Technical Report

Task assignment algorithms for two-type heterogeneous multiprocessors

Gurulingesh Raravi
Björn Andersson
Vincent Nélis
Konstantinos Bletsas

CISTER-TR-131002
Version:
Date: 10-08-2013

Task assignment algorithms for two-type heterogeneous multiprocessors

Gurulingesh Raravi, Björn Andersson, Vincent Nélis, Konstantinos Bletsas

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Consider the problem of assigning implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising two different types of processors - such a platform is referred to as two-type platform. We present two low degree polynomial time-complexity algorithms, SA and SA-P, each providing the following guarantee. For a given two-type platform and a task set, if there exists a task assignment such that tasks can be scheduled to meet deadlines by allowing them to migrate only between processors of the same type (intra-migrative), then (i) using SA, it is guaranteed to find such an assignment where the same restriction on task migration applies but given a platform in which processors are 1+a/2 times faster and (ii) SA-P succeeds in finding a task assignment where tasks are not allowed to migrate between processors (non-migrative) but given a platform in which processors are 1+a times faster. The parameter 0<'a'<=1 is a property of the task set; it is the maximum of all the task utilizations that are no greater than 1.

We evaluate average-case performance of both the algorithms by generating task sets randomly and measuring how much faster processors the algorithms need (which is upper bounded by 1+a/2 for SA and 1+a for SA-P) in order to output a feasible task assignment (intra-migrative for SA and non-migrative for SA-P). In our evaluations, for the vast majority of task sets, these algorithms require significantly smaller processor speedup than indicated by their theoretical bounds.

Finally, we consider a special case where no task utilization in the given task set can exceed one and for this case, we (re-)prove the performance guarantees of SA and SA-P. We show, for both of the algorithms, that changing the adversary from intra-migrative to a more powerful one, namely fully-migrative, in which tasks can migrate between processors of any type, does not deteriorate the performance guarantees. For this special case, we compare the average-case performance of SA-P and a state-of-the-art algorithm by generating task sets randomly. In our evaluations, SA-P outperforms the state-of-the-art by requiring much smaller processor speedup and by running orders of magnitude faster.

Journal of Time-Critical Computing Systems manuscript No.
(will be inserted by the editor)

Task Assignment Algorithms for Two-type Heterogeneous Multiprocessors

Gurulingesh Raravi · Björn Andersson · Vincent Nélis · Konstantinos Bletsas

Received: date / Accepted: date

Abstract Consider the problem of assigning implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising two different types of processors — such a platform is referred to as *two-type platform*. We present two low degree polynomial time-complexity algorithms, SA and SA-P, each providing the following guarantee. For a given two-type platform and a task set, if there exists a task assignment such that tasks can be scheduled to meet deadlines by allowing them to migrate only between processors of the *same type* (intra-migrative), then (i) using SA, it is guaranteed to find such an assignment where the same restriction on task migration applies but given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster and (ii) SA-P succeeds in finding a task assignment where tasks are not allowed to migrate between processors (non-migrative) but given a platform in which processors are $1 + \alpha$ times faster. The parameter $0 < \alpha \leq 1$ is a property of the task set; it is the maximum of all the task utilizations that are no greater than 1.

We evaluate average-case performance of both the algorithms by generating task sets randomly and measuring how much faster processors the algorithms need (which is upper bounded by $1 + \frac{\alpha}{2}$ for SA and $1 + \alpha$ for SA-P) in order to output a feasible task assignment (intra-migrative for SA and non-migrative for SA-P). In our evaluations, for the vast majority of task sets, these algorithms require significantly smaller processor speedup than indicated by their theoretical bounds.

Finally, we consider a *special case* where no task utilization in the given task set can exceed one and for this case, we (re-)prove the performance guarantees of SA and SA-P. We show, for both of the algorithms, that changing the *adversary* from intra-migrative to a more powerful one, namely *fully-migrative*, in which tasks can migrate between processors of *any type*, does not deteriorate the performance guarantees. For this special case, we compare the average-case performance of SA-P and a state-of-the-art algorithm by generating task sets randomly. In our evaluations, SA-P outperforms the state-of-the-art by requiring much smaller processor speedup and by running orders of magnitude faster.

CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Porto, Portugal
E-mail: {ghri, nelis, ksbs}@isep.ipp.pt
Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA
E-mail: baandersson@sei.cmu.edu

Keywords heterogeneous multiprocessors · real-time scheduling · resource augmentation bound

1 Introduction

This paper addresses the problem of assigning a set of implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising processors of two unrelated types: type-1 and type-2. We refer to such a computing platform as *two-type platform.* On such a platform, the execution time of a task depends on the type of processor on which it executes. Our interest in considering such a platform model is motivated by the fact that many chip makers offer chips having two types of processors, both for desktops and embedded devices — see, e.g., AMD Inc. (2013); Apple Inc. (2013); Intel Corporation (2013a,b); Nvidia Inc. (2013); Qualcomm Inc (2013); Samsung Inc. (2013); ST Ericsson (2013); Texas Instruments (2013). For scheduling tasks on such platforms, we consider three models for migration: *non-migrative*, *intra-migrative* and *fully-migrative.*

In the *non-migrative* model (sometimes referred to as *partitioned* model in the literature), every task is statically assigned to a processor before run time and all its jobs must execute only on that processor at run time. The challenge is to find, before run time, a *task-to-processor* assignment such that, at run time, the given scheduling algorithm meets all the deadlines while scheduling the tasks on their assigned processor. Scheduling tasks to meet deadlines is a well-understood problem in the non-migrative model. One may use Earliest Deadline First (EDF) (Liu and Layland, 1973) on each processor, for example. The EDF algorithm is an *optimal* scheduling algorithm on uniprocessor systems (Dertouzos, 1974; Liu and Layland, 1973), with the interpretation that, it always finds a schedule in which all the deadlines are met, if such a schedule exists. Therefore, assuming that an optimal scheduling algorithm is used on every processor, the challenging part is to find a task-to-processor assignment for which *there exists* a schedule that meets all deadlines — such an assignment is said to be a *feasible* assignment hereafter. A non-migrative task assignment algorithm is said to be *optimal* if, for each task set, it succeeds in finding a feasible task-to-processor assignment, provided such an assignment exists. Even in the simpler case of identical multiprocessors, finding a feasible task-to-processor assignment is NP-Complete in the strong sense (see, e.g., Johnson, 1973). Hence, this result continues to hold for two-type platforms as well. In this work, we propose an approximation algorithm, SA-P, for this problem which outperforms state-of-the-art.

In the *intra-migrative* model, every task is statically assigned to a *processor type* before run time, rather than to an individual processor. Then, the jobs of each task can migrate at run-time from one processor to another as long as these processors are of the same type. Similar to the non-migrative model, once tasks have been assigned, scheduling tasks to meet deadlines under the intra-migrative model is well-understood, e.g., one may use an optimal scheduling algorithm, such as, ERfair (Anderson and Srinivasan, 2000), DP-Fair (Levin et al, 2010) or Sporadic-EKG (Andersson and Bletsas, 2008) with $S = \gcd(T_1, T_2, \ldots, T_n)$, that is designed for identical multiprocessors. Once again, assuming that an optimal algorithm is used for scheduling tasks on processors of each type, the challenging part is to find a *feasible task-to-processor-type* assignment for which *there exists* a schedule that meets all the deadlines. An intra-migrative task assignment algorithm is said to be *optimal* if, for each task set, it succeeds in finding a feasible task-to-processor-type assignment, provided such an assignment exists. It is straightforward to see that the problem of determining a task-to-processor-type assignment on two-type platform (assuming that an optimal scheduling algorithm is used on each processor type) is equivalent to the problem of assigning tasks to two processors, each of different types, such that each processor is used at most 100% of its capacity. Even the simpler instance of this problem, in which tasks must be assigned to *two* identical processors, is NP-Complete (Theorem 18.1 in Korte and Vygen (2006), p. 426). Hence, this result continues to hold for two-type platforms as well. In this work, we propose an approximation algorithm, SA, for this problem, for which no previous algorithm is known to exist.

In the *fully-migrative* model, jobs are allowed to migrate from any processor to any other processor at run-time, irrespective of the processor types. Even though this model is powerful in theory, it is rarely applicable in practice because job migration between processors of different types is hard to achieve (if not impossible as discussed by DeVuyst et al (2012)) as different processor types typically differ in their register formats, instruction sets, etc. Hence, this model is not the main focus of this work and is only considered for the role of the *adversary* (i.e., a class of algorithms against which the performance guarantee of the algorithm under design is proven) when we discuss a *special case*, in which no task utilization in the given task set can exceed one.

Note: The fully-migrative model is more powerful than the intra-migrative model which in turn is more powerful than the non-migrative model, in the sense that, (i) a non-migrative solution can always be transformed into an intra-migrative solution and similarly, an intra-migrative solution can always be transformed into a fully-migrative solution whereas (ii) a fully-migrative solution cannot always be transformed into an intra-migrative solution and similarly, an intra-migrative solution cannot always be transformed into a non-migrative solution. The relation of these models can be expressed using set notations as follows: `fully-migrative model` $\supset$ `intra-migrative model` $\supset$ `non-migrative model`.

Commonly, the performance of an algorithm is characterized using the notion of *utilization bound* (see, Liu and Layland, 1973; Davis and Burns, 2011): an algorithm with a utilization bound of UB is always capable of scheduling any task set with a utilization up to UB so as to meet all deadlines. This definition has been used in uniprocessor scheduling (e.g., see, Liu and Layland, 1973), *identical* multiprocessor scheduling (e.g., see, Andersson et al, 2001) and *uniform* multiprocessor scheduling (e.g., see, Darera and Jenkins, 2006). However, it does not translate to heterogeneous multiprocessors, hence we rely on the *resource augmentation* framework (Phillips et al, 1997) to characterize the performance of the algorithm under design.

We define *approximation ratio* $\mathrm{AR_I}$ of an intra-migrative algorithm $\mathcal{A}_\mathcal{I}$ (resp., $\mathrm{AR_N}$ of a non-migrative algorithm $\mathcal{A}_\mathcal{N}$) against an intra-migrative adversary as the lowest number such that for every task set τ and computing platform π it holds that if it is possible for an intra-migrative algorithm (i.e., the adversary) to meet all deadlines of τ on π then algorithm $\mathcal{A}_\mathcal{I}$ (resp., $\mathcal{A}_\mathcal{N}$) outputs an intra-migrative assignment (resp., non-migrative assignment) which meets all deadlines of τ on a platform π' whose every processor is $\mathrm{AR_I}$ (resp., $\mathrm{AR_N}$) times faster than the corresponding processor in π.

A low approximation ratio indicates high performance; the best achievable is 1 (which reflects the optimal algorithm for a given problem). If a scheduling algorithm has an infinite approximation ratio then a task set exists which could be scheduled (by another intra-migrative algorithm) to meet deadlines but would miss deadlines with the actually used algorithm even if processor speeds were multiplied by an "infinite" factor. Thus, a scheduling algorithm with a *finite* (and ideally small) approximation 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 approximation ratio, (e.g., see, Andersson and Tovar, 2007; Baruah and Fisher, 2007; Chen and Chakraborty, 2011). Therefore, we aim for algorithms with finite (and ideally small) approximation ratios.

