Optimal Scheduling of Measurement-Based Parallel Real-Time Tasks

Kunal Agrawal · Sanjoy Baruah · Pontus Ekberg · Jing Li

Abstract In this work we consider a measurement-based model for parallel real-time tasks represented by the work and span parameters of directed acyclic graphs, with different bounds for nominal and overload scenarios. We address the corresponding real-time scheduling problem and propose an optimal scheduling strategy with a derived tight bound on the maximum response time of a task.

Keywords DAG scheduling · Uncertainty · Multiprocessors · Optimality

1 Introduction

Task models based upon directed acyclic graphs (DAGs) are widely used for representing recurrent real-time processes in a manner that exposes their internal parallelism, thereby enabling the exploitation of such parallelism upon multiprocessor and multicore platforms. These task models typically represent pieces of sequential (i.e., non-parallelizable) computation via vertices and their dependencies as edges between vertices; hence constructing such a model for a recurrent process requires detailed knowledge of the internal control-flow structure of the process.

This research was supported by NSF grants CCF-1733873, CCF-1618802, CCF-1439062, CNS-1814739, CPS-1932530, CNS-1911460, and CNS-1948457 and by Swedish Research Council grant 2018-04446.

Kunal Agrawal
Washington University in St. Louis, Campus Box 1045, St Louis, MO 63130, USA
E-mail: kunal@wustl.edu

Sanjoy Baruah
Washington University in St. Louis, Campus Box 1045, St Louis, MO 63130, USA
E-mail: baruah@wustl.edu

Pontus Ekberg
Uppsala University, Box 337, SE-751 05 Uppsala, Sweden
E-mail: pontus.ekberg@it.uu.se

Jing Li
New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
E-mail: jingli@njit.edu

Such knowledge is not always available. Furthermore, even when available, conservative estimates of the computational demands of individual vertices, e.g., via worst-case execution time (WCET) parameters, can result in severe under-utilization of computational resources during run-time. To ameliorate these problems, a *measurement-based* model was recently proposed [1]. This model deals with the lack of knowledge of the internal structure by representing the computation of a DAG with just the two parameters *work* (the cumulative computation of all the vertices in the DAG) and *span* (the maximum cumulative computation of any precedence-constrained sequence of vertices). This model deals with the potential pessimism by requiring that two estimates be provided for each parameters: $work_O$ and $span_O$ are very conservative upper bounds (safe even under *overload* conditions), while $work_N$ and $span_N$ are *nominal* upper bounds (i.e., upper bounds under "typical" circumstances) on the values of the work and span parameters respectively. It is assumed that $work_N \leq work_O$ and $span_N \leq span_O$.

Definition 1 (The scheduling problem) Suppose we are given a task represented by the four parameters $work_N$, $span_N$, $work_O$ and $span_O$, and a deadline D and two processor counts: m_N and m_O, where $m_N \leq m_O$. The scheduling problem is to finish the task with a makespan (response time) no larger than the deadline D, and we may use at most m_N processors to do so, unless it is *observed* during the execution that at least one of the nominal parameters $work_N$ and $span_N$ does not provide a valid upper bound for the current invocation of the task. If this is observed, we may switch to using up to m_O processors instead for the remainder of the execution, but we must still meet the original deadline D even if the computational demands of the task invocation turns out to be as high as $work_O$ and $span_O$. The scheduler does not know anything more about the internal details of the task than what can be deduced from the given parameters. □

The approach presented in [1] is a scheduling strategy that precomputes an upper bound D_N on the maximum makespan that is possible when executing a task with a total work at most $work_N$ and a span at most $span_N$ upon m_N processors using any greedy (work-conserving) scheduling [2]. It then starts to execute the given task upon m_N processors greedily, and after D_N time units checks whether the task has completed. If not at least one of $work_N$ or $span_N$ must have been exceeded, and so it activates the additional $(m_O - m_N)$ processors and continues the greedy execution until completion.

The new approach in this paper is also to begin executing the task greedily upon m_N processors, but rather than checking the progress of the task at a precomputed time point D_N, it instead monitors *the total amount of execution* occurring across all the m_N processors. If the invocation does not complete before the execution equals the nominal work parameter $work_N$, then it activates the additional $(m_O - m_N)$ processors and continues executing the task greedily until completion.

