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 + 6ρ 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
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 \(12 \times \frac{1 + \frac{m}{\rho}}{m+\rho}\) where \(m\) and \(\rho\) denote the number of processors and resources, respectively. Later, Ramani 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 \(4 + 6 \times \frac{e}{\min(m_1, m_2)}\) where \(m_1\) and \(m_2\) denote the number of processors of first and second kind, respectively. 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 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.

### 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 \(\rho\) resources \(R = \{r_1, r_2, \ldots, r_\rho\}\) on a uniform multiprocessor platform \(\pi = \{\pi_1, \pi_2, \ldots, \pi_\rho\}\) 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 \(\tau_i\) 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 \(\tau_i\) 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 \(\pi_k \in \pi\) is of speed \(s_k\) where \(1 \leq \rho \leq s_m\); for ease of explanation, we consider that processors are indexed such that \(s_1 \leq s_2 \leq \ldots \leq s_\rho\). If, for a time interval of duration \(L\) and for a processor \(\pi_k\) of speed \(s_k\), a job executes during the entire time interval on \(\pi_k\), then the job performs \(L \times s_k\) units of execution during this time interval.

The set of shared resources comprising \(\rho\) 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 \(\tau_i\) 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 \(\tau_i\) is denoted by \(u_i\) and is defined as \(u_i = \frac{C_i}{\rho_s}\). It is easy to see that if a job of a task \(\tau_i\) executes only on processor \(\pi_k\) of speed \(s_k\), then it performs its \(C_i\) units of execution by executing for \(T_i\) time units. For this reason, we define \(C_i \rho_s \equiv \frac{C_i}{\rho_s}\). Analogously, we define \(u_i \rho_s \equiv \frac{C_i}{\rho_s}\).

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 \(\tau_i\), its density is denoted as \(\delta_i\) and is defined by \(\delta_i \equiv \frac{C_i}{\min(D_i, T_i)} = \frac{C_i}{T_i}\). Similar to \(u_i \rho_s\), we define \(\delta_i \rho_s \equiv \frac{C_i}{\rho_s T_i}\).

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 \(1\) and we assume that it can be "emulated" on a physical processor of speed \(1\), using no more than \(\frac{1}{2}\) of its processing capacity. One intuitive way of achieving this is by dividing time into short slots of length \(S\) and using \(\frac{1}{2} \times S\) 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-
Subtasks of $\tau_i$    WCET     Deadline     Period
| $\tau_i^A$     | $C_i^A$     | $D_i^A = \frac{C_i^A}{T_i} \times \frac{T_i}{2}$     | $T_i^A = T_i$          |
| $\tau_i^B$     | $C_i^B$     | $D_i^B = \frac{C_i^B}{T_i}$     | $T_i^B = T_i$          |
| $\tau_i^C$     | $C_i^C$     | $D_i^C = \frac{C_i^C}{T_i} \times \frac{T_i}{2}$     | $T_i^C = T_i$          |

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

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 $\tau_i^A$, $\tau_i^B$ and $\tau_i^C$ corresponding to phase-A, phase-B and phase-C of the execution of $\tau_i$, 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 $\tau_i^A$, $\tau_i^B$ and $\tau_i^C$ 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, $\tau_i^A$, $\tau_i^B$ and $\tau_i^C$ derived from $\tau_i$ meet their deadlines then the original task $\tau_i$ meets its deadline as well. Finally, we group these derived subtasks into the following task sets:

$$\tau_i^A = \{ \tau_i^A | i \in [1, n] \} \quad (1)$$
$$\tau_i^B = \{ \tau_i^B | i \in [1, n] \text{ and } r(i) = r_k \} \quad (2)$$
$$\tau_i^C = \{ \tau_i^C | i \in [1, n] \} \quad (3)$$

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

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

Analogously,

$$\forall \tau_i^C : \delta_i^C = \frac{C_i^C}{D_i} = \frac{C_i^C}{T_i^C} \times \frac{T_i^C}{2} = \frac{2 C_i^C}{T_i} = 2 \mu_i \quad (5)$$

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

$$\forall \tau_i^+ \in \tau^+ : C_{i,p}^+ = \frac{C_i^A}{s_p}, \delta_{i,p}^+ = \frac{\delta_i^A}{s_p} \quad (5)$$
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 $V_{PAC}$): From each physical processor $p$, we create one virtual processor of speed $s_p$. So, in total, $m$ such virtual processors are created from $m$ physical processors.

