Optimal GEDF-Based Schedulers that Allow Intra-Task Parallelism on Heterogeneous Multiprocessors *

Kecheng Yang and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

A variant of the conventional sporadic task model is considered wherein successive invocations of the same task do not have to execute in precedence order. This model is motivated by stream-processing applications where successive data items can be processed independently. The considered hardware platform is assumed to be a heterogeneous multiprocessor with processors of different speeds. Such platforms can be utilized in embedded applications to enable performance guarantees to be made with acceptable energy costs. The main contribution of this paper is to show that preemptive and non-preemptive variants of the global earliest-deadline-first scheduler are optimal with respect to ensuring bounded response times under the considered task model and hardware platform. An experimental evaluation of both variants is presented as well.

1 Introduction

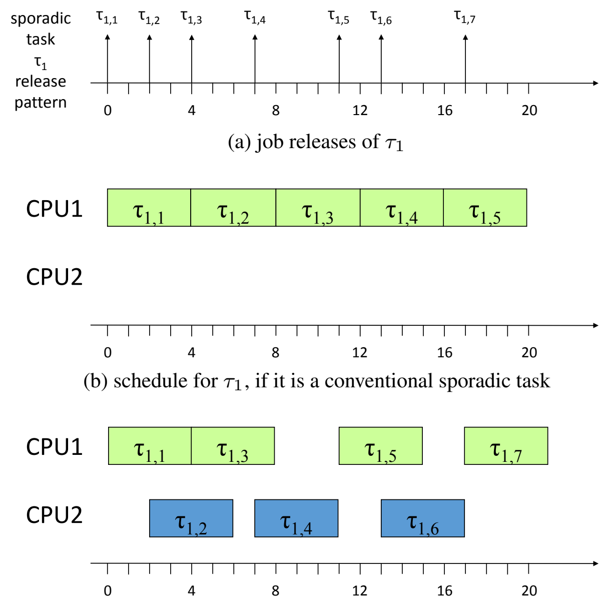
The computing industry recently experienced a major shift in CPU architectures with the advent of multicore chips. This shift has necessitated the adoption of new programming models, algorithms, and analysis methods to fully exploit the parallelism inherent in multicore chip designs. While advances in these areas are well underway, industry has already begun yet another architectural shift towards *heterogeneity* in order to achieve greater levels of performance and energy efficiency. Heterogeneity creates new challenges because the availability of different types of processing resources means that "choices" must be made when allocating hardware resources to software components. The need to resolve such choices can add considerable complexity to resource allocation.

An example platform that epitomizes the above-mentioned shift is recently proposed multicore technology by ARM called big.LITTLE [1], which integrates relatively slower, low-power processors with faster, high-power ones to balance performance and energy efficiency. This heterogeneous computing architecture is being used by Samsung in their new mobile SoC, Exynos 5422, which consists of four slower ARM Cortex-A7 cores and four faster ARM Cortex-A15 cores [2]. Using a big.LITTLE platform, energy can be saved under low-load situations by utilizing only slower cores, while still accommodating high-load situations, when necessary, by utilizing faster cores.

A big.LITTLE platform is an example of a *uniform heterogeneous multiprocessor*, wherein processors (conceptually) differ only with respect to processing speed. In this paper, we consider the real-time scheduling of data-processing streams on such a multiprocessor (hereafter, all references to "heterogeneous" should be taken to mean "uniform heterogeneous"). Specifically, we consider workloads in which each stream is processed by a single sporadic task, the data items processed in one step are handled by invoking the corresponding task once, and consecutive invocations, or *jobs*, of the same task have no precedence requirements and may execute in parallel if one such job is still unfinished when a later such job is released. (As explained later in Sec. 6, this model can be generalized so that multiple processing steps are handled by a sequence of tasks in a *pipelined* fashion.) We call the resulting task model the *npc-sporadic task model* (for "no precedence constraints"). As illustrated in Fig. 1, the npc-sporadic task model is different from the conventional sporadic model, which requires successive jobs of the same task to execute in sequence. In fact, as seen later, under the npc-sporadic model, an individual task may overutilize any single processor; this is not feasible under the conventional model if response times must be bounded. Our specific contribution lies in showing how to *optimally* schedule npc-sporadic task systems (even with pipelining) on a heterogeneous multiprocessor, where the real-time constraint of interest is that job response times are (analytically) upper-bounded. Before continuing, we examine two example use-case scenarios in which the elimination of intra-task precedence constraints is useful.

Example 1: Obstacle detection in automotive systems. Detecting and tracking objects (potential obstacles) is a fundamental part of automotive computer vision. While the computations for frame-to-frame tracking require the output of detectors to have been completed, the actual per-frame detectors may be run independently as long as their results are available in time to be used in frame-to-frame tracking computations. In some use cases (*e.g.*, detecting pedestrians in the road instead of tracking them), frame-to-frame computations may not even be required. In an automotive system, desired response times might be specified, for example, to reflect the reaction time of an alert driver, which

*Work supported by NSF grants CNS 1016954, CNS 1115284, CNS 1218693, and CNS 1239135, AFOSR grant FA9550-14-1-0161, and a grant from General Motors Corp.



(a) job releases of τ_1

(b) schedule for τ_1, if it is a conventional sporadic task

(c) schedule for τ_1, if it is an npc-sporadic task

Figure 1: Example showing the differences between conventional sporadic tasks and npc-sporadic tasks. τ_1 is either a conventional sporadic task (Fig. 1(b)) or an npc-sporadic task (Fig. 1(c)) with a minimum job-release separation of 2 time units, and each job $\tau_{1,j}$ needing 4 time units to complete its execution. We consider scheduling τ_1 on two identical CPUs.

is around 700 ms [13]. In the automotive industry, heterogeneous platforms are often used for energy and cost reasons.

Example 2: Video coding. Video decoding can be realized by defining a task that is invoked each time a frame of video is processed. Dependencies among frames are common, so such a task is often modeled as a conventional sporadic task with intra-task precedence constraints. However, I-frames, or *instantaneous decoder refresh* (IDR) frames, are not subject to such dependencies. Based on this observation, Ahmed *et al.* [3] showed that greater parallelism in video decoding can be achieved by defining tasks on a *group of pictures* (GOP) basis instead of a per-frame basis, where each GOP starts with an I-frame and contains all of the reference frames needed to process the entire GOP. Each task invocation processes an entire GOP, and as long as the response time of each such invocation is bounded, bounded buffers can be employed to display decoded frames in the proper sequence. Note that requiring each GOP to start with an I-frame eliminates intra-task dependencies. Such dependencies can be eliminated in video encoding as well [17].

Related work. Having motivated the usefulness of sometimes eliminating intra-task dependencies, we briefly review prior related work, before describing our major contributions more carefully. In our work, we assume that tasks are scheduled by a *global* multiprocessor scheduler that schedules tasks from a single run queue. In work on scheduling conventional sporadic task systems on homogeneous multiprocessors, a variety of global algorithms have been shown to be *optimal* with respect to ensuring per-task response time bounds (*i.e.*, such bounds can be ensured whenever possible) [5, 6, 7, 9, 15]. In all of this work, response-time bounds are actually derived by bounding *deadline tardiness* (or, the extent to which deadlines can be missed). Under the "bounded tardiness" definition of *soft real-time* (SRT) correctness [5], these algorithms are therefore *SRT-optimal*. Several of the just-cited papers focus on a particular global scheduler, *global earliest-deadline-first* (GEDF), which priotitizes jobs in earliest-deadline-first (EDF) order. GEDF-based algorithms are our focus as well. In other work on homogeneous multiprocessors, Erickson et al. [8] showed that GEDF retains the property of being SRT-optimal for npc-sporadic task systems. In this paper, we extend this result to heterogeneous multiprocessors.

