IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

Real-Time Scheduling with Resource Sharing on Uniform Multiprocessors

Gurulingesh Raravi

Vincent Nélis

Björn Andersson

HURRAY-TR-120902

Version:

Date: 09-10-2012

Real-Time Scheduling with Resource Sharing on Uniform Multiprocessors

Gurulingesh Raravi, Vincent Nélis, Björn Andersson

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

Consider the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a uniform multiprocessor platform where each task may access at most one of |R| shared resources and at most once by each job of that task. The resources have to be accessed in a mutually exclusive manner. We propose an algorithm, GIS-vpr, which offers the guarantee that if a task set is schedulable to meet deadlines by an optimal task assignment scheme that allows a task to migrate only when it accesses or releases a resource, then our algorithm also meets the deadlines with the same restriction on the task migration, if given processors 4 + 6|R| times as fast. The proposed algorithm, by design, limits the number of migrations per job to at most two. To the best of our knowledge, this is the first result for resource sharing on uniform multiprocessors with proven performance guarantee.

Real-Time Scheduling with Resource Sharing on Uniform Multiprocessors

Gurulingesh Raravi
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Porto, Portugal
ghri@isep.ipp.pt

Vincent Nélis
CISTER/INESC-TEC, ISEP,
Polytechnic Institute of Porto,
Porto, Portugal
nelis@isep.ipp.pt

Björn Andersson
Software Engineering Institute,
Carnegie Mellon University,
Pittsburgh, USA
baandersson@sei.cmu.edu

ABSTRACT

Consider the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a uniform multiprocessor platform where each task may access at most one of ρ shared resources and at most once by each job of that task. The resources have to be accessed in a mutually exclusive manner. We propose an algorithm, GIS-vpr, which offers the guarantee that if a task set is schedulable to meet deadlines by an optimal task assignment scheme that allows a task to migrate only when it accesses or releases a resource, then our algorithm also meets the deadlines with the same restriction on the task migration, if given processors $4 + 6\rho$ times as fast. The proposed algorithm, by design, limits the number of migrations per job to at most two. To the best of our knowledge, this is the first result for resource sharing on uniform multiprocessors with proven performance guarantee.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Organization and Design—*Real-time systems and embedded systems*; G.4 [**Mathematical Software**]: Algorithm design and analysis

General Terms

Theory

Keywords

real-time scheduling, resource sharing, uniform multiprocessors

1. INTRODUCTION

Computers are often used in applications where they interact with the physical world (for example, software in an autopilot ensures that airplane stays at the right altitude). Hence, at run-time, the software must finish computations at the right time; such systems are referred to as *real-time systems*.

A real-time software system is often modeled as a set of tasks where each task generates a (potentially infinite) sequence of jobs. Each job of a task may arrive at any time once a minimum inter-arrival time has elapsed since the arrival of the previous job of the same task. Each job has an execution time and a deadline within which it has to complete its execution. Tasks typically share a processor but in many computer systems, tasks also share other resources such as data structures, sensors, etc. and such tasks must be operated in a *mutually exclusive* manner while accessing the resource. Even on a uniprocessor platform, the sharing of such resources can have a profound effect on timing behavior as witnessed by the close-to-failure of the NASA mission Mars Pathfinder because the resource-sharing protocol in the operating system was not enabled [12]. Scheduling real-time tasks that share resources on a *multiprocessor* platform is more complex. Our goal in this work is to design an algorithm for scheduling the tasks that share resources on uniform multiprocessors so as to meet all the deadlines and to prove its performance.

Commonly, the performance of a scheduling algorithm is characterized using the notion of *utilization bound* [13]. This metric has been used to evaluate the scheduling algorithms on uniprocessor (e.g., [13]) and *identical* multiprocessors (e.g., [1]) where the speeds of all the processors are same. However, it does not translate to algorithms (for scheduling tasks that share resources) on *uniform* multiprocessors where processors are characterized by different speeds, hence we rely on the *resource augmentation* framework [15] to characterize the performance of the algorithm under design. We say that an algorithm A has a *speed competitive ratio* SCR_A if, for every real-time task set for which it is possible to meet the deadlines, it holds that A meets the deadlines as well if, the speed of each processor is multiplied by SCR_A.

A low speed competitive ratio indicates high performance; ideally it should be one. A scheduling algorithm with a *finite* speed competitive ratio is desirable as well because it can ensure the designer that deadlines will be met by using faster processors. Consequently, the real-time systems community has embraced the development of scheduling algorithms with finite speed competitive ratio, e.g., [3, 4, 8]. Unfortunately, the community has not yet developed a multiprocessor scheduling algorithm with finite speed competitive ratio for tasks that share resources on *uniform* multiprocessors. Therefore, in this paper, we present one and

RTNS'12, November 08–09 2012, Pont á Mousson, France

prove its performance.

Problem Statement: We consider the problem of scheduling implicit-deadline sporadic tasks (in which the deadline of a task is equal to its minimum inter-arrival time) that share resources on uniform multiprocessors. We assume that each task may request at most one resource (known at design time) and at most once by each job of that task.

Related Work: The problem of scheduling real-time tasks that share resources has been studied in the past for identical multiprocessors (e.g., [7, 14, 16]). However, none of these algorithms has a proven speed competitive ratio. In a recent significant development, Andersson et al. [2] proposed an algorithm for scheduling tasks that share resources on identical multiprocessors with a speed competitive ratio of $12 \times \frac{1+3\rho}{4m}$ where m and ρ denote the number of processors and resources, respectively. Later, Raravi et al. [17] proposed an algorithm for scheduling tasks that share resources on *heterogeneous* multiprocessors *with two distinct kinds of processors* and proved that its speed competitive ratio is $4 + 6 \times \left\lceil \frac{\rho}{\min(m_1,m_2)} \right\rceil$ where m_1 and m_2 denote the number of processors of first and second kind, respectively[1]. Since the uniform multiprocessor platform is *not* a special case of the heterogeneous multiprocessor platform with *two* distinct kinds of processors, *the result of [17] does not trivially translate to the problem under consideration.*

Contributions and Significance of this work: The recent result by Andersson et al. [2] for the problem of resource sharing on identical multiprocessors raised the following question: is it possible to design an algorithm for scheduling tasks that share resources on uniform multiprocessors with a finite speed competitive ratio. In this work, we answer this question in the affirmative by designing an algorithm, GIS-vpr, using some of the techniques discussed in [2].

The algorithm, GIS-vpr, offers the guarantee that if a task set is schedulable to meet deadlines by an optimal task assignment scheme that allows task migrations when it accesses or releases a resource, then our algorithm also meets deadlines with the same restriction on task migrations, if given processors $4 + 6\rho$ times as fast. It also ensures that the number of migrations per job is at most two.

We believe that the significance of this work is two-fold. First, for the problem of scheduling tasks that share resources on uniform multiprocessors, no previous algorithm exists and hence our algorithm is the first for this problem with a finite speed competitive ratio. Second, we hope that that this work will inspire some research towards designing new algorithms for the problem under consideration with a better speed competitive ratio or for a more generic resource sharing model where tasks can access multiple resources and/or every job of these tasks can access the resources more than once.

The remainder of this paper is organized as follows. Section 2 describes the task, resource and platform model that we are considering in this work. Section 3 gives an overview of the proposed algorithm. The algorithm is described in Section 4 and its performance in terms of speed competitive ratio is proven in Section 5. Finally, Section 6 concludes with a discussion on few interesting properties of the proposed algorithm.

[1]The resource sharing model considered in [2, 17] is *same* as in this work, i.e., each task can access at most one resource and at most once by each job of that task.