Related work. The scheduling problem on heterogeneous multiprocessors has been studied in the past (see, e.g., Baruah, 2004a,b,c; Correa et al, 2012; Lenstra et al, 1990; Raravi et al, 2011, 2013). The problem considered by Lenstra et al (1990) is to minimize the *makespan*, i.e., the duration of the schedule, for non-preemptive scheduling of a collection of jobs on heterogeneous multiprocessors. For this problem, Lenstra et al (1990) proposed an algorithm with an approximation ratio of 2. It is well-known that this problem is equivalent to the problem of preemptive, non-migrative scheduling of implicit-deadline sporadic tasks on heterogeneous multiprocessors using EDF on each processor. For this problem, Baruah (2004b,c) also proposed non-migrative algorithms with an approximation ratio of 2. All these approaches (Baruah, 2004b,c; Lenstra et al, 1990) focused on generic heterogeneous multiprocessor platforms, i.e., platforms having two or more processor types and their approximation ratios have been proven against a non-migrative adversary. Due to practical relevance, recent research (Raravi et al, 2013) considered the problem of non-migrative scheduling of tasks on *two-type* platforms and proposed an algorithm, FF-3C, based on the first-fit heuristic and a couple of variants of this algorithm. These had the same worst-case performance guarantee as the approaches in Baruah (2004b,c) and Lenstra et al (1990) (i.e., requiring processors twice as fast) but can be implemented more efficiently. Also, in average-case performance evaluations, for randomly generated task sets, these algorithms required far smaller processor speedups than their theoretical worst-case estimate and also performed better than the approaches in Baruah (2004b,c). The problem of fully-migrative feasibility of a task set on a heterogeneous multiprocessor platform has also been studied (Baruah, 2004a). Correa et al (2012) showed that if a task set can be scheduled by an optimal algorithm on a heterogeneous multiprocessor platform with full migrations then an optimal algorithm for scheduling tasks on heterogeneous multiprocessor platform with no migrations needs processors four times as fast. Raravi et al (2011) showed that if a task set in which "no task utilization can exceed one" can be scheduled to meet deadlines on a heterogeneous multiprocessor platform with full migrations then the algorithm in Baruah (2004c) also succeeds in scheduling the task set on a platform with no migrations in which processors are only twice as fast. In the previous sentence and in the rest of the manuscript, the phrase "no task utilization can exceed one" means that, for every task in the task set, it holds that all the utilizations of the task (note that

Platform	Adversary	Task Assignment Algorithms		
		Algorithm	Time-Complexity	Approx. ratio
t-type[a]	non-migrative	(Lenstra et al, 1990), non-migrative	$O(P)^{c}$	2
t-type	non-migrative	(Baruah, 2004c), non-migrative	$O(P \cdot 2^{m})$	2
t-type	non-migrative	(Baruah, 2004b) non-migrative	$O(P)$	2
2-type[b]	non-migrative	(Raravi et al, 2013) non-migrative	$O(n \cdot \max(\log n, m))$	2
t-type	fully-migrative	(Correa et al, 2012) non-migrative	$O(P)$	4
t-type	fully-migrative	(Raravi et al, 2011)[d] non-migrative	$O(P)$	2
2-type	intra-migrative	SA, intra-migrative	$O(n \log n)$	$1 + \frac{\alpha}{2} \leq 1.5$
2-type	intra-migrative	SA-P, non-migrative	$O(n \log n)$	$1 + \alpha \leq 2$
2-type	fully-migrative	SA[d], intra-migrative	$O(n \log n)$	$1 + \frac{\beta}{2} \leq 1.5$
2-type	fully-migrative	SA-P[d], non-migrative	$O(n \log n)$	$1 + \beta \leq 2$

[a] A heterogeneous multiprocessor platform having two or more processor types.
[b] A heterogeneous multiprocessor platform having only two processor types.
[c] The time-complexity $O(P)$ indicates that the algorithm relies on solving a Linear Program (LP) formulation — note that though a linear program can be solved in polynomial time, the polynomial generally has a higher degree.
[d] These algorithms apply only to those task sets in which utilization of any task on any processor type does not exceed one.

Table 1: Summary of state-of-the-art task assignment algorithms along with the algorithms proposed in this paper.

on a t-type heterogeneous multiprocessor platform, each task has t utilizations, one on each processor type, where $t \geq 2$) are less than or equal to one[1].

The state-of-the-art, along with the contributions of this paper, is summarized in Table 1. Each row in the table corresponds to a different algorithm. For example, the third row in the table is read as follows: for a generic heterogeneous multiprocessor platform in which there can be two or more types of processors (denoted as t-type), a non-migrative algorithm is proposed in Baruah (2004b) and this algorithm is shown to have an approximation ratio of 2 against a non-migrative adversary and the algorithm has a time-complexity of $O(P)$ (explained in Table 1).

Contributions and significance of this work. We present a task assignment algorithm, called SA, which has a $O(n \log n)$ time-complexity and offers the following guarantee. Consider a two-type platform π and an implicit-deadline sporadic task set τ in which, for every task in τ, it holds that: (i) utilization of the task on processors of type-1 is either no greater than α or is greater than 1 and (ii) utilization of the task on processors of type-2 is either no greater than α or is greater than 1, where $0 < \alpha \leq 1$. If there exists a feasible intra-migrative as-

[1] In a heterogeneous multiprocessor, under the assumption that a task cannot execute on multiple processors simultaneously at any time instant (which is stated in Section 2), if a task set has a task with all its utilizations greater than one then the task set *is infeasible* else the task set *may be feasible*. For example, a task set with a single task whose utilization is greater than 1 on both type-1 and type-2 processors is *infeasible* on a two-type platform (with any number of processors). As another example, a task set with a single task whose utilization is equal to 1 on type-1 processors and is equal to 2 on type-2 processors is *feasible* on a two-type platform with at least one processor of type-1 (number of type-2 processors is irrelevant here).

signment of τ on π (i.e., task-to-processor-type assignment) then, using SA, it is guaranteed to find such a feasible intra-migrative assignment of τ on $\pi^{(1+\frac{\alpha}{2})}$, where $\pi^{(1+\frac{\alpha}{2})}$ is a two-type platform in which every processor is $1+\frac{\alpha}{2}$ times faster than the corresponding processor in π. Then, we modify SA to obtain SA-P, a non-migrative algorithm of $O(n \log n)$ time-complexity which offers the following guarantee. For a given task set τ and a two-type platform π, if there exists a feasible *intra-migrative* assignment of τ on π then SA-P succeeds in finding a feasible *non-migrative* assignment of τ on $\pi^{(1+\alpha)}$ (i.e., task-to-processor assignment). We also show that the proven approximation ratio of each of these algorithms is a tight bound. We then consider a *special case* where the maximum utilization of any task on any processor in the given task set is no greater than one and (re-)prove the performance guarantees of SA and SA-P. We show, for both algorithms, that changing the *adversary* from intra-migrative to a more powerful one, namely *fully-migrative*, does not deteriorate the performance guarantees. Specifically, we show that for a given two-type platform and a given task set, if the task set is *fully-migrative* feasible on the platform, then (i) using SA, it is guaranteed to find a feasible *intra-migrative* task assignment on a platform in which processors are $1+\frac{\beta}{2}$ times faster and (ii) SA-P succeeds in finding a feasible *non-migrative* task assignment on a platform in which processors are $1+\beta$ times faster, where $0 < \beta \leq 1$. The parameter β is a property of the task set — it is the maximum utilization of any task in the given task set. We also evaluate the average-case performance of our new algorithms by generating task sets randomly and measuring how much faster processors the algorithms need (which is upper bounded by the approximation ratios of respective algorithms), for a given task set, in order to output a feasible task assignment (which is intra-migrative for SA and non-migrative for SA-P). Finally, for the special case where "no task utilization can exceed one", we compare the average-case performance of SA-P and a state-of-the-art algorithm (Raravi et al, 2011) (and a variation of the latter) by generating task sets randomly. We evaluate algorithms based on (i) their running times and (ii) the amount of extra speed of processors that the algorithm needs, for a given task set, so as to succeed, compared to an optimal fully-migrative algorithm.

We believe that the significance of this work is three-fold. First, for the problem of intra-migrative task assignment, no previous algorithm exists and hence our algorithm, SA, is the first for this problem[2]. Second, for the problem of non-migrative task assignment, our algorithm, SA-P, has superior performance compared to state-of-the-art. This can be seen from Table 1 since SA-P has (i) the same approximation ratio as algorithms in Baruah (2004b,c); Lenstra et al (1990); Raravi et al (2013) but with a stronger adversary and also a better time-complexity and (ii) among the algorithms with approximation ratio proven against an adversary with a migration model of intra-migrative or greater power (Correa et al, 2012), SA-P offers the best approximation ratio[3]. Similar observations hold for both SA and SA-P for the case in which no task utilization in the given task set can exceed one. Third, in our evaluations with randomly generated task sets, for the vast majority of task sets, our algorithms require significantly smaller proces-

[2] Although the approach presented in Lenstra et al (1990) can be "adapted" to obtain a solution for the intra-migrative model, it would incur a high time-complexity as it relies on solving a linear program.

[3] Since the work in Raravi et al (2011) applies only to a special case in which no task utilization in the given task set can exceed one, it is ignored here.

sor speedup than what is indicated by their theoretical bounds and for the special case where "no task utilization can exceed one", SA-P exhibits a better average-case performance by outperforming the prior state-of-the-art algorithm (Raravi et al, 2011).

Compared to the conference version of this paper (Raravi et al, 2012), the additional contributions of this work can be summarized as follows: (i) for randomly generated task sets, we evaluate the average-case performance of our algorithms in terms of the processor speedup required to output a feasible assignment and show that the algorithms exhibit better average-case performance than their theoretical bounds, (ii) for the sake of completeness, we show that the problem of intra-migrative task assignment on two-type platform is NP-Complete and the problem of non-migrative task assignment on two-type platform is NP-Complete in the strong sense, (iii) we show that the proven approximation ratio of each of the proposed algorithms is a tight bound, (iv) we extend the analysis of our algorithms to a *special case* where no task utilization can exceed one and show that for this case, changing the adversary to a more powerful one, namely fully-migrative, does not deteriorate the performance guarantees of our algorithms, (v) for this special case, we compare the average-case performance of SA-P and prior state-of-the-art algorithm for randomly generated task sets and show that SA-P outperforms state-of-the-art and (vi) we also analyze the performance guarantees of our algorithms in terms of additional processors required compared to an optimal algorithm, giving the designer a choice of either choosing the additional processors or increasing the speed of processors.

Organization of the paper. The rest of the paper is organized as follows. Section 2 describes the system model. Section 3 presents an optimal *intra-migrative* task assignment algorithm, MILP-Algo, that uses Mixed Integer Linear Programming (MILP) formulation. Since solving MILP typically takes a long time (MILP without restrictions is known to be NP-Complete; see pp. 201–202 in Papadimitriou (1994)), Section 4 presents another algorithm, LP-Algo, by relaxing the MILP formulation to Linear Programming (LP) formulation and derives its approximation ratio. As solving an LP formulation is also often time consuming, Section 5 presents a new intra-migrative algorithm SA of time-complexity $O(n \log n)$ that does not rely on solving an LP formulation but has the same approximation ratio as LP-Algo, which is proven in Section 6. Section 7 extends SA to obtain a non-migrative task assignment algorithm, SA-P, of time-complexity $O(n \log n)$. Section 8 presents the approximation ratio of SA-P. Section 9 offers average-case performance evaluations of SA and SA-P. Section 10 analyzes the performance guarantees of SA and SA-P for a special case in which no task utilization in the given task set can exceed one. Section 11 presents the average-case performance evaluation of SA-P for this special case and compares it with a prior state-of-the-art algorithm. Section 12 concludes. Finally, Appendix A discusses the hardness of the two problems that are under consideration.

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 on a two-type heterogeneous multiprocessor platform $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ comprising m processors, of which m_1 processors are of type-1 and m_2 processors are of type-2. Each task τ_i is characterized by two parameters: a *worst-case execution time* and a *minimum inter-arrival time* T_i. Each task τ_i releases a (potentially infinite) sequence of *jobs*, with the first job released at any time during the system execution and subsequent jobs released *at least* T_i time units apart. Each job released by a task τ_i has to complete its execution within T_i time units (also referred to as *deadline*) from its release.