It follows from work of Funk *et al.* [11] that the feasibility condition for ensuring response-time bounds for sporadic tasks in the heterogeneous case differs from the homogeneous case. Technically, Funk presented a condition for avoiding deadline misses in hard real-time (HRT) task systems, but such a condition implies that response times are bounded as well. Subsequently, a number of authors presented algorithms and analysis pertaining to HRT heterogeneous systems (we omit citations due to space constraints). To the best of our knowledge, the Level Algorithm [14] is the only prior algorithm for heterogeneous systems known to be optimal in any sense, but it is impractical to implement because it preempts and migrates tasks excessively.

Contributions. We consider the scheduling of npc-sporadic task systems on a heterogeneous multiprocessor under two GEDF-based algorithms, which we call *F-P-GEDF* (full-migration preemptive GEDF) and *N-P-GEDF* (non-preemptive GEDF), respectively. We show that both of these algorithms guarantee bounded job response times for any feasible npc-sporadic task system. Interestingly, the feasibility condition we establish for npc-sporadic tasks differs from that established by Funk for sporadic tasks. Of the two algorithms we consider, F-P-GEDF is more greedy in executing jobs on faster processors, and therefore ensures a better response-time bound; in contrast, N-P-GEDF only ensures a looser response-time bound, but does not migrate jobs away from slower processors when faster processors becomes available; this characteristic could lead to energy savings.

To the best of our knowledge, this is the first paper to consider the scheduling of npc-sporadic task systems on heterogeneous multiprocessors, the first to establish a feasibility condition for ensuring SRT-optimality (*i.e.*, guaranteed bounded response times), and the first to present scheduling algorithms that are SRT-optimal.

Organization. In the rest of the paper, we describe the considered platforms and task systems more carefully (Sec. 2), present proof-specific preliminaries (Sec. 3), present our response-time-bound proofs (Secs. 4 and 5), further elaborate on stream-processing use cases (Sec. 6), present an experimental evaluation (Sec. 7), and conclude (Sec. 8).

2 Model

We consider *uniform* heterogeneous multiprocessors, as defined in [12]. m is the number of processors. Processor i, where $1 \leq i \leq m$, has an associated *speed* denoted by a real number s_i, which represents the amount of work that can be done on this processor within one time unit. We also identify each processor by its speed and assume the processors are decreasingly ordered by their speeds, *i.e.*, we denote a uniform heterogeneous multiprocessor by $\pi = \{s_1, s_2, \ldots, s_m\}$ where $s_i \geq s_{i+1}$ for $i = 1, 2, \ldots, m-1$. The cumulative speed of the i fastest processors is $S_i = \sum_{k=1}^{i} s_k$. Also, we assume time is continuous.

A *task* is a sequential piece of code and a *job* is an instance (or an invocation) of a task. We let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote the task set to be scheduled. In the widely studied sporadic task model, each task τ_i is characterized by (C_i, T_i, D_i), where C_i is τ_i's *worst-case execution time* (WCET)[1] on a unit-speed processor, T_i is its *period* (the minimum separation of any two jobs), and D_i is its *relative deadline*. $\tau_{i,j}$ is the j^{th} job of τ_i. The release time of $\tau_{i,j}$ is $r_{i,j}$ and its absolute deadline is $d_{i,j}$, where $d_{i,j} = r_{i,j} + D_i$.

On a uniform heterogeneous multiprocessor, C_i, called the *worst-case execution requirement* of τ_i, is defined relative to a processor of speed 1.0. Thus, the WCET of τ_i when entirely executing on processor s_p is $\frac{C_i}{s_p}$ $(1 \leq p \leq m)$. We let $C_{max} = \max\{1 \leq i \leq n \mid C_i\}$. Note that, in the heterogeneous case, the *execution requirement* of a job may differ from its *execution time*. In prior work on identical multiprocessors, the two are the same, since speeds are usually normalized to 1.0. The *utilization* of a task τ_i is $u_i = \frac{C_i}{T_i}$. The total utilization of τ is $U_\tau = \sum_{i=1}^{n} u_i$. We also use τ to refer to the set of all jobs generated by tasks in τ.

Like in [8], we consider npc-sporadic tasks, which have no intra-task precedence constraints. The main difference between the conventional sporadic task model and the npc-sporadic task model is that the former requires successive jobs of each task to execute in sequence while the latter allows them to execute in parallel. That is, in the conventional sporadic task model, job $\tau_{i,j+1}$ cannot commence execution until its predecessor $\tau_{i,j}$ completes, even if $r_{i,j+1}$, the release time of $\tau_{i,j+1}$, has elapsed; in contrast, in npc-sporadic task model, any job can execute as soon as it is released. Additionally, in the npc-sporadic model, a task is allowed to have a utilization greater than the fastest processor's speed. Note that, although we allow intra-task parallelism, each individual job still must execute sequentially.

Feasibility conditions. In [11], a feasibility condition for conventional HRT implicit-deadline (*i.e.*, $D_i = T_i$) *periodic* task systems on uniform heterogeneous multiprocessors is derived. Let U_i denote the sum of the largest i utilizations in τ and assume the number of tasks n is at least the number of processors m. Then, an implicit-deadline periodic task set τ is feasible on a uniform heterogeneous multiprocessor π if and only if

$$U_i \leq S_i, \text{ for } i = 1, 2, \cdots, m-1, \tag{1}$$

and

$$U_\tau \leq S_m. \tag{2}$$

The proof in [11] actually shows this is also a necessary and sufficient feasibility condition for conventional implicit-deadline *sporadic* task systems in *both* HRT and SRT senses. The former requires all deadlines to be met while the latter only requires response times to be bounded.

In npc-sporadic task model, where intra-task precedence constraints are eliminated, it is clear that (1) and (2) are also a necessary and sufficient feasibility condition for HRT implicit-deadline task systems, since in such systems, every job has to complete before its successor is released and hence there is no difference between conventional sporadic task systems and npc-sporadic ones. However, for SRT systems, (1) is not required, as we will show that response-time bounds do not rely on (1). Furthermore, for arbitrary-deadline tasks, our analysis applies as well. On the other hand, (2) is always required. A violation of (2) means the system is overutilized, and therefore response times will increase unboundedly. To summarize, an npc-sporadic task system τ is feasible, or guarantees bounded response times, on π if and only if (2) holds.

3 Preliminaries

Def. 1. At a time instant t, a job $\tau_{i,j}$ is *unreleased* if $t < r_{i,j}$, *pending* if $t \geq r_{i,j}$ and $\tau_{i,j}$ has not completed execution by t, and *complete* if $\tau_{i,j}$ has completed by t.

Def. 2. We let $\mathsf{A}(\mathcal{R}, \tau_{i,j}, t_1, t_2)$ denote the cumulative processor capacity allocation to job $\tau_{i,j}$ in an arbitrary schedule $\mathcal{R}$ (for "real") within the time interval $[t_1, t_2]$.

Also, we let $\mathsf{A}(\mathcal{R}, \mathcal{J}, t_1, t_2)$ denote the cumulative processor capacity allocation to the jobs in job set $\mathcal{J}$ in an arbitrary schedule $\mathcal{R}$ within the time interval $[t_1, t_2]$, *i.e.*,

$$\mathsf{A}(\mathcal{R}, \mathcal{J}, t_1, t_2) = \sum_{\tau_{i,j} \in \mathcal{J}} \mathsf{A}(\mathcal{R}, \tau_{i,j}, t_1, t_2). \tag{3}$$

Ideal schedule. We let $\pi_{\text{IDEAL}} = \{u_1, u_2, \ldots, u_n\}$ denote an ideal multiprocessor for the task set τ, where π_{IDEAL} consists of n processors with speeds that exactly match the utilizations of the n tasks in τ, respectively. Let $\mathcal{I}$ be the partitioned schedule for τ on π_{IDEAL}, where each task τ_i in τ is assigned to the processor of speed u_i. Then, in $\mathcal{I}$, every job in τ commences execution at its release time and completes execution within one period (it exactly executes for one period if and only if its actual execution requirement matches its worst-case execution requirement).

