A note on task-parallelism upon multiprocessors

Joël Goossens
Université Libre de Bruxelles
Brussels, Belgium
Email: joel.goossens@ulb.ac.be

Shelby Funk
University of Georgia
Athens, GA, USA
Email: shelby@cs.uga.edu

We consider the scheduling of arbitrary deadline periodic task systems on multiprocessors. When the deadline of a task can exceed its period, it is possible that a task may have several jobs be simultaneously active. For uniprocessor systems these jobs are serialized (i.e., scheduled in FIFO order). For multiprocessors, we may be able to execute them simultaneously on different processors. Often, the jobs of all tasks are serialized without discussion (see, e.g., [1], [2], [3]). This type of execution would be required if each of a given task’s jobs uses data generated by the predecessor jobs. If, on the other hand, each job uses data gathered from environmental sensors, then multiple jobs generated by a single task may be able to execute simultaneously without conflict.

The conventional belief is that disallowing task parallelism is “worse” than allowing it. Below, we demonstrate that there are scenarios in which serialization and simultaneous execution are incomparable scheduling paradigms. In particular, we demonstrate these paradigms are incomparable for systems satisfying the following description:

- All tasks are synchronous and strictly periodic – i.e., a task $\tau_i$ with period $T_i$ will generate jobs at times $k \cdot T_i$, for $k = 0, 1, 2, \ldots$.
- The tasks are scheduled using either the Earliest Deadline First (EDF) algorithm or the Deadline Monotonic (DM) algorithm. EDF gives higher priority to jobs with earlier deadlines. DM gives higher priority to jobs generated by tasks with smaller relative deadlines.

While these assumptions are not uncommon, there are many other types of systems. For example, we might want to consider other scheduling algorithms or the more relaxed sporadic task model, in which the period $T_i$ indicates the minimum amount of time between the releases of $\tau_i$’s consecutive jobs.

Given that allowing and prohibiting task parallelism are incomparable scheduling variants for the specific scenarios described above, we must consider a number of questions. Specifically,

- Under what conditions does allowing / forbidding task parallelism improve schedulability?
- Are there specific analysis techniques that can be employed when task parallelism is permitted?
- Do these scheduling paradigms remain incomparable when using the sporadic task model?
- Do these paradigms continue to be incomparable for other scheduling algorithms such as LL-REF [4], BF [5] or DP-Wrap [6] or the Pfair [7] family of algorithms?

We now demonstrate that these two scheduling variants are incomparable. We begin by defining the terms we will use in the subsequent discussion.

**Model and definitions.** A periodic or sporadic task set is denoted $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task $\tau_i$ is described using the 3-tuple $(T_i, C_i, D_i)$, where $T_i, C_i$ and $D_i$ are $\tau_i$’s period, execution time and relative deadline, respectively. The jobs of a periodic tasks arrive exactly $T_i$ time units apart, with the first job arriving at time 0. The jobs of a sporadic task arrive at least $T_i$ time units apart, with the first job arriving no earlier than time 0. If a job of periodic or sporadic task $\tau_i$ arrives at time $t$, it must be allowed to execute for $C_i$ time units during the interval $[t, t + D_i)$. In the special case where the relative deadline is the same as the period, we describe $\tau_i$ using the pair $(T_i, C_i)$. We say task (job) parallelism occurs when a task (job) runs simultaneously on several processors. Assuming job parallelism is prohibited, we explore the implications of allowing or prohibiting task parallelism.

**Theorem 1.** Allowing / disallowing task parallelism are incomparable variants for both the EDF and DM scheduling of periodic task sets. Specifically,

- There are periodic task systems that are schedulable using EDF and DM when task parallelism is allowed, but are unschedulable using these algorithms when task parallelism is forbidden, and
There are periodic task systems that are schedulable using EDF and DM when task parallelism is forbidden, but are unschedulable using these algorithms when task parallelism is allowed.

Proof: We show each case separately. First, consider the task set \( \tau = \{(3,2), (3,2), (4,2,6)\} \). Figure 1(a) illustrates a schedule of this task set upon two processors for 12 time units (the hyperperiod of the system). The color of the tasks indicates which processor the job executes upon. The schedule in Figure 1(a) arises when using either the EDF or the DM scheduling policies. The schedule meets all deadlines only because the second job of task \( \tau_3 \) is allowed to execute simultaneously with the first and third jobs. If parallelism were forbidden for \( \tau_3 \) then its second job would clearly miss its deadline.

The theorem’s second statement is less intuitive. Even so, we show that this statement also holds. Consider the task set \( \tau = \{(7,2,2), (7,2,2), (10,7,11), (72,50)\} \). Figure 1(b) illustrates a schedule of this task set upon two processors. As in Figure 1(a), the color of the tasks indicates which processor the job executes upon and the figure illustrates the schedule generated by both EDF and DM. However, in Figure 1(b) task parallelism is forbidden. Because \( \tau_3 \)'s jobs overlap, its jobs have alternating light and dark outlines.

When parallelism is forbidden, the tasks never migrate. Tasks \( \tau_1 \) and \( \tau_3 \) execute on processor \( \pi_1 \), and tasks \( \tau_2 \) and \( \tau_4 \) execute on processor \( \pi_2 \). Observe that \( \tau_4 \) completes execution at time 70 and \( \tau_2 \) must execute during the interval [70, 72]. Therefore, if \( \tau_3 \) ever executes simultaneously on both processors then \( \tau_4 \) will miss a deadline. When task parallelism is permitted, \( \tau_3 \) would occupy both processors during the intervals [10, 11] and [30, 31], which would cause \( \tau_4 \) to miss its first deadline. Observe that in Figure 1(b) all tasks meet their deadlines during the interval [0, 72]. Because the tasks are effectively partitioned, this interval presents the worst case scenario for this system.

Consequently our intuition is incorrect in the sense that allowing/disallowing task-parallelism are incomparable scheduling variants – at least for EDF and for DM. This observation opens many new questions about the multiprocessor scheduling of tasks with unconstrained deadlines. A few of these questions are listed above. We believe that, as we gain deeper understanding of these two scheduling paradigms, additional questions will follow.

REFERENCES