On a two-type platform, the worst-case execution time of a task depends on the type of the processor on which the task executes. We denote by C_i^1 and C_i^2 the worst-case execution time of task τ_i when executed on processor of type-1 and type-2, respectively. We denote by $u_i^1 \overset{\text{def}}{=} C_i^1/T_i$ and $u_i^2 \overset{\text{def}}{=} C_i^2/T_i$ the utilizations of task τ_i on type-1 and type-2 processors, respectively. A task that cannot be executed upon a certain processor type is modeled by setting its worst-case execution time (and thus its utilization) on that processor type to ∞.
Let α be a real number defined as follows:

$$\alpha \overset{\text{def}}{=} \max_{\forall \tau_i \in \tau, t \in \{1,2\}} \left\{ u_i^t : u_i^t \leq 1 \right\} \tag{1}$$

Then it holds that the utilization of any task on any processor type is either no greater than α or is greater than 1, i.e.,

$$\forall \tau_i \in \tau : \quad \Big((u_i^1 \leq \alpha) \quad \vee \quad (u_i^1 > 1) \Big) \ \wedge \ \Big((u_i^2 \leq \alpha) \quad \vee \quad (u_i^2 > 1) \Big) \tag{2}$$

The following example illustrates how to determine the value of α from a given task set.

Example 1 Consider a task set comprising three tasks, $\tau = \{\tau_1, \tau_2, \tau_3\}$ whose utilizations on type-1 and type-2 processors are given by $u_1^1 = 0.5, u_1^2 = 1.5, u_2^1 = 1.2, u_2^2 = 0.8, u_3^1 = 0.7, u_3^2 = 0.9$. For this task set, $\alpha = 0.9$.

We assume that the tasks are independent, i.e., they do not share any resources except processors and do not have any data dependency. We assume that a job can be executing on at most one processor at any given time. When studying the intra-migrative model, we assume that all tasks assigned to type-1 (resp., type-2) processors are scheduled on the set of type-1 (resp., type-2) processors using an algorithm that is optimal for the problem of scheduling tasks on identical multiprocessors (e.g., ERfair (Anderson and Srinivasan, 2000), Sporadic-EKG (Andersson and Bletsas, 2008), DP-Fair (Levin et al, 2010)). When studying the non-migrative model, we assume that all the tasks assigned to a processor are scheduled on this processor using an algorithm that is optimal for the problem of scheduling tasks on a uniprocessor (e.g., EDF (Liu and Layland, 1973)).

For convenience, we sometimes denote a two-type platform π with m_1 processors of type-1 and m_2 processors of type-2 by $\pi(m_1, m_2)$. Also, we denote by $\pi^{(x)}$, a two-type platform in which every processor is $x > 0$ times faster than the corresponding processor in platform π.

3 MILP-Algo: An optimal intra-migrative task assignment algorithm

In this section, we provide an *optimal intra-migrative task assignment algorithm* for assigning tasks in τ to *processor types* on two-type platform π. Recall that a task assignment algorithm is said to be *optimal* if, for each task set, it succeeds in finding a feasible assignment, provided such an assignment exists. The proposed algorithm is based on solving Mixed Integer Linear Programming (MILP) formulation. As described earlier, once the tasks have been assigned to processor types, we assume that, an optimal scheduling algorithm (e.g., ERfair (Anderson and Srinivasan, 2000), DP-Fair (Levin et al, 2010) or Sporadic-EKG (Andersson and Bletsas, 2008) with $S = \gcd(T_1, T_2, \ldots, T_n))$ that is designed for identical multiprocessors, will be used to schedule the tasks on processors of each type. From the feasibility tests of identical multiprocessor scheduling (Horn, 1974), the following necessary and sufficient set of conditions must hold $\forall t \in \{1, 2\}$, for intra-migrative task assignment to be feasible:

$$\forall t \in \{1,2\} : \forall \tau_i \in \tau^t : u_i^t \leq 1 \tag{3}$$

$$\forall t \in \{1,2\} : \sum_{\tau_i \in \tau^t} u_i^t \leq m_t \tag{4}$$

where τ^t denotes the set of tasks that are assigned to processors of type-t. The first condition (Expression (3)) is essential since the system model does not allow a task to execute simultaneously on more than one processor at any time (as mentioned earlier in Section 2). The second condition (Expression (4)) is essential as it is a feasibility condition for implicit-deadline sporadic task on identical multiprocessors (Horn, 1974) which ensures that the computing *load* does not exceed the processing *capacity*.

Given these necessary and sufficient feasibility conditions, we now describe, how to obtain an optimal intra-migrative task assignment algorithm. We partition the task set τ into four subsets H12, H1, H2 and L as defined below.

H12 is the set of tasks whose utilization exceeds one on both processor types, i.e., these tasks violate the feasibility condition shown in Expression (3), irrespective of the processor type they are assigned to. Formally,

$$\text{H12} \overset{\text{def}}{=} \Big\{ \tau_i \in \tau : u_i^1 > 1 \quad \wedge \quad u_i^2 > 1 \Big\} \tag{5}$$

A task in H12 cannot be scheduled to meet its deadline unless it executes in parallel, which is forbidden in our system model. Hence, for task sets with H12 $\neq \emptyset$, no feasible task assignment exists and thus we assume this set to be empty hereafter.

H1 is the set of tasks that must be assigned to type-1 processors as their utilization on type-2 exceeds one (and hence assigning them to type-2 processors violates the feasibility condition shown in Expression (3)), i.e.,

$$\text{H1} \overset{\text{def}}{=} \Big\{ \tau_i \in \tau : u_i^1 \leq \alpha \quad \wedge \quad u_i^2 > 1 \Big\} \tag{6}$$

Analogously, H2 is the set of tasks that must be assigned to type-2 processors as their utilization on type-1 exceeds one (and hence assigning them to type-2 processors violates the feasibility condition shown in Expression (3)), i.e.,

$$\text{H2} \overset{\text{def}}{=} \Big\{ \tau_i \in \tau : u_i^1 > 1 \quad \wedge \quad u_i^2 \leq \alpha \Big\} \tag{7}$$

Finally, L is the set of tasks that can be assigned on either processor type as their utilizations on both processor types do not exceed one, i.e.,

Minimize Z subject to the following constraints:

I1.	$\forall \tau_i \in \mathrm{L}$: $x_i^1 + x_i^2 = 1$
I2.	$U^1 + \sum_{\tau_i \in \mathrm{L}} x_i^1 \times u_i^1 \leq Z \times m_1$
I3.	$U^2 + \sum_{\tau_i \in \mathrm{L}} x_i^2 \times u_i^2 \leq Z \times m_2$
I4.	$\forall \tau_i \in \mathrm{L}$: $x_i^1 \in \{0,1\}$ and $x_i^2 \in \{0,1\}$; Z is a non-negative real number

Fig. 1: MILP formulation – MILP-Feas$(\mathrm{L}, \pi, U^1, U^2)$ for assigning tasks in L to processor types in π.

$$\mathrm{L} \overset{\text{def}}{=} \left\{ \tau_i \in \tau : u_i^1 \leq \alpha \quad \wedge \quad u_i^2 \leq \alpha \right\} \tag{8}$$

In these definitions, we can intuitively understand the meaning of "H" as "heavy" and "L" as "light" tasks.

The optimal intra-migrative task assignment algorithm that we propose, namely MILP-Algo, works as follows.

First, assign the tasks in H1 to type-1 (resp., tasks in H2 to type-2) processors. Let U^1 denote the capacity consumed on type-1 processors after assigning H1 tasks, formally,

$$U^1 = \sum_{\tau_i \in \mathrm{H1}} u_i^1 \tag{9}$$

Analogously, let U^2 denote the capacity consumed on type-2 processors after assigning H2 tasks, formally,

$$U^2 = \sum_{\tau_i \in \mathrm{H2}} u_i^2 \tag{10}$$

If $U^1 > m_1$ or $U^2 > m_2$ then declare failure as this violates the feasibility condition shown in Expression (4).

Second, solve the MILP formulation shown in Figure 1 for assigning tasks in L. The formulation in Figure 1 is an MILP formulation on x_i^j variables and Z variable. In this formulation, variable Z denotes the average used capacity of either type-1 or type-2 processors, whichever is greater, and is set as the objective function to be minimized. Each variable x_i^t (where $t \in \{1, 2\}$) indicates the assignment of task τ_i to type-t processors. The first set of constraints specifies that every task must be assigned to a processor type. The second (resp., third) set of constraints asserts that at most $Z \times m_1$ capacity of type-1 (resp., $Z \times m_2$ capacity of type-2) processors can be used. The fourth set of constraints asserts that each task must be assigned entirely to either processors of type-1 or type-2. Using the solution of this MILP formulation, assign the tasks in L to processor types as follows: for each $\tau_i \in \mathrm{L}$, τ_i is assigned to type-t processors if and only if $x_i^t = 1$. If $Z > 1$ then declare failure as this indicates that the feasibility condition in Expression (4) is violated.

Theorem 1 *The MILP formulation MILP-Feas*$(\mathrm{L}, \pi, \sum_{\tau_i \in \mathrm{H1}} u_i^1, \sum_{\tau_i \in \mathrm{H2}} u_i^2)$ *shown in Figure 1 has a solution with* $Z \leq 1$ *if and only if the task set* τ *is intra-migrative feasible on two-type platform* π*.*

Proof Suppose that the task set τ is intra-migrative feasible on platform π and let $\mathcal{X}$ denote a feasible assignment. It then holds that, in this assignment, all tasks in H1 are assigned to processors of type-1 (otherwise, the condition shown in Expression (3) is violated) and analogously, all tasks in H2 are assigned to processors of type-2. It can be seen that, by assigning $U^1 \leftarrow \sum_{\tau_i \in \mathrm{H1}} u_i^1$ and by assigning $U^2 \leftarrow \sum_{\tau_i \in \mathrm{H2}} u_i^2$ and $\forall \tau_i \in \mathrm{L}$, by assigning values to x_i^t variables of MILP formulation of Figure 1 as:

$$\text{if } \mathcal{X}(i) = 1 \text{ then } x_i^1 \leftarrow 1, x_i^2 \leftarrow 0$$
$$\text{if } \mathcal{X}(i) = 2 \text{ then } x_i^1 \leftarrow 0, x_i^2 \leftarrow 1$$

gives a (feasible) solution to the MILP formulation in which $Z \leq 1$.

Now, suppose that there is a (feasible) solution with $Z \leq 1$ to the MILP formulation, MILP-Feas(L, π, $\sum_{\tau_i \in \mathrm{H1}} u_i^1$, $\sum_{\tau_i \in \mathrm{H2}} u_i^2$), of Figure 1. Using this solution, define the assignment of tasks to processor types as follows:

$$\begin{aligned}
\forall i \in \mathrm{H1} :\ & \mathcal{X}(i) \leftarrow 1 \\
\forall i \in \mathrm{H2} :\ & \mathcal{X}(i) \leftarrow 2 \\
\forall i \in \mathrm{L} \ \ :\ & \mathcal{X}(i) \leftarrow 1, \ \text{if } x_i^1 = 1 \wedge x_i^2 = 0 \\
& \mathcal{X}(i) \leftarrow 2, \ \text{if } x_i^1 = 0 \wedge x_i^2 = 1
\end{aligned}$$

By constraint I1 of the MILP formulation, each task is assigned to exactly one processor type in the assignment $\mathcal{X}$ obtained as shown above. By constraint I2 (resp., I3) of the MILP formulation, the capacity of type-2 (resp., type-3) processors is not exceeded in the assignment $\mathcal{X}$ (since $Z \leq 1$ in the feasible solution to MILP formulation). Hence, $\mathcal{X}$ is a feasible intra-migrative assignment. □

Corollary 1 *If there exists a feasible intra-migrative task assignment of τ on π then MILP-Algo is guaranteed to return such a feasible intra-migrative task assignment. In other words, MILP-Algo is an optimal intra-migrative task assignment algorithm.*

Proof Follows from Theorem 1. □

Since MILP-Algo relies on solving MILP formulation for which no polynomial time-complexity algorithm is known to exist (when there are no restrictions (Papadimitriou, 1994)), we now present a *sub-optimal* polynomial-time algorithm by relaxing the MILP formulation to an LP formulation.