2. SYSTEM MODEL AND ASSUMPTIONS

We consider the problem of scheduling a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n implicit-deadline sporadic tasks that share a set of ρ resources $R = \{r_1, r_2, \ldots, r_\rho\}$ on a uniform multiprocessor platform $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of m processors.

Each task $\tau_i \in \tau$ is characterized by three parameters: a *worst-case execution time* C_i, a *period* T_i and a deadline D_i (which is equal to its period T_i). Each task τ_i releases a (potentially infinite) sequence of *jobs*, with the first job released at any time during the system execution and subsequent jobs released *at least* T_i time units apart. Each job released by a task τ_i has to complete its execution within D_i time units from its release. We allow *preemptive scheduling* of tasks.

In the computing platform, a processor $\pi_p \in \pi$ is of speed s_p where $1 \leq p \leq m$; for ease of explanation, we consider that processors are indexed such that $s_1 \leq s_2 \leq \ldots \leq s_m$. If, for a time interval of duration L and for a processor π_p of speed s_p, a job executes during the entire time interval on π_p then the job performs $L \times s_p$ units of execution during this time interval.

The set of shared resources comprising ρ resources that tasks need in addition to the m processors is denoted by $R = \{r_1, r_2, \ldots, r_\rho\}$. We assume that each task may access at most one shared resource from R, and further each job of that task may request the resource at most once during its execution. The resource (if any) used by the task τ_i is denoted by $r(i)$, i.e., $r(i) = r_k$ or $r(i) = \phi$, where $k \in [1, \rho]$.

For convenience, we use the following notations. The *utilization* of the task τ_i is denoted by u_i and is defined as $u_i \stackrel{def}{=} \frac{C_i}{T_i}$. It is easy to see that if a job of task τ_i executes only on processor π_p of speed s_p then it performs its C_i units of execution by executing for $\frac{C_i}{s_p}$ time units. For this reason, we define $C_{i,p} \stackrel{def}{=} \frac{C_i}{s_p}$. Analogously, we define $u_{i,p} \stackrel{def}{=} \frac{C_i}{s_p \times T_i}$.

Also, we make use of *constrained-deadline* tasks — the deadline of such a task is less than or equal to its minimum inter-arrival time (i.e., $D_i \leq T_i$). For a constrained-deadline task τ_i, its *density* is denoted as δ_i and is defined by $\delta_i \stackrel{def}{=} \frac{C_i}{\min(D_i,T_i)} = \frac{C_i}{D_i}$. Similar to $u_{i,p}$, we define $\delta_{i,p}$ as $\delta_{i,p} \stackrel{def}{=} \frac{C_i}{s_p \times D_i}$.

Finally, we assume that a job cannot execute in parallel, i.e., it cannot run on two or more processors simultaneously.

3. OVERVIEW OF OUR ALGORITHM

The proposed algorithm, GIS-vpr, relies on the concept of *virtual processors* to design the solution for the problem under consideration. Virtual processors are logical constructs, used as task assignment targets by our algorithm. A virtual processor acts equivalent to a physical processor with speed $\frac{1}{f}$ and we assume that it can be "emulated" on a physical processor of speed 1, using no more than $\frac{1}{f}$ of its processing capacity. One intuitive way of achieving this is by dividing time into short slots of length S and using $\frac{1}{f} \times S$ time units in each slot to serve the workload of virtual processor. By selecting S, we can then make the speed of the emulated pro-

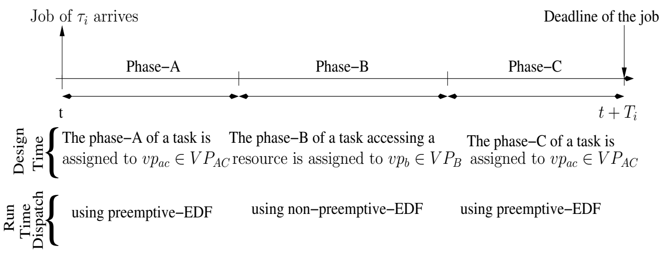

Figure 1: Three execution phases of a job along with the design-time and run-time decisions of GIS-vpr algorithm.

cessor arbitrarily close to $\frac{1}{f}$ (and in practice, S need rarely be impractically short) [6].

The algorithm, GIS-vpr, can be summarized in four steps. Steps 1-3 are executed at *design time* and only step 4 needs to be executed at *run time*.

Step 1–Creation of subtasks. Split the task execution into three *phases* as shown in Figure 1 — in essence create three constrained-deadline subtasks out of each implicit-deadline sporadic task and make different scheduling provisions for each of them.

- **phase-A:** the task has arrived and not yet made a request for the resource
- **phase-B:** the task has requested the resource and is either using (also referred to as *holding*) it or waiting for it and
- **phase-C:** the task has released the resource and yet to finish its execution

A task which does not access any of the shared resources is split into only phase-A.

The "arrival" of both phase-B and phase-C subtasks have fixed offsets from the arrival of the respective phase-A subtask. This guarantees that the subtasks have the same inter-arrival time as the original task thereby exhibiting no jitter in their arrival times. The manner in which these constrained-deadline subtasks are created and their parameters, i.e., worst-case execution time, period and deadline are determined is described in Section 4.1.

Step 2–Creation of virtual processors. Create two sets of virtual processors, namely, $\mathrm{VP_{AC}}$ and $\mathrm{VP_B}$ virtual processors from the given physical processors. The specification of the virtual processors and their creation is discussed in Section 4.2.

Step 3–Task assignment. The phase-A and phase-C subtasks created from each task are assigned to the same virtual processor in $\mathrm{VP_{AC}}$, whereas the corresponding phase-B subtask is assigned to a virtual processor in $\mathrm{VP_B}$. This step is discussed in Section 4.3.

Step 4–Task scheduling. All the phase-A and phase-C subtasks are scheduled using *preemptive* Earliest-Deadline-First (EDF) algorithm [13] on their assigned virtual processors in $\mathrm{VP_{AC}}$. All the phase-B subtasks are scheduled using *non-preemptive* EDF algorithm on their assigned virtual processors in $\mathrm{VP_B}$ to ensure mutual exclusion while accessing the shared resources.

Subtasks of τ_i	WCET	Deadline	Period
$\tau_\mathbf{i}^\mathbf{A}$	C_i^A	$D_i^A = \frac{C_i^A}{C_i} \times \frac{T_i}{2}$	$T_i^A = T_i$
$\tau_\mathbf{i}^\mathbf{B}$	C_i^B	$D_i^B = \frac{T_i}{2}$	$T_i^B = T_i$
$\tau_\mathbf{i}^\mathbf{C}$	C_i^C	$D_i^C = \frac{C_i^C}{C_i} \times \frac{T_i}{2}$	$T_i^C = T_i$

Table 1: The three constrained-deadline subtasks that are derived form a given implicit-deadline task τ_i

4. THE NEW ALGORITHM GIS-vpr

In this section, we describe the new algorithm, GIS-vpr, in detail and also provide its pseudo-code.

4.1 Creating the Subtasks

In this section, we describe how the algorithm, GIS-vpr, creates constrained-deadline subtasks, i.e., phase-A, phase-B and phase-C subtasks from the given set of implicit-deadline tasks and how it sets the parameters, i.e., execution time, period and deadline of these derived tasks.