- $m \times \rho$ virtual processors (denoted as $V_{PB}$): From each physical processor $p$, we create $\rho$ virtual processors of speed $s_p$. So, in total, $m \times \rho$ such virtual processors are created from $m$ physical processors.

In other words, from each physical processor $p$, we create $1 + \rho$ virtual processors, i.e., one $V_{PAC}$ and $\rho$ $V_{PB}$ 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 $p$ of speed $s_p$, it is indeed possible to create one $V_{PAC}$ and $\rho$ $V_{PB}$ virtual processors as per the specifications given earlier and hence the given uniform multiprocessor platform $\pi$ can be dimensioned accordingly to obtain the above specified set of virtual processors.

**Lemma 1.** The given uniform multiprocessor platform $\pi$ can be dimensioned as mentioned above to obtain the set of virtual processors $V_{PAC}$ and $V_{PB}$.

**Proof.** The proof is a direct consequence of the fact that each physical processor can emulate its associated $V_{PAC}$ virtual processor and its $\rho$ $V_{PB}$ virtual processors, as per the specifications of the virtual processors. Indeed, for each $p \in \pi$, we have

$$\frac{1 \times (s_p \times \frac{2}{2 + 3\rho})}{V_{PAC} \text{ virtual processor}} + \frac{\rho \times (s_p \times \frac{3}{2 + 3\rho})}{V_{PB} \text{ virtual processors}} = \frac{2s_p + 3\rho s_p}{2 + 3\rho} = s_p$$

Hence the proof.

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

**Algorithm 1:** GIS-vpr(\(\tau, \pi, R\)): for scheduling tasks that share resources on uniform multiprocessors

// Lines 1-9 execute offline; line 10 executes at run time.
1 Create the sets $\tau^A, \tau^B, r_s$ and $\tau^C$ of subtasks from the given task set $\tau$ as described in Section 4.1;
2 Create $V_{PAC}$ and $V_{PB}$ virtual processors from the given set $\pi$ of processors as described in Section 4.2;
3 Assign all the subtasks $\tau^i \in \tau^A$ to the $V_{PAC}$ virtual processors using the algorithm GIS (see [10] for details);
4 foreach $\tau_i \in \tau$ do
5 if \(\tau_i \text{ requests a resource } r_k \) then
6 Assign $\tau^B_i$ 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 $\tau^C_i \in \tau^C$ to that virtual processor in $V_{PAC}$ to which the corresponding subtask $\tau^A_i \in \tau^A$ has been assigned on line 3;
10 Schedule (i) all the subtasks of $\tau^A$ and $\tau^C$ on $V_{PAC}$ virtual processors using preemptive EDF and (ii) all the subtasks of $\tau^B$ on $V_{PB}$ virtual processor using non-preemptive EDF;

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

On line 1, it creates the sets $\tau^A, \tau^B, r_s$ and $\tau^C$ of constrained-deadline subtasks from the given set $\tau$ of implicit-deadline tasks as described in Section 4.1.

On line 2, it creates $m$ $V_{PAC}$ and $m \times \rho$ $V_{PB}$ 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, $\tau^A$, on $V_{PAC}$ virtual processors using GIS algorithm. The algorithm, GIS, was proposed by Gonzalez and Ibarra and Sahni [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 VPB virtual processor. Specifically, all the subtasks accessing the resource $r_k$, $\forall k \in \{1, p\}$, are assigned to the $k$th VPB virtual processor created from the fastest physical processor, $\pi_{r_k}$. This technique serves two purposes. 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 wherein 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 VPB virtual processors that are created from the fastest physical processor $\pi_{r_k}$, hence leaving all the other VPB virtual processors idle. It is therefore natural to think that creating VPB 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 VPB virtual processors involved in these “context switches”, the mutual exclusion property while accessing the resources is not affected. This is because a VPB 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, $\tau^C_i$, to that virtual processor in VPAC to which the corresponding phase-A subtask, $\tau^A_i$, has been assigned. Such an assignment does not endanger the schedulability of the tasks assigned on the VPAC 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 VPAC virtual processors using preemptive EDF and tasks executing in their phase-B on VPB 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 $\tau^p[\pi]$ denote the tasks assigned on a processor $\pi$ of speed $s_p$. If $\sum_{t_i \in \tau[\pi]} u_i \leq s_p$ and tasks are scheduled with preemptive EDF on $\pi$ 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 virtual processors.

