Worst-case response time analysis for partitioned fixed-priority DAG tasks on identical processors

Slim BEN-AMOR
Kopernic, INRIA
Paris, France
slim.ben-amor@inria.fr

Liliana CUCU-GROSJEAN
Kopernic, INRIA
Paris, France
liliana.cucu@inria.fr

Dorin MAXIM

Abstract—The continuous integration of new functionality increases the complexity of embedded systems, while each functionality might impose precedence constraints between the programs fulfilling it. In addition, the prevalence of several processors may create the illusion of higher computation capacity easing the associated scheduling problem. However, this capacity is not exploitable in critical real time systems because of the increased variability of the execution times due to processor features designed to provide excellent average time behaviour and not necessarily ensuring small worst case bounds. This difficulty is added to the existence of scheduling anomalies when the systems are built on top of several processors. In this paper, we study the feasibility of independent tasks scheduled according to a given preemptive fixed-priority partitioned policy on identical processors. Each task is composed of several dependent sub-tasks related between them according to a directed acyclic graph (DAG). We provide a worst case response time analysis for DAG tasks when each sub-tasks have an individual priority level. This assumption allows to decrease the number of possible execution scenarios, making our analysis easier and less pessimistic.

Index Terms—Precedence constraints, DAG graph, Multiprocessor, Schedulability, Partitioning

I. INTRODUCTION

The widespread of intelligent devices pushes embedded system designers to include more features. Each new functionality is ensured by a group of programs (or tasks). The latter are related by precedence constraints that should be satisfied in order to ensure the functional correctness. These precedence constraints are often described by the directed acyclic graph (DAG) task model. Besides, chips manufacturers aim to improve the performance while reducing power consumption. Thus, they incorporate several cores on the same processor seeking for speedup.

However, multiprocessors constitute unpredictable architectures by adding interference and communication delays between different tasks executed on different cores. These interactions increase the number of possible execution scenarios. Since timing analysis techniques for real-time systems are designed to analyze simple software that run on simple and predictable architecture, handling all execution scenarios may become strongly pessimistic.

In this paper, we focus on partitioned scheduling on identical processors as an interesting option to reduce the number of possible execution scenarios. This strategy may be suitable for hard real-time systems, making them simpler to analyze and validate. Besides, we consider fixed-priority scheduling policies where instances of the same task have the same priority during the entire schedule. The priority assignment technique is out of the scope of this paper. Thus, we assume that tasks priorities are given.

Our contribution: In this work, we present a schedulability analysis of DAG tasks scheduled according to a partitioned policy on identical processors. We propose a response time analysis based on fixed priorities defined on sub-task level.

II. RELATED WORK

The scheduling problem of real-time systems on multicore processors have been extensively studied [1] after the widespread of these architectures. Beside sequential task models, the problem of scheduling parallel tasks has been tackled in the literature under different task models. The fork-join model represents a task as an alternating sequence of sequential and parallel segments. The number of sub-tasks in parallel segments should be the same on all segments and it should not exceed the number of processors. Lakshmanan et al. [2] have proposed a stretch transformation for fork-join model to execute the parallel segments as sequential when possible. The synchronous parallel model is considered in [3]. In this later work, authors present a task decomposition algorithm that transforms implicit deadline tasks into constrained deadline tasks. It is also provided a resource augmentation bound for G-EDF and partitioned DM scheduling. This model removes some restrictions of the fork-join model. It allows different numbers of sub-tasks in each segment and this number could be greater than the number of processors. However, the synchronization is still required after each parallel segment.

A more general parallel structure is the DAG task model where each task is represented by a directed acyclic graph that describe precedence between sub-tasks. This model has been studied in case of global scheduling in [4], [5]. DAG task model is also explored under partitioned scheduling system by Fonseca et al. [6]. Authors use self-suspending task to model the DAG task. Then, they estimate the response time of each task by resolving a mixed integer linear problem. This approach provides a good estimation of response time compared to existing work but it is not scalable with high number of tasks due to the complexity of the associated Mixed ILP that has to be solved.
In addition, partitioned scheduling of parallel tasks is studied in the context of distributed system. Tindell and Clark [7] propose an end-to-end response time analysis of several independent sequences of sub-tasks. These independent sequences can execute in parallel on multiprocessor platform. This holistic approach was refined later by Palencia et al. [8]. It is used, now, in MAST tool [9] to analyze multi-path end-to-end flows. This approach is pessimistic since it assumes that a worst-case scenario occurs at each activation of a sub-task from the sequence.