From each implicit-deadline task $\tau_i \in \tau$, it creates three constrained-deadline subtasks τ_i^A, τ_i^B and τ_i^C corresponding to phase-A, phase-B and phase-C of the execution of τ_i, respectively. In the rest of the paper, the superscript A, B and C will be used in the notations corresponding to phase-A, phase-B and phase-C subtasks, respectively. For example, C_i^A, C_i^B and C_i^C denote the worst-case execution time of task $\tau_i \in \tau$ before accessing the resource $r(i)$ (phase-A), while holding the resource $r(i)$ (phase-B) and after releasing $r(i)$ (phase-C), respectively. Note that $\forall \tau_i \in \tau : C_i = C_i^A + C_i^B + C_i^C$.

The parameters of the three subtasks τ_i^A, τ_i^B and τ_i^C that are derived from each $\tau_i \in \tau$ are set as shown in Table 1. Note that $D_i^A + D_i^B + D_i^C \leq T_i = D_i$. This is essential as it ensures that if the subtasks, τ_i^A, τ_i^B and τ_i^C derived from τ_i meet their deadlines then the original task τ_i meets its deadline as well. Finally, we group these derived subtasks into the following task sets:

$$\tau^A = \{\tau_i^A \mid i \in [1, n]\} \quad (1)$$

$$\tau^{B,r_k} = \{\tau_i^B \mid i \in [1, n] \text{ and } r(i) = r_k\} \quad (2)$$

$$\tau^C = \{\tau_i^C \mid i \in [1, n]\} \quad (3)$$

As opposed to the given task set τ which contains implicit-deadline tasks, these derived task sets contain constrained-deadline tasks. Also, observe that the task set τ^A is derived such that the density of every subtask $\tau_i^A \in \tau^A$ is twice the utilization of the corresponding task $\tau_i \in \tau$. Formally,

$$\forall \tau_i^A \in \tau^A : \quad \delta_i^A = \frac{C_i^A}{D_i^A} = \frac{C_i^A}{\frac{C_i^A \times T_i}{2C_i}} = \frac{2C_i}{T_i} = 2u_i \quad (4)$$

Analogously,

$$\forall \tau_i^C \in \tau^C : \quad \delta_i^C = \frac{C_i^C}{D_i^C} = \frac{C_i^C}{\frac{C_i^C \times T_i}{2C_i}} = \frac{2C_i}{T_i} = 2u_i$$

Also, note that the execution requirements and densities of the derived subtasks are with respect to a processor of speed 1. On a processor π_p of speed s_p, these terms must be divided by s_p, i.e.,

$$\forall \tau_i^A \in \tau^A : C_{i,p}^A = \frac{C_i^A}{s_p}, \delta_{i,p}^A = \frac{\delta_i^A}{s_p} \quad (5)$$

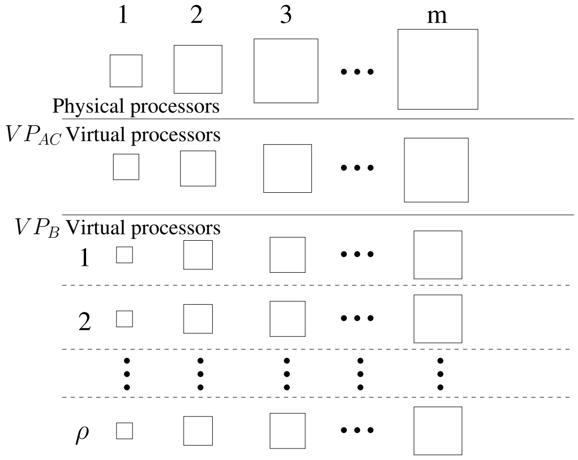

Figure 2: Creation of $m(1+\rho)$ virtual processors from m physical processors of a uniform multiprocessor platform.

In contrast, the periods and deadlines of these subtasks are independent of the processor speed. The same holds for each task $\tau_i^C \in \tau^C$ and $\tau_i^B \in \tau^{B,r_k}$, $\forall r_k \in R$.

4.2 Dimensioning the Virtual Processors on Uniform Multiprocessor Platform

In this section, we describe the creation of virtual processors from the given physical processors of a uniform multiprocessor computing platform.

We create $m(1+\rho)$ virtual processors from the given m physical processors. Precisely, we create the virtual processors with following specifications:

- **m virtual processors (denoted as $\mathrm{VP_{AC}}$):** From each physical processor π_p of speed s_p, we create one virtual processor of speed $s_p \times \frac{2}{2+3\rho}$. So, in total, m such virtual processors are created from m physical processors. These are used to schedule phase-A and phase-C subtasks and are referred to as '$\mathrm{VP_{AC}}$ virtual processors'.
- **$m \times \rho$ virtual processors (denoted as $\mathrm{VP_B}$):** From each physical processor π_p of speed s_p, we create ρ virtual processors of speed $s_p \times \frac{3}{2+3\rho}$. So, in total, $m \times \rho$ such virtual processors are created from m physical processors. These are used to schedule phase-B subtasks and are referred to as '$\mathrm{VP_B}$ virtual processors'.

In other words, from each physical processor π_p, we create $1+\rho$ virtual processors, i.e., one $\mathrm{VP_{AC}}$ and ρ $\mathrm{VP_B}$ virtual processors as shown by each column in Figure 2. Observe that no virtual processor is created using more than one physical processor, i.e., the capacity of a virtual processor comes only from one physical processor.

We now show that, from one physical processor π_p of speed s_p, it is indeed possible to create one $\mathrm{VP_{AC}}$ and ρ $\mathrm{VP_B}$ virtual processors as per the specifications given earlier and hence the given uniform multiprocessor platform π can be dimensioned accordingly to obtain the above specified set of virtual processors.

LEMMA 1. *The given uniform multiprocessor platform π can be dimensioned as mentioned above to obtain the set of virtual processors,* $\mathrm{VP_{AC}}$ *and* $\mathrm{VP_B}$.

Algorithm 1: GIS-vpr(τ, π, R): for scheduling tasks that share resources on uniform multiprocessors

---

`// Lines 1-9 execute offline; line 10 executes at run time.`

1 Create the sets τ^A, τ^{B,r_k} and τ^C of subtasks from the given task set τ as described in Section 4.1;
2 Create $\mathrm{VP_{AC}}$ and $\mathrm{VP_B}$ virtual processors from the given set π of processors as described in Section 4.2;
3 Assign all the subtasks $\tau_i^A \in \tau^A$ to the $\mathrm{VP_{AC}}$ virtual processors using the algorithm GIS (see [10] for details);
4 **foreach** $\tau_i \in \tau$ **do**
5   **if** τ_i *requests a resource* r_k *(i.e.,* $r(i) = r_k$*)* **then**
6   Assign τ_i^B to the k'th virtual processor created from the m'th (i.e., the fastest) physical processor;
7   **end**
8 **end**
9 Assign every subtask $\tau_i^C \in \tau^C$ to that virtual processor in $\mathrm{VP_{AC}}$ to which the corresponding subtask $\tau_i^A \in \tau^A$ has been assigned on line 3;
10 Schedule (i) all the subtasks of τ^A and τ^C on $\mathrm{VP_{AC}}$ virtual processors using *preemptive* EDF and (ii) all the subtasks of τ_i^B on $\mathrm{VP_B}$ virtual processor using *non-preemptive* EDF;

PROOF. The proof is a direct consequence of the fact that each physical processor can emulate its associated $\mathrm{VP_{AC}}$ virtual processor and its ρ $\mathrm{VP_B}$ virtual processors, as per the specifications of the virtual processors. Indeed, for each $\pi_p \in \pi$, we have

$$\underbrace{1 \times (s_p \times \frac{2}{2+3\rho})}_{\mathrm{VP_{AC}}\text{ virtual processor}} + \underbrace{\rho \times (s_p \times \frac{3}{2+3\rho})}_{\mathrm{VP_B}\text{ virtual processors}} = \frac{2s_p + 3\rho s_p}{2+3\rho} = s_p$$

Hence the proof. □

We now describe the rest of the steps in the algorithm to schedule the tasks that *share the resources* on uniform multiprocessors by providing the pseudo-code.