Thus, we have, $\mathsf{A}(\mathcal{I}, \tau_{i,j}, t_1, t_2) \leq u_i \cdot (t_2 - t_1)$, and for an arbitrary job set $\mathcal{J} \subseteq \tau$,

$$\mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) \leq U_\tau \cdot (t_2 - t_1). \tag{4}$$

[1]Techniques from [16] can be applied to instead provision tasks on an average-case basis; we do not consider such techniques further due to space constraints.

This is similar to the processor sharing (PS) schedule considered in prior work [5]. However, the above notion of an ideal schedule is preferable in the heterogeneous case, because it prevents a single job from executing in parallel with itself.

Note that, in the ideal schedule, a constrained-deadline task τ_i (*i.e.*, $D_i < T_i$) may not complete execution at its deadline. The following definition gives an upper bound on the amount of work that completes later than its deadline in the ideal schedule.

Def. 3. Let $L_i = u_i \cdot \max\{0, T_i - D_i\}$, and let $L_\tau = \sum_{i=1}^{n} L_i$. Then, in $\mathcal{I}$, at any time instant t, the amount of incomplete work with deadline at or before t is at most L_τ.

Def. 4. We denote the difference between the allocation to a job $\tau_{i,j}$ in $\mathcal{I}$ and in a schedule $\mathcal{R}$ within $[0, t]$ as

$$\mathsf{lag}(\tau_{i,j}, t, \mathcal{R}) = \mathsf{A}(\mathcal{I}, \tau_{i,j}, 0, t) - \mathsf{A}(\mathcal{R}, \tau_{i,j}, 0, t), \quad (5)$$

and such an allocation difference for an arbitrary job set $\mathcal{J}$ is

$$\mathsf{LAG}(\mathcal{J}, t, \mathcal{R}) = \sum_{\tau_{i,j} \in \mathcal{J}} \mathsf{lag}(\tau_{i,j}, t, \mathcal{R}). \quad (6)$$

By (5) and Def. 2, for any time interval $[t_1, t_2]$ we have

$$\mathsf{lag}(\tau_{i,j}, t_2, \mathcal{R}) = \mathsf{lag}(\tau_{i,j}, t_1, \mathcal{R}) + \mathsf{A}(\mathcal{I}, \tau_{i,j}, t_1, t_2) - \mathsf{A}(\mathcal{R}, \tau_{i,j}, t_1, t_2); \quad (7)$$

and by (3), (6), and (7),

$$\mathsf{LAG}(\mathcal{J}, t_2, \mathcal{R}) = \mathsf{LAG}(\mathcal{J}, t_1, \mathcal{R}) + \mathsf{A}(\mathcal{I}, \mathcal{J}, t_1, t_2) - \mathsf{A}(\mathcal{R}, \mathcal{J}, t_1, t_2). \quad (8)$$

Lemma 1. *If a job $\tau_{i,j}$ is unreleased or complete at t, then $\mathsf{lag}(\tau_{i,j}, t, \mathcal{R}) \leq 0$; if $\tau_{i,j}$ is pending at t, then $\mathsf{lag}(\tau_{i,j}, t, \mathcal{R}) \leq C_i$.*

Proof. Follows immediately from Defs. 1 and 4. □

Job of interest. To derive response-time bounds, we consider an arbitrary job $\tau_{k,l}$ in τ, and upper bound its response time. Let t_d be the absolute deadline of $\tau_{k,l}$, *i.e.*, $t_d = d_{k,l}$.

Def. 5. In the rest of this paper, we let Ψ be the job set consisting of all jobs with deadlines at or before t_d. The jobs in Ψ are called *competing jobs* for $\tau_{k,l}$. At time instant t, the total incomplete work due to all jobs in Ψ is called *competing work* at t.

Def. 6. If at a time instant t, all of the m processors are executing jobs in Ψ, then t is a *busy instant* for Ψ; otherwise t is a *non-busy instant* for Ψ. If in the time interval $[t_1, t_2]$ every time instant is a busy instant for Ψ, then $[t_1, t_2]$ is a *busy interval* for Ψ.

Lemma 2. *If in $\mathcal{R}$, $[t_1, t_2]$ is a busy interval for Ψ, then $\mathsf{LAG}(\Psi, t_1, \mathcal{R}) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{R})$.*

Proof. Follows from the previous definitions. □

Def. 7. For any time instant t, we let t^+ denote the time instant $(t + \epsilon)$ and we let t^- denote the time instant $(t - \epsilon)$, where $\epsilon \to 0^+$.

4 Response-Time Bounds under F-P-GEDF

The F-P-GEDF scheduler works as follows.

- At any time instant, if there are at most m pending jobs, then all of them are scheduled; if there are more than m pending jobs, then the m such jobs with the earliest deadlines are scheduled. Deadline ties are broken arbitrarily.

- For any two jobs $\tau_{i,j}$ scheduled on processor s_p and $\tau_{a,b}$ scheduled on processor s_q, where $p < q$, we have $d_{i,j} \leq d_{a,b}$ (note that, given how we order processors, $s_p \geq s_q$).

In this section, we let $\mathcal{R}$ denote an F-P-GEDF schedule of τ on π.

4.1 Basic Bounds

We first present a proof to derive basic response-time bounds. This proof is more similar to the SRT analysis framework for GEDF in [5], and therefore is easier to understand. We will improve the basic bounds proved here in Sec. 4.2. Recall that $\tau_{k,l}$ is the analyzed job.

Lemma 3. *At any non-busy instant t at or before t_d, $\mathsf{LAG}(\Psi, t, \mathcal{R}) \leq (m-1) \cdot C_{max}$.*

Proof. We decompose Ψ into three disjoint subsets: Ψ_1, Ψ_2, and Ψ_3 consisting of jobs that are *unreleased*, *pending*, and *complete*, respectively. Since in the npc-sporadic task model intra-task precedence constraints are removed, under F-P-GEDF, there can be at most $(m-1)$ jobs in Ψ that are pending at t, *i.e.*, $|\Psi_2| \leq m - 1$. Thus,

$$\begin{aligned}
&\mathsf{LAG}(\Psi, \mathcal{R}, t)\\
&= \{\text{by the definition of } \Psi_1, \Psi_2, \text{ and } \Psi_3\}\\
&\quad \mathsf{LAG}(\Psi_1, t, \mathcal{R}) + \mathsf{LAG}(\Psi_2, t, \mathcal{R}) + \mathsf{LAG}(\Psi_3, t, \mathcal{R})\\
&= \{\text{ by (6) }\}\\
&\quad \sum_{\tau_{i,j} \in \Psi_1} \mathsf{lag}(\tau_{i,j}, t, \mathcal{R}) + \sum_{\tau_{i,j} \in \Psi_2} \mathsf{lag}(\tau_{i,j}, t, \mathcal{R}) + \sum_{\tau_{i,j} \in \Psi_3} \mathsf{lag}(\tau_{i,j}, t, \mathcal{R})\\
&\leq \{\text{ by Lemma 1 }\}\\
&\quad \sum_{\tau_{i,j} \in \Psi_1} 0 + \sum_{\tau_{i,j} \in \Psi_2} C_i + \sum_{\tau_{i,j} \in \Psi_3} 0\\
&\leq |\Psi_2| \cdot C_{max}\\
&\leq (m-1) \cdot C_{max}.
\end{aligned}$$