4 LP-Algo: An intra-migrative task assignment algorithm

We relax our MILP formulation to LP as shown in Figure 2. In this LP formulation, variables Z and x_i^t have the same meaning as the corresponding variables in the MILP formulation and the first three constraints are same as well. Only the fourth constraint is different (i.e., *relaxed*) and it now asserts that a task can either be *integrally* or *fractionally* assigned to processor types. Since the LP formulation is less constrained than the MILP, the following lemma holds.

Minimize Z subject to the following constraints:

C1.	$\forall \tau_i \in \mathrm{L}$: $x_i^1 + x_i^2 = 1$
C2.	$U^1 + \sum_{\tau_i \in \mathrm{L}} x_i^1 \times u_i^1 \leq Z \times m_1$
C3.	$U^2 + \sum_{\tau_i \in \mathrm{L}} x_i^2 \times u_i^2 \leq Z \times m_2$
C4.	$\forall \tau_i \in \mathrm{L}$: x_i^1, x_i^2 are non-negative **real** numbers $\in [0, 1]$; Z is a non-negative real number

Fig. 2: Relaxed LP formulation – LP-Feas($\mathrm{L}, \pi, U^1, U^2$) for assigning tasks in L to processor types in π.

Lemma 1 *For any task set* L*, two-type platform* π *and non-negative real numbers* U^1 *and* U^2*, let* Z_{MILP} *be the value of the objective function that any MILP solver would return by solving MILP-Feas*($\mathrm{L}, \pi, U^1, U^2$) *shown in Figure 1. Similarly, let* Z_{LP} *be the value of the objective function that any LP solver would return by solving LP-Feas(*L*,* π*,* U^1*,* U^2*) shown in Figure 2. It then holds that* $Z_{\mathrm{LP}} \leq Z_{\mathrm{MILP}}$.

Our intra-migrative task assignment algorithm, LP-Algo, works as follows.

1. Assign the tasks in H1 to type-1 (resp., tasks in H2 to type-2) processors. Let U^1 and U^2 denote the same entities as before. If $U^1 > m_1$ or $U^2 > m_2$ then declare failure as it violates the feasibility condition shown in Expression (4).
2. Assign the tasks in L by solving the LP formulation shown in Figure 2. In the returned solution, if $x_i^t = 1$ (where $t \in \{1, 2\}$) then entirely (also referred to as *integrally*) assign the corresponding task τ_i to processors of type-t. If $0 < x_i^t < 1$ then assign a fraction x_i^t of task τ_i to processors of type-t; we say that such tasks are *fractionally* assigned and are referred to as *fractional* tasks in the rest of the paper. If $Z > 1$ then declare failure as this indicates that the feasibility condition shown in Expression (4) is violated.

Among all the optimal solutions to an LP problem, at least one solution lies at a *vertex* of the *feasible region*[4](see, pp. 117 in Luenberger and Ye (2008)). We are interested in such a solution, as we show below that it leads to a task assignment with at most one fractional task. For ease of discussion, we use index $1, 2, \ldots, \ell$ to refer to tasks in subset L hereafter.

Lemma 2 *Consider an optimal solution* $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ *to the LP formulation shown in Figure 2 that lies at the vertex of the feasible region. For such a solution, it holds that, there exists at most one task from* L *which is fractionally assigned to both processor types (and the rest are integrally assigned to either processors of type-1 or type-2) in the task assignment that* S *reflects, i.e., there exists at most one index* $f \in \{1, 2, \ldots, \ell\}$ *such that* $0 < x_f^1 < 1$ *and* $0 < x_f^2 < 1$.

Proof The proof is based on Fact 2 in Baruah (2004c): *"consider a linear program on* n *variables* $x_1, x_2, \ldots, x_n$*, in which each variable* x_i *is subject to the non-negativity constraint, i.e.,* $x_i \geq 0$*. Suppose that there are further* m *linear constraints. If* $m < n$*, then at each vertex of the feasible region (including the*

[4] The *feasible region* of a linear program in n-dimensional space is the region over which all the constraints hold.

basic solution), at most m of the variables have non-zero values". Clearly, the LP formulation of Figure 2 is a linear program on $n' = 2\ell + 1$ variables (i.e., 2ℓ variables x_i^t, plus variable Z), all subject to non-negativity constraint, and $m' = \ell + 2$ further linear constraints (ℓ constraints due to C1 plus one constraint each due to C2 and C3). As $m' < n'$ (we assume $\ell > 1$; otherwise the problem becomes trivial), we know from the above fact that in every optimal solution at the vertex of the feasible region, it holds that at most $m' = \ell + 2$ variables take non-zero values. Since Z is certain to be non-zero, at most $\ell + 1$ variables x_i^t can be non-zero.

Since there are only ℓ constraints $x_i^1 + x_i^2 = 1$ and at most $\ell + 1$ non-zero variables x_i^t, it can be seen that at most one constraint can have its two variables set to non-zero values. Indeed, for any $f \in \{1, 2, \ldots, \ell\}$, if we set the two variables x_f^1 and x_f^2 of the constraint $x_f^1 + x_f^2 = 1$ to fractional values, then there remain $\ell - 1$ non-zero values to distribute to the $\ell - 1$ remaining constraints $x_k^1 + x_k^2 = 1$ ($\forall k \in \{1, 2, \ldots, \ell\}$, $k \neq f$). Since none of those constraints can have its two variables set to 0, at least one variable (either x_k^1 or x_k^2) has to take a non-zero value in each of these $(\ell - 1)$ remaining constraints. Again, because $x_k^1 + x_k^2 = 1$ ($\forall k \in \{1, 2, \ldots, \ell\}$, $k \neq f$), all these non-zero values have to be equal to 1 and thus, at most one task (in this case, τ_f) can be fractionally assigned. □

Lemma 3 *Any solution, S_f^{LP}, to the LP formulation (shown in Figure 2) with at most one fractional task and $Z_f^{\mathrm{LP}} \leq 1$, can be converted to a solution, $S_{\mathrm{nf}}^{\mathrm{LP}}$, with no fractional task and*

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \frac{\alpha}{2} \leq 1 + \frac{\alpha}{2} \tag{11}$$

Proof Let $S_f^{\mathrm{LP}} = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z_f^{\mathrm{LP}}\}$ be a solution with only one index $f \in \{1, 2, \ldots, \ell\}$ such that $0 < x_f^1 < 1$ and $0 < x_f^2 < 1$ (i.e., τ_f is the fractional task). Now, let us convert this solution, S_f^{LP}, into $S_{\mathrm{nf}}^{\mathrm{LP}} = \{x_1^{1'}, x_1^{2'}, x_2^{1'}, x_2^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z_{\mathrm{nf}}^{\mathrm{LP}}\}$ such that $\forall i \in \{1, 2, \ldots, \ell\}$: $x_i^{1'} = 1 \vee x_i^{2'} = 1$, as follows:

$$\forall i \in \{1, 2, \ldots, \ell\}, i \neq f : x_i^{1'} \leftarrow x_i^1 \quad \wedge \quad x_i^{2'} \leftarrow x_i^2 \tag{12}$$

Now, for index f, two options remain:
either perform $x_f^{1'} \leftarrow x_f^1 + x_f^2 \;\wedge\; x_f^{2'} \leftarrow 0$ which results in

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \frac{x_f^2 \times u_f^1}{m_1}$$

or perform $x_f^{1'} \leftarrow 0 \;\wedge\; x_f^{2'} \leftarrow x_f^1 + x_f^2$ which results in

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \frac{x_f^1 \times u_f^2}{m_2}$$

None of the above two operations violate constraints C1-C4 of the LP formulation. So, let us choose the one that results in the lowest upper bound on $Z_{\mathrm{nf}}^{\mathrm{LP}}$, i.e.,

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq \min\left(Z_f^{\mathrm{LP}} + \frac{x_f^2 \times u_f^1}{m_1},\; Z_f^{\mathrm{LP}} + \frac{x_f^1 \times u_f^2}{m_2}\right)$$

Rewriting the above expression, we get:

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \min\left(\frac{x_f^2 \times u_f^1}{m_1},\ \frac{x_f^1 \times u_f^2}{m_2}\right)$$

The min term in the above expression increases as (i) m_1 and m_2 decrease and (ii) u_f^1 and u_f^2 increase. Hence, by setting m_1 and m_2 to their minimum values, i.e., $m_1 = m_2 = 1$, and by setting u_f^1 and u_f^2 to their maximum values, i.e., $u_f^1 = u_f^2 = \alpha$, we get:

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \min\left(\alpha \times x_f^2,\ \alpha \times x_f^1\right)$$

Using the fact $x_f^2 = 1 - x_f^1$ and rewriting yields:

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \alpha \times \min\left(1 - x_f^1,\ x_f^1\right)$$

The maximum values that Z_f^{LP} and the "min" term can take are 1.0 and 0.5, respectively. Hence, the above expression becomes:

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \frac{\alpha}{2} \leq 1 + \frac{\alpha}{2}$$

Thus, we showed that this transformed solution $S_{\mathrm{nf}}^{\mathrm{LP}} = \{x_1^{1'}, x_1^{2'}, x_2^{1'}, x_2^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z_{\mathrm{nf}}^{\mathrm{LP}}\}$ has no fractional tasks (i.e., indicator variables with fractional values) and satisfies Expression (11) and all the constraints of LP formulation. Hence the proof. □

Recall that $\pi^{(x)}$ denotes a two-type platform in which each processor is $x > 0$ times faster than the corresponding processor in platform π. We now prove the approximation ratio of LP-Algo.

Corollary 2 (Approximation ratio of LP-Algo)
If there exists a feasible intra-migrative assignment of τ on π then using LP-Algo, it is guaranteed to find such a feasible intra-migrative assignment of τ on $\pi^{(1+\frac{\alpha}{2})}$.

Proof We know that LP-Algo assigns tasks in H1 and H2 in the same way as an optimal intra-migrative task assignment algorithm does (as there is no other way to assign those tasks to meet deadlines). It then uses LP formulation to assign tasks in L. Combining Corollary 1, 1 and 2 gives us: if there exists a feasible intra-migrative task assignment of τ on π then LP-Algo returns an assignment of τ on π in which at most one task from L is fractionally assigned and the rest are integrally assigned to either type-1 or type-2 processors. Then, it follows from Lemma 3 that this fractional task can be assigned integrally to one of the processor types if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster. Hence the proof. □

We now show that the proven approximation ratio of LP-Algo is a tight bound.

Theorem 2 (Approximation ratio of LP-Algo is tight)
The proven approximation ratio 1.5 of algorithm LP-Algo is a tight bound.

Tasks	**Utilizations of tasks**	
	$\mathbf{u_i^1}$	$\mathbf{u_i^2}$
$\boldsymbol{\tau_1}$	0.5	0.5
$\boldsymbol{\tau_2}$	1.0	1.0
$\boldsymbol{\tau_3}$	0.5	0.5

Table 2: An example to illustrate that the proven approximation ratio of LP-Algo algorithm is a tight bound.

Processor types	**Tasks assigned**
type-1 (π_1)	τ_1 and τ_3
type-2 (π_2)	τ_2

Table 3: A feasible intra-migrative assignment for tasks shown in Table 2 on platform π.

Variables	**Values**
Z	1.0
x_1^1	1.0
x_1^2	0.0
x_2^1	0.5
x_2^2	0.5
x_3^1	0.0
x_3^2	1.0

Table 4: A solution output by the LP solver to the LP formulation shown in Figure 2 for the problem instance under consideration.

Proof In order to show that the proven approximation ratio of LP-Algo algorithm is a tight bound, it is sufficient to show that there exists a (feasible intra-migrative) problem instance for which LP-Algo needs 1.5 times faster processors to output a feasible intra-migrative assignment. We now show that such a problem instance exists.

