Real-time scheduling with resource sharing on heterogeneous multiprocessors

Björn Andersson & Gurulingesh Raravi

Abstract

Consider the problem of scheduling a task set τ of implicit-deadline spo- radic tasks to meet all deadlines on a t-type heterogeneous multiprocessor platform where tasks may access multiple shared resources. The multiprocessor platform has m_k processors of type-k, where $k \in \{1, 2, \ldots, t\}$. The execution time of a task de- pends on the type of processor on which it executes. The set of shared resources is denoted by R. For each task τ_i, there is a resource set $R_i \subseteq R$ such that for each job of τ_i, during one phase of its execution, the job requests to hold the resource set R_i exclusively with the interpretation that (i) the job makes a single request to hold all the resources in the resource set R_i and (ii) at all times, when a job of τ_i holds R_i, no other job holds any resource in R_i. Each job of task τ_i may request the resource set R_i at most once during its execution. A job is allowed to migrate when it requests a resource set and when it releases the resource set but a job is not allowed to migrate at other times. Our goal is to design a scheduling algorithm for this problem and prove its performance.

We propose an algorithm, LP-EE-vpr, which offers the guarantee that *if* an implicit-deadline sporadic task set is schedulable on a t-type heterogeneous multiprocessor platform by an optimal scheduling algorithm that allows a job to migrate only when it requests or releases a resource set, *then* our algorithm also meets the deadlines with the same restriction on job migration, if given processors $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$ times as fast. (Here MAXP and $|P|$ are computed based on the resource sets that tasks request.) For the special case that each task requests at most one resource, the bound of LP-EE-vpr collapses to

$4 \times (1 + \lceil \frac{|R|}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$. To the best of our knowledge, LP-EE-vpr is the first algorithm with proven performance guarantee for real-time scheduling of sporadic tasks with resource sharing on t-type heterogeneous multiprocessors.

Keywords

Heterogeneous multiprocessors, Real-time scheduling, Resource sharing

1 Introduction

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 tasks must operate on such resources in a *mutually exclusive* manner while accessing the resource, that is, at all times, when a job of a task holds a resource, no other job of any task can hold that resource. Even on a single processor, the sharing of such resources can have a profound effect on timing behavior as witnessed by the near failure of the NASA mission, *Mars Pathfinder*, because the resource-sharing protocol in the operating system was not enabled (Jones 1997). 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 real-time tasks that share resources (apart from processors) on t-type heterogeneous multiprocessors so as to meet all the deadlines.

In a t-type *heterogeneous* multiprocessor platform (also called *unrelated* parallel machine) (i) not all processors are of the same type, (ii) the execution time of a task depends on the type of processor on which it executes and (iii) the number of distinct *types* of processors is a constant and is given by $t \geq 2$. Many manufacturers offer chips combining different types of processors (AMD Inc. 2012; Apple Inc. 2012; Intel Corporation 2012; Intel Corporation 2013; Nvidia Inc. 2012; Qualcomm Inc. 2012; Samsung Inc. 2012; Ericsson 2012; Texas Instruments 2012; Alben 2013; Intel Corp. 2013). Clearly, such chips are key components in heterogeneous systems, and such systems are increasingly used in practice. Yet, despite this trend, state-of-the-art in real-time scheduling theory for heterogeneous multiprocessors is under-developed. The reasons include (i) processors typically share low-level hardware resources such as caches and interconnects, which make task execution times interdependent and (ii) dispatching limitations, for example, some processors depend on another processor for dispatching (Gschwind et al. 2006). Such idiosyncratic challenges must be addressed on a case-by-case basis, accounting for the particularities of the architecture. The state-of-the-art does offer some general ideas on analyzing shared low-level hardware resources (Dasari et al. 2011; Dasari and Nélis 2012; Li et al. 2009; Lv et al. 2010; Pellizzoni et al. 2010; Rosén et al. 2007; Schliecker et al. 2010) and scheduling co-processors (Bletsas 2007; Gai et al. 2002; Lakshmanan and Rajkumar 2010). Unlike the idiosyncratic challenges though, the dependency of the execution time of a task on the type of processor to which it is assigned is an inherent property of heterogeneous multiprocessors.

Therefore, designers using heterogeneous multiprocessors today and in the future can benefit from scheduling theories that consider this inherent property. And for this reason, in this work, we design an algorithm (considering this property) to schedule tasks that share resources (in addition to processors) on t-type heterogeneous multiprocessors and prove its performance.

Commonly, the performance of a scheduling algorithm is characterized using the notion of *utilization bound* (Liu and Layland 1973). This metric has been used to evaluate scheduling algorithms on uniprocessors (Liu and Layland 1973), *identical* multiprocessors (Andersson et al. 2001) in which the speeds of all processors are the same and *uniform* multiprocessors (Darera and Jenkins 2006) in which the speeds of the processors are different. However, it does not translate to algorithms on *heterogeneous* multiprocessors (even when tasks do not share resources), hence we rely on the *resource augmentation* framework (Phillips et al. 1997) 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 task set for which it is possible to meet all deadlines, A succeeds to schedule the tasks to meet all deadlines as well if the speed of each processor is SCR_A times faster. In the literature, speed competitive ratio is sometimes referred to as speedup factor (for example, see Baruah 2013; Wiese et al. 2013).

A low speed competitive ratio indicates high performance; the best achievable is *one* (which reflects the optimal algorithm for a given problem). If a scheduling algorithm has an infinite speed competitive ratio then a task set exists which could be scheduled (by another algorithm) to meet deadlines but would miss deadlines with the actually used algorithm even if processor speeds were multiplied by an "infinite" factor. Therefore, a scheduling algorithm with a *finite* (ideally small) speed competitive ratio is desirable 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., Andersson and Tovar (2007), Baruah and Fisher (2007) and Davis et al. (2009). Unfortunately, the community has not yet developed a multiprocessor scheduling algorithm with a proven speed competitive ratio for the problem of scheduling tasks that share resources on *t-type heterogeneous* multiprocessors. Therefore, in this paper, we present an algorithm for this problem and prove its performance.

Problem statement. We consider the problem of scheduling a task set τ of implicit-deadline sporadic tasks to meet all deadlines on a t-type heterogeneous multiprocessor platform where a task may access multiple shared resources. There are m_k processors of type-k, where $k \in \{1, 2, \ldots, t\}$. The execution time of a task depends on the processor type on which it executes. There is a set R of resources. For each task τ_i, there is a resource set $R_i \subseteq R$ such that for each job of τ_i, during one phase of its execution, the job requests to hold the resource set R_i exclusively with the interpretation that (i) the job makes a single request to hold all the resources in the resource set R_i and (ii) at all times, when a job of τ_i holds R_i, no other job holds any resource in R_i. We assume that each job of task τ_i may request the resource set R_i at most once during its execution. We also assume (like the previous work on D-PCP (Rajkumar et al. 1988)) that a job is allowed to migrate when it requests a

resource set and when it releases a resource set but a job is not allowed to migrate at other times. One can show (through mapping an instance of 3-PARTITION to an instance of our problem) that the problem under consideration is NP-Complete in the strong sense. Our goal is to design a scheduling algorithm for this problem and prove the speed competitive ratio of this algorithm.

Related work. Scheduling a collection of jobs that share resources is well-studied in operations research (see Blazewicz et al. 1983, for example) but unfortunately these algorithms deal with jobs which make them less suited for real-time systems because real-time systems tend to be implemented with tasks that generate a (potentially infinite) sequence of jobs. The problem of scheduling a set of implicit-deadline sporadic tasks on heterogeneous multiprocessors has been studied in the past (Baruah 2004a, 2004b, 2004c; Correa et al. 2012; Lenstra et al. 1990; Andersson et al. 2010; Raravi et al. 2012, 2013; Raravi and Nélis 2012; Wiese et al. 2013; Horowitz and Sahni 1976; Jansen and Porkolab 1999) but without considering the case when tasks share resources. However, recently, a run-time synchronization protocol, PSRP, is proposed in Holenderski et al. (2012) for the problem of scheduling parallel tasks on a platform comprising multiple heterogeneous resources. It considers a parallel task model in which a task may execute on several processors at the same time whereas we consider a sequential task model in which a task can execute on at most one processor at any time. In this respect, the task model considered in Holenderski et al. (2012) is more general than the one considered in this work. However, the PSRP algorithm of Holenderski et al. (2012) does not have a proven speed competitive ratio whereas we prove the speed competitive ratio for our algorithm. More importantly, the work in Holenderski et al. (2012) proposes a "run-time synchronization mechanism" and thus assumes that an assignment of tasks to processors is given; however, in this work, we propose an algorithm which assigns tasks to processors before run-time and handles synchronization at run-time. So, the problem addressed and the goals of Holenderski et al. (2012) are different than this work although both are related to sharing multiple resources on multiprocessors.

For the problem of scheduling tasks that share resources on heterogeneous multiprocessors, one might also consider an obvious solution of assigning tasks to processors and then applying a resource-sharing protocol conceived for identical multiprocessors, for example, D-PCP (Rajkumar et al. 1988). However, protocols for resource sharing on an identical multiprocessor (such as D-PCP) are less effective in minimizing priority inversion when used in heterogeneous multiprocessors as they are in minimizing priority inversion when used in identical multiprocessors. The reason for this is that, a task holding a shared resource may be executing on a processor where it runs slowly—causing large priority inversion to other tasks and poor schedulability. Therefore, a resource-sharing protocol for heterogeneous platforms ought to be cognizant of the execution rate of each task on each processor type. It should also provide a bound on how much worse it performs, compared to an optimal scheme.

This work. In this paper, we propose an algorithm, LP-EE-vpr, for scheduling implicit-deadline sporadic tasks that share resources on a t-type heterogeneous multiprocessor platform. We also prove the speed competitive ratio of LP-EE-vpr.

A key idea of our new algorithm is to organize the resource sets into *resource request partitions* so that for every pair of tasks τ_i and $\tau_{i'}$, if there is a resource shared between these two tasks (that is, if $R_i \cap R_{i'} \neq \emptyset$) then the resource sets (R_i and $R_{i'}$) belong to the same resource request partition. Hence, if two resource sets of different tasks belong to different resource set partitions then we know that these tasks do not share resources. We will create a procedure for forming the resource request partitions and then we let P denote the set of resource request partitions and MAXP denote the number of elements in the resource request partition with the largest number of elements. (P and MAXP will be defined formally in Sect. 2.)

The algorithm, LP-EE-vpr, offers the guarantee that *if* a task set is schedulable on a t-type heterogeneous multiprocessor platform by an optimal scheduling algorithm that allows a job to migrate only when it requests or releases the resources, *then* our algorithm also meets the deadlines with the same restriction on the job migration, if given processors $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$ times as fast. In order to prove this bound, we create a new algorithm, ra-np-pEDF-fav, which is used as one part of LP-EE-vpr and prove a lemma which compares feasibility of tasks on a multiprocessor with schedulability of tasks scheduled by ra-np-pEDF-fav and as a corollary of this lemma, we obtain a new, tighter, performance bound of uniprocessor non-preemptive EDF scheduling—we improve the (previously known (Andersson and Easwaran 2010)) bound from *three* to *two*. This is an interesting result in its own right. For the special case that each task requests at most one resource, the bound of LP-EE-vpr collapses to $4 \times (1 + \lceil \frac{|R|}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$.

Contributions and significance of this work. This paper presents two contributions. First, for the problem of scheduling implicit-deadline sporadic tasks that share multiple resources on t-type heterogeneous multiprocessors, no previous algorithm exists and hence our algorithm, LP-EE-vpr, is the first for this problem with a proven speed competitive ratio. Second, for the problem of non-preemptive scheduling of tasks on a uniprocessor, this paper improves the previously known (Andersson and Easwaran 2010) speed competitive ratio of uniprocessor non-preemptive EDF algorithm from *three* to *two*. This improvement is presented because it is a natural by-product of our proof of the performance bound of LP-EE-vpr.

Organization of the paper. The rest of the paper is organized as follows. Section 2 briefs the system model. Section 3 gives an overview of our algorithm and Sect. 4 describes the algorithm in detail. Section 5 proves the speed competitive ratio of ra-np-pEDF-fav (an intermediate result) as well as the speed competitive ratio of LP-EE-vpr (the main result of this paper). Section 6 discusses useful properties of the proposed algorithm and finally, Sect. 7 concludes.

2 System model

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 $R = \{r_1, r_2, \ldots, r_\rho\}$ of ρ resources on a t-type heterogeneous multiprocessor platform $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ comprising m processors of which m_k processors are of type-k, where $k \in \{1, 2, \ldots, t\}$.

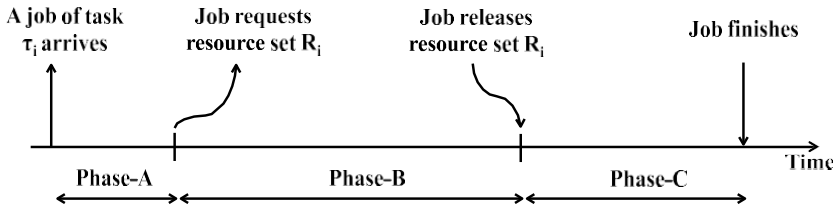


Fig. 1 Categorization of the execution of a task that requests a resource into three phases

In the task set, each implicit-deadline sporadic task τ_i generates a (potentially infinite) sequence of *jobs*, with the first job arriving at any time and subsequent jobs arriving *at least* T_i time units apart (referred to as *minimum inter-arrival time*). Each job of a task τ_i has to complete its execution within $D_i = T_i$ time units from its arrival (referred to as *deadline*).

In the computing platform, a processor $\pi_p \in \pi$ belongs to one of the t different types of processors. The computing platform consists of m_k processors of type-k, where $k \in \{1, 2, \ldots, t\}$, i.e., it consists of m_1 processors of type-1, m_2 processors of type-2, …, m_t processors of type-t; hence, $m_1 + m_2 + \ldots + m_t = m$.

The tasks share resources from the set $R = \{r_1, r_2, \ldots, r_\rho\}$ of ρ resources. Specifically, for each task $\tau_i \in \tau$, *there is a resource set* $R_i \subseteq R$ such that for each job of τ_i, during one phase of its execution, the job requests to hold the resource set R_i exclusively, that is, at all times, when a job of τ_i holds R_i, no other job holds any resource in R_i. We assume that each job of task τ_i may request the corresponding resource set R_i at most once during its execution and further each job must request all the resources in this set together. We also assume that a job of a task can execute on at most one processor at any given time.

For a *job of a task* τ_i *such that* $R_i \neq \emptyset$ we categorize the execution into three phases as follows. Let phase-A execution of a job of task τ_i denote the execution the job performs from when it arrives until it requests R_i. Let phase-B execution of a job of task τ_i denote the execution the job performs from when it requests R_i until it releases R_i. Let phase-C execution of a job of task τ_i denote the execution the job performs from when it releases R_i until it finishes execution. This is illustrated in Fig. 1. For a *job of a task* τ_i *such that* $R_i = \emptyset$ we categorize its execution into a single phase, phase-A, which denotes the entire execution of the job, i.e., the execution the job performs from when it arrives until it finishes execution.