## 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 $\pi$ denote a uniform multiprocessor platform of $m$ processors, $\{\pi_1, \pi_2, \ldots, \pi_m\}$. The speed of a processor $\pi_i$ is $s_i$. 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 $\pi_i$ is $s$ times that of the corresponding processor in $\pi$. The platform $\pi \times s$ is obtained by multiplying the speed of each processor in platform $\pi$ by a real number, $s > 0$. We use $\pi_m$ 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 sched($A, \tau, \pi$) denote a predicate to signify that a task set $\tau$ that does not share resources meets all its deadlines when scheduled by algorithm $A$ on platform $\pi$. 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 $\pi$’.

Let there exists a non-migrative-affine preemptive schedule which meets all deadlines of tasks in $\pi$ that do not share resources on platform $\pi$. 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 affine 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 sched($A, \tau, R, \pi$) signifies that the task set $\tau$ sharing the resources from a set $R$ meets all its deadlines when scheduled by algorithm $A$ on platform $\pi$ with restricted migration. In this and other predicates, the term ‘sharing the resources’ has the same meaning as discussed in Section 2 and ‘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 \( \tau \) share a single resource \( r_k \), where \( 1 \leq k \leq \rho \).

Let \( \text{feas}(\text{nmo}, \tau, R, \pi) \) denote a predicate to signify that there exists a restricted-migration-offline preemptive schedule which meets all the deadlines of tasks in \( \tau \) on platform \( \pi \) when tasks are ‘sharing the resources’ from \( R \).

Also, some of the above described predicates will be used by adding a suffix -\( \delta \) (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 -\( \delta \) signify that the schedulability of \( \tau \) 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 \( \tau[\pi] \) of tasks assigned on a processor \( \pi \), to meet deadlines, it must hold that \( \sum_{i \in \tau[\pi]} \delta_i \leq s_p \). For example, the predicate \( \text{sched}(A, \pi, \delta) \) signifies that the tasks in \( \tau \) which do not share resources is ascertainable schedulable by algorithm \( A \) on platform \( \pi \) using the density-based schedulability test of algorithm \( A \).

Finally, the term ‘multiply (resp., divide)’ the processor speeds in platform \( \pi \) by a real number, \( x > 0 \)’ means that multiply (resp., divide) the speed of every processor in \( \pi \) 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 uniprocessor 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, non-preemptive EDF is 3 for scheduling a set of implicit-deadline sporadic tasks that do not share resources on a single processor (say, \( \tau_m \in \tau \)).

**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 \( \tau \) 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 \( \tau \) that does not share resources is non-migrative-offline, non-preemptive schedulable on a uniprocessor, say \( \tau_m \), 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 \( \tau \) that does not share resources is non-migrative-offline schedulable on a uniform multiprocessor platform \( \pi \) then the constrained-deadline task set \( \tau^A \) (which does not share resources as well) that is derived from \( \tau \) (as described in Section 4.1) is also non-migrative offline schedulable on platform \( \pi \times 2 \) (e.g., by non-migrative preemptive EDF).

**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 \( \tau \) on platform \( \pi \) in which all the deadlines are met. Hence, in that schedule, from Lemma 2, it must hold that:

\[
\forall \tau_p \in \pi : \sum_{i \in \tau[\pi]} \delta_i \leq s_p \leq 1
\]

where \( \tau[\pi] \) denotes the set of tasks that are assigned to processor \( \pi_p \).

We now show that there must also exist a non-migrative-offline schedule for the derived task set \( \tau^A \) on platform \( \pi \times 2 \) in which all the deadlines are met. By definition of \( \tau^A \), 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 \( \delta_i^A \) of \( \tau_i^A \in \tau^A \) is twice the \( u_i \) of \( \tau_i \in \tau \).