III. TASK MODEL AND NOTATION

We consider a set of $n$ sporadic real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled according to a fixed-priority and preemptive policy on $m$ identical processors denoted $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$. Each task $\tau_i$ is specified by a 3-tuple $(G_i, D_i, T_i)$ where $G_i$ is a directed acyclic graph (DAG), and $D_i$ and $T_i$ are positive integers. In fact, each task $\tau_i$ is a recurrent process that releases an infinite sequence of "jobs" $\tau_i^j$, $j \in \mathbb{N}$. The first job is released at time instant zero. While, subsequent jobs are released at least after $T_i$ time units. Every job released by $\tau_i$ has to complete its execution within its deadline i.e. $D_i$ time units from its release. We assume that the task set has constrained deadline ($D_i \leq T_i, \forall i \in \{1, 2, \ldots, n\}$).

![Fig. 1: Example of precedence graph divided into levels according to topological ordering](image)

The internal structure of a task $\tau_i$ is described by the DAG $G_i = (V_i, E_i)$, where $V_i$ is a set of $n_i$ sub-tasks and $E_i \subseteq (V_i \times V_i)$ is a set of directed edges connecting two sub-tasks (it is required that these edges do not form any cycle). A sub-task $\tau_i,l = (C_i,l, p_i,l)$ is characterized by its (WCET) worst-case execution time $C_i,l$ and by the processor $p_i,l$ to which it is assigned. We suppose that the mapping of the sub-tasks to processors is given. Each directed edge $(\tau_i,a, \tau_i,b) \in E_i$ denotes a direct precedence constraint between sub-tasks $\tau_i,a$ and $\tau_i,b$ meaning that sub-task $\tau_i,b$ cannot start executing before sub-task $\tau_i,a$ completes its execution. In this case, $\tau_i,b$ is called a "successor" of $\tau_i,a$, whereas $\tau_i,a$ is called a "predecessor" of $\tau_i,b$. We denoted the set of direct predecessors of sub-task $\tau_i,l$ by $\text{pred}(i, l)$. While $\text{succ}(i, l)$ refer to its set of direct successors. We call a sub-task without any predecessors or successors, respectively, "source" or "sink" sub-task.

**Definition 1.** A sub-task $\tau_{i,a}$ is called transitive predecessor of $\tau_{i,b}$ if there is at least one sequence of connected sub-tasks that starts with $\tau_{i,a}$ and ends with $\tau_{i,b}$.

We denote the set of transitive and direct predecessors of sub-task $\tau_{i,l}$ including $\tau_{i,l}$ itself by $\text{pred}^\infty(i, l)$. $\text{pred}^*(i, l)$ refers to only the set of transitive and direct predecessors of sub-task $\tau_{i,l}$. Analogously, we define $\text{succ}^\infty(i, l)$ and $\text{succ}^*(i, l)$ as the set of all transitive successors of the sub-task $\tau_{i,l}$ including and excluding itself respectively. If sub-task $\tau_{i,a} \notin \text{pred}^\infty(i, l)$ and $\tau_{i,a} \notin \text{succ}^\infty(i, l)$, then the two sub-tasks are independent and they may execute in parallel whenever they are mapped to different processors.

IV. RESPONSE TIME ANALYSIS

In this section, we propose a response time analysis of DAG task model. We estimate the worst-case response time (WCRT) of a DAG task $\tau_i$ in presence of higher priority tasks and we take into consideration existing precedence constraints between different sub-tasks. Our estimation should be safe and never underestimates the actual WCRT. Indeed, in case of underestimation, the proposed analysis will guarantee that completion of any job occurs before its deadline while the actual response time may exceed the estimated WCRT and the deadline. Hence, the system may encounter some failure despite the feasibility analysis. In order to avoid underestimation, we calculate an upper bound of the WCRT and we provide a mathematical proof to ensure the safety of this bound.