In our model, we allow a job of task τ_i to migrate at the time when it requests the resource set R_i and when it releases the resource set R_i but the job is not allowed to migrate at other times. (This assumption is similar to previous work on D-PCP (Rajkumar et al. 1988).) We assume that the processors a job migrates to/from is determined by the task that generated the job and consequently, all jobs of the same task migrate between the same processors. Specifically, phase-A executions of all jobs of task τ_i are assigned to the same processor (let $p_{i,a}$ denote this processor). Analogously, phase-B executions of all jobs of task τ_i are assigned to the same processor (let $p_{i,b}$ denote this processor). Phase-C executions of all jobs of task τ_i are assigned to the same processor (let $p_{i,c}$ denote this processor). Thus, all jobs of task τ_i only migrate between these (at most three) processors. Note that for a given task τ_i, it can

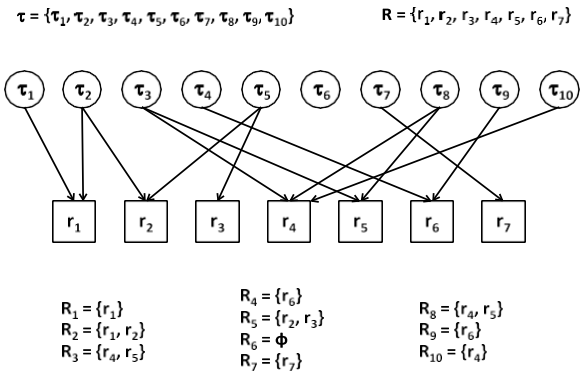
happen that the processors $p_{i,a}$, $p_{i,b}$ and $p_{i,c}$ are of different types. We refer to such assumption of migration as *restricted migration*.

Since a job executing within a phase cannot migrate, we can speak about the execution time of a job in a phase for a given processor type. Let CA_i^k denote an upper bound on the execution time of phase-A of a job of task τ_i if this phase-A execution is assigned to a processor of type-k. Analogously, let CB_i^k denote an upper bound on the execution time of phase-B of a job of task τ_i if this phase-B execution is assigned to a processor of type-k. Let CC_i^k denote an upper bound on the execution time of phase-C of a job of task τ_i if this phase-C execution is assigned to a processor of type-k. For convenience, we introduce the symbol C_i^k as follows: For a task τ_i whose jobs access a resource set, $C_i^k \stackrel{\text{def}}{=} CA_i^k + CB_i^k + CC_i^k$. For a task τ_i whose jobs do not access a resource set, $C_i^k \stackrel{\text{def}}{=} CA_i^k$. Intuitively, C_i^k denotes an upper bound on the execution time of a job of task τ_i *if* all its phases would be assigned to a processor of type-k. For convenience, we also use the following notation. The *utilization* of a task τ_i on a type-k processor (assuming that all phases of the task are assigned to processors of type-k) is denoted by u_i^k and is defined as $u_i^k \stackrel{\text{def}}{=} \frac{C_i^k}{T_i}$.

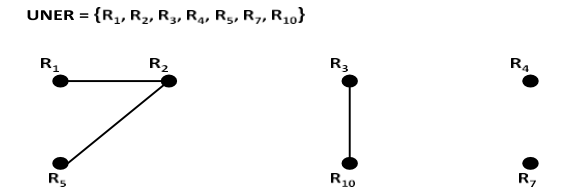
As mentioned earlier, in this work, we consider *implicit-deadline* sporadic tasks, that is, for each task $\tau_i : D_i = T_i$. In some parts of our discussion, however, we discuss *constrained-deadline* sporadic tasks, that is, for each task $\tau_i : D_i \leq T_i$. For a constrained-deadline sporadic task τ_i, its *density* on a type-k processor is denoted as δ_i^k and is defined by $\delta_i^k \stackrel{\text{def}}{=} \frac{C_i^k}{\min(D_i, T_i)} = \frac{C_i^k}{D_i}$.

Recall that tasks request resources from set R of resources. This is illustrated in Fig. 2a. It is helpful to introduce auxiliary variables and form a graph describing the potential conflicts of resource requests. Let UNER denote the set of unique non-empty resource sets that tasks request. Formally UNER is defined as $\text{UNER} \stackrel{\text{def}}{=} \bigcup_{\tau_i \in \tau \wedge R_i \neq \emptyset} \{R_i\}$. The graph (V, E), with the set of vertices V and the set of edges E is then formed as follows: (i) there is a function FUN that maps an element in UNER to an element in V, and this is a one-to-one correspondence, and (ii) there is an edge between vertex V_{k1} and vertex V_{k2} if and only if $(\text{FUN}^{-1}(V_{k1})) \cap (\text{FUN}^{-1}(V_{k2})) \neq \emptyset$. Such a graph is shown in Fig. 2b. Let $\text{PV} = \{\text{PV}_1, \text{PV}_2, \ldots, \text{PV}_{|\text{PV}|}\}$ denote the set of $|\text{PV}|$ connected components of this graph. The connected components in a graph can be found in linear time using a standard technique (Hopcroft and Tarjan 1973). For a connected component and the set of connected components, we introduce symbols that describe potential conflicts between resource sets. Let P_j denote the set of unique non-empty resource sets that correspond to the vertices in PV_j. We refer to P_j as a *resource request partition*. Formally, $P_j \stackrel{\text{def}}{=} \{\text{UNER}_k : (\text{UNER}_k \in \text{UNER}) \wedge (\text{FUN}(\text{UNER}_k) \in \text{PV}_j)\}$. Let P be defined as follows: $P \stackrel{\text{def}}{=} \{P_j : \text{PV}_j \in \text{PV}\}$ and let MAXP be defined as follows: $\text{MAXP} \stackrel{\text{def}}{=} \max_{P_j \in P} |P_j|$. These concepts are illustrated in Fig. 2c. Let $\mathbb{R}(P_j)$ be defined as follows: $\mathbb{R}(P_j) \stackrel{\text{def}}{=} \{r_\ell : \exists \tau_i \in \tau \text{ such that } R_i \in P_j \text{ and } r_\ell \in R_i\}$. Informally, $\mathbb{R}(P_j)$ denotes all the resources in resource request partition P_j. We refer to $\mathbb{R}(P_j)$ as a *resource partition*.

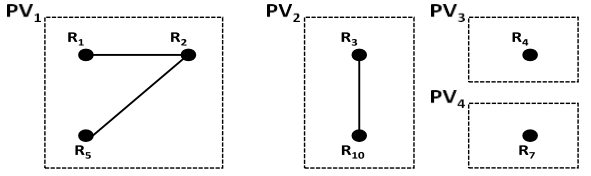
Note that for each $P_j \in P$, $P_{j'} \in P$ such that $P_j \neq P_{j'}$, both of the following statements are true:



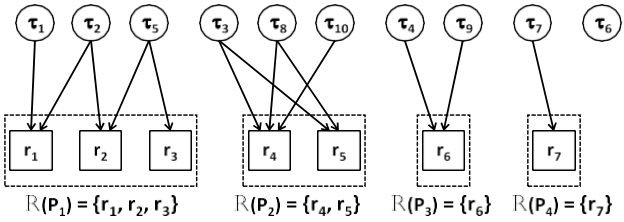
(a) A visualization of the resources requested by tasks. An arrow from a task to a resource indicates that the task requests the resource.



(b) Construction of the graph from resource sets requested. Each vertex has an associated resource set.



(c) The set $PV = \{PV_1, PV_2, PV_3, PV_4\}$ of connected components. From PV, we obtain set $P = \{P_1, P_2, P_3, P_4\}$ of resource request partitions where $P_1 = \{R_1, R_2, R_5\}$, $P_2 = \{R_3, R_{10}\}$, $P_3 = \{R_4\}$, $P_4 = \{R_7\}$ and $\mathrm{MAXI} = 3$.



(d) The resource partition $\mathbb{R}(P_j)$ for each resource request partition P_j.

Fig. 2 An example to illustrate the resource request information of tasks and how to construct the graph and connected components using this information

ra-np-pEDF (Resource-Aware-Non-Preemptive-Partitioned-EDF) algorithm

Assumptions:	Consider R, a set of resources and a task set such that whenever a task performs execution it must be holding its resource set. Consider a computer platform with $\lvert UNER \rvert$ or more identical processors.
Before run-time:	Select $\lvert UNER \rvert$ processors and call them ACT-processors and call the other processors NACT-processors. For ACT-processors, associate a resource set to each ACT-processor so that the following holds: (i) no two ACT-processors are associated with the same resource set in UNER and (ii) no two resource sets in UNER are associated with the same ACT processor and (iii) every ACT processor is associated with exactly one resource set in UNER and (iv) every resource set in UNER is associated with exactly one ACT processor. For NACT-processors, do not associate any resource set to these processors. A task is assigned to an ACT-processor whose associated resource set is equal to the resource set of the task.
At run-time:	A job is said to be *active* at time t if the arrival time of the job is $\leq t$ and the finishing time of the job is $\geq$. A job J is said to be *eligible* at time t if it is active and no currently executing job holds a resource set that intersects with the resource set of job J. At each instant t, consider the set of active jobs in earliest-deadline-first order. If the current job is eligible then start its execution on the processor to which its corresponding task is assigned. If the current job is not eligible then do not execute it; consider the next job in the set of active jobs.

Fig. 3 The description of ra-np-pEDF algorithm

1. $R(P_j) \cap R(P_{j'}) = \emptyset$
2. $\forall R_i \in P_j$, $\forall R_{i'} \in P_{j'}$ it holds that $R_i \cap R_{i'} = \emptyset$

Also, note that for each task τ_i, it holds that there is at most one element $P_k \in P$ such that $R_i \in P_k$. Hence, the tasks in the given task set can be partitioned based on the resources they request. With this partitioning, it holds that for two tasks in different partitions, there is no resource that they share. This is illustrated in Fig. 2d.

Figures 3 and 4 show two algorithms ra-np-pEDF and ra-np-pEDF-fav which we will use as building blocks in the design of our new algorithm. The algorithm ra-np-pEDF runs on an identical multiprocessor whereas the algorithm ra-np-pEDF-fav runs on a t-type heterogeneous multiprocessor. The algorithm ra-np-pEDF executes a task on a processor specific for its resource set and hence the execution of a task can only be delayed because of execution of another task whose resource set intersects with it. The algorithm ra-np-pEDF-fav works like ra-np-pEDF but ra-np-pEDF-fav assumes that each task is assigned to a processor that is its favorite type (a type such that there is no other type for which the task has smaller execution time).

3 Overview of our algorithm

The algorithm, LP-EE-vpr, can be summarized in four steps as shown in Fig. 5. Steps 1–3 are executed *before* run-time and only step 4 is executed *at* run-time. Step 1 produces subtasks from each task so that if the deadlines are met for these subtasks

ra-np-pEDF-fav (Resource-Aware-Non-Preemptive-Partitioned-EDF-Favorite-Processor) algorithm	
Assumptions:	Consider R, a set of resources and a task set such that whenever a task performs execution it must be holding its resource set. Consider a t-type heterogeneous multiprocessor platform with $\lfloor UNER \rfloor$ or more identical processors of each type.
Before run-time:	For each type $k \in \{1, 2, \ldots, t\}$ select $\lfloor UNER \rfloor$ processors and call them ACT-processors and call the other processors NACT-processors. For ACT-processors, associate a resource set to each ACT-processor so that for each type $k \in \{1, 2, \ldots, t\}$ the following holds: (i) no two ACT-processors of type-k are associated with the same resource set in UNER and (ii) no two resource sets in UNER are associated with the same ACT processor of type-k and (iii) every ACT processor of type-k is associated with exactly one resource set in UNER and (iv) every resource set in UNER is associated with exactly one ACT processor of type-k. For NACT-processors, do not associate any resource set to these processors. A task is assigned to an ACT-processor whose associated resource set is equal to the resource set of the task and whose type is such that there is no other type where the task has smaller execution time.
At run-time:	A job is said to be *active* at time t if the arrival time of the job is $\leq t$ and the finishing time of the job is $\geq$. A job J is said to be *eligible* at time t if it is active and no currently executing job holds a resource set that intersects with the resource set of job J. At each instant t, consider the set of active jobs in earliest-deadline-first order. If the current job is eligible then start its execution on the processor to which its corresponding task is assigned. (Note that since every task is assumed to be assigned to its favorite processor type, the jobs of each task execute on the respective favorite processor types). If the current job is not eligible then do not execute it; consider the next job in the set of active jobs.

Fig. 4 The description of ra-np-pEDF-fav algorithm

then the original task meets its deadline as well. Step 2 creates virtual processors from physical processors. Step 3 assigns subtasks to virtual processors. Finally, in Step 4, jobs are dispatched at run-time. We now provide more details about each of these steps.

Step 1—Creation of subtasks. Categorize the execution of a task that requests a resource set into three phases as shown in Fig. 6. The three phases of execution are phase-A, phase-B and phase-C, as mentioned in Sect. 2. Then create three constrained-deadline sporadic subtasks (one corresponding to each phase) out of each implicit-deadline sporadic task that requests a resource set and make different scheduling provisions for each of these subtasks. A task which does not request a resource set is categorized into phase-A alone and only one subtask is created for such a task.

For a task *that requests* a resource set, 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. Section 4.1 shows how these constrained-deadline subtasks are created and their parameters (worst-case execution times, minimum inter-arrival times and deadlines) are determined.

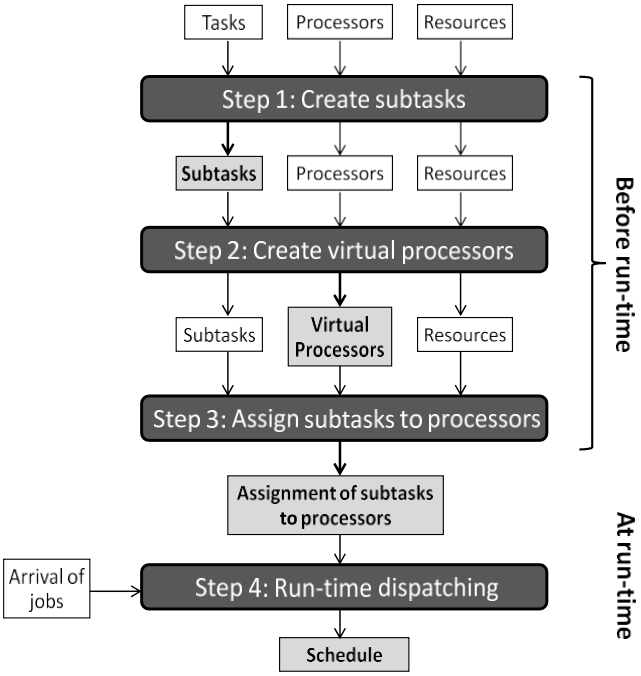


Fig. 5 Four steps of our new algorithm LP-EE-vpr. Each of the three first steps takes three inputs and produces outputs. Some outputs are identical to the inputs (e.g., in Step 1, "processors" are inputs and they are outputs) and they are marked in white. Some outputs, however, are produced (e.g., "subtasks" are outputs from Step 1 and they are not inputs to Step 1) and they are marked in gray

Step 2—Creation of virtual processors. Virtual processors are logical constructs, used as task assignment targets by our algorithm.[1] Create two sets of virtual processors, namely, VP_{AC} and VP_B virtual processors from the given physical processors. The VP_B virtual processors are then grouped together so as to create $|P|$ virtual processor groups, one group for every resource request partition in P. The virtual processor group corresponding to the resource request partition P_j is denoted as $\mathrm{Group}_B[j]$. The specification of the virtual processors (i.e., number of virtual processors and their speeds), their creation and grouping technique is discussed in Sect. 4.2.