Consider a problem instance with a task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ comprising three tasks and a two-type platform $\pi = \{\pi_1, \pi_2\}$ comprising two processors. Let π_1 be a processor of type-1 and π_2 be a processor of type-2. The utilizations of tasks are shown in Table 2.

Observe that the given task set τ is intra-migrative feasible on the given platform π. A feasible intra-migrative assignment is obtained by assigning (i) τ_1 and τ_3 to type-1 processors (which has a single processor, π_1) and (ii) τ_2 to type-2 processors (which has a single processor, π_2). This assignment is shown in Table 3.

Now consider algorithm LP-Algo. Initially, the task set is partitioned as follows using Expressions (5)–(8): $H12 = \emptyset$, $H1 = \emptyset$, $H2 = \emptyset$ and $L = \{\tau_1, \tau_2, \tau_3\}$. Since there are no heavy tasks, LP-Algo solves LP formulation shown in Figure 2 for assigning light tasks. Upon solving the LP formulation, we obtain a solution shown in Table 4. Upon assigning tasks to processor types using the solution output by the solver (which is shown in Table 4), it holds that:

– type-1 processors are fully utilized
– type-2 processors are fully utilized and
– task τ_2 is equally split between type-1 and type-2 processors

It can be seen that, in order to assign τ_2 integrally to type-1 processors, the speed of type-1 processors *must be* increased to 1.5. Analogously, for assigning τ_2 integrally to type-2 processors, the speed of type-2 processors *must be* increased to 1.5 as well. Therefore, a speedup of 1.5 is required to assign τ_2 integrally to one of the processor types.

Hence, the proven approximation ratio 1.5 of LP-Algo algorithm is a tight bound. □

Remark 1 Although Corollary 2 states that, for an intra-migrative feasible task set, LP-Algo needs a platform in which *every processor* is $1 + \frac{\alpha}{2}$ times faster, in order to output an intra-migrative feasible task assignment, it is trivial to see from the proof of Corollary 2 that a platform in which only *one processor* is $1+\frac{\alpha}{2}$ times faster is sufficient (to which the fractional task can be integrally assigned).

Recall that $\pi(m_1, m_2)$ denotes a two-type platform in which $m_1 > 0$ processors are of type-1 and $m_2 > 0$ processors are of type-2. We now state the performance of LP-Algo in terms of additional number of processors.

Corollary 3 *If there exists a feasible intra-migrative assignment of τ on $\pi(m_1, m_2)$ then, using LP-Algo, it is guaranteed to obtain such a feasible intra-migrative assignment of τ on $\pi'(m_1 + 1, m_2)$, which has one additional processor of type-1 compared to π.*

Proof Combining Corollary 1, 1 and 2 gives us: if there exists a feasible intra-migrative task assignment of τ on π then LP-Algo returns an assignment of τ on π in which at most one task from L, say τ_f, is fractionally assigned to both processor types and the rest are integrally assigned to either type-1 or type-2 processors. From the definition of L, we know that $u_f^1 \leq \alpha$ and $u_f^2 \leq \alpha$ where $0 < \alpha \leq 1$. Hence, if such a task τ_f exists then it could be integrally assigned to the set of type-1 processors, which has an additional processor in π'. Hence the proof. □

Remark 2 It is trivial to see that Corollary 3 holds true if LP-Algo is given a platform $\pi'(m_1, m_2 + 1)$ which has one additional processor of type-2 compared to π.

It is well known that the assignment techniques that rely on solving LP formulations take considerable amount of time to output a solution compared to techniques that do not solve LP formulations (e.g., see, Raravi et al, 2013). So, we now propose an algorithm, namely SA, that has the same approximation ratio as LP-Algo but does not solve LP formulation and instead uses a simple assignment technique.

5 SA: An intra-migrative task assignment algorithm

In this section, we describe the working of algorithm, SA, and show that it has a time-complexity of $O(n \log n)$.

5.1 The description of algorithm SA

SA is an intra-migrative task assignment algorithm and works as follows.

1. Partition the task set τ into subsets H12, H1, H2 and L as shown in Expression (5) to Expression (8). If H12 $\neq \emptyset$ then declare failure.
2. Assign tasks in H1 to type-1 (resp., H2 to type-2) processors on platform π. If $U^1 = \sum_{\tau_i \in \mathrm{H1}} u_i^1 > m_1$ or $U^2 = \sum_{\tau_i \in \mathrm{H2}} u_i^2 > m_2$ then declare failure.
3. Sort the tasks in L in non-increasing order of $\frac{u_i^2}{u_i^1}$, i.e., in non-increasing order of their preference to be assigned to type-1 processors.
4. Traverse this sorted list from "left to right" and assign the tasks one after the other to type-1 processors until there is no capacity left on type-1 processors to assign a task integrally (or all the tasks in L are assigned to type-1 processors leading to a successful assignment).
5. Traverse the sorted list from "right to left" and assign the remaining tasks one after the other to type-2 processors until there is no capacity left on type-2 processors to assign a task integrally (or the task that could not be assigned in the previous step is assigned to type-2 processors thereby resulting in a successful assignment).
6. Finally, assign the remaining task, if there is one, fractionally to both processor types (we show in Theorem 3 that there can be at most *one* such task, if there exists a feasible intra-migrative assignment of τ on π). While assigning this remaining task, assign as big a fraction of the task as possible to type-1 processors (i.e., the entire remaining capacity of type-1 processors is used), and assign the remaining fraction to type-2 processors. If there is not enough capacity left to assign this remaining task fractionally then declare failure.

SA is named so because we "**S**ort and **A**ssign" the tasks in L.

5.2 Time-complexity of algorithm SA

We now show that the time-complexity of SA is a low-degree polynomial function of the number of tasks (n). By inspecting the six steps of algorithm, SA, described above, we know that:

- H1 tasks are assigned to type-1 processors (i.e., at most n tasks). The time-complexity of this operation is $O(n)$.
- H2 tasks are assigned to type-2 processors (i.e., at most n tasks). The time-complexity of this operation is $O(n)$.
- Sorting is performed over a subset of τ (i.e., at most n tasks). The time-complexity of this operation is $O(n \cdot \log n)$ e.g., using Heapsort.
- Traverse the sorted list L (i.e., at most n tasks) and assign the tasks to processor types. The time-complexity of this operation is $O(n)$.

Thus, the time-complexity of the algorithm is at most

$$\underbrace{O(n)}_{\text{assign H1 tasks}} + \underbrace{O(n)}_{\text{assign H2 tasks}} + \underbrace{O(n \cdot \log n)}_{\text{sort L tasks}} + \underbrace{O(n)}_{\text{assign L tasks}} = O(n \cdot \log n)$$

6 Performance analysis of algorithm SA

In this section, we derive the approximation ratio of SA. For this, we mainly focus on the assignment of tasks in L since SA assigns tasks in H1 and H2 in the same way as an optimal intra-migrative assignment algorithm does.

First, we introduce a term, *swap solution*, that is extensively used in the rest of this section.

Definition 1 (***Swap solution***) A solution $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ to the LP formulation of Figure 2 is said to be a swap solution if and only if $\forall \tau_i, \tau_j \in$ L such that $\tau_i \neq \tau_j$ and $\frac{u_i^2}{u_i^1} \geq \frac{u_j^2}{u_j^1}$, it holds that: $x_i^1 = 1 \lor x_j^2 = 1$.

Property 1 (A single fractional task) From Definition 1, it can be easily shown that, in any swap solution $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$, there exists at most one task which is fractionally assigned to both processor types, i.e., there exists at most one index $f \in \{1, 2, \ldots, \ell\}$ such that $0 < x_f^1 < 1$ and $0 < x_f^2 < 1$.

The remainder of this section is organized as follows. In subsection 6.1, we describe a method to transform any feasible solution of the LP formulation (shown in Figure 2) into a feasible swap solution (Lemma 4). Then, in subsection 6.2, we show that the solution returned by SA for assigning tasks in L is similar to the *swap solution*, in the sense that, at most one task is fractionally assigned to both processor types and the rest are integrally assigned to type-1 and type-2 processors (Theorem 3). Finally, we show that, this fractional task can be integrally assigned to a processor type if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster (Theorem 4). Using all this information and considering that SA assigns tasks in H1 and H2 in a same way as an optimal intra-migrative task assignment algorithm does, we establish that, its approximation ratio is $1 + \frac{\alpha}{2}$.

6.1 The swapping method

We now show that any feasible solution to our LP formulation can be transformed into a feasible swap solution.

Lemma 4 *Any feasible solution $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ to the LP formulation of Figure 2 can be transformed into a feasible swap solution $S' = \{x_1^{1'}$, $x_1^{2'}$, $x_2^{1'}$, $x_2^{2'}$, $\ldots$, $x_\ell^{1'}$, $x_\ell^{2'}$, $Z'\}$ for which $Z' = Z$.*

Proof If S is not a swap solution, then we know by definition that there exists $\tau_p, \tau_q \in$ L such that:

$$\tau_p \neq \tau_q \quad \text{and} \quad \frac{u_p^2}{u_p^1} \geq \frac{u_q^2}{u_q^1} \quad \text{and} \quad x_p^1 < 1 \ \land \ x_q^2 < 1 \tag{13}$$

We prove the claim by (iteratively) transforming this solution S into another solution S' in which the following properties hold:

P1. $\forall \tau_i \in$ L, $\tau_i \neq \tau_p, \tau_i \neq \tau_q$: $x_i^{1'} = x_i^1$ and $x_i^{2'} = x_i^2$
P2. $x_p^{1'} = 1 \lor x_q^{2'} = 1$
P3. Constraints C1-C4 of LP formulation hold and $Z' = Z$

The steps involved in transforming solution S into S' are described below. Performing those steps iteratively as long as such a pair $\tau_p, \tau_q \in \mathrm{L}$ fulfilling Expression (13) exists, will ultimately lead to a feasible swap solution S' with Z' equal to Z. Property P1 and P2 ensure that, with each iteration, the solution is moving closer towards the swap solution and P3 ensures that this (intermediate) solution is feasible. At each iteration, we denote by $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ the feasible solution computed in the previous iteration (in the first iteration, this solution is the given one) and by $S' = \{x_1^{1'}, x_1^{2'}, x_2^{1'}, x_2^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z'\}$ the modified feasible solution after the current iteration (note that S' of iteration k acts as S in iteration $k+1$). The solution obtained after the final iteration is the feasible swap solution. Each iteration is performed as follows:
$\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$:

$$x_i^{1'} \leftarrow x_i^1 \tag{14}$$
$$x_i^{2'} \leftarrow x_i^2 \tag{15}$$

and

$$x_p^{1'} \leftarrow x_p^1 + \delta_1 \tag{16}$$
$$x_p^{2'} \leftarrow x_p^2 - \delta_1 \tag{17}$$
$$x_q^{1'} \leftarrow x_q^1 - \delta_2 \tag{18}$$
$$x_q^{2'} \leftarrow x_q^2 + \delta_2 \tag{19}$$

where $\delta_1 \overset{\text{def}}{=} \min(x_p^2, x_q^1 \times \frac{u_q^1}{u_p^1})$ and $\delta_2 \overset{\text{def}}{=} \min(x_p^2 \times \frac{u_p^1}{u_q^1}, x_q^1)$.

Proof of P1. From Expressions (14) and (15), it is trivial to see that Property **P1** holds.

Proof of P2. We have to consider two cases:
Case (i): $x_p^2 \leq x_q^1 \times \frac{u_q^1}{u_p^1}$. In this case, $\delta_1 = x_p^2$ and $\delta_2 = x_p^2 \times \frac{u_p^1}{u_q^1}$. Substituting the value of δ_1 in Expression (16) gives: $x_p^{1'} \leftarrow x_p^1 + x_p^2$. Since we know that $x_p^1 + x_p^2 = 1$ (it is true in the initial solution S and it holds true in all the subsequent iterations as well, as will be shown in **Proof of P3**), we get $x_p^{1'} \leftarrow 1$ and hence Property **P2** is satisfied.
Case (ii): $x_p^2 > x_q^1 \times \frac{u_q^1}{u_p^1}$. This case is analogous to the previous case. In this case, $\delta_1 = x_q^1 \times \frac{u_q^1}{u_p^1}$ and $\delta_2 = x_q^1$. Substituting the value of δ_2 in Expression (19) gives: $x_q^{2'} \leftarrow x_q^1 + x_q^2$. Since we know that $x_q^1 + x_q^2 = 1$ (it is true in the initial solution S and it holds true in all the subsequent iterations as well, as will be shown in **Proof of P3**), we get $x_q^{2'} \leftarrow 1$ and hence Property **P2** is satisfied.