4.3 Pseudo-code of GIS-vpr

The pseudo-code of GIS-vpr is shown in Algorithm 1. The algorithm works as follows.

On line 1, it creates the sets τ^A, τ^{B,r_k} and τ^C of constrained-deadline subtasks from the given set τ of implicit-deadline tasks as described in Section 4.1.

On line 2, it creates m $\mathrm{VP_{AC}}$ and $m \times \rho$ $\mathrm{VP_B}$ virtual processors from the given m physical processors as discussed in Section 4.2.

On line 3, it assigns the set of phase-A subtasks, τ^A, on $\mathrm{VP_{AC}}$ virtual processors using GIS algorithm. The algorithm, GIS, was proposed by **G**onzalez and **I**barra and **S**ahni [10] for non-migratively scheduling a set of implicit-deadline sporadic tasks that do not share resources on uniform multiprocessors. It has a speed competitive ratio of two. Actually, [10] studied the problem of non-preemptively scheduling non-periodic tasks that do not share resources on uniform multiprocessors for minimizing the *makespan*. It is easily shown that these two problems are equivalent. The abbreviation GIS comes from author names of [10].

On lines 4–8, it assigns all the phase-B subtasks that access the same shared resource to the same $\mathrm{VP_B}$ virtual processor. Specifically, all the subtasks accessing the resource r_k, $\forall k \in [1, \rho]$, are assigned to the k'th $\mathrm{VP_B}$ virtual processor created from the *fastest physical processor*, π_m. This technique serves two purpose. Firstly, it ensures mutual exclusion between the tasks accessing the same resource (as all the subtasks sharing the resource are assigned to the same virtual processor and are executed in a non-preemptive way). Secondly, it minimizes the *blocking time* of a task related to resource sharing by effectively creating the equivalent of a hypothetical single virtual processor whereupon every task would execute as fast as on the fastest processor in the system. The *blocking time* of a task that wants to access a resource is defined as the time duration during which it is blocked by a lower priority task holding that resource.

Observe that GIS-vpr assigns all the phase-B subtasks only to those $\mathrm{VP_B}$ virtual processors that are created from the fastest physical processor π_m, hence leaving all the other $\mathrm{VP_B}$ virtual processors idle. It is therefore natural to think that creating $\mathrm{VP_B}$ virtual processors only from the fastest physical processor might be a good idea. However, it turns out that, from the perspective of speed competitive ratio, it offers no benefit.

Since more than one virtual processors are created from a single physical processor, there might be frequent "context switches" between those virtual processors. Even with $\mathrm{VP_B}$ virtual processors involved in these "context switches", the mutual exclusion property while accessing the resources is not affected. This is because a $\mathrm{VP_B}$ virtual processor can only be preempted by a virtual processor that is either running a task that does *not* access the *same resource* or does not access a resource at all.

On line 9, it assigns every phase-C subtask, τ_i^C, to that virtual processor in $\mathrm{VP_{AC}}$ to which the corresponding phase-A subtask, τ_i^A, has been assigned. Such an assignment does not endanger the schedulability of the tasks assigned on the $\mathrm{VP_{AC}}$ virtual processors as there is a precedence constraint between these subtasks — this is formally proven later in Lemma 9 in Section 5.2. Also, such an assignment ensures that the number of migrations per job is restricted to at most two. This is easy to verify because both phase-A and phase-C of a task execute on the same physical processor as they are assigned to the same virtual processor and only phase-B subtask *might* have to execute on different physical processor as the virtual processor to which phase-B of the task is assigned might have been created from a different physical processor.

On line 10, it schedules the tasks executing in their phase-A and phase-C on $\mathrm{VP_{AC}}$ virtual processors using *preemptive EDF* and tasks executing in their phase-B on $\mathrm{VP_B}$ virtual processors using *non-preemptive EDF*.

The reason for using *preemptive EDF* for scheduling phase-A and phase-C subtasks is that it is an *optimal* uniprocessor scheduling algorithm [9, 13]. EDF is optimal in the sense that it always meets all the deadlines of tasks assigned to a processor if there exists a schedule that meets all the deadlines. The reason for using *non-preemptive EDF* to schedule phase-B subtasks is twofold: (i) its non-preemptive property facilitates in achieving mutual exclusion while accessing the shared resources and (ii) its speed competitive ratio is known [2].

Also, for *preemptive EDF* scheduling, the following result has been shown in [3] (an easily obtained generalization of the result in [13]) which we make use of while proving the performance of GIS-vpr.

LEMMA 2. *(From Theorem 2 in [3]: utilization-based schedulability test)*
Let $\tau[\pi_p]$ denote the tasks assigned on a processor π_p of speed s_p. If $\sum_{\tau_i \in \tau[\pi_p]} u_i \leq s_p$ and tasks are scheduled with preemptive EDF on π_p then all deadlines are met.

Note that in Algorithm 1, lines 1–9 execute at *design time* and only line 10 executes at *run time*.

The algorithm, GIS-vpr, is named after the fact that it makes use of the algorithm, GIS, for assigning some of the subtasks on **v**irtual **pr**ocessors.

5. PERFORMANCE ANALYSIS OF THE ALGORITHM GIS-vpr

In this section, we prove the speed competitive ratio of the proposed algorithm. But first we present some notations and useful results that are used later while proving the performance of GIS-vpr.

5.1 Few Notations

Let π denote a uniform multiprocessor platform of m processors, $\{\pi_1, \pi_2, \ldots, \pi_m\}$. The speed of a processor π_p is s_p. For ease of explanation, we consider that processors are ordered in increasing order of their speed, i.e., $s_1 \leq s_2 \leq \ldots \leq s_m$. Let $\pi \times s$ denote a uniform multiprocessor platform in which the speed of each processor π_p is s times that of the corresponding processor in π. The platform $\pi \times s$ is obtained by multiplying the speed of each processor in platform π by a real number, $s > 0$. We use π_m to denote a uniprocessor platform with speed s_m and $\pi_m \times s$ to denote a uniprocessor platform with speed $s_m \times s$.

Let $\mathrm{sched}(A, \tau, \pi)$ denote a predicate to signify that a task set τ that does not share resources *meets all its deadlines* when scheduled by algorithm A on platform π. The term *meets all its deadlines* in this and other predicates means 'meets deadlines for every possible arrival of tasks that is valid as per the given parameters of τ'.

Let $\mathrm{feas}(\mathrm{nmo}, \tau, \pi)$ signify that there exists a *non-migrative-offline* preemptive schedule which meets all deadlines of tasks in τ that do not share resources on platform π. Here, *non-migrative* schedule (also referred to as *partitioned* schedule) refers to a schedule in which all the jobs of a task execute on the same processor to which the task has been assigned (and hence different jobs of the same task are not allowed to migrate to a different processor). In this and other predicates, the term *offline* schedule refers to a schedule which (i) can contain inserted idle times and (ii) can be generated using the knowledge of future job arrival times (irrespective of whether such knowledge is available in practice).

The predicate $\mathrm{sched}(A, \tau, R, \pi)$ signifies that the task set τ *sharing the resources* from a set R *meets all its deadlines* when scheduled by algorithm A on platform π with *restricted migration*. In this and other predicates, the term (i) 'sharing the resources' has the same meaning as discussed in Section 2 and (ii) 'restricted migration' indicates that a job can only migrate when it accesses or releases the resource. Also, replacing the set R of resources by a single resource r_k in this and other related predicates signifies that the tasks in τ share a single resource r_k, where $1 \le k \le \rho$.