The studied task model defines a partial order between sub-tasks composing a DAG task through the precedence graph. This incomplete order may create several ambiguous cases for the scheduler where different sub-tasks with the same priority are ready to execute on the same core. We know that a core could execute only one sub-task at time. Hence, the scheduler has to choose one sub-task arbitrary since all ready sub-tasks have the same priority or order. This random choice will create several possible execution scenarios making the estimation of the upper bound of WCRT harder and it may cause an overestimation of the actual response time. We define priorities in sub-task level in order to decrease the number of possible execution scenario and reduce the pessimism of the estimation.

**A. Motivation example**

We illustrate the purpose of sub-priority by the following example. We schedule the DAG task defined by Figure 1 with two different sub-tasks ordering. The execution time of $\tau_{1,1}, \tau_{1,2}$ and $\tau_{1,3}$ is 1 and the execution time of $\tau_{1,4}$ and $\tau_{1,6}$ is 2 while $\tau_{1,5}$ has an execution time of 3 time units. We assume that sub-tasks $\tau_{1,1}, \tau_{1,2}$ and $\tau_{1,5}$ execute on the same processor and the rest of sub-tasks execute in another processor. After the execution of $\tau_{1,1}$, sub-tasks $\tau_{1,2}$ and $\tau_{1,5}$ become ready for execution on processor $\tau_1$. If we start with $\tau_{1,5}$, as in the first schedule (see Figure 2a), the response time would be equal to $R_1 = 9$. While, if we start with $\tau_{1,2}$ (see Figure 2b) the response time would be equal to $R_1 = 7$. Hence, the order between sub-tasks influences the response time of a DAG task by exploiting or not the possible parallelism.
Theorem 1. The worst-case response time in isolation of a sub-task \( \tau_{i,j} \) in a graph \( G_i \) is upper bounded by:

\[
R_{i,j}^{isol} = C_{i,j} + \max_{\tau_{i,l} \in pred^\infty(i,j)} \left\{ R_{i,l}^{isol} + I_{i,l}(pred^*(i,j)) \right\}
\]

Proof. We prove this theorem using mathematical induction. We divide the graph \( G_i \) into levels according to topological ordering described by Kahn [10]. Then, we proceed it level by level (see Figure 1).

First, we verify Equation (2) for the first level of source sub-tasks. Since, these nodes have no predecessors, the response time according to the formulas is equal to their execution time. This is correct because we consider each task separately and there is no previous sub-tasks that may affect their response time in isolation.

Now, we assume that Equation (2) is correct for all levels from the first level to level \( s \) and we prove it for level \( s + 1 \). Let \( \tau_{i,j} \) be a sub-task in the level \( s + 1 \), we know that all its predecessors \( \tau_{i,l} \in pred(i,j) \) are located in some levels prior or equal to \( s \). Thus, we can apply Equation (2) to the predecessors by using the induction hypothesis. This means that the upper bound \( r_{i,j} \) on response time of one of these predecessors \( \tau_{i,l} \) gives enough time for \( \tau_{i,l} \) and all sub-tasks of \( pred^*(i,l) \) to be executed. The term \( r_{i,j} + I_{i,l}(pred^*(i,l)) \) counts the required time for all sub-tasks of \( pred^*(i,l) \) and all the interference caused by other predecessors of \( \tau_{i,j} \). Hence, this term provides sufficient time for all transitive predecessors of \( \tau_{i,j} \) to be executed. If we take the maximum of this term over all direct predecessor we cover the worst scenario of execution. Then, we add the execution time \( C_{i,j} \) of the concerned sub-task. Thus, the upper bound given by Equation (2) is sufficient for the completion of the sub-task \( \tau_{i,j} \) and all its transitive predecessors.

In conclusion, we verify Equation (2) for any sub-task on level \( s + 1 \). Then, it is true for any level and any sub-task in the graph \( G_i \).

The local response time formulation considers the interference of all sub-tasks from the same graph while the response time in isolation counts only transitive predecessor sub-tasks. Thus, we use response time in isolation to compute the upper bound of local response time.

