [8] Hoon Sung Chwa, Jinkyu Lee, Kieu-My Phan, Arvind Easwaran, and Insik Shin. 2013. Global edf schedulability analysis for synchronous parallel tasks on multicore platforms. In *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 25–34.
- [9] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. 2010. Random graph generation for scheduling simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and), 60.
- [10] R. L. Graham. 1969. Bounds on multiprocessing timing anomalies. *Siam Journal on Applied Mathematics* 17, 2 (1969), 416–429.
- [11] Yu-Kwong Kwok and Ishfaq Ahmad. 1996. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 7, 5 (1996), 506–521.
- [12] Yu-Kwong Kwok and Ishfaq Ahmad. 1999. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)* 31, 4 (1999), 406–471.
- [13] Karthik Lakshmanan, Shinpei Kato, and Ragunathan Rajkumar. 2010. Scheduling parallel real-time tasks on multi-core processors. In *2010 31st IEEE Real-Time Systems Symposium*. IEEE, 259–268.
- [14] Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. 2014. Analysis of federated and global scheduling for parallel real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 85–96.
- [15] Thomas Lundqvist and Per Stenstrom. 1999. Timing anomalies in dynamically scheduled microprocessors. In *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*. IEEE, 12–21.

- [16] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. 2017. Schedulability analysis of conditional parallel task graphs in multicore systems. *IEEE Trans. Comput.* 66, 2 (2017), 339–353.
- [17] Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio C. Buttazzo. 2015. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *2015 27th Euromicro Conference on Real-Time Systems*. IEEE, 211–221.
- [18] Risat Mahmud Pathan, Petros Voudouris, and Per Stenstrom. 2018. Scheduling parallel real-time recurrent tasks on multicore platforms. *IEEE Transactions on Parallel Distributed Systems* PP, 99 (2018), 1–1.
- [19] Jan Reineke, Björn Wachter, Stefan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. 2006. A definition and classification of timing anomalies. In *6th International Workshop on Worst-Case Execution Time Analysis (WCET'06)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- [20] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2013. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems* 49, 4 (2013), 404–435.
- [21] Petros Voudouris, Per Stenström, and Risat Pathan. 2017. Timing-anomaly free dynamic scheduling of task-based parallel applications. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 365–376.