□

Lemma 4. *After t_d, once $\tau_{k,l}$ executes, it will continuously execute until it completes.*

Proof. After t_d, no job with deadline earlier than t_d can be released, *i.e.*, no job that can preempt $\tau_{k,l}$ can be released. Thus, once $\tau_{k,l}$ executes, it will continually execute until it completes, though it could migrate among processors. □

Lemma 5. *In $\mathcal{R}$, the competing work for $\tau_{k,l}$ at t_d is at most $L_\tau + (m-1) \cdot C_{max}$.*

Proof. By Defs. 3 and 4, the competing work pending at t_d in $\mathcal{R}$ is at most $L_\tau + \mathsf{LAG}(\Psi, \mathcal{R}, t_d)$. Let t' be the latest non-busy instant at or before t_d (or time 0 if no such non-busy instant exists). Then, by Lemma 2, $\mathsf{LAG}(\Psi, t', \mathcal{R}) \geq \mathsf{LAG}(\Psi, t_d, \mathcal{R})$. Also, by Lemma 3, $\mathsf{LAG}(\Psi, t', \mathcal{R}) \leq (m-1) \cdot C_{max}$. Thus, $\mathsf{LAG}(\Psi, t_d, \mathcal{R}) \leq (m-1) \cdot C_{max}$ and therefore the lemma follows. □

Lemma 6. *Let W be the competing work for $\tau_{k,l}$ at t_d. Then the job of interest, $\tau_{k,l}$, will complete execution no later than time*

$$t_d + \frac{W - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. Suppose that $\tau_{k,l}$ is not complete at or before t_d. Let δ be the amount of work of $\tau_{k,l}$ that has been completed by t_d and $e_{k,l}$ be the real execution requirement of $\tau_{k,l}$. Then the remaining execution work of $\tau_{k,l}$ at t_d is $e_{k,l} - \delta$. If $\tau_{k,l}$ does not execute within $[t_d, t_d + \frac{W-(e_{k,l}-\delta)}{S_m})$, then $[t_d, t_d + \frac{W-(e_{k,l}-\delta)}{S_m})$ must be a busy interval for Ψ. In this case, the competing work that is completed within $[t_d, t_d + \frac{W-(e_{k,l}-\delta)}{S_m})$ is $W - (e_{k,l} - \delta)$ (since within a busy interval, all processors execute competing work and the total speed is S_m), and the remaining competing work at $t_d + \frac{W-(e_{k,l}-\delta)}{S_m}$ is $e_{k,l} - \delta$, which must be totally due to $\tau_{k,l}$. Therefore, $\tau_{k,l}$ will execute at time $t_d + \frac{W-(e_{k,l}-\delta)}{S_m}$. Thus, if $\tau_{k,l}$ is not complete at or before t_d, then the latest time when $\tau_{k,l}$ commences execution after t_d is $t_d + \frac{W-(e_{k,l}-\delta)}{S_m}$. By Lemma 4, $\tau_{k,l}$ will not be preempted once it executes after t_d. Also, since the minimum execution speed is s_m, $\tau_{k,l}$ will complete within $\frac{e_{k,l}-\delta}{s_m}$ time units. Therefore, $\tau_{k,l}$ will complete by

$$\begin{aligned}
& t_d + \frac{W - (e_{k,l} - \delta)}{S_m} + \frac{e_{k,l} - \delta}{s_m} \\
= & \{\text{rearranging}\} \\
& t_d + \frac{W}{S_m} - \frac{e_{k,l}}{S_m} + \frac{e_{k,l}}{s_m} + (\frac{\delta}{S_m} - \frac{\delta}{s_m}) \\
\leq & \{\text{since } \delta \geq 0 \text{ and } S_m \geq s_m\} \\
& t_d + \frac{W}{S_m} - \frac{e_{k,l}}{S_m} + \frac{e_{k,l}}{s_m} \\
\leq & \{\text{since } e_{k,l} \leq C_k \text{ and } S_m \geq s_m\} \\
& t_d + \frac{W}{S_m} - \frac{C_k}{S_m} + \frac{C_k}{s_m} \\
= & \, t_d + \frac{W - C_k}{S_m} + \frac{C_k}{s_m}.
\end{aligned}$$

□

Theorem 1. *The response time of an arbitrary job $\tau_{k,l}$ in τ under F-P-GEDF scheduling on π is at most*

$$D_k + \frac{L_\tau + (m-1) \cdot C_{max} - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. Follows from Lemmas 5 and 6. □

4.2 Improved Bounds

We now show that the response-time bound above can be improved in several ways.

First, we can derive a better bound on the LAG at t_d or even an arbitrary time instant t by considering *LAG non-increasing intervals* instead of busy intervals.

Def. 8. We introduce an integer Λ such that $S_{\Lambda-1} < U_\tau$ and $S_\Lambda \geq U_\tau$ $(1 \leq \Lambda \leq m)$. If at time instant t, at least Λ processors are executing jobs in Ψ, then t is a *LAG non-increasing instant*. If in the time interval $[t_1, t_2]$ every time instant is a LAG non-increasing instant, then $[t_1, t_2]$ is a *LAG non-increasing interval* for Ψ.

By the rules of F-P-GEDF, it is clear that at any time instant, if p processors $(1 \leq p \leq m)$ execute jobs in Ψ, then they must be the p fastest ones. This property ensures that LAG for Ψ cannot increase within a LAG non-increasing interval. Then, we can derive following lemma.

Lemma 7. *For any time instant t, $\mathsf{LAG}(\Psi, t, \mathcal{R}) \leq (\Lambda - 1) \cdot C_{max}$.*

Proof. This proof is similar to Lemma 5. We instead consider the latest time instant that is not a LAG non-increasing instant at or before t. The definition of LAG non-increasing instant and the property in the prior paragraph ensure counterparts for Lemmas 3 and 2, respectively. Thus, the lemma follows. □

Furthermore, in Sec. 4.1, we only considered the execution of $\tau_{k,l}$ after t_d, and we pessimistically assumed $\tau_{k,l}$ is executed at the minimum speed s_m. Actually, we can consider the execution of $\tau_{k,l}$ as early as it is released, and derive several linear constraints, and solve a corresponding linear program. The following lemma shows this.

Def. 9. As defined in prior work [10, 11, 12], the *identicalness* of the multiprocessor platform π is

$$\lambda = \max_{1 \leq i \leq m-1} \left\{ \frac{s_{i+1} + s_{i+2} + \cdots + s_m}{s_i} \right\}.$$

That is,

$$\lambda = \max_{1 \leq i \leq m-1} \left\{ \frac{S_m - S_i}{s_i} \right\}. \tag{9}$$

Note that, $\lambda \leq m-1$. Also, $\lambda = m-1$ if and only if π is an identical multiprocessor.

Lemma 8. *Suppose the competing work for $\tau_{k,l}$ at $r_{k,l}$ is W. Then the response time of $\tau_{k,l}$ is upper bounded by*

$$\frac{W}{S_m} + \frac{\lambda}{S_m} C_k.$$

Proof. In the time interval between $r_{k,l}$ and $\tau_{k,l}$'s completion, let x_0 denote the cumulative time in which $\tau_{k,l}$ is not executing, and let x_i $(1 \le i \le m)$ denote the cumulative time in which $\tau_{k,l}$ is executing on processor s_i. Then, the response time of $\tau_{k,l}$ is $\sum_{i=0}^{m} x_i$.