Step 3—Task assignment. The phase-A and phase-C subtasks created from a task τ_i are assigned to the same virtual processor in VP_{AC}. The phase-B subtask created from task τ_i requesting the resource set R_i which is in a resource request partition,

[1] 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 processor arbitrarily close to $\frac{1}{f}$ and in practice, S need rarely be impractically short (Bletsas and Andersson 2009).

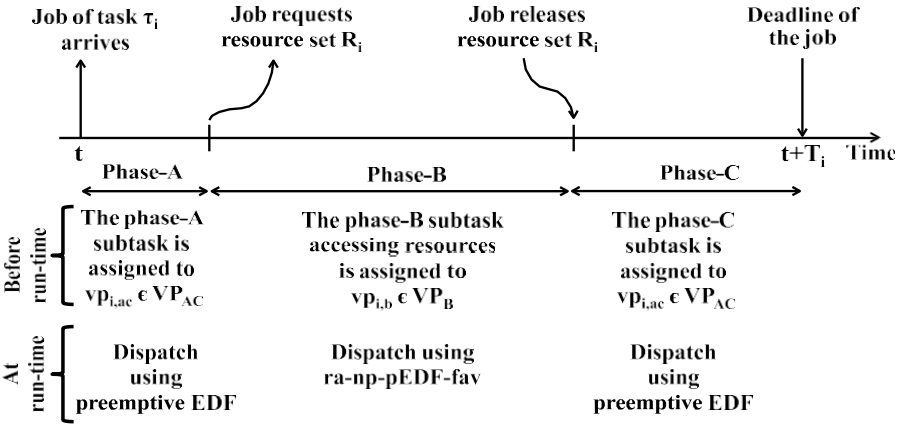


Fig. 6 Three execution phases of a job along with the design-time and run-time decisions of LP-EE-vpr algorithm

say P_j, i.e., $R_i \subseteq R(P_j)$, is assigned to $\text{Group}_B[j]$. This step is discussed in detail in Sect. 4.3.

Step 4—Task scheduling. All phase-A and phase-C subtasks are scheduled using *preemptive* Earliest-Deadline-First (EDF) algorithm (Liu and Layland 1973) on their assigned virtual processors in VP_{AC}. All phase-B subtasks that are assigned to virtual processors in a VP_B virtual processor group are scheduled using ra-np-pEDF-fav.

Remark: In the rest of the manuscript, to avoid tedium, we skip special mentioning of tasks that *do not request* a resource set (which are split into only phase-A) and hence, for such tasks, the discussion about phase-B and phase-C does not apply.

4 The new algorithm: LP-EE-vpr

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

4.1 Creating the subtasks

LP-EE-vpr creates subtasks. It creates three subtasks from each task, one subtask for each phase of the task and it assigns minimum inter-arrival time, deadlines and execution times to each subtask. Specifically, each subtask will have t different execution times, one for each type of processor and each subtask will also have t different deadlines, one for each type of processor. When a subtask is assigned to a processor, only one of its execution times is applicable and only one of its deadlines is applicable; the type of processor on which the subtask is assigned determines this. The algorithm assigns parameters (minimum inter-arrival time, deadlines and execution times) to subtasks and assigns subtasks to processors so that when subtasks are scheduled at

Table 1 The three constrained-deadline subtasks that are derived from a given implicit-deadline sporadic task τ_i that requests a resource set. For a task that does not request a resource set, only one subtask corresponding to phase-A execution, i.e., $\tau_{i,A}$, is derived and hence for such a task, $\tau_{i,B}$ and $\tau_{i,C}$ do not exist

Subtasks of τ_i	WCET on type-k	Deadline on type-k	Minimum inter-arrival time
$\tau_{i,A}$	$C^k_{i,A} = CA^k_i$	$D^k_{i,A} = \frac{C^k_{i,A}}{C^k_i} \times \frac{T_i}{2}$	$T_{i,A} = T_i$
$\tau_{i,B}$	$C^k_{i,B} = CB^k_i$	$D^k_{i,B} = \frac{T_i}{2}$	$T_{i,B} = T_i$
$\tau_{i,C}$	$C^k_{i,C} = CC^k_i$	$D^k_{i,C} = \frac{C^k_{i,C}}{C^k_i} \times \frac{T_i}{2}$	$T_{i,C} = T_i$

run-time it holds that (i) the three subtasks of a task execute in sequence (that is, one of the subtasks of τ_i must finish execution before another subtask of τ_i can start execution) and (ii) if each subtask meets its deadline then the task from which it was formed meets its deadline as well.

From each implicit-deadline sporadic task $\tau_i \in \tau$, the algorithm creates three constrained-deadline sporadic subtasks denoted by $\tau_{i,A}$, $\tau_{i,B}$ and $\tau_{i,C}$ corresponding to phase-A, phase-B and phase-C execution of task τ_i , respectively. In the rest of the paper, the *subscript A, B* and *C* will be used in the notations corresponding to phase-A, phase-B and phase-C *subtasks*, respectively. Also, the superscript *k* will be used in the notations corresponding to a processor of type-k. For example, $C^k_{i,A}$,$C^k_{i,B}$ and $C^k_{i,C}$denote the worst-case execution time of task $\tau_i \in \tau$ on a processor of type-k before requesting the resource set R_i (phase-A subtask), while holding the resource set (phase-B subtask) and after releasing the resource set (phase-C subtask), respectively.[2]

The parameters of the three subtasks $\tau_{i,A}$, $\tau_{i,B}$ and $\tau_{i,C}$ that are derived from the corresponding task $\tau_i \in \tau$ are set as shown in Table 1. It is easy to see that the following property holds: for each task $\tau_i \in \tau$ and for each pair of processor types *k* and $k^!$, it holds that $D^k_{i,A} + D^{k^!}_{i,B} + D^k_{i,C} \leq T_i = D_i$. This implies that if for each task $\tau_i \in \tau$ it holds that phase-A and phase-C of τ_i are assigned to the same processor type then if at run-time we can ensure that all subtasks meet their deadlines then the corresponding tasks meet all their deadlines as well. Indeed, later in Sect. 4.3 while assigning subtasks to processors, we ensure that this property holds.

We group these derived subtasks into the following task sets:

$$\tau^A = \left\{\tau_{i,A} \mid i \in \{1, 2, \ldots, n\}\right\}$$

$$\tau^{B,\mathbb{R}(P_j)} = \left\{\tau_{i,B} \mid i \in \{1, 2, \ldots, n\} \wedge R_i \subseteq \mathbb{R}(P_j)\right\}$$

$$\tau^C = \left\{\tau_{i,C} \mid i \in \{1, 2, \ldots, n\}\right\}$$

Note that $\tau_{i,A}$ refers to a *subtask* and τ^A refers to a *set of subtasks*. Analogously, for $\tau_{i,B}$ and $\tau^{B,\mathsf{R}(P_j)}$. Analogously, for $\tau_{i,C}$ and τ^C.

[2]Recall that, for a task that does not request a resource set, $C^k_{i,B}$ and $C^k_{i,C}$ do not exist.

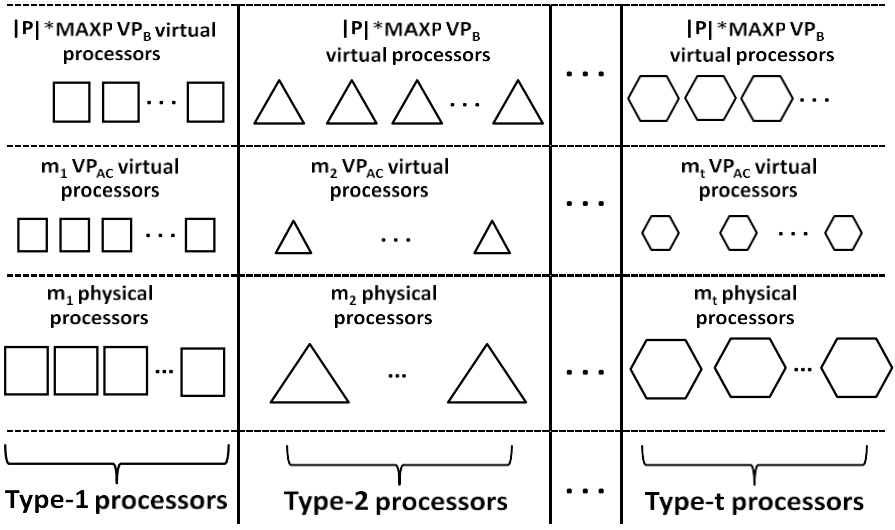


Fig. 7 $m + t \times |P| \times$ MAXP virtual processors created from m physical processors of a t-type heterogeneous multiprocessor platform

As opposed to the given task set τ which contains implicit-deadline sporadic tasks, these derived task sets contain constrained-deadline sporadic subtasks. Also, observe that the task set τ^A is derived such that, on a processor of type-k, the density of every subtask $\tau_{i,A} \in \tau^A$ is twice the utilization of the corresponding task $\tau_i \in \tau$. Formally,

$$\forall \tau_{i,A} \in \tau^A : \quad \delta^k_{i,A} = \frac{C^k_{i,A}}{D^k_{i,A}} = \frac{C^k_{i,A}}{\frac{C^k_{i,A} \times T_i}{C^k_i \times 2}} = \frac{2 \times C^k_i}{T_i} = 2 \times u^k_i \quad \text{of } \tau_i \in \tau \qquad (1)$$

Analogously, it can be seen that, the density of every subtask $\tau_{i,C} \in \tau^C$ is twice the utilization of the corresponding task $\tau_i \in \tau$.

4.2 Creating virtual processors from a t-type heterogeneous multiprocessor platform

In this section, we describe the creation of virtual processors from the given physical processors of a t-type heterogeneous multiprocessor platform.

We create $m + t \times |P| \times$ MAXP virtual processors from the given m physical processors as shown in Fig. 7. The main idea is as follows. We treat physical processors of each type as an identical multiprocessor platform and create a certain number of virtual processors of the corresponding type from this platform. To be precise, m_k physical processors of type-k are treated as an identical multiprocessor platform and $m_k \times |P| \times$ MAXP virtual processors of type-k are created from them (see different columns in Fig. 7, separated by "solid vertical lines") and ordered as shown in Fig. 7. Now, if we look at the first and the second row in Fig. 7 (separated by "dashed horizontal lines"), each of these rows represent a t-type heterogeneous multiprocessor platform of virtual processors—the first row represents a t-type heterogeneous multiprocessor platform with $t \times |P| \times$ MAXP virtual processors of which

$|P| \times \text{MAXP}$ virtual processors are of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) and the second row represents a t-type heterogeneous multiprocessor platform with m virtual processors of which m_k virtual processors are of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$). In this manner, $m + t \times |P| \times \text{MAXP}$ virtual processors are created from m physical processors of a t-type heterogeneous multiprocessor platform. Precisely, we create the virtual processors with following specifications:

- m virtual processors (denoted as VP_{AC}): From m_k physical processors of type-k, we create m_k virtual processors of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) each of speed $\frac{1}{1+\text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}$ times the speed of a corresponding physical processor of type-k. So, in total, m such virtual processors are created from m physical processors. These are later used to schedule phase-A and phase-C subtasks and are referred to as 'VP_{AC} virtual processors'.
- $t \times |P| \times \text{MAXP}$ virtual processors (denoted as VP_B): From m_k physical processors of type-k, we create $|P| \times \text{MAXP}$ virtual processors of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) each of speed $\frac{\text{MAXP}}{1+\text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}$ times the speed of a corresponding physical processor of type-k. So, in total, $t \times |P| \times \text{MAXP}$ such virtual processors are created from m physical processors of a t-type heterogeneous multiprocessor platform. These are later used to schedule phase-B subtasks and are referred to as 'VP_B virtual processors'.

In other words, from each processor type, say type-k, we create $m_k + |P| \times \text{MAXP}$ virtual processors of type-k, i.e., m_k VP_{AC} virtual processors of type-k and $|P| \times \text{MAXP}$ VP_B virtual processors of type-k. The way these virtual processors are created is as follows. From each processor π_p of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$):

- first create one VP_{AC} virtual processor of type-k of speed $\frac{1}{1+\text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}$ times the speed of π_p
- then create $\lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil$ VP_B virtual processors of type-k of speed $\frac{\text{MAXP}}{1+\text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}$ times the speed of π_p

Lemma 1 *The earlier specified set of virtual processors, VP_{AC} and VP_B, can be created from the given t-type heterogeneous multiprocessor platform π as described above. This procedure to create the virtual processors ensures that the capacity of a virtual processor comes from a single physical processor.*

Proof The proof is a direct consequence of the fact that each physical processor of type-k can emulate one VP_{AC} virtual processor of type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) and $\lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil$ VP_B virtual processors of type-k, as per the specifications of the virtual processors. Indeed, for each $\pi_p \in \pi$, we have

$$\underbrace{1 \times \frac{1}{1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}}_{\text{VP}_{AC} \text{ virtual processor}}$$

$$+\underbrace{\left\lceil \frac{|P| \times \text{MAXP}}{m_k} \right\rceil \times \frac{\text{MAXP}}{1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil}}_{\text{VP}_B \text{ virtual processors}} = 1$$

Thus, m_k physical processors of type-k can emulate m_k VP_{AC} virtual processors of type-k and $\lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil \times m_k \geq |P| \times \text{MAXP}$ VP_B virtual processors of type-k. Overall, m physical processors of a t-type heterogeneous multiprocessor platform can emulate m VP_{AC} virtual processors and $t \times |P| \times \text{MAXP}$ VP_B virtual processors.

From the above discussion, it is trivial to see that no virtual processor is created using two or more physical processors and hence it holds that the capacity of a virtual processor comes from a single physical processor alone. Hence the proof. □

We now describe the rest of the steps in the algorithm, LP-EE-vpr, for assigning and scheduling the tasks that *share resources* on t-type heterogeneous multiprocessors with the help of pseudo-code.

4.3 Pseudo-code of LP-EE-vpr

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

On line 1, it creates the sets τ^A, $\tau^{B,\mathsf{R}(P_j)}$ and τ^C of constrained-deadline sporadic subtasks from the given set τ of implicit-deadline sporadic tasks as described in Sect. 4.1.

On line 2, it creates m VP_{AC} and $t \times |P| \times \text{MAXP}$ VP_B virtual processors from the given t-type heterogeneous multiprocessor platform of m physical processors as discussed in Sect. 4.2.