Proof of P3. Since the initial solution S is feasible, constraint C1 holds by definition, i.e., $\forall \tau_i \in \mathrm{L} : x_i^1 + x_i^2 = 1$. Let us see whether this holds in solution S' which is obtained from S with the help of Expressions (14)-(19). Let us consider the following two cases:
Case (i): $\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$. Adding Expressions (14) and (15), we get: $x_i^{1'} + x_i^{2'} = x_i^1 + x_i^2$. Since we know that $\forall \tau_i \in \mathrm{L} : x_i^1 + x_i^2 = 1$, we obtain: $x_i^{1'} + x_i^{2'} = 1$. Recall that, in the next iteration, this solution S' acts as S while

computing another S'. Hence, this holds in that iteration and all subsequent iterations. Hence, constraint C1 holds true.
Case (ii): $\tau_i = \tau_p \vee \tau_i = \tau_q$. Analogous to the previous case, adding Expressions (16) and (17), gives: $x_p^{1'} + x_p^{2'} = 1$ and adding Expressions (18) and (19), gives: $x_q^{1'} + x_q^{2'} = 1$. This holds true in all the iterations for the reasons stated in the previous case. Hence, $\forall \tau_i \in L$, constraint C1 holds true.
Now, we show that constraint C2 holds. From Equations (14)–(19), we have:

$$\begin{aligned}\sum_{i=1}^{\ell}(x_i^{1'} \times u_i^1) = & \sum_{\substack{i=1\\ i\neq p, i\neq q}}^{\ell} (x_i^1 \times u_i^1)\\ & + \left(x_p^1 + \min\left(x_p^2, x_q^1 \times \frac{u_q^1}{u_p^1}\right)\right) \times u_p^1\\ & + \left(x_q^1 - \min\left(x_p^2 \times \frac{u_p^1}{u_q^1}, x_q^1\right)\right) \times u_q^1 \end{aligned} \quad (20)$$

We need to consider two sub-cases:
Case (iia): $x_p^2 \leq x_q^1 \times \frac{u_q^1}{u_p^1}$. In this case, Expression (20) becomes:

$$\sum_{i=1}^{\ell}(x_i^{1'} \times u_i^1) = \sum_{\substack{i=1\\ i\neq p, i\neq q}}^{\ell} (x_i^1 \times u_i^1) + x_p^1 \times u_p^1 + x_p^2 \times u_p^1 + x_q^1 \times u_q^1 - x_p^2 \times u_p^1$$

which can be rewritten as:

$$\sum_{i=1}^{\ell}(x_i^{1'} \times u_i^1) = \sum_{i=1}^{\ell}(x_i^1 \times u_i^1) \leq Z \times m_1 \quad (21)$$

Case (iib): $x_p^2 > x_q^1 \times \frac{u_q^1}{u_p^1}$. This case is analogous to the previous case and can be shown that Expression (20) simplifies to Expression (21).
Hence, Constraint C2 is not violated.
With analogous reasoning, it can be shown, for both the sub-cases (i.e, $x_p^2 \leq x_q^1 \times \frac{u_q^1}{u_p^1}$ and $x_p^2 > x_q^1 \times \frac{u_q^1}{u_p^1}$) that:

$$\sum_{i=1}^{\ell}(x_i^{2'} \times u_i^2) = \sum_{i=1}^{\ell}(x_i^2 \times u_i^2) \leq Z \times m_2 \quad (22)$$

Hence, Constraint C3 is also not violated.

Now let us consider constraint C4. We know by definition that in solution S, $\forall \tau_i \in$ L, it holds that $x_i^1 \geq 0$ and $x_i^2 \geq 0$. Hence, from Expressions (14) and (15), in solution S', $\forall \tau_i \in$ L$, \tau_i \neq \tau_p, \tau_i \neq \tau_q$, it holds that $x_i^{1'} \geq 0$ and $x_i^{2'} \geq 0$. Now for $\tau_i = \tau_p \vee \tau_i = \tau_q$, we have two cases:
Case (i): $x_p^2 \leq x_q^1 \times \frac{u_q^1}{u_p^1}$. In this case, we have $\delta_1 = x_p^2$ and $\delta_2 = x_p^2 \times \frac{u_p^1}{u_q^1}$. Since we have shown that constraint C1 holds, substituting the value of δ_1 in Expression (16) and (17), we get $x_p^{1'} = 1$ and $x_p^{2'} = 0$, respectively. From the case,

we have: $x_q^1 \geq x_p^2 \times \frac{u_p^1}{u_q^1} > 0$. So, substituting the value of δ_2 in Expression (18) and (19) gives us $x_q^{1'} \geq 0$ and $x_q^{2'} > 0$, respectively. Hence, constraint C4 holds in this case.

Case (ii): $x_p^2 > x_q^1 \times \frac{u_q^1}{u_p^1}$. This case is analogous to the previous case. In this case, we have $\delta_1 = x_q^1 \times \frac{u_q^1}{u_p^1}$ and $\delta_2 = x_q^1$. Since we have shown that constraint C1 holds, substituting the value of δ_2 in Expression (18) and (19), we get $x_q^{1'} = 0$ and $x_q^{2'} = 1$, respectively. From the case, we have: $x_p^2 \geq x_q^1 \times \frac{u_q^1}{u_p^1} > 0$. So, substituting the value of δ_1 in Expression (16) and (17) gives us $x_p^{1'} > 0$ and $x_p^{2'} \geq 0$, respectively. Hence, constraint C4 holds in this case. Thus, $\forall \tau_i \in L$, constraint C4 holds true.

Since none of the constraints, C1-C4, of LP formulation are violated, the transformed solution remains feasible, and from Expression (21) and Expression (22), we can conclude that $Z' = Z$. Thus, at the end of an iteration, for a pair of tasks τ_p, τ_q that we considered in the iteration, it holds that either $x_p^1 = 1 \vee x_q^2 = 1$. Hence, applying the transformation shown in Expressions (14)–(19) repeatedly, we obtain a feasible swap solution. $\square$

Lemma 5 *For any feasible swap solution $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ to the LP formulation, we can re-index tasks in* L *such that $\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1}$ (with ties broken favoring the task with lower index before re-indexing) and with this order, there is an index $f \in \{0, 1, 2, \ldots, \ell, \ell+1\}$ such that:*

$$\forall i \in \{1, 2, \ldots, L\} \text{ such that } i < f, \text{ it holds that: } x_i^1 = 1 \text{ and}$$
$$\forall i \in \{1, 2, \ldots, L\} \text{ such that } i > f, \text{ it holds that: } x_i^2 = 1$$

Proof Let $S = \{x_1^1, x_1^2, x_2^1, x_2^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ be any feasible swap solution. We re-index the tasks (together with x_i^1 and x_i^2 values in S, $\forall \tau_i \in$ L) such that

$$\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1} \tag{23}$$

with ties broken as described in the claim. We now prove that there exists $f \in \{0, 1, 2, \ldots, \ell, \ell+1\}$ such that $\forall \tau_i \in$ L, if $i < f$ then $x_i^1 = 1$ and if $i > f$ then $x_i^2 = 1$. The following three cases may arise (recall from Property 1 that, in a swap solution, there is at most one fractional task): (1) all the tasks in L are assigned to the same processor type or (2) tasks in L are assigned to both processor types and there is *one* fractional task or (3) tasks in L are assigned to both processor types and there is *no* fractional task. We now consider each of these cases separately below.

Case (1): All the tasks in L are assigned to processors of type-1 (resp., type-2); The claim trivially holds for $f = \ell + 1$ (resp., $f = 0$).

Case (2): The tasks in L are assigned to both processor types and there is *one* fractional task; let f be the index of this fractional task, i.e., there exists $\tau_f \in$ L for which $0 < x_f^1 < 1$ and $0 < x_f^2 < 1$. We need to consider two sub-cases:

Case 2.1 ($\forall \tau_i \in$ L **such that** $i < f$)**:** Since $\frac{u_i^2}{u_i^1} \geq \frac{u_f^2}{u_f^1}$, we know from Definition 1 that $x_i^1 = 1 \vee x_f^2 = 1$. However, by definition of f we know that τ_f is fractionally assigned and thus, $0 < x_f^2 < 1$; so, it must hold that $x_i^1 = 1$. Consequently, every task $\tau_i \in \mathrm{L}$ with $i < f$, is integrally assigned to type-1 processors.

Case 2.2 ($\forall \tau_i \in \mathrm{L}$ such that $i > f$): Since $\frac{u_f^2}{u_f^1} \geq \frac{u_i^2}{u_i^1}$, we know from Definition 1 that $x_f^1 = 1 \vee x_i^2 = 1$. Following the same reasoning as above, we have $0 < x_f^1 < 1$ and thus, it must hold that $x_i^2 = 1$. Hence, every task $\tau_i \in \mathrm{L}$ with $i > f$, is integrally assigned to type-2 processors.

Case (3): The tasks in L are assigned to both processor types and there is *no* fractional task. In this case, let f be the index of the first task in the sorted order (of tasks in L as shown in Expression (23)) that is integrally assigned to type-2 processors. By definition of τ_f, we know that all the tasks $\tau_i \in \mathrm{L}$ with $i < f$ must be integrally assigned to type-1 processors. Now consider any task $\tau_i \in \mathrm{L}$ with $i > f$. Since $\frac{u_f^2}{u_f^1} \geq \frac{u_i^2}{u_i^1}$, we know from Definition 1 that $x_f^1 = 1 \vee x_i^2 = 1$. But, we know that $x_f^1 = 0$, so it must hold that $x_i^2 = 1$. Hence, all tasks $\tau_i \in \mathrm{L}$ with $i > f$ are integrally assigned to type-2 processors.

We showed that the claim holds for all the cases, i.e., there exists an index $f \in \{0, 1, 2, \ldots, \ell, \ell + 1\}$ such that all the tasks in L (sorted as shown in Expression (23)) to its *left* are assigned to type-1 processors and all the tasks in L to its *right* are assigned to type-2 processors. Hence the proof. $\square$

6.2 The approximation ratio of SA

In this section, we show that the approximation ratio of algorithm, SA, is $1 + \frac{\alpha}{2}$. Before that, we prove a property of SA which in turn helps us to prove its approximation ratio.

Theorem 3 *If there exists an intra-migrative feasible assignment of τ on π then SA succeeds in finding a feasible assignment of τ on π in which at most one task from* L *is fractionally assigned to both processor types and the rest are integrally assigned to type-1 and type-2 processors.*

Proof We know from Corollary 1 that if τ is intra-migrative feasible on π then MILP-Algo succeeds in finding such an intra-migrative feasible assignment. This implies that there exists a feasible solution to the MILP formulation of Figure 1 with $Z_{\text{MILP}} \leq 1$. Then, we know from Lemma 1 that, since there exists a solution to the MILP formulation with $Z_{\text{MILP}} \leq 1$, there also exists a feasible solution to the LP formulation of Figure 2 with $Z_{\text{LP}} \leq 1$. We also know from Lemma 4 that such a solution can be converted into a feasible swap solution in which at most one task from L is fractionally assigned. Finally, we know from Lemma 5 that in this feasible swap solution, tasks in L can be re-indexed such that $\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1}$ (with ties broken, during re-indexing favoring the task with lower index before re-indexing) and with this order, there is an index $f \in \{0, 1, \ldots, \ell, \ell + 1\}$ such that:

$$\forall i \in \{1, 2, \ldots, L\} \text{ such that } i < f, \text{ it holds that: } x_i^1 = 1 \text{ and}$$
$$\forall i \in \{1, 2, \ldots, L\} \text{ such that } i > f, \text{ it holds that: } x_i^2 = 1$$

For the sake of readability, henceforth we simply denote by $S = \{x_1^1,\ x_1^2,\ x_2^1,\ x_2^2,\ \ldots,\ x_\ell^1,\ x_\ell^2,\ Z\}$ this sorted feasible swap solution (in which tasks are sorted as mentioned above). With this background, we now prove the theorem. The intuition behind the proof is that, SA always succeeds in returning a solution similar to the sorted feasible swap solution S (from the reasoning above, we already know that, such a swap solution always exists if τ is intra-migrative feasible on π).