Contributions and comparisons. The approach of [1] only requires that the runtime detect whether the task has completed by time D_N. In contrast, our approach requires the capability to monitor the total progress on the work — that is, the amount of execution done across the processors. Assuming this capability is available, we will show below that our approach is, in fact, *optimal* — no other scheduler can guarantee to meet the deadline D under the constraints of the scheduling problem specified above if this approach cannot also do so. Note that, our approach also has the advantage that it only needs three parameters; $work_N$,

$work_O$, and $span_O$ since it does not need to monitor whether the span exceeds $span_N$. In contrast, the approach in [1] needs $span_N$ to calculate the intermediate deadline D_N with the approach of [1].

In addition, (Expression (1) of Theorem 2) is a tight bound on the maximum makespan with this new scheduling approach. In addition to its use as a schedulability test, this expression can be used to, e.g., minimize the processor counts m_N and m_O needed to meet the deadline. Note that this is exactly what we want to do if the task is periodically or sporadically activated and we wanted to schedule it in a federated manner similar to [3].

2 Schedulability conditions

We use a well-known result about scheduling DAG tasks characterized by single *work* and *span* parameters (i.e., where we don't separate nominal and overload scenarios).

Theorem 1 (Graham [2]) *The maximum makespan of a given DAG executed on m processors by a greedy (work-conserving) scheduler is no larger than* $M = (\frac{work - span}{m} + span)$. □

In the following, we derive a tight bound on the makespan for our new scheduling approach for DAG tasks that are characterized by parameters $work_N$, $span_N$, $work_O$ and $span_O$ for nominal and overload scenarios. Comparing this bound with a deadline is a sufficient schedulability condition for our proposed strategy and also a necessary condition for *any* scheduler following the rules of the scheduling problem described in Definition 1.

Theorem 2 *Our proposed scheduling strategy will execute a task with a makespan that is no larger than*

$$M = \begin{cases} \frac{work_O - span_O}{m_N} + span_O, & \text{if } work_N > work_O - span_O \\ \frac{work_N}{m_N} + \frac{work_O - work_N - span_O}{m_O} + span_O, & \text{if } work_N \leq work_O - span_O. \end{cases} \quad (1)$$

In addition, no scheduler can guarantee a smaller makespan. □

Theorem 2 follows directly from lemmas 1 to 4, proven below. We start with lemmas 1 and 2, which demonstrate that no scheduler can guarantee a smaller makespan bound. Recall from Definition 1 that schedulers are assumed to not know the internal structure of the DAG, except for what can be deduced from the four parameters $work_N$, $span_N$, $work_O$ and $span_O$. The actual structure of the DAG may be anything consistent with those parameters.

Lemma 1 *If* $work_N > work_O - span_O$*, then no scheduler can guarantee to complete the task with a makespan smaller than* $\frac{work_O - span_O}{m_N} + span_O$.

Proof Consider a task invocation where the first $work_O - span_O$ units of work that can be executed is fully parallel (i.e., not on the critical path of the DAG) and the remaining $span_O$ units of work is sequential. Because $work_N > work_O - span_O$, no scheduler may activate the extra $m_O - m_N$ processors until some time after finishing the first $work_O - span_O$ units of work. This initial work cannot be finished in less than $(work_O - span_O)/m_N$ time units. After finishing these $work_O - span_O$ units of work, the task invocation is left with the sequential workload that takes $span_O$ time units to finish no matter how many processors are available. Therefore, the task can finish earliest after $(work_O - span_O)/m_N + span_O$ time units. □

Lemma 2 *If $work_N \leq work_o - span_o$, then no scheduler can guarantee to complete the task with a makespan smaller than $\frac{work_N}{m_N} + \frac{work_O - work_N - span_O}{m_O} + span_o$.*

Proof Let the task invocation be such that the first $work_N$ units of work executed are fully parallel, which is possible since $work_N \leq work_o - span_o$. Then, no scheduler may activate the extra processors before finishing a total of $work_N$ units of work, which can happen earliest after $work_N/m_N$ time units. After finishing the first $work_N$ units of work and m_o processors are allowed to be used, the task invocation still has $work_o - work_N - span_o$ units of work that are fully parallel, which takes $\frac{work_O - work_N - span_O}{m_O}$ time units to finish. Lastly, the task invocation is left with an entirely sequential part that cannot be finished in less than $span_o$ time units. The total time to completion is then at least $\frac{work_N}{m_N} + \frac{work_O - work_N - span_O}{m_O} + span_o$. □

We now show with lemmas 3 and 4 that our proposed scheduling strategy can finish within a makespan no larger than the one specified in Theorem 2.