On line 3, it groups $t \times |P| \times \text{MAXP}$ VP_B virtual processors into $|P|$ *groups of* VP_B *virtual processors*; each group contains $t \times \text{MAXP}$ VP_B virtual processors, with MAXP virtual processors of each type, i.e., MAXP virtual processors of type-1, MAXP virtual processors of type-2 and so on. Each group of virtual processors, denoted by $\text{Group}_B[j]$, where $j = \{1, 2, \ldots, |P|\}$, is used for scheduling phase-B subtasks that access a subset of resources from resource partition $\mathsf{R}(P_j)$.

On line 4, it assigns the set of phase-A subtasks, τ^A, to VP_{AC} virtual processors using LP-EE algorithm[3] (Baruah 2004c). The algorithm, LP-EE, is designed for non-migratively scheduling a set of implicit-deadline sporadic tasks that *do not* share resources on t-type heterogeneous multiprocessors. The internals of LP-EE and its performance bound are described in detail in Baruah (2004c). The average-case performance of LP-EE is discussed in Raravi et al. (2013). Therefore, we only give an overview of LP-EE here. The algorithm, LP-EE, has two steps: first, it assigns the tasks to processors and then schedules the tasks on each processor using preemptive EDF. The task assignment step works as follows:

[3]We selected LP-EE because it is simple to implement and easy to explain and it has a proven speed competitive ratio. However, a couple of other algorithms can be used instead as discussed later in Sect. 6.5

Algorithm 1: LP-EE-vpr(τ, $\Pi(m_1, m_2, ..., m_t)$, R): for scheduling implicit-deadline sporadic tasks that share resources on t-type heterogeneous multiprocessors

```
// Lines 1-10 execute before run-time; line 11
   executes at run-time.
```

1 Create the sets τ^A, $\tau^{B,R(P_j)}$ and τ^C of constrained-deadline sporadic subtasks from the given task set τ of implicit-deadline sporadic tasks as described in Sect. 4.1.

2 Create m VP_{AC} and $t \times |P| \times$ MAXP VP_B virtual processors from the given m physical processors of a t-type heterogeneous multiprocessor platform as described in Sect. 4.2.

3 Form $|P|$ virtual processor groups out of $t \times |P|$ VP_B virtual processors as follows. Take MAXP VP_B virtual processors of each type (i.e., $t \times$ MAXP virtual processors, in total) and form a virtual processor group, $Group_B[1]$; then take MAXP more VP_B virtual processors of each type and form another virtual processor group, $Group_B[2]$ and so on. Overall, we will have $|P|$ VP_B virtual processor groups; every group containing $t \times$ MAXP VP_B virtual processors; MAXP virtual processors of each type.

4 Assign all the subtasks $\tau_{i,A} \in \tau^A$ to VP_{AC} virtual processors using the algorithm LP-EE (Baruah 2004c) (more details in the description of the algorithm in Sect. 4.3).

5 **foreach** $\tau_i \in \tau$ **do**

6 | **if** $(\exists j : j \in \{1, 2, \ldots, |P|\} \wedge R_i \in R(P_j))$ **then**

7 | | Assign $\tau_{i,B}$ to the MAXP virtual processors in the j'th VP_B virtual processor *group*, $Group_B[j]$, on which subtask $\tau_{i,B}$ has the smallest execution time.

8 | **end**

9 **end**

10 Assign every subtask $\tau_{i,C} \in \tau^C$ to that virtual processor in VP_{AC} to which the corresponding subtask $\tau_{i,A} \in \tau^A$ has been assigned on line 4.

11 Schedule the subtasks of τ^A and τ^C that are assigned on each VP_{AC} virtual processor using *preemptive* EDF on that virtual processor. Schedule the subtasks of $\tau_{i,B}$ that are assigned to each VP_B virtual processor group using ra-np-pEDF-fav, on the respective virtual processor group.

– The assignment problem is formulated as Mixed Integer Linear Program (MILP) and then relaxed to Linear Program (LP). The LP formulation is solved using an LP solver (such as GUROBI Optimizer 2012 or IBM ILOG CPLEX 2012). Tasks are then assigned to the processors according to the values of the respective *indicator variables* in the solution provided by the solver. Using certain tricks (Potts 1985), it is shown that there exists a solution (for example, the solution that lies on the vertex of the feasible region) to the LP formulation in which all but at most $m-1$ tasks are integrally assigned to processors where m denotes the number of processors.

– The remaining at most $m - 1$ tasks are integrally assigned on the remaining capacity of the processors using "exhaustive enumeration".

The abbreviation LP-EE comes from the fact that the algorithm makes use of **L**inear **P**rogramming and **E**xhaustive **E**numeration techniques to provide the solution (Baruah 2004c).

On lines 5–9, it assigns all the phase-B subtasks that request the "related" resources, i.e., resources that belong to the same resource partition, to the same VP_B virtual processor group. Specifically, all the subtasks requesting (a subset of) resources from resource partition $\mathsf{R}(P_j)$, $\forall j \in \{1, 2, \ldots, |P|\}$, are assigned to the virtual processors in the j'th VP_B virtual processor group, $\text{Group}_B[j]$, on which these subtasks have the smallest execution time.

On line 10, it assigns every phase-C subtask, $\tau_{i,C}$, to that virtual processor in VP_{AC} to which the corresponding phase-A subtask, $\tau_{i,A}$, has been assigned. Such an assignment does not endanger the schedulability of the tasks assigned on the VP_{AC} virtual processors as there is a precedence constraint between these subtasks—this is formally proven later in Lemma 9 in Sect. 5.3. 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 (recall that the capacity of a virtual processor comes from a single physical processor—Lemma 1) and only the phase-B subtask *might* have to execute on a 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 11, it schedules the subtasks of τ^A and τ^C that are assigned to each VP_{AC} virtual processor using *preemptive* EDF on that virtual processor. It schedules the subtasks of $\tau^{B,\mathsf{R}(P_j)}$ that are assigned to each VP_B virtual processor group, $\text{Group}_B[j]$, using ra-np-pEDF-fav, on the respective virtual processor group. Recall that all the tasks in $\tau^{B,\mathsf{R}(P_j)}$ request (a subset of) resources from resource partition $\mathsf{R}(P_j)$ and hence are assigned to VP_B virtual processor group, $\text{Group}_B[j]$.

For *preemptive EDF* scheduling, the following result is well-known (an easily obtained generalization of the result shown in Liu and Layland 1973), which we make use of while proving the performance of LP-EE-vpr.

Lemma 2 (Utilization-based schedulability test) *Let $\tau[\pi_p]$ denote the tasks assigned on a processor π_p of type-k. If $\sum_{\tau_i \in \tau[\pi_p]} u_i^k \leq 1$ and tasks are scheduled with preemptive EDF on π_p then all deadlines are met.*

Note that in Algorithm 1, lines 1–10 execute *before* run-time and only line 11 executes *at* run-time. The algorithm, LP-EE-vpr, is named after the fact that it makes use of the algorithm, LP-EE, for assigning some of the subtasks on **v**irtual **pr**ocessors.

5 Performance analysis of LP-EE-vpr algorithm

In this section, we prove the speed competitive ratio of the proposed algorithm. But first we present notations (in Sect. 5.1), then prove the speed competitive ra-

tio of ra-np-pEDF (in Sect. 5.2). After that, we present some useful results (a previously known and a few new results, in Sect. 5.3) and the speed competitive ratio of ra-np-pEDF-fav that are used later while proving the speed competitive ratio of LP-EE-vpr (in Sect. 5.4).

5.1 Notations

Let $\Pi(m_1, m_2, \ldots, m_t)$ denote a t-type heterogeneous multiprocessor platform of m processors of which m_k processors are of type-k, where $k \in \{1, 2, \ldots, t\}$ and $\forall k : m_k > 0$; note that $m = m_1 + m_2 + \cdots + m_t$.

Let $\Pi(m_1, m_2, \ldots, m_t) \times (s_1, s_2, \ldots, s_t)$ denote a t-type platform in which, for each $k \in \{1, 2, \ldots, t\}$, the speed of every type-k processor is s_k times the speed of a corresponding type-k processor in $\Pi(m_1, m_2, \ldots, m_t)$, where $s_k > 0$ is a real number. As a special case of the above, we use $\Pi(m_1, m_2, \ldots, m_t) \times (s, s, \ldots, s)$ to denote a t-type platform in which, for each $k \in \{1, 2, \ldots, t\}$, the speed of every type-k processor is s times the speed of a corresponding type-k processor in $\Pi(m_1, m_2, \ldots, m_t)$, where $s > 0$ is a real number. For convenience, we sometimes denote $\Pi(m_1, m_2, \ldots, m_t) \times (s, s, \ldots, s)$ as $\Pi(m_1, m_2, \ldots, m_t) \times s$.

If τ is a task set and $y, y^{!}, y^{!!}$ are positive real numbers then we let the symbol $\text{mulCDT}(\tau, y, y^{!}, y^{!!})$ denote a task set where for each task in τ: its execution time is multiplied by y; its deadline is multiplied by $y^{!}$ and its minimum inter-arrival time is multiplied by $y^{!!}$.

We will now introduce three types of predicates (i) predicates that state if a task set is *schedulable* for a given scheduling algorithm, (ii) predicates that state if a task set is *feasible* and (iii) predicates that state if a task set is *schedulable* for a given scheduling algorithm *according to a certain class of schedulability tests*.

For a task set τ where tasks do not share any resources, we let the symbol $\text{sched}(A, \tau, \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that if τ is scheduled by algorithm A on platform $\Pi(m_1, m_2, \ldots, m_t)$ then for each set of jobs that τ can generate according to the model in Sect. 2, it holds that all jobs meet their deadlines and the constraint of restricted migration is satisfied (which in this case means that no migration is allowed because there are only phase-A executions).

For a task set τ where tasks may share resources in R, we let the symbol $\text{sched}(A, \tau, R, \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that if τ is scheduled by algorithm A on platform $\Pi(m_1, m_2, \ldots, m_t)$ then for each set of jobs that τ can generate according to the model in Sect. 2, it holds that all jobs meet their deadlines and the constraint of restricted migration is satisfied and there is no instant where a resource in R is held by more than one job. Analogously, for a task set τ where tasks may share resources in R, and where P_j is a resource set and $\tau^{B,\mathsf{R}(P_j)}$ is the task set derived as in Sect. 4.1, we let the symbol $\text{sched}(A, \tau^{B,\mathsf{R}(P_j)}, \mathsf{R}(P_j), \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that if $\tau^{B,\mathsf{R}(P_j)}$ is scheduled by algorithm A on platform $\Pi(m_1, m_2, \ldots, m_t)$ then for each set of jobs that $\tau^{B,\mathsf{R}(P_j)}$ can generate according to the model in Sect. 2, it holds that all jobs meet their deadlines and the constraint of restricted migration is satisfied (which in this case means that no migration is allowed because there are only phase-B executions) and there is no instant where a resource in $\mathsf{R}(P_j)$ is held by more than one job.

For a task set τ where tasks do not share any resources, we let the symbol nmig-feas$(\tau, \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that for each set of jobs that τ can generate according to the model in Sect. 2, it holds that there exist a schedule that meets all deadlines of all jobs and the constraint of restricted migration is satisfied (which in this case means that no migration is allowed because there are only phase-A executions).

For a task set τ where tasks may share resources in R, we let the symbol rmig-feas$(\tau, R, \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that for each set of jobs that τ can generate according to the model in Sect. 2, it holds that there exist a schedule that meets all deadlines of all jobs and the constraint of restricted migration is satisfied and there is no instant where a resource in R is held by more than one job. Analogously, for a task set τ where tasks may share resources in R, and where P_j is a resource set and $\tau^{B,\mathsf{R}(P_j)}$ is the task set derived from τ as in Sect. 4.1, we let the symbol rmig-feas$(\tau^{B,\mathsf{R}(P_j)}, \mathsf{R}(P_j), \Pi(m_1, m_2, \ldots, m_t))$ be a predicate that indicates that for each set of jobs that $\tau^{B,\mathsf{R}(P_j)}$ can generate according to the model in Sect. 2, it holds that there exist a schedule that meets all deadlines of all jobs and the constraint of restricted migration is satisfied (which in this case means that no migration is allowed because there are only phase-B executions) and there is no instant where a resource in $\mathsf{R}(P_j)$ is held by more than one job.

Some of these predicates will be used by adding a suffix "$-\delta$" to the scheduling algorithm or algorithm class where applicable, for example, for non-migrative scheduling of constrained-deadline sporadic subtasks corresponding to different phases. Such predicates with suffix $-\delta$ signify that the *schedulability* of the task set 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 type-k, to meet deadlines, it must hold that $\sum_{\tau_i \in \tau[\pi_p]} \delta_i^k \leq 1$. For example, sched(A-$\delta$, τ, $\Pi(m_1, m_2, \ldots, m_t)$) denotes a predicate that is true if for the task set τ which does *not* share resources is ascertained schedulable by algorithm A on platform $\Pi(m_1, m_2, \ldots, m_t)$ using the above mentioned density-based schedulability test.

We use a function `create-fav-taskset`$(\tau, \Pi(m_1, m_2, \ldots, m_t))$. This function takes a task set τ as input in which each task $\tau_i \in \tau$ is characterized by its minimum inter-arrival time T_i and its deadline D_i and its t worst-case execution times (one WCET on each processor type) $C_i^1, C_i^2, \ldots, C_i^t$. The function outputs a task set $\tau^!$ in which each task $\tau_i^! \in \tau^!$ is characterized by its minimum inter-arrival time $T_i^!$ and its deadline $D_i^!$ and its single worst-case execution time $C_i^!$. For each task $\tau_i^! \in \tau^!$, it sets $T_i^! = T_i$ and $D_i^! = D_i$ and $C_i^! = \min_{k \in \{1,2,\ldots,t\}} C_i^k$. Informally, from the given task set, it constructs another task set in which, the execution time of each task is equal to the execution time of its corresponding task on its favorite processor type and the minimum inter-arrival time of each task is equal to the minimum inter-arrival time of its corresponding task and the deadline of each task is equal to the deadline of its corresponding task.

We also use a function `create-fav-platform`$(\tau, \Pi(m_1, m_2, \ldots, m_t), m^!)$ which generates a multiprocessor platform with $m^!$ identical processors where each processor is such that for each task in τ it holds that the execution time is as if it executed on the processor type in $\Pi(m_1, m_2, \ldots, m_t)$ for which its execution time is the smallest.

5.2 The speed competitive ratio of ra-np-pEDF-fav algorithm

Recall from step 11 of Algorithm 1 in Sect. 4.3, that the algorithm LP-EE-vpr uses the algorithm ra-np-pEDF-fav (defined in Sect. 2) to schedule phase-B execution of tasks. For this reason, we need to show that ra-np-pEDF-fav has a finite speed competitive ratio. We will do so by showing the speed competitive ratio of ra-np-pEDF and later show (in Sect. 5.3) how it translates to a heterogeneous multiprocessor.

As a by-product of our proof of the speed competitive ratio of ra-np-pEDF, we obtain a corollary which is a new result on the speed competitive ratio of non-preemptive EDF on a single processor. Previously, it was known that the speed competitive ratio of non-preemptive EDF on a single processor is at most three. In this section, we see that it is at most two.

We start by proving a relationship between feasibility of a set of tasks that executes always holding a resource and the feasibility of this task set on an identical multiprocessor.