We prove the theorem by contradiction. Let us assume that the task set τ is intra-migrative feasible on π but SA fails to find an assignment of τ on π in which at most one task from L is fractionally assigned. We consider all the scenarios and show that it is impossible for this to happen.

Let us study the behavior of SA. It assigns tasks in H1 and H2 in the same manner as an optimal intra-migrative task assignment algorithm does (see the algorithm, MILP-Algo, in Section 3). Hence, we only need to look at the assignment of tasks in L. It considers these tasks in the order:

$$\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1} \tag{24}$$

with ties broken favoring the task with lower index before re-indexing. It considers tasks one by one from the left-hand side in the sorted order (as shown in Expression (24)) and starts assigning them to type-1 processors. It stops assigning tasks to type-1 processors upon failing to assign a task say, τ_x, integrally on type-1 processors or all the tasks are successfully assigned, thereby resulting in a successful assignment — whichever happens first. If it stops at τ_x then it considers tasks one by one from the right-hand side in the sorted order and starts assigning them to type-2 processors. It stops assigning tasks to processors of type-2 as soon as it fails to assign a task integrally (if τ is intra-migrative feasible on π then this task can be none other than τ_x as shown later in the theorem) or it successfully assigns τ_x integrally to processors of type-2, thereby resulting in a successful assignment — whichever happens first. If it stopped because it could not assign τ_x integrally to type-2 processor then it fractionally assigns τ_x to type-1 and type-2 processors.

We now compare the output of SA with that of the sorted feasible swap solution S and show that it is impossible for SA to fail (i.e., not to return an assignment with at most one fractional task) when τ is intra-migrative feasible on π. Note that the tasks are indexed in the same manner in both SA and S, i.e., $\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1}$, with ties broken in the same way.

We need to consider two cases with respect to the existence of a fractional task in S, i.e., a task τ_f for which $0 < x_f^1 < 1$ and $0 < x_f^2 < 1$. The remainder of the proof consists in exploring all the possible scenarios (and showing that each case leads to contradiction): it is first split into two parts, corresponding to the two cases 'such a fractional task exists or not', and each part is further divided into three cases.

Part 1: There exists a task $\tau_f \in$ L in the swap solution S which is fractionally assigned to both processor types, i.e., $0 < x_f^1 < 1$ and $0 < x_f^2 < 1$. In this part, we need to consider three cases with respect to the position of x and f.

Case 1.1 ($x < f$)**:** We know that tasks $\{\tau_1, \tau_2, \ldots, \tau_{f-1}\} \in$ L have been integrally assigned to type-1 processors in solution S, i.e., $\forall i \in \{1, 2, \ldots, f-1\}$: $x_i^1 = 1 \ \wedge\ x_i^2 = 0$. This means that $U^1 + \sum_{i=1}^{f-1} u_i^1 \leq m_1$ where $U^1 = \sum_{\tau_i \in \mathrm{H1}} u_i^1$ and since $x < f$, it must hold that:

$$U^1 + \sum_{i=1}^{x} u_i^1 \leq m_1 \tag{25}$$

i.e., tasks $\{\tau_1, \tau_2, \ldots, \tau_x\} \in$ L have been integrally assigned to processors of type-1 in S. However, we know that SA failed to integrally assign those tasks $\{\tau_1, \tau_2, \ldots, \tau_x\}$ to type-1 processors, which means that $U^1 + \sum_{i=1}^{x} u_i^1 > m_1$. This contradicts Expression (25).

Case 1.2 ($x > f$): This case is symmetrical to Case 1.1 and also leads to a contradiction. We know that tasks $\{\tau_{f+1}, \tau_{f+2}, \ldots, \tau_\ell\} \in$ L have been integrally assigned to type-2 processors in solution S, i.e., $\forall i \in \{f+1, f+2, \ldots, \ell\}$: $x_i^1 = 0 \;\wedge\; x_i^2 = 1$. This means that $U^2 + \sum_{i=f+1}^{\ell} u_i^2 \leq m_2$, where $U^2 = \sum_{\tau_i \in \text{H2}} u_i^2$. Further, since $x > f$, it must also hold that:

$$U^2 + \sum_{i=x}^{\ell} u_i^2 \leq m_2 \tag{26}$$

i.e., tasks $\{\tau_x, \tau_{x+1}, \ldots, \tau_\ell\} \in$ L have been integrally assigned to processors of type-2 in S. However, we know that SA failed to integrally assign those tasks $\{\tau_x, \tau_{x+1}, \ldots, \tau_\ell\}$ to type-2 processors, which means that $U^2 + \sum_{i=x}^{\ell} u_i^2 > m_2$. This contradicts Expression (26).

Case 1.3 ($x = f$): This indicates that the two sets of tasks $\{\tau_1, \tau_2, \ldots, \tau_{x-1}\} \in$ L and $\{\tau_{x+1}, \tau_{x+2}, \ldots, \tau_\ell\} \in$ L are integrally assigned to type-1 and type-2 processors, respectively, in both S and the solution returned by SA. Let $x_f^{1,S}$ denote the fraction of $\tau_f \in$ L assigned to type-1 processors in S, and similarly let $x_x^{1,\text{SA}}$ denote the fraction of $\tau_x \in$ L assigned to type-1 processors in the solution returned by SA. Since S is feasible, we know that, $U^1 + \sum_{i=1}^{f-1} u_i^1 + x_f^{1,S} \times u_f^1 \leq m_1$, and since $f = x$ we have:

$$U^1 + \sum_{i=1}^{x-1} u_i^1 + x_f^{1,S} \times u_x^1 \leq m_1 \tag{27}$$

But, by design (see step 6 of SA algorithm in Section 5), we also know that τ_x is split under SA such that:

$$U^1 + \sum_{i=1}^{x-1} u_i^1 + x_x^{1,\text{SA}} \times u_x^1 = m_1 \tag{28}$$

From Expression (27) and (28), we then observe that $x_f^{1,S} \leq x_x^{1,\text{SA}}$. As a first conclusion, SA is thus able to integrally assign to type-1 processors all the tasks in τ that are integrally assigned to type-1 processors in solution S, plus (at least) the same fraction of task τ_x as that of task τ_f assigned to type-1 processors in S. Also, $x_f^{1,S} \leq x_x^{1,\text{SA}}$ implies that $x_f^{2,S} \geq x_x^{2,\text{SA}}$, which in turn, yields:

$$U^2 + \sum_{i=f+1}^{n} u_i^2 + x_f^{2,S} \times u_f^2 \;\geq\; U^2 + \sum_{i=x+1}^{n} u_i^2 + x_x^{2,\text{SA}} \times u_x^2$$

The left-hand (resp., right-hand) side of the above expression denotes the utilization of the tasks, including the fractional assignment of τ_f (which is same task as

τ_x — from the case), assigned to type-2 processors in the solution S (resp., in the solution returned by SA). As a second conclusion, SA is thus able to integrally assign to type-2 processors all the tasks in τ that are integrally assigned to type-2 processors in solution S, and assign no greater fraction of the task τ_x (which is same task as τ_f) to type-2 processors than in solution S. So, SA succeeds in assigning all the tasks and hence this leads to a contradiction.

Thus, for the case when there is a fractional task in the swap solution, we have shown that all the three sub-cases lead to contradiction.

Part 2: There is no fractional task in solution S. Let τ_f be the first task that is integrally assigned to type-2 processor in S. Again, we need to consider three cases with respect to the position of x and f.

Case 2.1 ($x < f$): This case is analogous to Case 1.1 and leads to a contradiction.
Case 2.2 ($x > f$): This case is analogous to Case 1.2 and leads to a contradiction.
Case 2.3 ($f = x$): This indicates that SA was able to integrally assign tasks $\{\tau_1, \ldots, \tau_{x-1}\} \in L$ to type-1 processors as in S. However, it failed to integrally assign tasks $\{\tau_x, \ldots, \tau_\ell\} \in L$ to type-2 processors that are integrally assigned in solution S. This means $U^2 + \sum_{i=x}^{\ell} u_i^2 > m_2$ whereas $U^2 + \sum_{i=f}^{\ell} u_i^2 \leq m_2$. From the case (i.e., $f = x$), this is a contradiction and hence SA would also succeed in assigning those tasks to type-2 processors.

Thus, for the case when there is no fractional task in the swap solution, we have shown that all the three sub-cases lead to contradiction.

From Parts 1 and 2 of the proof, we have shown that all the cases lead to contradiction, hence proving the theorem. □

Theorem 4 (Approximation ratio of SA)
If there exists a feasible intra-migrative assignment of τ on π then, using SA, it is guaranteed to obtain such a feasible intra-migrative assignment of τ on $\pi^{(1+\frac{\alpha}{2})}$.

Proof We know from Theorem 3 that, if τ is intra-migrative feasible on π then SA succeeds in returning a feasible assignment of τ on π, in which, at most one task from L is fractionally assigned and the rest are integrally assigned to type-1 and type-2 processors. It follows from Lemma 3 (on page 13) that, this fractional task can also be assigned integrally to one of the processor types, if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster. Hence the proof. □

We now show that the proven approximation ratio of SA algorithm is a tight bound. This is shown using the same technique that was used earlier (Theorem 2 in Section 4) to show that the proven approximation ratio of LP-Algo is tight and also the same problem instance is used here (and for the sake of convenience, the problem instance is repeated).

Theorem 5 (Approximation ratio of SA algorithm is tight)
The proven approximation ratio 1.5 *of algorithm SA is a tight bound.*

Proof In order to show that the proven approximation ratio is tight for algorithm SA, it is sufficient to show that, there exists a (feasible intra-migrative) problem

Tasks	**Utilizations of tasks**	
	$\mathbf{u_i^1}$	$\mathbf{u_i^2}$
τ_1	0.5	0.5
τ_2	1.0	1.0
τ_3	0.5	0.5

Table 5: An example to illustrate that the proven approximation ratio of SA algorithm is a tight bound.

Processor types	**Tasks assigned**
type-1 (π_1)	τ_1 and τ_3
type-2 (π_2)	τ_2

Table 6: A feasible intra-migrative assignment for tasks shown in Table 5 on platform π.

Processor types	**Tasks assigned by SA**
type-1 (π_1)	100% of τ_1 and 50% of τ_2
type-2 (π_2)	100% of τ_3 and 50% of τ_2

Table 7: The assignment output by SA for tasks shown in Table 5 on platform π.

instance for which SA needs 1.5 times faster processors to output a feasible intra-migrative assignment. We now show that such a problem instance exists.

Consider a problem instance with a task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ comprising three tasks and a two-type platform $\pi = \{\pi_1, \pi_2\}$ comprising two processors. Let π_1 be a processor of type-1 and π_2 be a processor of type-2. The utilizations of tasks are shown in Table 5.

Observe that the given task set τ is intra-migrative feasible on the given platform π. A feasible intra-migrative assignment is obtained by assigning (i) τ_1 and τ_3 to type-1 processors (which has a single processor, π_1) and (ii) τ_2 to type-2 processors (which has a single processor, π_2). This assignment is shown in Table 6.