Let feas(rmo, τ, R, π) denote a predicate to signify that there exists a *restricted-migration-offline* preemptive schedule which meets all the deadlines of tasks in τ on platform π when tasks are 'sharing the resources' from R.

Also, some of the above described predicates will be used by adding a suffix -δ (where applicable, i.e., for non-migrative scheduling of constrained-deadline subtasks corresponding to different phases) to the scheduling algorithm (or algorithm class). Such predicates with suffix -δ signify that the *schedulability* of τ other than just being established via some exact test, must additionally be ascertainable via a (potentially pessimistic) *density-based* uniprocessor schedulability test (similar to Lemma 2). That is, for $\tau[\pi_p]$ of tasks assigned on a processor π_p of speed s_p, to meet deadlines, it must hold that $\sum_{\tau_i \in \tau[\pi_p]} \delta_i \le s_p$. For example, the predicate sched(A-δ, τ, π) signifies that the tasks in τ which do *not* share resources is ascertained schedulable by algorithm A on platform π using the density-based schedulability test of algorithm A.

Finally, the term 'multiply (resp., divide) the processor speeds in platform π by a real number, $x > 0$' means that multiply (resp., divide) the speed of *every processor* in π by x resulting in a new platform, $\pi \times x$.

5.2 Useful Results

In this section, we present few previously known (Lemma 3-6) and some new results (Lemma 7-11 and Corollary 1) that we use while proving the speed competitive ratio of our algorithm, GIS-vpr, in Section 5.3.

Lemma 3 states that the speed competitive ratio of algorithm, GIS, proposed in [10] is 2. As mentioned earlier, the algorithm, GIS, non-migratively schedules the implicit-deadline sporadic tasks that do not share resources on uniform multiprocessor platform.

Lemma 3 (From Theorem 2.1 in [10]).

$$\text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{GIS}, \tau, \pi \times 2)$$

Lemma 4 states that the speed competitive ratio of non-migrative-offline, non-preemptive EDF is 3 for scheduling a set of implicit-deadline sporadic tasks that do not share resources on a single processor (say, $\pi_m \in \pi$).

Lemma 4 (From Lemma 2 in [2])).

$$\text{feas}(\text{nmo-np}, \tau, \pi_m) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi_m \times 3)$$

The following lemma states that if an implicit-deadline sporadic task set τ that does not share resources is schedulable by nm-np-EDF on a uniprocessor platform say, $\pi_m \times 3$, then the task set is also schedulable by nm-np-EDF on a uniform multiprocessor platform $\pi \times 3$. This trivially holds if tasks are only scheduled on processor $\pi_m \in \pi \times 3$, keeping the additional processors idle.

Lemma 5.

$$\text{sched}(\text{nm-np-EDF}, \tau, \pi_m \times 3) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi \times 3)$$

Combining Lemma 4 and Lemma 5 gives: if an implicit-deadline sporadic task set τ that does not share resources is non-migrative-offline, non-preemptive schedulable on a uniprocessor, say π_m, then the task set is also schedulable by nm-np-EDF on a uniform multiprocessor platform $\pi \times 3$.

Lemma 6.

$$\text{feas}(\text{nmo-np}, \tau, \pi_m) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi \times 3)$$

We now show that if an implicit-deadline task set τ that does not share resources is non-migrative-offline schedulable on a uniform multiprocessor platform π then the constrained-deadline task set τ^A (which does not share resources as well) that is derived from τ (as described in Section 4.1) is also non-migrative offline schedulable on platform $\pi \times 2$ (e.g., by non-migrative preemptive EDF). This is shown with the help of a density-based schedulability test by exploiting the fact that, on a processor π_p, the density of a task $\tau_i^A \in \tau^A$ is always twice the utilization of the corresponding task $\tau_i \in \tau$ (see Expression (4)). Hence, the density of the task $\tau_i^A \in \tau^A$ on a twice faster platform $\pi \times 2$ is equal to the utilization of the corresponding task $\tau_i \in \tau$ on platform π.

Lemma 7.

$$\text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{feas}(\text{nmo-}\delta, \tau^A, \pi \times 2)$$

Proof. Suppose there exists a non-migrative-offline schedule for task set τ on platform π in which all the deadlines are met. Hence, in that schedule, from Lemma 2, it must hold that:

$$\forall \pi_p \in \pi : \sum_{\tau_i \in \tau[\pi_p]} \frac{u_i}{s_p} \le 1 \tag{6}$$

where $\tau[\pi_p]$ denotes the set of tasks that are assigned to processor π_p.

We now show that there must also exist a non-migrative-offline schedule for the derived task set τ^A on platform $\pi \times 2$ in which all the deadlines are met. By definition of τ^A, we know that, for every task $\tau_i \in \tau$ there exists a task $\tau_i^A \in \tau^A$. Also, from Expression (4), we know that δ_i^A of $\tau_i^A \in \tau^A$ is twice the u_i of $\tau_i \in \tau$.

Let us assign the tasks in τ^A on platform $\pi \times 2$ as follows: if $\tau_i \in \tau$ is assigned to $\pi_p \in \pi$ in the non-migrative-offline schedule which meets all the deadlines, then we also assign τ_i^A to $\pi_p \in \pi \times 2$. From the fact that this assignment of τ^A (which is identical to the assignment of τ) is made on a platform twice faster (on which the densities of tasks will be halved according to expression (5)) and from Expressions (4) and (6), we get:

$$\forall \pi_p \in \pi \times 2 : \sum_{\tau_i^A \in \tau^A[\pi_p]} \frac{\delta_i^A}{s_p} \le 1 \tag{7}$$

which satisfies density-based schedulability test of non-migrative EDF on uniform multiprocessors. Hence, τ^A is non-migrative-offline schedulable on $\pi \times 2$. □

The following lemma is an extension of Lemma 3 obtained by applying density-based test instead of utilization-based test and on faster platforms.

Lemma 8.

$$\text{feas}(\text{nmo-}\delta, \tau^A, \pi \times 2) \Rightarrow \text{sched}(\text{GIS-}\delta, \tau^A, \pi \times 4)$$

Proof. Let us assume that the left-hand side predicate feas(nmo-$\delta, \tau^A, \pi \times 2$) holds true. From Expression (4), we know that the density of every task in τ^A is twice the utilization of the corresponding task in τ. Hence, from the reasoning similar to the one provided in the previous lemma, the

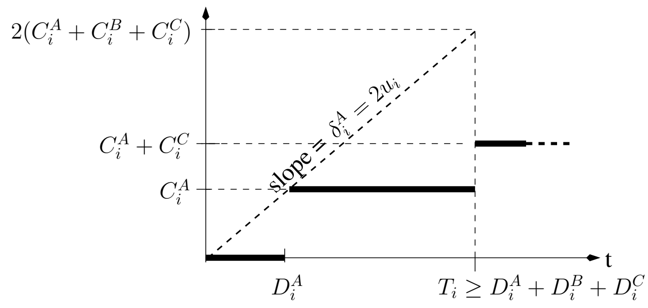

Figure 3: Assigning phase-C subtasks to the same virtual processor as the respective phase-A subtasks (earlier assigned using a density-based test) preserves schedulability.

predicate feas(nmo, τ, π) must hold true as well. Then, from Lemma 3, sched(GIS, $\tau, \pi \times 2$) must hold true. But again, using the fact that density of every task in τ^A is twice the utilization of the corresponding task in τ, sched(GIS-$\delta, \tau^A, \pi \times 4$) must hold true as well, from a similar reasoning as used in the previous lemma. Hence the proof. $\Box$

The following lemma states that if tasks from τ^A are preemptive EDF schedulable on a processor π_p then we can assign the respective phase-C subtasks from τ^C as well onto processor π_p and after this assignment, the entire set of tasks assigned to processor π_p is preemptive EDF schedulable.