Since we cannot execute $\tau_{k,l}$ for more than its worst-case execution requirement, we have the linear constraint

$$\sum_{i=1}^{m} s_i x_i \le C_k.$$

By the rules of F-P-GEDF, after $r_{k,l}$, when $\tau_{k,l}$ is not complete and is not currently executing, all of the m processors must execute jobs in Ψ; and when $\tau_{k,l}$ is executing on processor s_i, the fastest i processors, *i.e.*, s_1 to s_i, must execute jobs in Ψ. Since W is the competing work at $r_{k,l}$, the execution of jobs in Ψ after $r_{k,l}$ cannot exceed W. Therefore, we have the linear constraint

$$S_m x_0 + \sum_{i=1}^{m} S_i x_i \le W.$$

We now manually solve this linear programming problem by the Simplex Algorithm [4], assuming that C_k, W, s_i, and S_i $(1 \le i \le m)$ are constants and each x_i $(0 \le i \le m)$ is a variable. To do so, we introduce two auxiliary variables, x_{m+1} and x_{m+2}, to rewrite this problem in slack form. Specifically, we maximize

$$z = \sum_{i=0}^{m} x_i,$$

subject to

$$\begin{cases} x_{m+1} = C_k - \sum_{i=1}^{m} s_i x_i, \\ x_{m+2} = W - S_m x_0 - \sum_{i=1}^{m} S_i x_i, \\ x_0, x_1, x_2, \cdots, x_{m+2} \ge 0. \end{cases}$$

First, we pivot x_0 with x_{m+2}. Then, in the resulting program, we pivot x_h, where h satisfies $\frac{S_m - S_h}{s_h} = \max_{1 \le i \le m}\{\frac{S_m - S_i}{s_i}\} = \lambda$, with x_{m+1}. Due to space constraint, we omit the details here. The final program is to maximize

$$z = \sum_{1 \le i \le m \,\wedge\, i \ne h} \left(\left(\frac{S_m - S_i}{s_i} \right) - \left(\frac{S_m - S_h}{s_h} \right) \right) \frac{s_i}{S_m} x_i - \left(1 - \frac{S_h}{S_m}\right) \frac{x_{m+1}}{s_h} - \frac{x_{m+2}}{S_m} + \frac{W}{S_m} + \left(1 - \frac{S_h}{S_m}\right) \frac{C_k}{s_h}, \tag{10}$$

subject to

$$\begin{cases} x_h = \frac{C_k}{s_h} - \sum_{1 \le i \le m \,\wedge\, i \ne h} \frac{s_i}{s_h} x_i - \frac{x_{m+1}}{s_h}, \\ x_0 = \frac{W}{S_m} - \sum_{1 \le i \le m \,\wedge\, i \ne h} \frac{S_i}{S_m} x_i - \frac{x_{m+2}}{S_m} \\ \quad - \frac{S_h}{S_m} \left(\frac{C_k}{s_h} - \sum_{1 \le i \le m \,\wedge\, i \ne h} \frac{s_i}{s_h} x_i - \frac{x_{m+1}}{s_h} \right), \\ x_0, x_1, x_2, \cdots, x_{m+2} \ge 0. \end{cases}$$

By the definition of h, all the coefficients of the x terms of z in (10) are negative or zero. Therefore, when $x_h = \frac{C_k}{s_h}$, $x_0 = (\frac{W}{S_m} - \frac{S_h}{S_m} \cdot \frac{C_k}{s_h})$, and $x_i = 0$ (for all $i \ne 0$ and $i \ne h$), z has its maximum value, which is

$$\begin{aligned} z_{max} &= \frac{W}{S_m} + \frac{S_m - S_h}{S_m s_h} C_k \\ &= \{\text{by (9) and by the definition of } h\} \\ &\quad \frac{W}{S_m} + \frac{\lambda}{S_m} C_k. \end{aligned}$$

Thus, the lemma follows. □

We now upper bound the competing work for $\tau_{k,l}$ at $r_{k,l}$ by the following lemma.

Lemma 9. *The competing work for $\tau_{k,l}$ at $r_{k,l}$ is at most*

$$U_\tau \cdot D_k + L_\tau + (\Lambda - 1) \cdot C_{max}.$$

Proof. By Defs. 3, 4, and 5, the competing work for $\tau_{k,l}$ at $r_{k,l}$ is at most

$$\begin{aligned} & L_\tau + \mathsf{A}(\mathcal{I}, \Psi, 0, t_d) - \mathsf{A}(\mathcal{R}, \Psi, 0, r_{k,l}) \\ = & \{\text{by Def. 2}\} \\ & L_\tau + \mathsf{A}(\mathcal{I}, \Psi, 0, r_{k,l}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,l}, t_d) - \mathsf{A}(\mathcal{R}, \Psi, 0, r_{k,l}) \\ = & \{\text{by (5) and (6)}\} \\ & \mathsf{A}(\mathcal{I}, \Psi, r_{k,l}, t_d) + L_\tau + \mathsf{LAG}(\Psi, r_{k,l}, \mathcal{R}) \\ \le & \{\text{by (4)}\} \\ & U_\tau \cdot (t_d - r_{k,l}) + L_\tau + \mathsf{LAG}(\Psi, r_{k,l}, \mathcal{R}) \\ = & \{\text{since } t_d = d_{k,l} = r_{k,l} + D_k \text{ and by Lemma 7}\} \\ & U_\tau \cdot D_k + L_\tau + (\Lambda - 1) \cdot C_{max}. \end{aligned}$$

□

Theorem 2. *The response time of an arbitrary job $\tau_{k,l}$ in τ under F-P-GEDF scheduling on π is at most*

$$\frac{U_\tau}{S_m} \cdot D_k + \frac{1}{S_m} \cdot L_\tau + \frac{(\Lambda - 1)}{S_m} \cdot C_{max} + \frac{\lambda}{S_m} C_k.$$

Proof. Follows from Lemmas 8 and 9. □

5 Response-Time Bounds under N-P-GEDF

The N-P-GEDF scheduler is similar to the F-P-GEDF scheduler, except that once a job is selected for execution, it runs to completion without preemption or migration. Also, we do not require faster processors to be favored when scheduling jobs; instead, we can always favor slower ones for energy efficiency (if desired). In this section, we let $\mathcal{R}$ be the N-P-GEDF schedule of τ on π.

Def. 10. Suppose time instant t is a non-busy time instant for Ψ. If every pending job in Ψ is currently executing, then t is a *non-blocking non-busy instant* for Ψ; otherwise (*i.e.*, some pending job in Ψ is blocked by jobs that are not in Ψ), t is a *blocking non-busy instant* for Ψ. If in a time interval $[t_1, t_2]$ every time instant is a blocking non-busy instant for Ψ, then $[t_1, t_2]$ is a *blocking non-busy interval* for Ψ.

Def. 11. The set of jobs that are not in Ψ but currently executing in schedule $\mathcal{R}$ at time instant t is denoted $\mathcal{B}(t)$ and the incomplete work of jobs in $\mathcal{B}(t)$ is denoted $\mathsf{B}(t)$, called *blocking work*.

It is clear that $|\mathcal{B}(t)| \leq m$ for all t, and therefore $\mathsf{B}(t) \leq m \cdot C_{max}$ for all t.

5.1 Basic Bounds

As before, we first derive basic response-time bounds. We will improve the basic bounds in Sec. 5.2.