Corollary 1. The worst-case local response time of sub-task \( \tau_{i,j} \) in a graph \( G_i \) is upper bounded by:

\[
R_{i,j}^{loc} = R_{i,j}^{isol} + \sum_{\pi(\tau_{i,k}) \in i, hp(pred^\infty(i,j)), \tau_{i,k} \notin pred^\infty(i,j)} C_{i,k}
\]

In order to evaluate the upper bound of the local response time of the whole graph \( G_i \), we add a sub-task at the end of the DAG that succeeds all sink sub-tasks. This additional sub-task has a zero execution time and its response time will consider possible interference between the different predecessor branches.

2) Global response time: The global WCRT of a sub-task \( \tau_{i,j} \) is deduced from the response time in isolation by adding the worst-case workload caused by the preemption of higher priority tasks.

\[
R_{i,j}^{glob} = R_{i,j}^{isol} + w_{i,j}
\]
The workload caused by higher priority tasks is calculated recurrently by the following equation:

\[ w^{n+1}_{i,j} = \sum_{\pi(p,j)=\pi(t_{i,j})} \left[ w^n_{i,j} + L_{i,j} + J_{p,q} \right] C_{p,q} \]  

(5)

where \( J_{p,q} \) is the jitter term as introduced in [8].

\[ J_{p,q} = \max_{\tau_{p,s} \in \text{pred}(p,q)} \{ J^{\text{low}}_{p,s} \} \]

\( L_{i,j} \) is the initial interval used for computing how many pre-emption of higher priority tasks could occur. It is expressed as a difference between two terms. The first is the latest possible completion time \( R^{\text{local}}_{i,j} \) of the sub-task \( \tau_{i,j} \) considering only the effect of sub-tasks from task \( \tau_i \). The second term is the earlier release of the sub-task \( \tau_{i,l} \) that belongs to the connected sub-graph \( G^{\text{local}}_{i,j} \) composed only of the transitive successor and predecessor of \( \tau_{i,j} \) and executes on the same processor as \( \tau_{i,j} \).

\[ L_{i,j} = R^{\text{local}}_{i,j} - \min_{\tau_{i,l} \in \text{successors}(i)} \left\{ \max_{\tau_{i,k} \in \text{pred}(i,l)} \left\{ J^{\text{low}}_{i,k} \right\} \right\} \]

The earliest possible release of sub-task \( \tau_{i,l} \) is estimated using the maximum lower bound of the response time of all predecessor. A safe lower bound of response time is expressed by the following equation:

\[ R^{\text{low}}_{i,k} = C_{i,k} + \max_{P \in \pi} \left\{ \sum_{\tau_{i,k} \in \text{pred}(i,l)} C_{i,l} \right\} \]  

(6)

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed response time analysis. We also compare our approach to the holistic analysis proposed by Palencia et al. [8]. This approach assumes that a worst-case scenario occurs in processor \( \pi(t_{i,j}) \) at each activation of sub-task \( \tau_{i,j} \).

We generate 100 DAG task sets using “randfixedsum” [11]. Then, we compute the actual worst-case response time in the schedulability interval using the real-time scheduling simulator SimSo [12]. Finally, we evaluate the average percentage of overestimation introduced by each approach compared to the simulation under various system configuration (see Figure 3).

Our approach introduces less pessimism between 27% and 54%. While the holistic approach causes between 43% and 157% of overestimation. The overestimation increases significantly for holistic approach when number of tasks and sub-tasks increase (see Figures 3b-3c) because the interaction between sub-tasks expands. Nonetheless, our approach is not very affected because interactions are predictable due to the priorities between sub-tasks.

VI. CONCLUSION

In this paper, we provide a response time analysis for parallel real-time system defined by DAG task model scheduled according to a fixed priority preemptive partitioned policy on identical processors. We assume that priorities are defined individually for each sub-task. Then, we evaluate and compare our method to the holistic approach. As future work, we will propose a priority assignment and partitioning heuristics using the DAG structure to take advantage of possible parallelism.

REFERENCES