Let us assign the tasks in \( \tau^A \) 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 \( \tau_i^A \) to \( \pi_p \in \pi \times 2 \). From the fact that this assignment of \( \tau^A \) (which is identical to the assignment of \( \tau \)) 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:

\[
\forall \tau_p \in \pi \times 2 : \sum_{i \in \tau[\pi]} \delta_i^A \leq s_p \leq 1
\]

which satisfies density-based schedulability test of non-migrative EDF on uniform multiprocessors. Hence, \( \tau^A \) 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 \( \text{feas}(\text{nmo-\delta}, \tau^A, \pi \times 2) \) holds true. From Expression (4), we know that the density of every task in \( \tau^A \) is twice the utilization of the corresponding task in \( \tau \). Hence, from the reasoning similar to the one provided in the previous lemma, the
The demand bound function of a task set choice of arrival of respective phaseA subtasks whose arrivals have fixed offset onto processor πp and after this assignment, the entire set of tasks assigned to processor πp is preemptive EDF schedulable.

**Lemma 9.** Let τA[πp] denote the set of phase-A subtasks assigned on processor πp of speed sp. If τA[πp] is preemptive-EDF schedulable on πp, i.e.,

\[ \delta_{A[πp]} \geq \sum_{τi_{A[πp]} \in \tau_{A[πp]}} \delta_{i_{A[πp]}} \leq sp, \]

then τA[πp] ∪ τC[πp] (where τC[πp] 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 πp.

**Proof.** We know that the task set τA[πp] is preemptive-EDF schedulable on πp, i.e., δA[πp] ≤ sp. To show that τA[πp] ∪ τC[πp] is also schedulable on processor πp, it is sufficient to show that the demand bound function \( \text{DBF}(\tau_{A[πp]} \cup \tau_{C[πp]}, t) \), of task set τA[πp] ∪ τC[πp], never exceeds δA[πp] × t at any instant t [5].

The following holds for every phase-A subtask τiA ∈ τA and respective phase-C task τiC ∈ τC:

\[ \text{DBF}\left(\{τA\} \cup \{τC\}, t\right) \leq t \times \delta_{A[πp]} = t \times C_{A[πp]} \frac{1}{D_{A[πp]}} \quad (8) \]

This can be verified from Figure 3 since the maximum "slope" to any point in the graph of DBF(\{τA\} ∪ \{τC\}, t) from the origin is δiA = C_{A[πp]} / D_{A[πp]} which is equal to 2ui, of τi ∈ τ, as per our choice of D_{A[πp]}, at absissa t = D_{A[πp]} for all the tasks τiA ∈ τA[πp] and the corresponding tasks

\[ \text{DBF}(τ_i, t) = \sum_{τi_{A[πp]} \in \tau_{A[πp]}} \text{dbf}(τ_i, t) \] [5].
can be easily seen that the resulting schedule for this task set \( \langle C_i, D_i, T_i \rangle \) also meets all the deadlines (see the third schedule in Figure 4). Hence the lemma.

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, 2 \times T_i \rangle, r_k, \pi)
\]

**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 \( \langle C_i, D_i, 2 \times T_i \rangle \). Let \( a_{i,j} \) denote the arrival time of the \( j \)th job of \( \tau_i \) 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 \( \text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \) 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 \( \tau_i \)) 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.

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau^{B,r_k}, r_k, \pi_m \times 2)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, r_k, \pi)
\]

From Lemma 11, we know that:

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi \times 2)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi \times 2)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi_m)
\]

Applying the above expression to a task set \( \tau \langle C_i, D_i, T_i \rangle \) and twice faster platform, we get:

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi_m \times 2)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi_m \times 2) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau^{B,r_k}, r_k, \pi_m \times 2)
\]

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

\[
\text{feas}(\text{mno-mp}, \tau \langle C_i, D_i, T_i \rangle, r_k, \pi) \Rightarrow \\
\text{feas}(\text{mno-mp}, \tau^{B,r_k}, r_k, \pi_m \times 2)
\]
Finally, merging Expression (16) and (20) yields:
\[ \text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi \times 3) \] (25)