Lemma 10. *At any non-blocking non-busy instant t, $\mathsf{LAG}(\Psi, \mathcal{R}, t) + \mathsf{B}(t) \leq m \cdot C_{max}$.*

Proof. Let $b = |\mathcal{B}(t)|$ $(0 \leq b \leq m)$. Then $\mathsf{B}(t) \leq b \cdot C_{max}$, and by the definition of a non-blocking non-busy instant, the number of pending jobs in Ψ is at most $m - b$. Similarly to Lemma 3, we have $\mathsf{LAG}(\Psi, \mathcal{R}, t) \leq (m - b) \cdot C_{max}$. Thus, $\mathsf{LAG}(\Psi, \mathcal{R}, t) + \mathsf{B}(t) \leq (m-b) \cdot C_{max} + b \cdot C_{max} = m \cdot C_{max}$. $\square$

Lemma 11. *If $[t_1, t_2]$ is a busy interval for Ψ in $\mathcal{R}$, then $\mathsf{LAG}(\Psi, t_1, \mathcal{R}) + \mathsf{B}(t_1) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{R}) + \mathsf{B}(t_2)$.*

Proof. Since $t_1, t_2 \in [t_1, t_2]$ are busy instants for Ψ, $\mathcal{B}(t_1) = \mathcal{B}(t_2) = \emptyset$ and therefore $\mathsf{B}(t_1) = \mathsf{B}(t_2) = 0$. Also, by Lemma 2, $\mathsf{LAG}(\Psi, t_1, \mathcal{R}) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{R})$. Thus, $\mathsf{LAG}(\Psi, t_1, \mathcal{R}) + \mathsf{B}(t_1) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{R}) + \mathsf{B}(t_2)$. $\square$

Lemma 12. *If $[t_1, t_2]$ is a blocking non-busy interval for Ψ in $\mathcal{R}$, then any blocking job (i.e., any job that is not in Ψ but executing at some time instant t in $[t_1, t_2]$), must execute continuously in $[t_1^-, t]$.*

Proof. Because $[t_1, t_2]$ is a blocking non-busy interval for Ψ, at any time instant within $[t_1, t_2]$, there is at least one job in Ψ that is pending but not executing. Also, since any job in Ψ has an earlier deadline, or higher priority, than any job not in Ψ, no job that is not in Ψ and not executing at t_1^- can execute in $[t_1, t_2]$. Thus, the lemma follows. $\square$

Lemma 13. *If $[t_1, t_2]$ is a blocking non-busy interval for Ψ in $\mathcal{R}$, then $\mathsf{LAG}(\Psi, t_1, \mathcal{R}) + \mathsf{B}(t_1) \geq \mathsf{LAG}(\Psi, t_2, \mathcal{R}) + \mathsf{B}(t_2)$.*

Proof. Let $[t, t']$ be a subinterval in $[t_1, t_2]$ such that $|\mathcal{B}(t)| = |\mathcal{B}(t')|$. By Lemma 12, the blocking jobs at every time instant in $[t, t']$ are exactly the jobs in $\mathcal{B}(t)$. Let $\mathcal{P}$ denote the set of processors on which those blocking jobs execute in $[t, t']$. Then,

$$\mathsf{B}(t') = \mathsf{B}(t) - \sum_{s_i \in \mathcal{P}} s_i \cdot (t' - t).$$

Since $[t, t'] \subseteq [t_1, t_2]$ is a blocking non-busy interval, the processors not in $\mathcal{P}$ must execute jobs in Ψ; otherwise, it would be a non-blocking non-busy interval. Therefore,

$$\begin{aligned}
&\mathsf{LAG}(\Psi, t', \mathcal{R}) \\
&= \mathsf{LAG}(\Psi, t, \mathcal{R}) + \mathsf{A}(\mathcal{I}, \Psi, t, t') - \mathsf{A}(\mathcal{R}, \Psi, t, t') \\
&\leq \mathsf{LAG}(\Psi, t, \mathcal{R}) + U_\tau \cdot (t' - t) - \sum_{s_i \notin \mathcal{P}} s_i \cdot (t' - t).
\end{aligned}$$

Thus,

$$\begin{aligned}
&\mathsf{LAG}(\Psi, t', \mathcal{R}) + \mathsf{B}(t') \\
&\leq \mathsf{LAG}(\Psi, t, \mathcal{R}) + U_\tau \cdot (t' - t) - \sum_{s_i \notin \mathcal{P}} s_i \cdot (t' - t) + \\
&\qquad \mathsf{B}(t) - \sum_{s_i \in \mathcal{P}} s_i \cdot (t' - t) \\
&= \mathsf{LAG}(\Psi, t, \mathcal{R}) + \mathsf{B}(t) + U_\tau \cdot (t' - t) - \\
&\qquad (\sum_{s_i \notin \mathcal{P}} s_i + \sum_{s_i \in \mathcal{P}} s_i) \cdot (t' - t) \\
&= \mathsf{LAG}(\Psi, t, \mathcal{R}) + \mathsf{B}(t) + U_\tau \cdot (t' - t) - S_m \cdot (t' - t) \\
&\leq \{\text{since } U_\tau \leq S_m\} \\
&\quad \mathsf{LAG}(\Psi, t, \mathcal{R}) + \mathsf{B}(t).
\end{aligned}$$

That is, for every such subinterval $[t, t'] \subseteq [t_1, t_2]$, we have

$$\mathsf{LAG}(\Psi, t, \mathcal{R}) + \mathsf{B}(t) \geq \mathsf{LAG}(\Psi, t', \mathcal{R}) + \mathsf{B}(t').$$

By induction, the lemma follows. $\square$

Lemma 14. *In $\mathcal{R}$, the competing work for $\tau_{k,l}$ plus the blocking work at t_d is at most $L_\tau + m \cdot C_{max}$.*

Proof. Similarly to Lemma 5, the competing work pending at t_d is at most $L_\tau + \mathsf{LAG}(\Psi, \mathcal{R}, t_d)$. Also, the blocking work at t_d is $\mathsf{B}(t_d)$, so the competing work for $\tau_{i,j}$ plus the blocking work at t_d is at most $L_\tau + \mathsf{LAG}(\Psi, \mathcal{R}, t_d) + \mathsf{B}(t_d)$.

Let t' be the latest non-blocking non-busy instant at or before t_d (or time 0 if no such non-blocking non-busy instant exists). Then by Lemma 10,

$$\mathsf{LAG}(\Psi, \mathcal{R}, t') + \mathsf{B}(t') \leq m \cdot C_{max}.$$

Moreover, by the definition of t', $[t'^+, t_d]$ consists of busy intervals and/or blocking non-busy intervals. Therefore, by Lemmas 11 and 13,

$$\mathsf{LAG}(\Psi, \mathcal{R}, t') + \mathsf{B}(t') \geq \mathsf{LAG}(\Psi, \mathcal{R}, t_d) + \mathsf{B}(t_d).$$

Thus, $\mathsf{LAG}(\Psi, \mathcal{R}, t_d) + \mathsf{B}(t_d) \leq m \cdot C_{max}$ and the lemma follows. $\square$

Lemma 15. *Let W be the competing work plus the block-*

ing work for $\tau_{k,l}$ at t_d. Then the job of interest, $\tau_{k,l}$, will complete execution no later than time

$$t_d + \frac{W - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. The proof of this lemma is exactly the same as Lemma 6. □

Theorem 3. *The response time of an arbitrary job $\tau_{k,l}$ in τ under N-P-GEDF scheduling on π is at most*

$$D_k + \frac{L_\tau + m \cdot C_{max} - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. Follows from Lemmas 14 and 15. □

5.2 Improved Bounds

We now show that the response-time bound for N-P-GEDF can also be improved. However, in contrast to the situation in Sec. 4.2, we still have to pessimistically assume the job of interest executes entirely at the minimum speed s_m, since in N-P-GEDF, a scheduled job cannot migrate among processors. Nevertheless, we can still consider the execution of $\tau_{k,l}$ before t_d.

The following three lemmas are similar to Lemmas 4, 14, and 15.