Lemma 3 $\forall \tau, \forall \Pi(m_1, m_2, \ldots, m_t), \forall R, v \geq |\text{UNER}|$ *such that* τ *is an implicit-deadline sporadic task set and* $\forall \tau_i \in \tau : R_i \neq \emptyset$ *and* $\forall \tau_i \in \tau$ *it holds that whenever* τ_i *executes it holds resource set* R_i*:*

$$\text{rmig-feas}(\tau, R, \Pi(m_1, m_2, \ldots, m_t)) \Rightarrow \text{rmig-feas}(\texttt{create\_fav\_taskset}(\tau, \Pi(m_1, m_2, \ldots, m_t)), R, \texttt{create\_fav\_platform}(\tau, \Pi(m_1, m_2, \ldots, m_t), v))$$

Proof The lemma follows from two observations:

1. The task set τ is such that at each instant, there can be at most $|\text{UNER}|$ jobs executing at this instant.
2. If a task set is feasible then giving each task an execution time as if it executed on the processor where its execution time is smallest cannot violate feasibility.

The truth of first observation can be seen as follows: Suppose that the first observation was false. Then there would exist a feasible schedule such that there exists an instant where $|\text{UNER}| + 1$ or more jobs execute at that instant. Then it follows that there are two or more jobs that execute holding the same resource set in UNER. Consequently, this schedule is not feasible. Hence the first observation is true.

The truth of the second observation can be seen as follows: For a feasible schedule, if we change the execution time of a job to a smaller value then we can simply idle the processor so that the schedule for all other jobs are the same and hence feasibility is not violated by reducing the execution time of a job. D

We can then show (below) how feasibility relates to schedulability of ra-np-pEDF.

Lemma 4 $\forall \tau, \forall \Pi(m_1, m_2, \ldots, m_t), \forall R, \forall x \geq 1, v \geq |\text{UNER}|$ *such that is an implicit-deadline sporadic task set and* $\forall \tau_i \in \tau : R_i \neq \emptyset$ *and* $\forall \tau_i \in \tau$ *it holds that whenever* τ_i *executes it holds resource set* R_i*:*

$$\text{rmig-feas}\big(\texttt{create\_fav\_taskset}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big), R,$$
$$\texttt{create\_fav\_platform}\big(\tau, \Pi(m_1, m_2, \ldots, m_t), v\big)\big)$$
$$\Rightarrow \quad \text{sched}\Big(\text{ra-np-pEDF},$$
$$\text{mulCDT}\Big(\texttt{create\_fav\_taskset}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big),$$
$$\frac{1}{2 \times v \times x}, \frac{1}{x}, 1\Big), R,$$
$$\texttt{create\_fav\_platform}\big(\tau, \Pi(m_1, m_2, \ldots, m_t), v\big)\Big)$$

Proof The proof is by contradiction. Suppose that the claim is false. Then there exists a τ, $\Pi(m_1, m_2, \ldots, m_t)$, R, $x \geq 1$, $v \geq$ UNER such that τ is an implicit-deadline sporadic task set and $\forall \tau_i \in \tau : R_i \neq \emptyset$ and $\forall i \in$ it holds that whenever τ_i executes it holds resource set R_i for which it holds that ((2) is true) $\wedge$ ((3) is false) where (2) and (3) are defined as:

$$\text{rmig-feas}\big(\texttt{create\_fav\_taskset}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big), R,$$
$$\texttt{create\_fav\_platform}\big(\tau, \Pi(m_1, m_2, \ldots, m_t), v\big)\big) \qquad (2)$$
$$\text{sched}\Big(\text{ra-np-pEDF},$$
$$\text{mulCDT}\Big(\texttt{create\_fav\_taskset}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big),$$
$$\frac{1}{2 \times v \times x}, \frac{1}{x}, 1\Big), R,$$
$$\texttt{create\_fav\_platform}\big(\tau, \Pi(m_1, m_2, \ldots, m_t), v\big)\Big) \qquad (3)$$

Note that both (2) and (3) make statements about a task set and a multiprocessor platform with identical processors. Since it is an identical multiprocessor, we do not need to specify execution times as depending on processor type and hence, we let C_j denote the execution time of task τ_j for the task set in (2). Because of our assumption that the task set τ is an implicit-deadline sporadic task set and because (2), it follows that:

$$(C_1 \leq D_1 = T_1) \wedge (C_2 \leq D_2 = T_2) \wedge \cdots \wedge (C_n \leq D_n = T_n) \qquad (4)$$

We will now discuss the implication of (3) being false. Since (3) is false, it follows that there exist an assignment of arrival times to jobs such that a deadline is missed. Let t_0 denote the earliest time when a deadline is missed. Let us choose a job whose deadline expires at time t_0 and let us call it DMJ (deadline miss job). Let t_2 denote

the arrival time of the job DMJ. Let τ_k denote the task that generated DMJ. From (4) we get:

$$C_k \leq D_k = T_k \tag{5}$$

Let $S(\tau_k)$ be defined as:

$$S(\tau_k) = \{\tau_{k'} : (\tau_{k'} \in \texttt{create\_fav\_taskset}(\tau, \Pi(m_1, m_2, \ldots, m_t))) \wedge (\tau_{k'} \neq \tau_k) \wedge (|R_{k'} \cap R_k| \geq 1)\} \tag{6}$$

$S(\tau_k)$ is the set of tasks that can share a resource with task τ_k. If $|S(\tau_k)| = 0$ then DMJ would have executed immediately when it arrived and because of (5) and because $\frac{1}{2 \times v \times x} \leq \frac{1}{x}$ it would follow that τ_k would have met its deadline and this would be a contradiction. Hence, we know that:

$$|S(\tau_k)| \geq 1 \tag{7}$$

Let $\text{BLT}(\tau_k, \text{DMJ}, t_2)$ be defined as:

$$\text{BLT}(\tau_k, \text{DMJ}, t_2) = \{\tau_{k'} : (\tau_{k'} \in S(\tau_k)) \wedge (\text{there is a job of task } \tau_{k'} \text{ executing at time } t_2)\} \tag{8}$$

BLT(τ_k, DMJ, t_2) is the set of tasks in $S(\tau_k)$ such that these tasks executed at time t_2. Let BLJ(τ_k, DMJ, t_2) be defined as the set of jobs generated by BLT(τ_k, DMJ, t_2) such that the jobs executed at time t_2. Clearly, for each element in BLJ(τ_k, DMJ, t_2), there is a corresponding element in BLT(τ_k, DMJ, t_2). Intuitively, BLT means "blocking-tasks" and BLJ means "blocking-jobs".

Let us explore two cases:

1. $|\text{BLT}(\tau_k, \text{DMJ}, t_2)| \geq 1$

   Let t_1 denote maximum of the finishing times of the jobs in BLJ(τ_k, DMJ, t_2). Let us choose a job in BLJ(τ_k, DMJ, t_2) that finished at time t_1 and let the task that generated this job be denoted τ_i and let t_b denote the starting time of this job. From the definition of t_2, we have $t_b \leq t_2$.

   We will now discuss the time interval $[t_b, t_0)$ and we let L denote the duration of this time interval (that is $L = t_0 - t_b$). During this time, at each instant t , at least one of the following is true: (i) the set of jobs executing at time t includes a job of task τ_i or (ii) the set of jobs executing at time t includes DMJ (the job of task τ_k) or (iii) the set of jobs executing at time t includes a job of a task in $S(\tau_k) \setminus \{\tau_i\}$

   Since we had a deadline miss, we obtain that:

$$\frac{C_i}{2 \times v \times x} + \max\left(0, \left\lfloor \frac{L - \frac{D_k}{x}}{T_k} \right\rfloor + 1\right) \times \frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{L - \frac{D_{i'}}{x}}{T_{i'}} \right\rfloor + 1\right) \times \frac{C_{i'}}{2 \times v \times x} > L \tag{9}$$

Using (4) on (9) and rewriting yields:

$$\frac{C_i}{2 \times v \times x} + \max\left(0, \left\lfloor \frac{L - \frac{T_k}{x} + T_k}{T_k} \right\rfloor\right) \times \frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times \frac{C_{i'}}{2 \times v \times x} > L \quad (10)$$

Since at time t_2, there is a job of task τ_i executing, it follows that this job of task τ_i started to execute at time t_2 or earlier. Since t_b is defined as the starting time of this job we obtain: $t_b \leq t_2$. This gives us:

$$t_0 - t_2 \leq t_0 - t_b \quad (11)$$

Note that $t_0 - t_2 = D_k/x$. Also note that $t_0 - t_b = L$. This gives us:

$$\frac{D_k}{x} \leq L \quad (12)$$

Using (4) on (12) yields:

$$\frac{T_k}{x} \leq L \quad (13)$$

We will now discuss the implication of (2) being true. Since (2) is true, it follows that for every possible assignment of arrival times to jobs in the task set `create-fav-taskset`$(\tau, \Pi(m_1, m_2, \ldots, m_t))$, all deadlines are met on an identical multiprocessor with v processors and where it is required that the resource sharing constraints are respected. Let us consider the case that tasks arrive periodically. Then it follows that there exist a time when a job of task τ_i arrives. And since deadlines are met, this job must have finished at most T_i time units later and hence there exist a time when a job of task τ_i executed. Let tarbegin denote the time when this job of task τ_i started to execute and let tarend denote the time $L^!$ time units later. (Clearly, tarend − tarbegin $= L^!$.) We can also observe that for some other task $\tau_{i^!}$, it holds that at each instant, a job of task $\tau_{i^!}$ arrives at most $T_{i^!}$ time units later. Hence, during this time interval [tarbegin, tarend] (of duration $L^!$), there are at least

$$\left\lfloor \frac{L'}{T_{i'}} \right\rfloor \quad (14)$$

jobs of task $\tau_{i^!}$ with arrival time within [tarbegin, tarend].

Hence, during this time interval [tarbegin, tarend] (of duration $L^!$), there are at least

$$\max\left(0, \left\lfloor \frac{L' - D_{i'}}{T_{i'}} \right\rfloor\right) \quad (15)$$

jobs of task $\tau_{i^!}$ with arrival time and deadline within [tarbegin, tarend].

Using (4) gives us that during this time interval [tarbegin, tarend] (of duration $L^!$), there are at least

$$\max\left(0, \left\lfloor \frac{L' - T_{i'}}{T_{i'}} \right\rfloor\right) \tag{16}$$

jobs of task $\tau_{i^!}$ with arrival time and deadline within [tarbegin, tarend].

Note that for the feasible schedule, at each instant, there can be at most v jobs executing (because otherwise there would be two jobs executing while holding the same resource set). With this observation and using (16) gives us:

$$\min(C_i, L') + \max\left(0, \left\lfloor \frac{L' - T_k}{T_k} \right\rfloor\right) \times C_k + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{L' - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \leq v \times L' \tag{17}$$

This expression (17) applies for any choice of $L^!$. Applying it with $L^! = 2L \times x$ gives us:

$$\min(C_i, 2L \times x) + \max\left(0, \left\lfloor \frac{2L \times x - T_k}{T_k} \right\rfloor \times C_k\right) + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \leq v \times 2L \times x \tag{18}$$

Let us explore two cases.

(a) $C_i > 2L \times x$

We will show that if this case is true then it contradicts (2). Note that τ_i and τ_k share at least one resource and hence it is impossible for them to execute simultaneously. Recall that (2) states that there is a feasible schedule so in this feasible schedule, it must hold that τ_i and τ_k never execute simultaneously. With reasoning similar to (16), we obtain that, for the case of periodically arriving tasks, in a time interval of duration $2T_k$, there is at least one job of task τ_k that has arrived and whose deadline expired. Hence, from (2), it follows that in a time interval of duration $2T_k$, there is at least one job of task τ_k that has executed entirely. Using (13) and the condition of the case gives us that $C_i > 2T_k$. Hence, during the time when a job of τ_i executes, there is at least one job of τ_k executing. But this is impossible because τ_i and τ_k share resources. Hence, this is a contradiction.

(b) $C_i \leq 2L \times x$

Using the condition of the case on (18) and dividing by $2v \times x$ gives us:

$$\frac{C_i}{2 \times v \times x} + \max\left(0, \left\lfloor \frac{2L \times x - T_k}{T_k} \right\rfloor\right) \times \frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times \frac{C_{i'}}{2 \times v \times x} \leq L \tag{19}$$

Combining (19) with (10) and multiplying by $2v\,x$ and observing that the resulting equation has the same term on both sides and this can be canceled out gives us:

$$\begin{aligned}
&\max\left(0, \left\lfloor \frac{2L \times x - T_k}{T_k} \right\rfloor\right) \times C_k \\
&\quad + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&< \max\left(0, \left\lfloor \frac{L - \frac{T_k}{x} + T_k}{T_k} \right\rfloor\right) \times C_k \\
&\quad + \sum_{\tau_{i'} \in (S(\tau_k) \setminus \{\tau_i\})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \qquad (20)
\end{aligned}$$

Observe that the left-hand side can be rewritten as a single sum. And also observe that the right-hand side can be rewritten as a single sum. Rewriting each of the sums as two sums gives us:

$$\begin{aligned}
&\sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} \le L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&\quad + \sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} > L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&< \sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} \le L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&\quad + \sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} > L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&\qquad (21)
\end{aligned}$$

Observing that the last sum is zero and relaxing the second term on the left-hand side gives us:

$$\begin{aligned}
&\sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} \le L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&< \sum_{\tau_{i'} \in ((S(\tau_k) \cup \{\tau_k\}) \setminus \{\tau_i\}) \wedge (T_{i'} \le L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \\
&\qquad (22)
\end{aligned}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in \left(\left(S(\tau_k) \cup \{\tau_k\}\right) \setminus \{\tau_i\}\right)\right) \wedge \left(T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'}\right)$$
$$\wedge \left(\max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} < \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'}\right) \tag{23}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in \left(\left(S(\tau_k) \cup \{\tau_k\}\right) \setminus \{\tau_i\}\right)\right) \wedge \left(T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'}\right)$$
$$\wedge \left(2L \times x - T_{i'} < L - \frac{T_{i'}}{x} + T_{i'}\right) \tag{24}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in \left(\left(S(\tau_k) \cup \{\tau_k\}\right) \setminus \{\tau_i\}\right)\right) \wedge \left(\frac{T_{i'}}{x} \leq L\right)$$
$$\wedge \left((2x - 1) \times L < (2 - 1/x) \times T_{i'}\right) \tag{25}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in \left(\left(S(\tau_k) \cup \{\tau_k\}\right) \setminus \{\tau_i\}\right)\right)$$
$$\wedge \left((2x - 1) \times \frac{T_{i'}}{x} < (2 - 1/x) \times T_{i'}\right) \tag{26}$$

This is a contradiction.

2. $|\mathrm{BLT}(\tau_k, \mathrm{DMJ}, t_2)| = 0$

From the case, we obtain that there is no task in $S(\tau_k)$ such that this task executed at the time when DMJ arrived. We will now discuss the time interval $[t_2, t_0)$. We let L denote the duration of this time interval. Clearly,

$$L = \frac{D_k}{x} \tag{27}$$

Using (4) on (27) yields:

$$L = \frac{T_k}{x} \tag{28}$$

During this time interval $[t_2, t_0)$, at each instant, either (i) the set of jobs executing includes a job of task τ_k or (ii) the set of jobs executing includes a job of a task in $S(\tau_k)$.