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

Consider phase-C scheduling. Note that GIS-vpr assigned a phase-C subtask, \(\tau^C_k \in \tau^C\), to the same virtual processor to which the corresponding phase-A subtask, \(\tau^A_k \in \tau^A\), is assigned (see line 13 in Algorithm 1). For convenience, let GIS-\delta-\text{cp} denote such a task assignment policy, i.e., using GIS-\delta 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:
\[ \text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{GIS-\delta-\text{cp}}, \tau^A \cup \tau^C, \pi \times 4) \] (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 \(\tau \) sharing the resource \(r_k\) are non-preemptive-offline, non-preemptive schedulable on platform \(\pi\) then the derived task set \(\tau^B, r_k\) is also non-preemptive-offline, non-preemptive schedulable on a single dedicated processor which is the fastest, i.e., \(\pi_m\), 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:
\[ \forall r_k \in R : \text{feas}(\text{nmo}, \tau, r_k, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau^B, r_k, \pi_m \times 2) \] (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:
\[ \forall r_k \in R : \text{feas}(\text{nmo}, \tau, r_k, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau^B, r_k, \pi \times 2) \] (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:
\[ \text{feas}(\text{nmo}, \tau, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau, \pi \times 3) \] (25)

Applying Expression (25) to task set \(\tau^B, r_k\) and multiplying the processor speeds by 2 on both left-hand and right-hand side platforms, yields:
\[ \forall r_k \in R : \text{feas}(\text{nmo}, \tau^B, r_k, \pi \times 2) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau^B, r_k, \pi \times 6) \] (26)

Combining Expression (24) and Expression (26), we get:
\[ \forall r_k \in R : \text{feas}(\text{nmo}, \tau, r_k, \pi) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau^B, r_k, \pi \times 6) \] (27)

Let us combine the results obtained for task sets \(\tau^A \cup \tau^C\) and \(\tau^B, r_k\). “Dividing the processor speeds” in Expression (22) by 4 + 6\(\rho\), we get:
\[ \text{feas}(\text{nmo}, \tau, \pi \times \frac{1}{4 + 6\rho}) \Rightarrow \text{sched}(\text{GIS-\delta-\text{cp}}, \tau^A \cup \tau^C, \pi \times \frac{2}{2 + 3\rho}) \] (28)

“Dividing the processor speeds” in Expression (27) by 4 + 6\(\rho\), we get:
\[ \text{feas}(\text{nmo}, \tau, \pi \times \frac{1}{4 + 6\rho}) \Rightarrow \text{sched}(\text{nm-np-EDF}, \tau^B, r_k, \pi \times \frac{3}{2 + 3\rho}) \] (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 \(\text{VP}_A\) virtual processors and phase-B subtasks to \(\text{VP}_B\) virtual processors. Hence, combining Expression (28) and \(\rho\) instances of Expression (29), yields:
\[ \text{feas}(\text{rmo}, \tau, R, \pi \times \frac{1}{4 + 6\rho}) \Rightarrow \text{sched}(\text{GIS-vpr}, \tau, R, \pi) \] (30)

Finally, “multiplying the processor speeds” in Expression (30) by 4 + 6\(\rho\) yields:
\[ \text{feas}(\text{rmo}, \tau, R, \pi) \Rightarrow \text{sched}(\text{GIS-vpr}, \tau, R, \pi \times (4 + 6\rho)) \]

Hence the theorem. \[\square\]

6. DISCUSSIONS AND CONCLUSIONS

We now highlight a 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 \(\tau_i\) to the same \(\text{VP}_A\) virtual processor say, \(\text{vp}_A \in \text{VP}_A\), and phase-B of the task \(\tau_i\) to another \(\text{VP}_B\) virtual processor say, \(\text{vp}_B \in \text{VP}_B\). 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 \(\text{VP}_B\) to which phase-B of task \(\tau_i\) is assigned can come from a different physical processor, migration of a task can only occur at time instants when task \(\tau_i\) 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 $n$ 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 $n$ 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, this 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-FEDER-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