Lemma 16. *Under N-P-GEDF, once $\tau_{k,l}$ executes, it will continuously execute until it completes.*

Proof. Follows from the non-preemptive property. □

Lemma 17. *For any time instant t at or before t_d, $\mathsf{LAG}(\Psi, \mathcal{R}, t) + \mathsf{B}(t) \le m \cdot C_{max}$.*

Proof. This proof is exactly the same as that for upper bounding $\mathsf{LAG}(\Psi, \mathcal{R}, t_d) + \mathsf{B}(t_d)$ in Lemma 14. □

Lemma 18. *Let W be the competing work plus the blocking work for $\tau_{k,l}$ at $r_{k,l}$. Then the job of interest, $\tau_{k,l}$, will complete execution no later than time*

$$r_{k,l} + \frac{W - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. By Lemma 16, this proof is exactly the same as Lemma 15. □

We now upper bound the competing work plus the blocking work for $\tau_{k,l}$ at $r_{k,l}$ by the following lemma.

Lemma 19. *The competing work plus the blocking work for $\tau_{k,l}$ at $r_{k,l}$ is at most*

$$U_\tau \cdot D_k + L_\tau + m \cdot C_{max}.$$

Proof. By Defs. 3, 4, 5, and 11, the competing work plus the blocking work for $\tau_{k,l}$ at $r_{k,l}$ is at most

$$\begin{aligned}
&L_\tau + \mathsf{A}(\mathcal{I}, \Psi, 0, t_d) - \mathsf{A}(\mathcal{R}, \Psi, 0, r_{k,l}) + \mathsf{B}(r_{k,l}) \\
= &\{\text{by Def. 2}\} \\
&L_\tau + \mathsf{A}(\mathcal{I}, \Psi, 0, r_{k,l}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,l}, t_d) \\
&\quad - \mathsf{A}(\mathcal{R}, \Psi, 0, r_{k,l}) + \mathsf{B}(r_{k,l}) \\
= &\{\text{by (5) and (6)}\} \\
&L_\tau + \mathsf{LAG}(\Psi, \mathcal{R}, r_{k,l}) + \mathsf{A}(\mathcal{I}, \Psi, r_{k,l}, t_d) + \mathsf{B}(r_{k,l}) \\
\le &\{\text{by (4)}\} \\
&U_\tau \cdot (t_d - r_{k,l}) + L_\tau + \mathsf{LAG}(\Psi, \mathcal{R}, r_{k,l}) + \mathsf{B}(r_{k,l}) \\
\le &\{\text{since } t_d = d_{k,l} = r_{k,l} + D_k \text{ and by Lemma 17}\} \\
&U_\tau \cdot D_k + L_\tau + m \cdot C_{max}.
\end{aligned}$$

□

Theorem 4. *The response time of an arbitrary job $\tau_{k,l}$ in τ under N-P-GEDF scheduling on π is at most*

$$\frac{U_\tau}{S_m} \cdot D_k + \frac{L_\tau + m \cdot C_{max} - C_k}{S_m} + \frac{C_k}{s_m}.$$

Proof. Follows from Lemmas 18 and 19. □

6 Relevance to Stream Processing Systems

By eliminating intra-task precedence constraints, we have shown that both F-P-GEDF and N-P-GEDF can guarantee bounded response times in processing data streams, when it is possible to do so, provided successive task invocations are independent. Note that, for those systems where the output of processed data needs to be in order, we can use an ordered buffer to store output. Furthermore, our derived response-time bounds can be used to determine buffer sizes.

Our npc-sporadic task model can be generalized to optimally support stream processing in a pipelined fashion. In this case, we model each stage of the pipeline as an npc-sporadic task; all such tasks have a common period that depends on the stream. Within a stage, the processing due to successive invocations must be independent, *i.e.*, no precedence constraints are required. Each stage usually does depend on data output from the stage before it, but such a dependency can be eliminated by increasing the phase of each stage to match our derived response-time bounds. Fig. 2 shows this.

7 Evaluation

We evaluated the proposed algorithms and derived response-time bounds by randomly generating task sets and then calculating response-time bounds for each task on certain selected platforms.

In the case of homogeneous multiprocessors, the considered platform is implicitly determined by the number of processors. However, for uniform heterogeneous multiprocessors, even if given the number of processors, there are an infinite number of speed combinations to consider. Therefore,

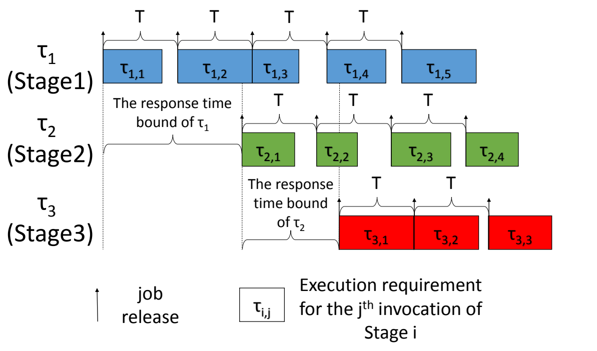
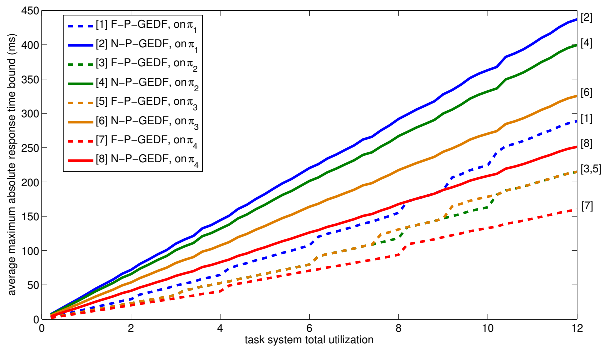

T
T
T
T
τ1 (Stage1)
τ1,1
τ1,2
τ1,3
τ1,4
τ1,5
The response time bound of τ1
τ2 (Stage2)
τ2,1
τ2,2
τ2,3
τ2,4
The response time bound of τ2
τ3 (Stage3)
τ3,1
τ3,2
τ3,3
job release
τi,j
Execution requirement for the jth invocation of Stage i

Figure 2: Pipelining example, where T is the period of the data stream.

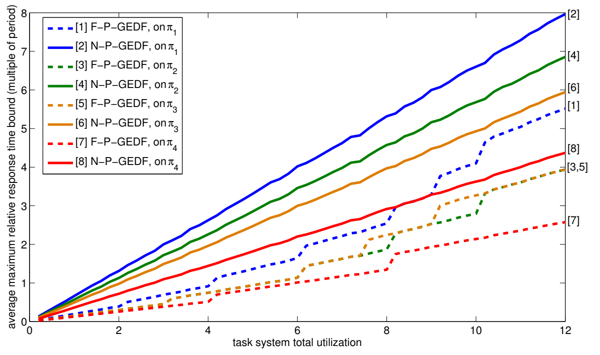
there is no way to systematically choose platforms to evaluate by varying the number of processors. Thus, we chose the following four multiprocessors as representatives of different heterogeneous multiprocessors in terms of both processor number and speed combination: $\pi_1 = \{2, 2, 2, 2, 1, 1, 1, 1\}$, $\pi_2 = \{3, 3, 2, 2, 1, 1\}$, $\pi_3 = \{3, 3, 1.5, 1.5, 1.5, 1.5\}$, and $\pi_4 = \{4, 4, 2, 2\}$. We could also normalize the slowest processor of the latter two platforms to be 1.0; however, we chose instead to scale all four platforms to have the same total processor capacity to enable comparisons among different platforms.

In our experiments, we assumed implicit deadlines for simplicity, *i.e.*, relative deadlines are equal to periods ($D_i = T_i$). For a given total utilization cap, we generated a task set by first randomly selecting its task count uniformly over $[1, 20]$. Note that we allow the number of tasks to be less than the number of processors. Next, we randomly assigned a *relative utilization* or *weight*, by uniformly generating a number in $(0, 1]$, for each task. We then scaled the relative utilizations to obtain real utilizations by letting the total utilization match the pre-set cap. Note that, by the scaling step, we may generate tasks with a utilization greater than the fastest processor's speed; that is allowed in our task model and analysis. Since all of the four considered platforms have a total capacity of 12, we varied task-set utilization caps in $[0, 12]$ by increments of 0.2. For each given total utilization cap, we generated 10,000 task sets. The period for each task was selected uniformly within $[10\text{ ms}, 100\text{ ms}]$; its worst-case execution requirement was then determined based on its utilization and period.