Since we had a deadline miss, we obtain that:

$$\frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in S(\tau_k)} \max\left(0, \left\lfloor \frac{L - \frac{D_{i'}}{x}}{T_{i'}} \right\rfloor + 1\right) \times \frac{C_{i'}}{2 \times v \times x} > L \tag{29}$$

Using (4) on (29) and rewriting yields:

$$\frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in S(\tau_k)} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times \frac{C_{i'}}{2 \times v \times x} > L \quad (30)$$

We can discuss the implication of (2) being true just like in Case 1 and this gives us:

$$\max\left(0, \left\lfloor \frac{2L \times x - T_k}{T_k} \right\rfloor\right) \times \frac{C_k}{2 \times v \times x} + \sum_{\tau_{i'} \in S(\tau_k)} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times \frac{C_{i'}}{2 \times v \times x} \leq L \quad (31)$$

Combining (31) with (30) and multiplying by $2v \times x$ and observing that $\max(0, \lfloor \frac{2L \times x - T_k}{T_k} \rfloor) = \max(0, \lfloor \frac{2T_k - T_k}{T_k} \rfloor) = 1$ and rewriting gives us:

$$\sum_{\tau_{i'} \in S(\tau_k)} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} < \sum_{\tau_{i'} \in S(\tau_k)} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \quad (32)$$

Rewriting each of the sums as two sums gives us:

$$\sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} + \sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} > L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} < \sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} + \sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} > L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \quad (33)$$

Observing that the last sum is zero and relaxing the second term on the left-hand side gives us:

$$\sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'})} \max\left(0, \left\lfloor \frac{2L \times x - T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} < \sum_{\tau_{i'} \in (S(\tau_k)) \wedge (T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'})} \max\left(\left\lfloor \frac{L - \frac{T_{i'}}{x} + T_{i'}}{T_{i'}} \right\rfloor\right) \times C_{i'} \quad (34)$$

Observing (34) gives us that there is at least one term on the left-hand side that is smaller than the corresponding term on the right-hand side. This together with (28) give us that there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in S(\tau_k)\right) \wedge \left(T_{i'} \leq L - \frac{T_{i'}}{x} + T_{i'}\right) \wedge \left(L = \frac{T_k}{x}\right) \wedge \left(2L \times x - T_{i'} < L - \frac{T_{i'}}{x} + T_{i'}\right) \tag{35}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in S(\tau_k)\right) \wedge \left(\frac{T_{i'}}{x} \leq L\right) \wedge \left(L = \frac{T_k}{x}\right) \wedge \left((2x-1) \times L < \left(2 - \frac{1}{x}\right) \times T_{i'}\right) \tag{36}$$

Hence, there exists a task $\tau_{i'}$ such that

$$\left(\tau_{i'} \in S(\tau_k)\right) \wedge \left(\frac{T_{i'}}{x} \leq L\right) \wedge \left((2x-1) \times \frac{T_{i'}}{x} < \left(2 - \frac{1}{x}\right) \times T_{i'}\right) \tag{37}$$

This is a contradiction.

Hence, if the lemma is false then we obtain a contradiction. Consequently, the lemma is true. D

Combining the two previous lemmas gives us (below) a relationship between feasibility on a heterogeneous multiprocessor and schedulability of ra-np-pEDF.

Lemma 5 *$\forall \tau$, $\forall \Pi(m_1, m_2, \ldots, m_t)$, $\forall R$, $\forall x \geq 1$, $v \geq |\text{UNER}|$ such that τ is an implicit-deadline sporadic task set and $\forall \tau_i \in \tau : R_i \neq \emptyset$ and $\forall \tau_i \in \tau$ it holds that whenever τ_i executes it holds resource set R_i:*

$$\text{rmig-feas}\left(\tau, R, \Pi(m_1, m_2, \ldots, m_t)\right) \Rightarrow \text{sched}\Bigg(\text{ra-np-pEDF}, \text{mulCDT}\Bigg(\texttt{create\_fav\_taskset}\left(\tau, \Pi(m_1, m_2, \ldots, m_t)\right), \frac{1}{2 \times v \times x}, \frac{1}{x}, 1\Bigg), R, \texttt{create\_fav\_platform}\left(\tau, \Pi(m_1, m_2, \ldots, m_t), v\right)\Bigg)$$

Proof Follows from Lemmas 3 and 4. D

Corollary 1 *Consider an implicit-deadline sporadic tasks set that is offline non-preemptive feasible on a single processor. If this task set is scheduled by the algorithm* non-preemptive EDF *on a processor with twice the speed then this task set is schedulable.*

Proof Follows from specializing Lemma 5 with $v = 1$ and $x = 1$ and a system with a single processor and a single resource and all tasks share this single resource and whenever a task executes it needs to hold this resource. D

5.3 Useful results

In this section, we present a previously known (Lemma 6) result and some new results (Lemmas 7–10 and Corollary 2) that we use while proving the speed competitive ratio of our algorithm, LP-EE-vpr, in Sect. 5.4.

Lemma 6 states that the speed competitive ratio of algorithm, LP-EE, proposed in Baruah (2004c) is two. The algorithm, LP-EE, non-migratively schedules a set of implicit-deadline sporadic tasks that do not share resources on a t-type heterogeneous multiprocessor platform.

Lemma 6 (*From Theorem 3 in* Baruah 2004c)

$$\text{nmig-feas}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big) \Rightarrow \text{sched}\big(\text{LP-EE}, \tau, \Pi(m_1, m_2, \ldots, m_t) \times 2\big)$$

We now show that if an implicit-deadline sporadic task set τ in which tasks do not share resources is non-migrative-offline schedulable on a t-type heterogeneous multiprocessor platform $\Pi(m_1, m_2, \ldots, m_t)$ then the constrained-deadline sporadic task set τ^A (in which tasks do not share resources as well) which is derived from τ (as described in Sect. 4.1) is also non-migrative offline schedulable but on platform $\Pi(m_1, m_2, \ldots, m_t) \times 2$ (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 of type-k, the density $\delta^k_{i,A}$ of a task $\tau_{i,A} \in \tau^A$ is always twice the utilization u^k_i of the corresponding task $\tau_i \in \tau$ (see Expression (1)). Hence, the density of the task $\tau_{i,A} \in \tau^A$ on a twice faster platform $\Pi(m_1, m_2, \ldots, m_t) \times 2$ is equal to the utilization of the corresponding task $\tau_i \in \tau$ on platform $\Pi(m_1, m_2, \ldots, m_t)$.

Lemma 7

$$\text{nmig-feas}\big(\tau, \Pi(m_1, m_2, \ldots, m_t)\big) \Rightarrow \text{nmig-feas-}\delta\big(\tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 2\big)$$

Proof Suppose that the left-hand side, nmig-feas$(\tau, \Pi(m_1, m_2, \ldots, m_t))$, is true. Then let us arbitrarily choose one set of jobs JS generated by τ. Since it holds that nmig-feas$(\tau, \Pi(m_1, m_2, \ldots, m_t))$ is true, there exists a non-migrative-offline schedule for this job set on platform $\Pi(m_1, m_2, \ldots, m_t)$ in which all the deadlines are met. Since jobs do not migrate and since there is only one phase per job (because there are no resource requests) and since it holds (as stated in Sect. 2) that all phase-A executions of a given task execute on the same processor, we can form, from this schedule, a partitioning of the tasks. In this schedule, let $\tau[\pi_p]$ be the set of tasks assigned to processor π_p. This gives us:

$$\forall k: \quad \forall \pi_p \text{ of type-k} \in \Pi(m_1, m_2, \ldots, m_t): \sum_{\tau_i \in \tau[\pi_p]} u_i^k \leq 1 \qquad (38)$$

We now show that there must also exist a non-migrative-offline schedule for the derived task set τ^A on platform $\Pi(m_1, m_2, \ldots, m_t) \times 2$ 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 corresponding task $\tau_{i,A} \in \tau^A$. Also, from Expression (1), we know that, on a processor of type-k, where $k \in \{1, 2, \ldots, t\}$, density δ^k of task $\tau_{i,A} \in \tau^A$ is twice the utilization u_i^k of the corresponding task $\tau_i \in \tau$.

Let us assign the tasks in τ^A on platform $\Pi(m_1, m_2, \ldots, m_t) \times 2$ as follows: if $\tau_i \in \tau$ is assigned to a processor of type-k, say π_p of type-k $\in \Pi(m_1, m_2, \ldots, m_t)$, in the non-migrative-offline schedule which meets all deadlines, then we assign its corresponding task $\tau_{i,A}$ to the corresponding processor in the faster platform, i.e., to processor π_p of type-k $\in \Pi(m_1, m_2, \ldots, m_t) \times 2$. 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) and from Expressions (1) and (38), we get:

$$\forall k: \quad \forall \pi_p \text{ of type-k} \in \Pi(m_1, m_2, \ldots, m_t) \times 2: \sum_{\tau_{i,A} \in \tau^A[\pi_p]} \delta_{i,A}^k \leq 1 \qquad (39)$$

which satisfies density-based schedulability test of non-migrative EDF on a t-type heterogeneous multiprocessor platform. We can repeat this reasoning for any choice of JS. Hence, τ^A is non-migrative-offline feasible on $\Pi(m_1, m_2, \ldots, m_t) \times 2$. Hence the lemma. D

Corollary 2

$$\text{nmig-feas-}\delta\left(\tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 2\right) \Rightarrow \text{nmig-feas}\left(\tau, \Pi(m_1, m_2, \ldots, m_t)\right)$$

Proof Follows from reasoning analogous to the reasoning for the proof of Lemma 7. D

The following lemma is an extension of Lemma 6 obtained by applying density-based test instead of utilization-based test and on twice faster platforms.

Lemma 8

$$\text{nmig-feas-}\delta\left(\tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 2\right)$$

$$\Rightarrow \quad \text{sched}\left(\text{LP-EE-}\delta, \tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 4\right)$$

Pr(.e., nmig-feas-$\delta(\tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 2)$ is true. Using Corollary 2, we obtain nmig-feas$(\tau, \Pi(m_1, m_2, \ldots, m_t))$. Then, from Lemma 6, the predicate sched(LP-EE, $\tau, \Pi(m_1, m_2, \ldots, m_t) \times 2$) must hold true. From Expression (1), we know that on a processor of type-k, density $\delta^k_{i,A}$ of every task $\tau_{i,A} \in \tau^A$ is twice the utilization u^k_i of the corresponding task $\tau_i \in \tau$, and hence sched(LP-EE-δ, $\tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 4$) must hold true as well, from a similar reasoning as used in Lemma 7. Hence the proof. D

The following lemma states that if tasks from τ^A are preemptive EDF schedulable on a processor π_p of type-k 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 type-k. If $\tau^A[\pi_p]$ is preemptive-EDF schedulable ascertainable with a density-based test on π_p, i.e.,*

$$\delta^k_{\tau^A[\pi_p]} \stackrel{\text{def}}{=} \sum_{\tau_{i,A} \in \tau^A[\pi_p]} \delta^k_{i,A} \leq 1$$

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 preemptive-EDF schedulable on processor π_p of type-k.

Proof We know that the task set $\tau^A[\pi_p]$ is preemptive-EDF schedulable, ascertainable with a density-based test, on processor π_p of type-k, i.e., $\delta^k_{\tau^A[\pi_p]} \leq 1$. To show that $\tau^A[\pi_p] \cup \tau^C[\pi_p]$ is schedulable on processor π_p, it is sufficient to show that the *demand-bound function*,[4] 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 $\delta^k_{\tau^A[\pi_p]} \times t$ at any instant t (Baruah et al. 1990).

The following holds for every phase-A subtask $\tau_{i,A} \in \tau^A$ and respective phase-C subtask $\tau_{i,C} \in \tau^C$:

$$\text{DBF}\left(\{\tau_{i,A}\} \cup \{\tau_{i,C}\}, t\right) \leq t \times \delta^k_{i,A} = t \times \frac{C^k_{i,A}}{D^k_{i,A}} \qquad (40)$$

[4]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: DBF$(\tau, t) = \sum_{\tau_i \in \tau}$ dbf(τ_i, t) (Baruah et al. 1990).

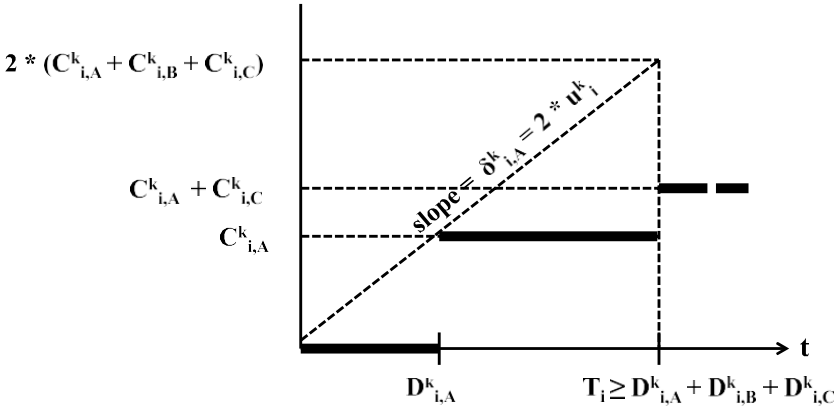


Fig. 8 Assigning phase-C subtasks to the same virtual processor as the respective phase-A subtasks (earlier assigned using a density-based test) preserves schedulability

This can be verified from Fig. 8 since the maximum "slope" to any point in the graph of DBF($\{\tau_{i,A}\} \cup \{\tau_{i,C}\}, t$) from the origin is $\delta^k_{i,A} = \frac{C^k_{i,A}}{D^k_{i,A}}$ (which is equal to $2 \times u^k$ of $\tau_i \in \tau$, as per our choice of $D^k_{i,A}$), at abscissa $t = D^k_{i,A}$. Summing Expression (40) for all the subtasks $\tau_{i,A} \in \tau^A[\pi_p]$ and the corresponding subtasks $\tau_{i,C} \in \tau^C[\pi_p]$ yields:

$$\mathrm{DBF}\left(\tau^A[\pi_p] \cup \tau^C[\pi_p], t\right) \leq t \times \sum_{\tau_{i,A} \in \tau^A[\pi_p]} \delta^k_{i,A} = t \times \delta^k_{\tau^A[\pi_p]}$$

Hence the proof. D

We will now prove a guarantee on the schedulability of ra-np-pEDF-fav.

Lemma 10 *Let τ denote an implicit-deadline sporadic task set. Let R denote the set of resources in the system. Let P_j denote one resource request partition of R and let $\mathbb{R}(P_j)$ denote the resources belonging to this resource request partition.*

$$\begin{aligned} &\text{rmig-feas}\left(\tau, R, \Pi(m_1, m_2, \ldots, m_t)\right) \\ &\Rightarrow \quad \text{sched}\left(\text{ra-np-pEDF-fav}, \tau^{B,\mathbb{R}(P_j)}, \mathbb{R}(P_j), \right. \\ &\qquad \left. \Pi\left(|P_j|, |P_j|, \ldots, |P_j|\right) \times 4 \times |P_j|\right) \end{aligned} \tag{41}$$

Proof Let $\tau^!$ denote the subset of tasks in τ that request a resource set in P_j. Let $\tau^{!!}$ denote a set of tasks derived from $\tau^!$ but where a task in $\tau^{!!}$ does not perform any execution before requesting a resource set and a task in $\tau^{!!}$ does not perform any execution after releasing a resource set.