LEMMA 9. *Let $\tau^A[\pi_p]$ denote the set of phase-A subtasks assigned on processor π_p of speed s_p. If $\tau^A[\pi_p]$ is preemptive-EDF schedulable on π_p, i.e.,*

$$\delta_{\tau^A[\pi_p]} \stackrel{def}{=} \sum_{\tau_i^A \in \tau^A[p]} \delta_i^A \leq s_p$$

then $\tau^A[\pi_p] \cup \tau^C[\pi_p]$ (where $\tau^C[\pi_p]$ is the set of respective phase-C subtasks whose arrivals have fixed offset from the arrival of respective phase-A subtasks) is also preemptive-EDF schedulable on processor π_p.

PROOF. We know that the task set $\tau^A[\pi_p]$ is preemptive-EDF schedulable on π_p, i.e., $\delta_{\tau^A[\pi_p]} \leq s_p$. To show that $\tau^A[\pi_p] \cup \tau^C[\pi_p]$ is also schedulable on processor π_p, it is sufficient to show that the *demand-bound function*[2], DBF($\tau^A[\pi_p] \cup \tau^C[\pi_p], t$), of task set $\tau^A[\pi_p] \cup \tau^C[\pi_p]$, never exceeds $\delta_{\tau^A[\pi_p]} \times t$ at any instant t [5].

The following must hold for every phase-A subtask $\tau_i^A \in \tau^A$ and respective phase-C task $\tau_i^C \in \tau^C$:

$$\mathrm{DBF}\left(\{\tau_i^A\} \cup \{\tau_i^C\}, t\right) \leq t \times \delta_i^A = t \times \frac{C_i^A}{D_i^A} \qquad (8)$$

This can be verified from Figure 3 since the maximum "slope" to any point in the graph of DBF($\{\tau_i^A\} \cup \{\tau_i^C\}, t$) from the origin is $\delta_i^A = \frac{C_i^A}{D_i^A}$ (which is equal to $2u_i$ of $\tau_i \in \tau$, as per our choice of D_i^A), at abscissa $t = D_i^A$. Summing Equation (8) for all the tasks $\tau_i^A \in \tau^A[\pi_p]$ and the corresponding tasks

[2]The *demand bound function* of a task τ_i, dbf(τ_i, t), is the maximum possible execution demand by jobs of τ_i, that have both arrival and deadline within any interval of length t. The demand bound function of a task set τ is defined as: $\mathrm{DBF}(\tau, t) = \sum_{\tau_i \in \tau} \mathrm{dbf}(\tau_i, t)$ [5].

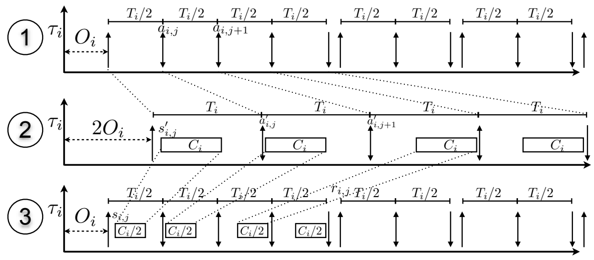

Figure 4: Visualization of the reasoning of Lemma 10.

$\tau_i^C \in \tau^C[\pi_p]$ yields:

$$\mathrm{DBF}(\tau^A[\pi_p] \cup \tau^C[\pi_p], t) \leq t \times \sum_{\tau_i^A \in \tau^A[\pi_p]} \delta_i^A = t \times \delta_{\tau^A[\pi_p]}$$

Hence the proof. $\Box$

We now show that if tasks in τ sharing the resource $r_k \in R$ are non-migrative-offline, non-preemptive schedulable on platform π then the derived task set τ^{B,r_k} is also non-migrative-offline, non-preemptive schedulable on a single dedicated processor which is the fastest, i.e., on processor π_m, but with its speed multiplied by 2. Intuitively, this speedup factor of 2 comes from the fact that we have halved the deadlines of tasks in τ^{B,r_k} compared to the deadlines of corresponding tasks in τ — this is formally proven in Corollary 1. This corollary is a consequence of the following two lemmas. The reader may want to skip the remainder of this section now and revisit it later.

LEMMA 10. *If $\tau\langle C_i, D_i, T_i\rangle$ denotes a task set in which each task τ_i is characterized by the 3-tuple $\langle C_i, D_i, T_i\rangle$ and shares the resource r_k, then it holds that*

$$\begin{aligned} &\text{feas(nmo-np}, \tau\langle C_i, D_i, T_i\rangle, r_k, \pi) \Rightarrow \\ &\text{feas(nmo-np}, \tau\left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2}\right\rangle, r_k, \pi) \qquad (9)\end{aligned}$$

PROOF. We assume that the left-hand side predicate holds true and then we show that the right-hand side predicate holds true as well. Let us denote by P any job-arrival pattern that can be generated by the task set $\tau\left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2}\right\rangle$. Let $a_{i,j}$ denote the arrival time of the jth job of τ_i in P (see the first schedule in Figure 4). The question is: "Given all these arrival times $a_{i,j}$, does there exist any schedule of $\tau\left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2}\right\rangle$ in which all the deadlines of jobs are met?". To answer this question, let us create the job-arrival pattern P' as follows. For every job-arrival $a_{i,j}$, we create a job-arrival $a'_{i,j} = 2 \times a_{i,j}$ and P' is defined as the set of all the job arrival times $a'_{i,j}$. One can easily see that the job-arrival pattern P' can be generated by the task set $\tau\langle C_i, D_i, T_i\rangle$. Since the predicate feas(nmo-np, $\tau\langle C_i, D_i, T_i\rangle, r_k, \pi$) is true we know that there exists a schedule for this job-arrival pattern P' in which every job of each task $\tau_i \in \tau\langle C_i, D_i, T_i\rangle$ executes for C_i time units and meets its deadline (see the second schedule in Figure 4). If $s'_{i,j}$ denotes the start-of-execution time of the jth job of task $\tau_i \in \tau\langle C_i, D_i, T_i\rangle$ in that schedule, then we define by $s_{i,j} = \frac{s'_{i,j}}{2}$ the start-of-execution time of the jth job of task $\tau_i \in \tau\left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2}\right\rangle$. It

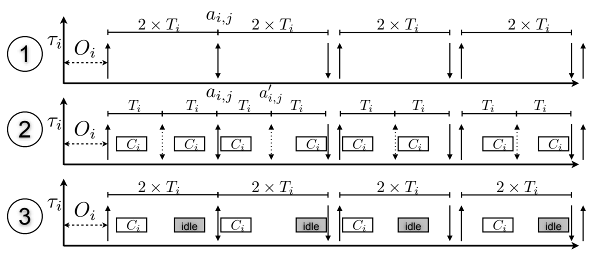

Figure 5: Visualization of the reasoning of Lemma 11.

can be easily seen that the resulting schedule for this task set $\tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2} \right\rangle$ also meets all the deadlines (see the third schedule in Figure 4). Hence the lemma. $\square$

LEMMA 11. *If $\tau \langle C_i, D_i, T_i \rangle$ denotes a task set in which each task τ_i is characterized by the 3-tuple $\langle C_i, D_i, T_i \rangle$ and shares the resource r_k, then it holds that*

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, 2 \times T_i \rangle, r_k, \pi) \end{aligned} \tag{10}$$