Response-time bounds. We evaluated response-time bounds in terms of both absolute values and relative values. The former are directly computed by Theorems 2 and 4; the latter are defined by the ratio of a task's absolute response-time bound and its period. We focus here on the maximum response-time bound for each task set. Fig. 3(a) shows absolute response-time bound results while Fig. 3(b) shows the relative response-time bound results. Each data point in each figure is the average of the maximum response-time bounds (absolute or relative) of the 10,000 generated task sets for a given total utilization.



(a) Absolute response-time bounds

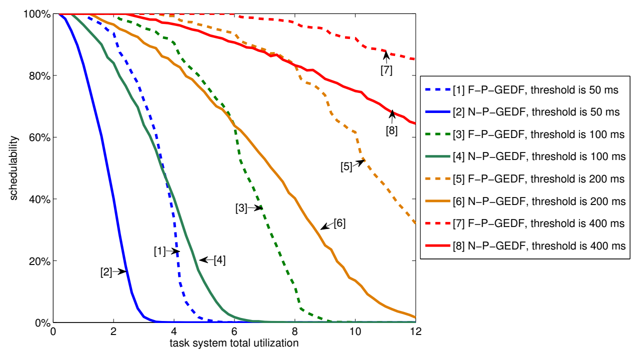


(b) Relative response-time bounds

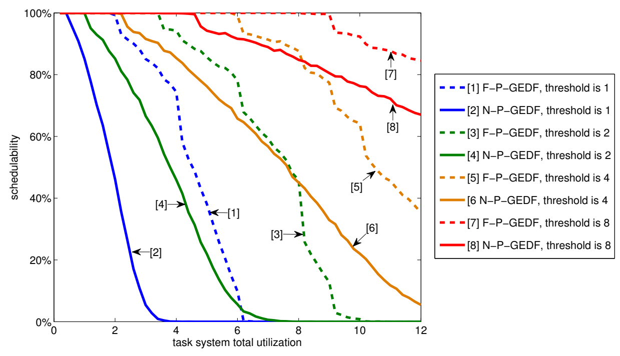
Figure 3: Average maximum response-time bounds.

The resulting absolute and relative response-time bounds are under 450 ms and 8 periods, respectively. Generally, the lower the total utilization cap, the better the bounds; given a total processor capacity, having fewer processors with faster speeds yields better bounds.

Schedulability. Since in our analysis we do not have any utilization constraints except for that the total task system utilization is at most the total processor capacity, we have shown that both F-P-GEDF and N-P-GEDF are SRT-optimal, *i.e.*, for any feasible system, either algorithm can ensure a bounded response time for every job. However, many systems require not only that response times are bounded, but also that they do not exceed a certain threshold. For this reason, we also evaluated F-P-GEDF and N-P-GEDF with respect to response-time thresholds. Again, we considered both absolute response times and relative response times. For the former, we considered thresholds of 50 ms, 100 ms, 200 ms, and 400 ms; for the latter, we considered thresholds of 1, 2, 4, and 8. In all cases, we assessed *schedulability* by determining the fraction of the randomly generated task systems for which given response-time requirement could be met. Due to space constraints, we only show results for platform π_1. Fig. 4(a) shows the schedulability on π_1 in terms of absolute-response-time thresholds while Fig. 4(b) shows the schedulability on π_1 in terms of relative-response-time thresholds. When the threshold is



(a) Absolute response-time thresholds



(b) Relative response-time thresholds

Figure 4: Schedulability in terms of response-time thresholds.

400 ms (or 8 periods for relative response times), even in the fully utilized case, the achieved schedulability was around 80%. Overall, the lower the total utilization, the better the schedulability.

8 Conclusion

We have considered two GEDF-based schedulers, F-P-GEDF and N-P-GEDF, on uniform heterogeneous multiprocessors in the absence of intra-task precedence constraints. Both ensure bounded job response times for npc-sporadic tasks as long as the underlying multiprocessor platform is not overutilized. To the best of our knowledge, this is the first paper to consider such tasks in the context of heterogeneous multiprocessor platforms.

It follows from our work that, on such platforms, different feasibility conditions apply for HRT and SRT npc-sporadic systems (where "SRT" is interpreted to mean that, although response times are bounded, deadlines can be missed). This stands in contrast to the conventional sporadic task model. For the npc-sporadic model, the HRT feasibility condition is the same as that for the conventional sporadic task model; however, the SRT feasibility condition for the npc-sporadic model merely requires that the system is not overutilized, in contrast to the more complicated condition for the sporadic case that requires (1). Note that both F-P-GEDF and N-P-GEDF are SRT-optimal for scheduling npc-sporadic task systems.

F-P-GEDF is more greedy in executing jobs on faster processors and hence has a better response-time bound, at the expense of potentially greater preemption and migration frequencies. On the other hand, N-P-GEDF does not preempt or migrate jobs, but its guaranteed response-time bounds are relatively higher. Our analysis for N-P-GEDF applies even under the scheduling rule that, when multiple processors are available to a job, the slowest one is chosen to execute that job. This may yield benefits from an energy point of view.

References

[1] big.LITTLE Processing. http://www.arm.com/products/processors/technologies/biglittleprocessing.php

[2] Samsung's Exynos 5422 & The Ideal big.LITTLE: Exynos 5 Hexa (5260). http://www.anandtech.com/show/7811/samsungs-exynos-5422-the-ideal-biglittle-exynos-5-hexa-5260

[3] I. Ahmad, Y. He, and M. Liou. Video compression with parallel processing. *Parallel Computing*, 28(7):1039-1078, 2002.

[4] G. Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[5] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. *Real-Time Sys.*, 38(2):133-189, 2008.

[6] J. Erickson, U. Devi, and S. Baruah. Improved tardiness bounds for global EDF. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 14-23, 2010.

[7] J. Erickson, N. Guan, and S. Baruah. Tardiness bounds for global EDF with deadlines different from periods. In *Proceedings of the 14th International Conference On Principles Of Distributed Systems (OPODIS)*, pp. 286-301, 2010.

[8] J. Erickson and J. Anderson. Response time bounds for G-EDF without intra-task precedence constraints. In *Proceedings of the 15th International Conference On Principles Of Distributed Systems (OPODIS)*, pp. 128-142, 2011.

[9] J. Erickson and J. Anderson. Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 3-11, 2012.

[10] S. Funk and S. Baruah. Characteristics of EDF schedulability on uniform multiprocessors. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS)*, pp. 211-218, 2003.

[11] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS)*, pp. 183-192, 2001.

[12] S. Funk. *EDF scheduling on heterogeneous multiprocessors*. PhD thesis, University of North Carolina, NC, 2004.

[13] M. Green. "How long does it take to stop?" Methodological analysis of driver perception-brake times. *Transportation Human Factors*, 2(3):195-216, 2000.

[14] E. Horvath, S. Lam, and R. Sethi. A level algorithm for preemptive scheduling. *Journal of the ACM*, 24(1):32-43, 1977.

[15] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Sys.*, 44(1):26-71, 2010.

[16] A. Mills and J. Anderson. A multiprocessor server-based scheduler for soft real-time tasks with stochastic execution demand. In *Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 207-217, 2011.

[17] J. Nang and J. Kim. An Effective parallelizing scheme of MPEG-1 video encoding on ethernet-connected workstations. In *Advances in Parallel and Distributed Computing*, 1997:4-11, 1997.