Then consider the three claims below:

1. $\text{rmig-feas}(\tau, R, \Pi(m_1, m_2, \ldots, m_t)) \Rightarrow \text{rmig-feas}(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t))$

2. rmig-feas$(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t)) \Rightarrow$ sched(ra-np-pEDF-fav, $\tau''^{B,\mathbb{R}(P_j)}$, $\mathbb{R}(P_j), \Pi(|P_j|, |P_j|, \ldots, |P_j|) \times 4 \times |P_j|)$
3. $\tau''^{B,\mathbb{R}(P_j)} = \tau^{B,\mathbb{R}(P_j)}$

If we can prove these three claims then the correctness of the lemma follows. Hence, we prove the claims below.

Proving 1. This claim follows from the fact that feasibility cannot be violated by only considering a subset of the tasks and by only considering a subset of the resources and by only considering some of the execution of a task.

Proving 2. Applying Lemma 5 with the task set $\tau^{!!}$ and the resource set $\mathsf{R}(P_j)$ and with $x = 2$ and $v = |P_j|$ yields:

$$\begin{aligned}
&\text{rmig-feas}(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t)) \\
&\quad \Rightarrow \quad \text{sched}\Big(\text{ra-np-pEDF}, \\
&\qquad\qquad \text{mulCDT}\Big(\texttt{create_fav_taskset}(\tau'', \Pi(m_1, m_2, \ldots, m_t)), \\
&\qquad\qquad 1, \frac{1}{2}, 1\Big), \mathbb{R}(P_j), \\
&\qquad\qquad \texttt{create_fav_platform}(\tau'', \Pi(m_1, m_2, \ldots, m_t), v) \times 4 \times |P_j|\Big)
\end{aligned} \tag{42}$$

The order in which the functions mulCDT and `create_fav_taskset` are applied can be changed without affecting the result. And the result of the function `create_fav_platform` when taken $\tau^{!!}$ as input is the same as when taken mulCDT$(\tau^{!!}, 1, \frac{1}{2}, 1)$ as input. This gives us:

$$\begin{aligned}
&\text{rmig-feas}(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t)) \\
&\quad \Rightarrow \quad \text{sched}\Big(\text{ra-np-pEDF}, \texttt{create_fav_taskset}\Big(\text{mulCDT}\Big(\tau'', 1, \frac{1}{2}, 1\Big), \\
&\qquad\qquad \Pi(m_1, m_2, \ldots, m_t)\Big), \mathbb{R}(P_j), \\
&\qquad\qquad \texttt{create_fav_platform}\Big(\text{mulCDT}\Big(\tau'', 1, \frac{1}{2}, 1\Big), \\
&\qquad\qquad \Pi(m_1, m_2, \ldots, m_t), v\Big) \times 4 \times |P_j|\Big)
\end{aligned} \tag{43}$$

Observing that mulCDT$(\tau'', 1, \frac{1}{2}, 1) = \tau''^{B,\mathbb{R}(P_j)}$ gives us:

$$\begin{aligned}
&\text{rmig-feas}(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t)) \\
&\quad \Rightarrow \quad \text{sched}(\text{ra-np-pEDF}, \texttt{create_fav_taskset}(\tau''^{B,\mathbb{R}(P_j)},
\end{aligned}$$

$$\Pi(m_1, m_2, \ldots, m_t)), \mathbb{R}(P_j),$$
$$\texttt{create\_fav\_platform}(\tau''^{B,\mathbb{R}(P_j)}, \Pi(m_1, m_2, \ldots, m_t), v)$$
$$\times 4 \times |P_j|) \tag{44}$$

Observe that the schedule generated by ra-np-pEDF scheduling of tasks in the task set $\texttt{create\_fav\_taskset}(\tau^{!!B,\mathbb{R}(P_j)}, \Pi(m_1, m_2, \ldots, m_t))$ on processors in the platform $\texttt{create\_fav\_platform}(\tau^{!!B,\mathbb{R}(P_j)}, \Pi(m_1, m_2, \ldots, m_t), v)$ is identical to the schedule generated by ra-np-pEDF-fav scheduling of tasks in $\tau^{!!B,\mathbb{R}(P_j)}$ on $\Pi(|P_j|, |P_j|, \ldots, |P_j|)$. Combining this observation with (44) gives us:

$$\text{rmig-feas}(\tau'', \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t))$$
$$\Rightarrow \quad \text{sched}(\text{ra-np-pEDF-fav}, \tau''^{B,\mathbb{R}(P_j)}, \mathbb{R}(P_j),$$
$$\Pi(|P_j|, |P_j|, \ldots, |P_j|) \times 4 \times |P_j|) \tag{45}$$

This states the Claim 2.

Proving 3. The correctness of this claim ($\tau^{!!B,\mathbb{R}(P_j)} = \tau^{B,\mathbb{R}(P_j)}$) can be seen directly from the definition of $\tau^{!!B,\mathbb{R}(P_j)}$.

Hence the lemma. D

5.4 The speed competitive ratio of LP-EE-vpr algorithm

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

Theorem 1 *The algorithm* LP-EE-vpr *has the following speed competitive ratio*: $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$.

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 Lemmas 7 and 8, yields:

$$\text{rmig-feas}(\tau, \Pi(m_1, m_2, \ldots, m_t))$$
$$\Rightarrow \quad \text{sched}(\text{LP-EE-}\delta, \tau^A, \Pi(m_1, m_2, \ldots, m_t) \times 4) \tag{46}$$

Consider phase-C scheduling. Note that LP-EE-vpr assigns a phase-C subtask, $\tau_{i,C}$ $\tau_{i,C} \in^C$, to the same VP_{AC} virtual processor to which the corresponding phase-A subtask, $\tau_{i,A}$ $\tau_{i,A} \in^A$, is assigned (see line 10 in Algorithm 1). For convenience, let LP-EE-δ-cp denote such a task assignment policy, i.e., using LP-EE-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 (46), we get:

$$\text{rmig-feas}(\tau, \Pi(m_1, m_2, \ldots, m_t))$$
$$\Rightarrow \quad \text{sched}(\text{LP-EE-}\delta\text{-cp}, \tau^A \cup \tau^C, \Pi(m_1, m_2, \ldots, m_t) \times 4) \tag{47}$$

Now let us discuss phase-B scheduling. From Lemma 10 we obtain:

$$\forall P_j \in P: \quad \text{rmig-feas}\big(\tau, R, \Pi(m_1, m_2, \ldots, m_t)\big) \Rightarrow \text{sched}\big(\text{ra-np-pEDF-fav}, \tau^{B,\mathbb{R}(P_j)}, \mathbb{R}(P_j), P_j, \Pi\big(|P_j|, |P_j|, \ldots, |P_j|\big) \times 4 \times |P_j|\big) \tag{48}$$

We know that, $\text{MAXP} = \max_{P_j \in P} |P_j|$. Using this, Expression (48) can be rewritten as:

$$\forall P_j \in P: \quad \text{rmig-feas}\big(\tau, \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t)\big) \Rightarrow \text{sched}\big(\text{ra-np-pEDF-fav}, \tau^{B,\mathbb{R}(P_j)}, \mathbb{R}(P_j), \Pi\big(|P_j|, |P_j|, \ldots, |P_j|\big) \times \text{MAXP} \times 4\big) \tag{49}$$

Let us now combine the results obtained for task sets $\tau^A \cup \tau^C$ and $\tau^{B,\mathbb{R}(P_j)}$. Dividing the type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) processor speeds in Expression (47) by $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil)$, we get:

$$\text{rmig-feas}\left(\tau, \Pi(m_1, m_2, \ldots, m_t) \times \left\langle \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil)}, \ldots, \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil)} \right\rangle\right)$$
$$\Rightarrow \quad \text{sched}\left(\text{LP-EE-}\delta\text{-cp}, \tau^A \cup \tau^C, \Pi(m_1, m_2, \ldots, m_t) \times \left\langle \frac{1}{1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil}, \ldots, \frac{1}{1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil} \right\rangle\right) \tag{50}$$

Dividing the type-k ($\forall k : k \in \{1, 2, \ldots, t\}$) processor speeds in Expression (49) by $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil)$, we get:

$$\forall P_j \in P: \quad \text{rmig-feas}\left(\tau, \mathbb{R}(P_j), \Pi(m_1, m_2, \ldots, m_t) \times \left\langle \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil)}, \ldots, \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil)} \right\rangle\right)$$
$$\Rightarrow \quad \text{sched}\left(\text{ra-np-pEDF-fav}, \tau^{B,\mathbb{R}(P_j)}, \mathbb{R}(P_j),\right.$$

$$\Pi\big(|P_j|, |P_j|, \ldots, |P_j|\big) \times \left\langle \frac{\text{MAXP}}{1 + \text{MAXP} \times \lceil \frac{|P|}{m_1} \rceil}, \ldots, \frac{\text{MAXP}}{1 + \text{MAXP} \times \lceil \frac{|P|}{m_t} \rceil} \right\rangle \right) \tag{51}$$

The specifications of the processors in the right-hand side predicates of Expression (50) and Expression (51) match those of the virtual processors that LP-EE-vpr created (see Sect. 4.2). Recall that LP-EE-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 (50) and $|P|$ instances of Expression (51), yields:

$$\begin{aligned} &\text{rmig-feas}\Big(\tau, R, \Pi(m_1, m_2, \ldots, m_t) \times \Big\langle \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil)}, \ldots, \\ &\qquad \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil)} \Big\rangle \Big) \\ &\Rightarrow \quad \text{sched}\big(\text{LP-EE-vpr}, \tau, R, \Pi(m_1, m_2, \ldots, m_t)\big) \end{aligned} \tag{52}$$

We know that higher speed processors do not jeopardize the feasibility of a task set. Hence, we can write:

$$\begin{aligned} &\text{rmig-feas}\big(\tau, R, \Pi(m_1, m_2, \ldots, m_t) \\ &\qquad \times \big\langle \min\{s_1, s_2, \ldots, s_t\}, \ldots, \min\{s_1, s_2, \ldots, s_t\} \big\rangle \big) \\ &\Rightarrow \quad \text{rmig-feas}\big(\text{rmo}, \tau, R, \Pi(m_1, m_2, \ldots, m_t) \times \langle s_1, s_2, \ldots, s_t \rangle\big) \end{aligned}$$

Substituting $s_k = \frac{1}{4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{m_k} \rceil)}$, $\forall k : k \in \{1, 2, \ldots, t\}$, in the above expression and combining with Expression (52) and rewriting gives:

$$\begin{aligned} &\text{rmig-feas}\Big(\tau, R, \Pi(m_1, m_2, \ldots, m_t) \\ &\qquad \times \Big\langle \frac{1}{4 \times (1 + \text{MAXP} \times \max\{\lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil, \ldots, \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil\})}, \\ &\qquad \ldots, \frac{1}{4 \times (1 + \text{MAXP} \times \max\{\lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil, \ldots, \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil\})} \Big\rangle \Big) \\ &\Rightarrow \quad \text{sched}\big(\text{LP-EE-vpr}, \tau, R, \Pi(m_1, m_2, \ldots, m_t)\big) \end{aligned} \tag{53}$$

Multiplying the speeds of all the processors in Expression (53) by a factor of $4 \times (1 + \text{MAXP} \times \max\{\lceil \frac{|P| \times \text{MAXP}}{m_1} \rceil, \ldots, \lceil \frac{|P| \times \text{MAXP}}{m_t} \rceil\})$, we get:

$$\begin{aligned} &\text{rmig-feas}\big(\tau, R, \Pi(m_1, m_2, \ldots, m_t)\big) \\ &\qquad \times \Big\langle 4 \times \Big(1 + \text{MAXP} \times \max\Big\{\Big\lceil \frac{|P| \times \text{MAXP}}{m_1} \Big\rceil, \ldots, \\ &\qquad \Big\lceil \frac{|P| \times \text{MAXP}}{m_t} \Big\rceil\Big\}\Big), \ldots, \\ &\qquad 4 \times \Big(1 + \text{MAXP} \times \max\Big\{\Big\lceil \frac{|P| \times \text{MAXP}}{m_1} \Big\rceil, \ldots, \\ &\qquad \Big\lceil \frac{|P| \times \text{MAXP}}{m_t} \Big\rceil\Big\}\Big)\Big\rangle\Big) \end{aligned}$$

Hence the theorem.

Theorem 2 *Consider the case in which each task can request at most one resource, i.e.,* $\forall \tau_i \in \tau : |R_i| \leq 1$. *For this case,* LP-EE-vpr *has a speed competitive ratio of* $4 \times (1 + \lceil \frac{|R|}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$.

Proof If $\forall \tau_i \in \tau : |R_i| \leq 1$ then every connected component in the graph has one vertex and hence every resource request partition has one element. Thus, MAXP = 1. Also, the number of resource request partitions $|P|$ is no greater than $|R|$, i.e., $|P| \leq |R|$. Applying this on Theorem 1 gives us the theorem. □

6 Discussion

In this section, we briefly discuss run-time mechanisms for realizing virtual processors and the preemptions generated and also highlight a couple of useful properties of LP-EE-vpr such as deadlock-free property, nested resource access and the bound on number of migrations per job. Also, a couple of tricks to improve the performance of LP-EE-vpr are discussed as well.

6.1 Run-time mechanism for realizing virtual processors and the preemptions generated

Given that the research literature has been lacking a scheduling algorithm for heterogeneous multiprocessors with resource sharing such that the algorithm has a proven

speed competitive ratio, our focus in this paper has been to create one. We did not deal with the cost of preemption.

Assuming that there is no cost of a preemption, one can create a set of virtual processors from a single physical processor without losing capacity as follows. Choose a timeslot size (denoted as S) and subdivide time into time intervals, each being of duration equal to the timeslot size S. Then if we want to create a set $VP = \{ vp_1, vp_2, ..., vp_{|VP|} \}$ of virtual processors where virtual processor vp_l (where $l \in \{1, 2, \ldots, |VP|\}$) has speed SP_l and accomplish this as long as $\sum_{l \in \{1,2,\ldots,|VP|\}} SP_l \leq 1$, then this can be done as follows. Create a reserve for vp_l in the timeslot so that this reserve has the duration $S \times vp_l$ and let the time of this reserve supply time to the virtual processor vp_l. Then let S be arbitrarily small. This gives us the desired virtual processors and this is the idea we have assumed in this paper.