PROOF. We assume that the left-hand side predicate holds true and then we show that the right-hand side predicate holds true as well. Let us denote by P any job-arrival pattern that can be generated by the task set $\tau \langle C_i, D_i, 2 \times T_i \rangle$. Let $a_{i,j}$ denote the arrival time of the jth job of τ_i in P (see the first schedule in Figure 5). The question is: "Given all these arrival times $a_{i,j}$, does there exist any schedule of $\tau \langle C_i, D_i, 2 \times T_i \rangle$ in which all the deadlines of jobs are met?". To answer this question, let us create the job-arrival pattern P' as follows. For every job arrival $a_{i,j}$, insert a new job-arrival $a'_{i,j} = a_{i,j} + T_i$ (see the second schedule in Figure 5 — marked by dotted arrows). One can easily see that the job-arrival pattern P' composed of all the $a_{i,j}$ and $a'_{i,j}$ can be generated by the task set $\tau \langle C_i, D_i, T_i \rangle$. Since $\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi)$ is true we know that there exists a schedule for this job-arrival pattern P' in which all the deadlines are met. Thus, by copying this schedule but replacing the execution of every job arriving at one of the instants $a'_{i,j}$ (for all task τ_i) with an idle time of length C_i, we obtain a schedule for the task set $\tau \langle C_i, D_i, 2 \times T_i \rangle$ in which all the deadlines are met as well (as seen in the third schedule in Figure 5). Hence the lemma. $\square$

COROLLARY 1. *Let $\tau \langle C_i, D_i, T_i \rangle$ denote a task set in which each task τ_i is characterized by the 3-tuple $\langle C_i, D_i, T_i \rangle$ and shares the resource r_k, and let τ^{B,r_k} denote the set of phase-B subtasks derived from $\tau \langle C_i, D_i, T_i \rangle$. It holds that*

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi_{\mathbf{m}} \times 2) \end{aligned} \tag{11}$$

PROOF. The proof is a consequence of Lemma 10 and Lemma 11. From Lemma 10, we know that:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2} \right\rangle, r_k, \pi) \end{aligned} \tag{12}$$

From Lemma 11, we know that:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, \frac{T_i}{2} \right\rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi) \end{aligned} \tag{13}$$

Combining Expression (12) and Expression (13), we get:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi) \end{aligned} \tag{14}$$

Since dividing the execution requirements of the tasks by a factor of 2 is equivalent to doubling the speed of the computing platform, the following must hold:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \left\langle \frac{C_i}{2}, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi \times 2) \end{aligned} \tag{15}$$

Combining Expression (14) and Expression (15), we get:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi \times 2) \end{aligned} \tag{16}$$

Now consider phase-B scheduling. For each resource $r_k \in R$, since r_k is accessed in a mutually exclusive way, all the tasks that access resource r_k must execute sequentially. So, if tasks in τ sharing the resource r_k are non-migrative-offline, non-preemptive schedulable on platform π comprising processors $\pi_1, \pi_2, \ldots, \pi_m$ with speeds s_1, s_2, …, s_m (with π_m being the processor with highest speed) then the task set τ is also non-migrative-offline, non-preemptive schedulable on a single dedicated processor which is the fastest, i.e., π_m. This can be written as:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi_{\mathbf{m}}) \end{aligned} \tag{17}$$

Applying the above expression to a task set $\tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle$ and twice faster platform, we get:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi \times 2) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi_{\mathbf{m}} \times 2) \end{aligned} \tag{18}$$

Since the tasks in τ^{B,r_k} have the same parameters as the tasks in the task set $\tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle$ (with $C_i^B \leq C_i$), the following must hold:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi_{\mathbf{m}} \times 2) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi_{\mathbf{m}} \times 2) \end{aligned} \tag{19}$$

Now combining Expressions (18) and (19), we get:

$$\begin{aligned} &\text{feas}(\text{nmo-np}, \tau \left\langle C_i, \frac{D_i}{2}, T_i \right\rangle, r_k, \pi \times 2) \Rightarrow \\ &\text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi_{\mathbf{m}} \times 2) \end{aligned} \tag{20}$$

Finally, merging Expression (16) and (20) yields:

$$\begin{aligned}&\text{feas}(\text{nmo-np}, \tau \langle C_i, D_i, T_i\rangle, r_k, \pi) \Rightarrow\\ &\quad \text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi_{\mathbf{m}} \times 2)\end{aligned}$$

Hence the proof. $\square$

5.3 The Speed Competitive Ratio of the Algorithm GIS-vpr

We now prove the speed competitive ratio of the proposed algorithm.

THEOREM 1. *The speed competitive ratio of the algorithm,* GIS-vpr*, is* $4 + 6\rho$*.*

PROOF. We prove the claim by considering the scheduling of tasks in each of the three phases independently and then merging the results from these three scenarios.

Consider phase-A scheduling. Combining Lemma 7 and Lemma 8, yields:

$$\text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{GIS-}\delta, \tau^A, \pi \times 4) \tag{21}$$

Consider phase-C scheduling. Note that GIS-vpr assigned a phase-C subtask, $\tau_i^C \in \tau^C$, to the same virtual processor to which the corresponding phase-A subtask, $\tau_i^A \in \tau^A$, is assigned (see line 13 in Algorithm 1). For convenience, let GIS-δ-cp denote such a task assignment policy, i.e., using GIS-δ to assign phase-A subtasks and 'copying' the assignment for respective phase-C subtasks. Lemma 9 showed that such an assignment preserves schedulability of the relevant tasks. From Lemma 9 and Expression (21), we get:

$$\text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{GIS-}\delta\text{-cp}, \tau^A \cup \tau^C, \pi \times 4) \tag{22}$$

Now consider phase-B scheduling. For each resource $r_k \in R$, since r_k is accessed in a mutually exclusive way, all the tasks that access resource r_k must execute sequentially. So, if tasks in τ sharing the resource r_k are non-migrative-offline, non-preemptive schedulable on platform π then the derived task set τ^{B,r_k} is also non-migrative-offline, non-preemptive schedulable on a single dedicated processor which is the fastest, i.e., π_m but with its speed multiplied by a factor of 2. Recall that this was formally proven in Corollary 1 (with the help of Lemma 10 and Lemma 11). Hence, we can write:

$$\begin{aligned}&\forall r_k \in R : \text{feas}(\text{nmo-np}, \tau, r_k, \pi) \Rightarrow\\ &\quad \text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi_m \times 2)\end{aligned} \tag{23}$$

If we add additional $m - 1$ processors (from the left-hand side predicate) to the right-hand side predicate and leave these additional processors idle, then the above result can be re-written as:

$$\begin{aligned}&\forall r_k \in R : \text{feas}(\text{nmo-np}, \tau, r_k, \pi) \Rightarrow\\ &\quad \text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi \times 2)\end{aligned} \tag{24}$$

Let us add additional processors to the left-hand side predicate of Lemma 6 and keep these processors idle while scheduling. Using this information, we can rewrite Lemma 6 as follows:

$$\text{feas}(\text{nmo-np}, \tau, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi \times 3) \tag{25}$$

Applying Expression (25) to task set τ^{B,r_k} and multiplying the processor speeds by 2 on both left-hand and right-hand side platforms, yields:

$$\begin{aligned}&\forall r_k \in R : \text{feas}(\text{nmo-np}, \tau^{B,r_k}, r_k, \pi \times 2) \Rightarrow\\ &\quad \text{sched}(\text{nm-np-EDF}, \tau^{B,r_k}, r_k, \pi \times 6)\end{aligned} \tag{26}$$