Lemma 3 *If $work_N > work_o - span_o$, then our proposed scheduling strategy will complete the task with a makespan no larger than $\frac{work_O - span_O}{m_N} + span_o$.*

Proof Follows from using Theorem 1 with the more conservative task parameters $work_o$ and $span_o$ and the smaller number of processors m_N that we are always guaranteed. □

Lemma 4 *If $work_N \leq work_o - span_o$, then our proposed scheduling strategy will complete the task with a makespan no larger than $\frac{work_N}{m_N} + \frac{work_O - work_N - span_O}{m_O} + span_o$.*

Proof We separately consider the cases where the nominal parameter $work_N$ holds or not during the execution of the task invocation.

Case 1 (The total workload of the current invocation is no larger than $work_N$): In this case the extra processors will never be activated. By Theorem 1 the makespan is no larger than $\frac{work_N - span_O}{m_N} + span_o$, and using the assumption $0 \leq work_o - work_N - span_o$ we have

$$\frac{work_N - span_o}{m_N} + span_o \quad \leq \quad \frac{work_N}{m_N} + \frac{work_o - work_N - span_o}{m_o} + span_o.$$

Case 2 (The total workload of the current invocation is larger than $work_N$): In this case, the extra $m_o - m_N$ processors will get activated by our proposed approach, say after t time units. Let t_{busy} denote the total amount of time before t where all m_N processors are busy, and let $t_{\text{idle}} = t - t_{\text{busy}}$ denote the total time during which at least one processor is idling. Let $work'$ and $span'$ denote the actual remaining work and span after the first t time units and note that $work' \leq work_o - work_N$ and $span' \leq span_o$.

Because a greedy scheduler never idles all processors unless the invocation completes and we have completed exactly $work_N$ units of execution after t time units, we have $work_N \geq t_{\text{busy}} \times m_N + t_{\text{idle}}$, which implies that $t_{\text{busy}} \leq \frac{work_N - t_{\text{idle}}}{m_N}$. Note that the first vertex in any path is always available for execution, and so if any processor is idle we know that all critical paths must currently be executing and therefore the remaining span is also being shortened. We must then have $span' \leq span_o - t_{\text{idle}}$, which implies $t_{\text{idle}} \leq span_o - span'$. Thus,

$$t \;=\; \left(t_{\text{busy}} + t_{\text{idle}}\right) \;\leq\; \frac{work_N - t_{\text{idle}}}{m_N} + t_{\text{idle}} \;\leq\; \frac{work_N}{m_N} + (span_o - span')\left(1 - \frac{1}{m_N}\right). \quad (2)$$

Using Eq. (2) and Theorem 1 we see that the total makespan cannot be larger than

$$\begin{aligned} t + \frac{work' - span'}{m_O} + span' &\leq \frac{work_N}{m_N} + (span_O - span') \left(1 - \frac{1}{m_N}\right) + \frac{work' - span'}{m_O} + span' \\ &= \frac{work_N}{m_N} + \frac{work'}{m_O} + span_O - \frac{span_O}{m_N} + \frac{span'}{m_N} - \frac{span'}{m_O} \\ &\leq \frac{work_N}{m_N} + \frac{work'}{m_O} + span_O - \frac{span_O}{m_N} + \left(\frac{1}{m_N} - \frac{1}{m_O}\right) span_O \\ &\leq \frac{work_N}{m_N} + \frac{work_O - work_N - span_O}{m_O} + span_O, \end{aligned}$$

which finishes the proof. □

References

1. Kunal Agrawal and Sanjoy Baruah. A measurement-based model for parallel real-time tasks. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, July 2018.
2. R. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.
3. J. Li, D. Ferry, S. Ahuja, K. Agrawal, C. Gill, and C. Lu. Mixed-criticality federated scheduling for parallel real-time tasks. In *Proceedings of the 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.