Unfortunately, this approach generates an infinite number of preemptions. One could generate virtual processors in two other ways. First, by choosing S being the greatest common denominator of the parameters of the subtasks, one can still form virtual processors as mentioned above and still utilize 100 % of the capacity of a physical processor (Andersson and Bletsas 2008). This approach has two problems (i) the greatest common divisor of the parameters of the subtasks may not exist (this is an issue for the case that parameters are not rational numbers) and (ii) even if the greatest common divisor of the parameters of the subtasks exists, it may still be very small and hence may generate a very large number of preemptions. A second way to choose S (which avoids this drawback) is to choose a positive integer δ and then choose S as the minimum of all parameters of subtasks divided by δ. This approach has been used for creating virtual processors in Andersson and Bletsas (2008) and Bletsas and Andersson (2009) so that as long as the sum of the speeds of the virtual processors desired to be formed does not exceed a given bound $UB(\delta)$ (higher than 60 % but lower than 100 %), which is a function of δ, then all virtual processors can be formed. We can use such approaches at the cost of having a speed competitive ratio being multiplied by $1/UB(\delta)$.

6.2 Bound on the number of migrations per job

The algorithm, LP-EE-vpr, by design, limits the number of migrations per job to at most two. Recall that, LP-EE-vpr assigns both phase-A and phase-C executions of a task τ_i to the same VP_{AC} virtual processor and phase-B of that task to another VP_B virtual processor. Since the algorithm creates the virtual processors in such a manner that the capacity of no virtual processor comes from more than one physical processor (Lemma 1 in Sect. 4.2), 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 may come from a different physical processor, migration of a job of task τ_i can only occur at time instants when the job requests or releases the resource set R_i. Thus, the algorithm limits the number of migrations per job to at most two.

6.3 Nested resource access

To enable our algorithm for handling tasks with nested resource access, one of the two below mentioned techniques can be used.

– Group locking. It is a previously known technique (Block et al. 2007) in which the inner locks of a nested resource access are removed and only an outer lock (referred to as a *group lock*) is retained. The following example illustrates how nested resource access can be handled with the help of group locks. Consider a nested resource access in which jobs of a task τ_i request and release the resources in the following order: Each job of task τ_i does the following (in order):

request(r_1)

request(r_2)

release(r_2)

request(r_3)

release(r_3)

release(r_1)

With group locking, a new lock would be created, say r_{123} and then task τ_i would be changed such that each job of τ_i now does the following (in order):

request(r_{123})

release(r_{123})

If there is any other task that requests one or more of these resources (i.e., resource r_1, r_2 and r_3) then these tasks need to be changed as well.

– A variant of group locking. Another way to handle nested resource access is to request all the resources in the nested block at the beginning of the nested block and release all the resources at the end of this block. With this technique, in the above example, task τ_i would be changed such that each job of task τ_i now does the following (in order):

request(r_1 and r_2 and r_3)

release(r_1 and r_2 and r_3)

Since we allow multiple resources to be requested simultaneously, we can use any of the above two techniques for handling tasks with nested resource access.

6.4 Deadlock free property

Partial allocation describes a situation where a task is “waiting” for additional resource(s) while “holding” previously acquired one(s). Partial allocation is a necessary condition for deadlock to occur—see Chap. 7 in Silberschatz et al. (2009). Recall that,

we assume (as mentioned in Sect. 2) that a job of task τ_i performs a single request for the resource set R_i and then releases all the resources in the resource set R_i at once. And hence with this assumption, partial allocation never happens. And consequently, the algorithm LP-EE-vpr, for the assumptions stated in Sect. 2, cannot enter a deadlocked state.

6.5 Performance improvement

In this section, we describe a couple of tricks to improve the performance of the algorithm.

First, we dimensioned the phase-B virtual processors without considering the parameters of the subtasks that will execute on this virtual processor. A possible way to increase the performance of our algorithm though would be to determine, for each resource request partition, what is the lowest speed that is needed in order for the subtasks requesting the resources from the corresponding resource partition to be ra-np-pEDF-fav schedulable.

Second, our algorithm is based on LP-EE (Baruah 2004c) for assigning phase-A and phase-C subtasks. We selected LP-EE because it is simple to implement and easy to explain and it has a proven speed competitive ratio. Unfortunately, this algorithm has a time-complexity that is exponential with the number of processors. But we can replace LP-EE with another algorithm (Baruah 2004b) which has the same speed competitive ratio but runs with polynomial time-complexity because it does not perform exhaustive enumeration. In addition, one could replace LP-EE with the task assignment algorithm in Wiese et al. (2013) (which has a better speed competitive ratio than LP-EE). Then we would have a scheduling algorithm for our problem (with resource sharing), with a better speed competitive ratio but at the expense of having a time-complexity that is a polynomial of very high degree.

7 Conclusions

The heterogeneous multiprocessor model is more generic than identical or uniform multiprocessor model, in terms of the systems that it can accommodate. Hence, it is interesting to study heterogeneous multiprocessor systems since a solution designed for such systems can also be applied to identical and uniform multiprocessor systems. In addition, heterogeneous multiprocessors are increasingly becoming relevant as many chip manufacturers offer chips with different types of processors (AMD Inc. 2012; Apple Inc. 2012; Intel Corporation 2012; Intel Corporation 2013; Nvidia Inc. 2012; Qualcomm Inc. 2012; Samsung Inc. 2012; Ericsson 2012; Texas Instruments 2012; Alben 2013; Intel Corp. 2013). In many computer systems, apart from processors, tasks also share resources such as data structures, sensors, etc. and tasks must operate on such resources in a mutually exclusive manner while accessing the resource. Scheduling real-time tasks that share resources on a heterogeneous multiprocessor platform is a complex problem. In this work, we took the first step to solve the issue via a scheduling algorithm with a proven speed competitive ratio for heterogeneous multiprocessors.

This work considered the problem of scheduling a task set of implicit-deadline sporadic tasks to meet all deadlines on a t-type heterogeneous multiprocessor platform where tasks may share multiple resources. The tasks must operate on such resources in a mutually exclusive manner while accessing the resource, that is, at all times, when a job of a task holds a resource, no other job of any task can hold that resource. Each job may request (a subset of) resources at most once during its execution and it has to request all the resources in the subset together. A job is allowed to migrate when it requests/releases the resources but a job is not allowed to migrate at other times.

We presented an algorithm LP-EE-vpr and proved its performance bound. Specifically, we proved that if an implicit-deadline sporadic task set is schedulable on a t-type heterogeneous multiprocessor platform by an optimal scheduling algorithm that allows a job to migrate only when it requests or releases a resource set, then our algorithm also meets the deadlines with the same restriction on job migration, if given processors $4 \times (1 + \text{MAXP} \times \lceil \frac{|P| \times \text{MAXP}}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$ times as fast. For the special case that each task requests at most one resource, the bound of LP-EE-vpr collapses to $4 \times (1 + \lceil \frac{|R|}{\min\{m_1, m_2, \ldots, m_t\}} \rceil)$. To the best of our knowledge, LP-EE-vpr is the first algorithm with proven performance guarantee for real-time scheduling of sporadic tasks with resource sharing on t-type heterogeneous multiprocessors.

Acknowledgements This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References

AMD Inc (2012) AMD Accelerated Processing Units. http://www.amd.com/fusion

Andersson B, Baruah S, Jonsson J (2001) Static-priority scheduling on multiprocessors. In: Proceedings of the 22nd IEEE real-time systems symposium, pp 193–202

Andersson B, Bletsas K (2008) Sporadic multiprocessor scheduling with few preemptions. In: Proceedings of the 20th euromicro conference on real-time systems, pp 243–252

Andersson B, Easwaran A (2010) Provably good multiprocessor scheduling with resource sharing. Real-Time Syst 46(2):153–159

Andersson B, Raravi G, Bletsas K (2010) Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In: Proceedings of the 31st IEEE international real-time systems symposium, pp 239–248

Andersson B, Tovar E (2007) Competitive analysis of partitioned scheduling on uniform multiprocessors. In: Proceedings of the 15th international workshop on parallel and distributed real-time systems, pp 1–8

Apple Inc (2012) Apple A5X: Dual-core CPU and Quad-core GPU. http://www.apple.com/ipad/specs/

Baruah S (2004a) Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In: Proceedings of the 25th IEEE international real-time systems symposium, pp 37–46

Baruah S (2004b) Partitioning real-time tasks among heterogeneous multiprocessors. In: Proceedings of the 33rd international conference on parallel processing, pp 467–474
Baruah S (2004c) Task partitioning upon heterogeneous multiprocessor platforms. In: Proceedings of the 10th IEEE international real-time and embedded technology and applications symposium, pp 536–543
Baruah S (2013) Partitioned EDF scheduling: a closer look. Real-Time Syst 49(6):715–729
Baruah S, Fisher N (2007) The partitioned dynamic-priority scheduling of sporadic task systems. Real-Time Syst 36(3):199–226
Baruah S, Mok A, Rosier L (1990) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: Proceedings of the IEEE real-time systems symposium, pp 182–190
Blazewicz J, Lenstra J, Kan A (1983) Scheduling subject to resource constraints: classification and complexity. Discrete Appl Math 5(1):11–24
Bletsas K (2007) Worst-case and best-case timing analysis for real-time embedded systems with limited parallelism. PhD thesis, The University of York
Bletsas K, Andersson B (2009) Notional processors: an approach for multiprocessor scheduling. In: Proceedings of the 15th IEEE international real-time and embedded technology and applications symposium, pp 3–12
Block A, Leontyev H, Brandenburg B, Anderson J (2007) 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, pp 47–56
Correa J, Skutella M, Verschae J (2012) The power of preemption on unrelated machines and applications to scheduling orders. Math Oper Res 37(2):379–398
Darera V, Jenkins L (2006) Utilization bounds for RM scheduling on uniform multiprocessors. In: Proceedings of the 12th IEEE international conference on embedded and real-time computing systems and applications, pp 315–321
Dasari D, Andersson B, Nélis V, Petters S, Easwaran A, Lee J (2011) Response time analysis of COTS-based multicores considering the contention on the shared memory bus. In: Proceedings of the 8th IEEE international conference on embedded software and systems, pp 1068–1075
Dasari D, Nélis V (2012) An analysis of the impact of bus contention on the WCET in multicores. In: Proceedings of the 9th IEEE international conference on embedded software and systems, pp 1450–1457
Davis R, Rothvoß T, Baruah S, Burns A (2009) Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. Real-Time Syst 43(3)
Gai P, Abeni L, Buttazzo GC (2002) Multiprocessor DSP scheduling in system-on-a-chip architectures. In: Proceedings of the 14th euromicro conference on real-time systems (ECRTS 2002), Vienna, Austria, pp 231–238
Gschwind M, Hofstee HP, Flachs B, Hopkins M, Watanabe Y, Yamazaki T (2006) Synergistic processing in cell's multicore architecture. IEEE MICRO 26(2):10–24
Gurobi Optimization Inc (2012) Gurobi optimizer reference manual. http://www.gurobi.com
Holenderski M, Bril RJ, Lukkien JJ (2012) Parallel-task scheduling on multiple resources. In: Proceedings of the 24th euromicro conference on real-time systems, pp 233–244
Hopcroft J, Tarjan R (1973) Efficient algorithms for graph manipulation. Commun ACM 16(6):372–378
Horowitz E, Sahni S (1976) Exact and approximate algorithms for scheduling nonidentical processors. J ACM 23:317–327
IBM (2012). IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/
Intel Corp (2013) Bay Trail: Multicore SoC Family for Mobile Devices. http://www.intel.com/newsroom/kits/idf/2013_fall/pdfs/bay_trail_fact_sheet.pdf
Intel Corporation (2012). Intel Atom Processor Z6xx Series with Intel SM35 Express Chipset. http://www.intel.com/p/en_US/embedded/hwsw/hardware/atom-z6xx/overview
Intel Corporation (2013) The 4th Generation Intel® Core™ i7 Processo. http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html
Jansen K, Porkolab L (1999) Improved approximation schemes for scheduling unrelated parallel machines. In: Proceedings of the 31st annual ACM symposium on theory of computing (STOC'99), pp 408–417 Alben J (2013) NVIDIA Brings Kepler, World's Most Advanced Graphics Architecture, to Mobile Devices. http://blogs.nvidia.com/blog/2013/07/24/kepler-to-mobile/
Jones M (1997) What Happened on Mars? http://www.ece.cmu.edu/~raj/mars.html

Lakshmanan K, Rajkumar R (2010) Scheduling self-suspending real-time tasks with rate-monotonic priorities. In: Proceedings of the 16th IEEE international real-time and embedded technology and applications symposium, pp 3–12

Lenstra J, Shmoys D, Tardos É (1990) Approximation algorithms for scheduling unrelated parallel machines. Math Program 46:259–271

Li Y, Suhendra V, Liang Y, Mitra T, Roychoudhury A (2009) Timing analysis of concurrent programs running on shared cache multi-cores. In: Proceedings of the 30th IEEE real-time systems symposium, pp 57–67

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20:46–61

Lv M, Yi W, Guan N, Yu G (2010) Combining abstract interpretation with model checking for timing analysis of multicore software. In: Proceedings of the 31st IEEE real-time systems symposium, pp 339–349

Nvidia Inc (2012) Tegra 2 and Tegra 3 Super Chip Processors. http://www.nvidia.com/object/tegra-3-processor.html

Pellizzoni R, Schranzhofer A, Chen JJ, Caccamo M, Thiele L (2010) Worst case delay analysis for memory interference in multicore systems. In: Proceedings of the conference on design, automation and test in Europe, pp 741–746

Phillips CA, Stein C, Torng E, Wein J (1997) Optimal time-critical scheduling via resource augmentation. In: Proceedings of the 29th ACM symposium on theory of computing, pp 140–149

Potts CN (1985) Analysis of a linear programming heuristic for scheduling unrelated parallel machines. Discrete Appl Math 10:155–164

Qualcomm Inc (2012) Snapdragon Processors. http://www.qualcomm.com/chipsets/snapdragon

Rajkumar R, Sha L, Lehoczky J (1988) Real-time synchronization protocols for multiprocessors. In: Proceedings of the 9th IEEE real-time systems symposium, pp 259–269

Raravi G, Andersson B, Bletsas K (2013) Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. Real-Time Syst 49(1):29–72

Raravi G, Andersson B, Bletsas K, Nélis V (2012) Task assignment algorithms for two-type heterogeneous multiprocessors. In: Proceedings of the 24th euromicro conference on real-time systems, pp 34–43

Raravi G, Nélis V (2012) A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors. In: Proceedings of the 33rd IEEE international real-time systems symposium, pp 117–126

Rosén J, Andrei A, Eles P, Peng Z (2007) Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In: Proceedings of the 28th IEEE international real-time systems symposium, pp 49–60

Samsung Inc (2012) Samsung Exynos processor. www.samsung.com/exynos/

Schliecker S, Negrean M, Ernst R (2010) Bounding the shared resource load for the performance analysis of multiprocessor systems. In: Proceedings of the conference on design, automation and test in Europe, pp 759–764

Silberschatz A, Galvin P, Gagne G (2009) Operating system concepts, 8th edn. Wiley, New York

Ericsson ST (2012) In: NOVA processor family—highest performance application processors. http://www.stericsson.com/products/application_processors.jsp

Texas Instruments (2012) OMAP mobile processors. http://www.ti.com/omap

Wiese A, Bonifaci V, Baruah S (2013) Partitioned EDF scheduling on a few types of unrelated multiprocessors. Real-Time Syst 49(2):219–238