Combining Expression (24) and Expression (26), we get:

$$\begin{aligned}&\forall r_k \in R : \text{feas}(\text{nmo-np}, \tau, r_k, \pi) \Rightarrow\\ &\quad \text{sched}(\text{nm-np-EDF}, \tau^{B,r_k}, r_k, \pi \times 6)\end{aligned} \tag{27}$$

Let us combine the results obtained for task sets $\tau^A \cup \tau^C$ and τ^{B,r_k}. "Dividing the processor speeds" in Expression (22) by $4 + 6\rho$, we get:

$$\begin{aligned}&\text{feas}\left(\text{nmo}, \tau, \pi \times \frac{1}{4+6\rho}\right) \Rightarrow\\ &\text{sched}\left(\text{GIS-}\delta\text{-cp}, \tau^A \cup \tau^C, \pi \times \frac{2}{2+3\rho}\right)\end{aligned} \tag{28}$$

"Dividing the processor speeds" in Expression (27) by $4+6\rho$, we get:

$\forall r_k \in R :$

$$\begin{aligned}&\text{feas}\left(\text{nmo-np}, \tau, r_k, \pi \times \frac{1}{4+6\rho}\right) \Rightarrow\\ &\text{sched}\left(\text{nm-np-EDF}, \tau^{B,r_k}, r_k, \pi \times \frac{3}{2+3\rho}\right)\end{aligned} \tag{29}$$

The specifications of the processors in the right-hand side predicates of Expression (28) and Expression (29) match those of the virtual processors that GIS-vpr created. Recall that GIS-vpr assigned phase-A and phase-C subtasks to VP_{AC} virtual processors and phase-B subtasks to VP_{B} virtual processors. Hence, combining Expression (28) and ρ instances of Expression (29), yields:

$$\begin{aligned}&\text{feas}\left(\text{rmo}, \tau, R, \pi \times \frac{1}{4+6\rho}\right) \Rightarrow\\ &\quad \text{sched}\left(\text{GIS-vpr}, \tau, R, \pi\right)\end{aligned} \tag{30}$$

Finally, "multiplying the processor speeds" in Expression (30) by $4 + 6\rho$ yields:

$$\begin{aligned}&\text{feas}\left(\text{rmo}, \tau, R, \pi\right) \Rightarrow\\ &\text{sched}\left(\text{GIS-vpr}, \tau, R, \pi \times (4+6\rho)\right)\end{aligned}$$

Hence the theorem. $\square$

6. DISCUSSIONS AND CONCLUSIONS

We now highlight couple of interesting features of the proposed solution.

First, the algorithm, GIS-vpr, by design, limits the number of migrations per job to at most two. Recall that GIS-vpr assigns both phase-A and phase-C executions of a task τ_i to the same VP_{AC} virtual processor say, $vp_{ac} \in \text{VP}_{\text{AC}}$, and phase-B of the task τ_i to another VP_{B} virtual processor say, $vp_b \in \text{VP}_{\text{B}}$. Since the algorithm creates each virtual processor from a single physical processor, it is clear that both phase-A and phase-C of a task are assigned to the same physical processor. Since the virtual processor in VP_{B} to which phase-B of task τ_i is assigned can come from a different physical processor, migration of a task can only occur at time instants when task τ_i requests or releases the resource.

Thus, the algorithm limits the number of migrations per job to at most two.

Second, the solution proposed in this work can be used as a framework. That is, the designer replaces algorithm GIS with any partitioning algorithm for uniform multiprocessors (whose speed competitive ratio is known) and a new speed competitive ratio can be derived for the resulting algorithm accordingly. For example, by replacing GIS with a fully polynomial-time approximation scheme [11] and by changing the specification of virtual processors accordingly, an algorithm with a better speed competitive ratio can be obtained.

Finally, recall that we use only one VP_B virtual processor (i.e., the one created from the fastest physical processor) from each group while assigning the phase-B subtasks and keep the rest of the VP_B virtual processors idle. It is natural to think that creating phase-B virtual processors only from the fastest physical processor might be a good idea. However, it turns out that, from the perspective of speed competitive ratio, it offers no benefit.

To conclude, in this work, we presented an algorithm, GIS-vpr, to schedule implicit-deadline sporadic tasks on uniform multiprocessors where each task can access at most one resource. We proved the speed competitive ratio of GIS-vpr to be $4 + 6\rho$. We also showed that this algorithm limits the number of migrations per job to at most two. To the best of our knowledge, this is the first algorithm for scheduling tasks that share resources on uniform multiprocessors with a proven speed competitive ratio. As part of the future work, we intend to extend this algorithm to a generic resource sharing model where tasks can access more than one resource and can access the resource more than once.

7. ACKNOWLEDGMENTS

This work was partially supported by FCT(Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within REHEAT project, ref. FCOMP-01-0124-FED-ER-010045; by FCT and the EU ARTEMIS JU funding, within RECOMP project, ref. ARTEMIS/0202/2009, JU grant nr. 100202; by FCT and ESF through POPH, under PhD grant SFRH/BD/66771/2009.

8. REFERENCES

[1] B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proceedings of the* 22^{nd} *IEEE Real-Time Systems Symposium*, pages 193–202, 2001.

[2] B. Andersson and A. Easwaran. Provably Good Multiprocessor Scheduling with Resource Sharing. *Journal of Real-Time System*, 46(2):153–159, 2010.

[3] B. Andersson and E. Tovar. Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors. In *Proceedings of the* 15^{th} *International Workshop on Parallel and Distributed Real-Time Systems*, pages 1–8, 2007.

[4] S. Baruah and N. Fisher. The Partitioned Dynamic-priority Scheduling of Sporadic Task Systems. *Real-Time Systems*, 36(3):199–226, Aug. 2007.

[5] S. Baruah, A. Mok, and L. Rosier. Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990.

[6] K. Bletsas and B. Andersson. Notional Processors: An Approach for Multiprocessor Scheduling. In *Proceedings of the 15th IEEE International Real-Time and Embedded Technology and Applications Symposium*, pages 3–12, 2009.

[7] A. Block, H. Leontyev, B. Brandenburg, and J. Anderson. A Flexible Real-Time Locking Protocol for Multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 47–56, 2007.

[8] R. Davis, T. Rothvoß, S. Baruah, and A. Burns. Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. *Real-Time Systems*, 43(3), Nov. 2009.

[9] M. Dertouzos. Control Robotics: The Procedural Control of Physical Processes. In *Proceedings of IFIP Congress (IFIP'74)*, pages 807–813, 1974.

[10] T. Gonzalez, O. Ibarra, and S. Sahni. Bounds for LPT Schedules on Uniform Processors. *SIAM Journal on Computing*, 6(1):155–166, 1977.

[11] E. Horowitz and S. Sahni. Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *Journal of ACM*, 23:317–327, 1976.

[12] M. Jones. What Happened on Mars? http://www.ece.cmu.edu/ raj/mars.html, 1997.

[13] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[14] M. López, J. L. Díaz, and D. F. García. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems Journal*, 28:39–68, 2004.

[15] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal Time-Critical Scheduling via Resource Augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.

[16] R. Rajkumar, L. Sha, and J. Lehoczky. Real-Time Synchronization Protocols for Multiprocessors. In *9th IEEE Real-Time Systems Symposium*, pages 259 –269, 1988.

[17] G. Raravi, B. Andersson, and K. Bletsas. Provably Good Scheduling of Sporadic Tasks with Resource Sharing on a Two-type Heterogeneous Multiprocessor Platform. In *15th International Conference on Principles of Distributed Systems (OPODIS)*, Lecture Notes in Computer Science, pages 528–543. Springer, 2011.