Now consider algorithm SA. Initially, the task set is partitioned as follows using Expressions (5)–(8): $H12 = \emptyset$, $H1 = \emptyset$, $H2 = \emptyset$ and $L = \{\tau_1, \tau_2, \tau_3\}$. Since all the tasks in the task set are light, SA sorts the tasks in non-increasing order of $\frac{u_i^2}{u_i^1}$. Since this ratio is same for all the three tasks, a sorted order is as follows: $\tau_1 \to \tau_2 \to \tau_3$. With this sorted order, SA assigns the tasks as shown in Table 7. In the assignment output by SA (which is shown in Table 7), it holds that:

– type-1 processors are fully utilized
– type-2 processors are fully utilized and
– task τ_2 is equally split between type-1 and type-2 processors

In order to assign τ_2 integrally to type-1 processors, the speed of type-1 processors *must be* increased to 1.5. Analogously, for assigning τ_2 integrally to type-2 processors, the speed of type-2 processors *must be* increased to 1.5 as well. Therefore, a speedup of 1.5 is required to assign τ_2 integrally to one of the processor types.

Hence, the proven approximation ratio 1.5 of algorithm SA is a tight bound. □

Remark 3 Although Theorem 4 states that, for an intra-migrative feasible task set, SA needs a platform in which *every processor* is $1+\frac{\alpha}{2}$ times faster, in order to output such a feasible intra-type assignment, it is trivial to see that a platform in which only *one processor* is $1+\frac{\alpha}{2}$ times faster is sufficient (to which the fractional task can be integrally assigned).

Corollary 4 *If there exists a feasible intra-migrative assignment of τ on $\pi(m_1, m_2)$ then, using SA, it is guaranteed to obtain such a feasible intra-migrative assignment of τ on $\pi'(m_1+1, m_2)$, which has one additional processor of type-1 compared to π.*

Proof It follows from Theorem 3 that if there exists an intra-migrative feasible assignment of τ on π then SA succeeds in returning a feasible assignment of τ on π in which at most one task from L, say τ_f, is fractionally assigned to both the processor types and the rest are integrally assigned to type-1 and type-2 processors. From Corollary 3, we know that, if such a task τ_f exists then it can be integrally assigned to the set of type-1 processors in π', which has an additional processor compared to π. Hence the proof. □

Remark 4 It is trivial to see that Corollary 4 holds true if SA is given a platform $\pi'(m_1, m_2+1)$, which has one additional processor of type-2 compared to π.

7 SA-P: A non-migrative task assignment algorithm

We now present a *non-migrative* task assignment algorithm, SA-P, an enhanced version of SA, for assigning tasks in τ to individual processors on a two-type platform π. We also evaluate its performance, against a *powerful adversary*, i.e., against an optimal intra-migrative assignment algorithm.

7.1 The description of algorithm SA-P

For this algorithm, we consider that the processors are indexed in some order and this indexing is maintained throughout the algorithm. The new algorithm, SA-P, for assigning tasks to processors, works as follows.

1. Assign tasks in τ to processor types on π using SA.
   - SA assigns tasks to only processor types (and not to individual processors); let τ^1 (resp., τ^2) be the subset of tasks assigned to type-1 (resp., type-2) processors.
   - SA guarantees that, for an intra-migrative feasible task set, at most one task is fractionally assigned to both processor types; let τ_f be this task and let fraction x_f^1 of τ_f be assigned to type-1 and fraction $x_f^2 = 1 - x_f^1$ be assigned to type-2.

2. Assign tasks from τ^1 (resp., τ^2) to individual processors of type-1 (resp., type-2) using next-fit but allowing *splitting* of tasks between consecutive processors (also referred to as "wrap-around" assignment in literature). Assign fraction x_f^1 of τ_f to the last processor (i.e., the m_1^{th} processor) of type-1 and fraction x_f^2 to the last processor (i.e., the m_2^{th} processor) of type-2. It is trivial to see that such an assignment ensures following properties:
   - at most $m_1 - 1$ tasks are *split* between processors of type-1 with one task split between each pair of consecutive processors
   - at most $m_2 - 1$ tasks are *split* between processors of type-2 with one task split between each pair of consecutive processors and
   - at most *one* task, τ_f, is fractionally assigned between processors of type-1 and type-2; specifically, τ_f is split between the m_1^{th} processor of type-1 and the m_2^{th} processor of type-2
3. Copy this assignment of tasks onto a faster platform π' (we show in Theorem 6 that a platform in which every processor is $1 + \alpha$ times faster than the corresponding processor in π is sufficient).
4. On platform π', assign a task split between processor p and $p + 1$ of type-1 to processor p, where $1 \leq p < m_1$; similarly, assign a task split between processor q and $q+1$ of type-2 to processor q, where $1 \leq q < m_2$. Finally, assign the task τ_f to the m_1^{th} processor of type-1 (or to the m_2^{th} processor of type-2).

SA-P is named so because it is the "**P**artitioned" (i.e., non-migrative) version of algorithm SA.

7.2 Time-complexity of algorithm SA-P

We now show that the time-complexity of SA-P algorithm is a low-degree polynomial function of the number of tasks (n). By inspecting the four steps of algorithm, SA-P, described above, we know that:

- In Step 1, tasks are assigned to processor types using SA. The time-complexity of this operation is $O(n \cdot \log n)$ — see Section 5.2.
- In Step 2, tasks that are assigned to type-1 (resp., type-2) processors by SA (at most n) are assigned to individual processors of type-1 (resp., type-2) using "wrap-around" technique. The time-complexity of each of these operations is $O(n)$.
- In Step 3, the assignment (of n tasks) is copied onto a faster platform. The time-complexity of this operation is $O(n)$.
- In Step 4, tasks that are fractionally assigned (at most m) are integrally assigned. The time-complexity of this operation is $O(n)$ since the number of fractionally assigned tasks is upper bounded by n.

Thus, the time-complexity of the algorithm is at most

$$\underbrace{O(n \cdot \log n)}_{\text{Step 1}} + \underbrace{O(n)}_{\text{Step 2}} + \underbrace{O(n)}_{\text{Step 3}} + \underbrace{O(n)}_{\text{Step 4}} = O(n \cdot \log n)$$

8 Performance analysis of algorithm SA-P

In this section, we derive the approximation ratio of SA-P.

Theorem 6 (Approximation ratio of SA-P)
If there exists a feasible intra-migrative assignment of τ on π then SA-P is guaranteed to find a feasible non-migrative assignment of τ on $\pi^{(1+\alpha)}$.

Proof We know from Theorem 3 that if τ is intra-migrative feasible on π then SA succeeds in returning an assignment of tasks in τ to processor types on π, in which, at most one task from L is fractionally assigned to both processor types and the rest are integrally assigned to type-1 and type-2 processors. Hence, we only need to show that, if SA assigns tasks in τ to processor types on π with at most one fractional task then SA-P can assign tasks in τ to individual processors on $\pi^{(1+\alpha)}$ in which the speed of each processor is $1+\alpha$ times faster than that of the corresponding processor in π.

Let us consider the assignment of tasks in τ to processor types on π, returned by SA, with at most one fractional task. We know that, SA assigns tasks to processor types (and not to individual processors) — let τ^1 (resp., τ^2) denote the subset of tasks that are assigned to processors of type-1 (resp., type-2). Let τ_f denote the task that is fractionally assigned to both processor types — fraction x_f^1 to type-1 and fraction $x_f^2 = 1 - x_f^1$ to type-2 processors. Clearly, $\tau = \tau^1 \cup \tau^2 \cup \{\tau_f\}$ and $\tau^1 \cap \{\tau_f\} = \emptyset$ and $\tau^2 \cap \{\tau_f\} = \emptyset$ and finally $\tau^1 \cap \tau^2 = \emptyset$. We also know that:

$$\forall \tau_i \in \tau^1 : \; u_i^1 \leq \alpha \qquad \text{and} \tag{29}$$

$$\forall \tau_i \in \tau^2 : \; u_i^2 \leq \alpha \qquad \text{and} \tag{30}$$

$$\tau_f \in \tau : \; u_f^1 \leq \alpha \;\; \wedge \;\; u_f^2 \leq \alpha \tag{31}$$

SA-P uses this assignment information and assigns tasks to individual processors (using "wrap-around" technique, which allows splitting of tasks between processors of same type), as described earlier in Step 2 of SA-P algorithm. After this step, it must hold that:

$$\forall p \in \pi : U[p] \leq 1 \tag{32}$$

where $U[p]$ denotes the utilization of tasks that are assigned to processor p. Let $\tau^1_{p_1,p_1+1}$ denote the task split between the p_1^{th} processor and the $(p_1+1)^{th}$ processor of type-1 where $1 \leq p_1 < m_1$. Analogously, let $\tau^2_{p_2,p_2+1}$ denote the task split between the p_2^{th} processor and the $(p_2+1)^{th}$ processor of type-2 where $1 \leq p_2 < m_2$.

On step 3, SA-P copies this assignment onto the faster platform $\pi^{(1+\alpha)}$. Let $u_i^{1'}$ and $u_i^{2'}$ denote the utilizations of task τ_i on platform $\pi^{(1+\alpha)}$. Then, it holds that:

$$\forall \tau_i \in \tau : \frac{u_i^{2'}}{u_i^2} = \frac{u_i^{1'}}{u_i^1} = \frac{1}{1+\alpha} \tag{33}$$

Combining Expression (32) and (33) gives us:

$$\forall p \in \pi^{(1+\alpha)} : U[p] \leq \frac{1}{1+\alpha} \tag{34}$$

Also, combining Expressions (29)-(31) and (33), we get:

$$\forall \tau_i \in \tau^1 : \;\; u_i^{1'} \leq \frac{\alpha}{1+\alpha} \qquad \text{and} \tag{35}$$

$$\forall \tau_i \in \tau^2 : \;\; u_i^{2'} \leq \frac{\alpha}{1+\alpha} \qquad \text{and} \tag{36}$$

$$\tau_f \in \tau : \; u_f^{1'} \leq \frac{\alpha}{1+\alpha} \;\; \wedge \;\; u_f^{2'} \leq \frac{\alpha}{1+\alpha} \tag{37}$$

On step 4, SA-P assigns the split tasks integrally. So, $\forall p_1 \in$ type-1 of $\pi^{(1+\alpha)}$, it moves the fraction of the task $\tau^1_{p_1,p_1+1}$ that is assigned to the $(p_1+1)^{th}$ processor of type-1 to the p_1^{th} processor of type-1. After this re-assignment, it follows from Expressions (34) and (35) that:

$$\forall p_1 \in \text{type-1 of } \pi^{(1+\alpha)} \ \wedge \ p_1 \neq m_1 : \ U[p_1] \leq 1.0 \tag{38}$$

Note that the m_1^{th} processor of type-1 is still utilized at most $\frac{1}{1+\alpha}$ of its capacity as no fraction of any task is moved to this processor in the above step.

Analogously, $\forall p_2 \in$ type-2 of $\pi^{(1+\alpha)}$, SA-P moves the fraction of the task $\tau^2_{p_2,p_2+1}$ that is assigned to the $(p_2+1)^{th}$ processor of type-2 to the p_2^{th} processor of type-2. After this re-assignment, it follows from Expressions (34) and (36) that:

$$\forall p_2 \in \text{type-2 of } \pi^{(1+\alpha)} \ \wedge \ p_2 \neq m_2 : \ U[p_2] \leq 1.0 \tag{39}$$

Once again, since no fraction of any task is moved to the m_2^{th} processor of type-2 in the above step, this processor is still utilized at most $\frac{1}{1+\alpha}$ of its capacity.

Finally, the task τ_f (split between the m_1^{th} processor of type-1 and the m_2^{th} processor of type-2) remains to be integrally assigned. It turns out that this task can be entirely assigned to either the m_1^{th} processor or the m_2^{th} processor. Consider the case that, it is integrally assigned to the m_1^{th} processor of type-1. Since, this processor is used at most $\frac{1}{1+\alpha}$ of its capacity and since $u_f^{1'} \leq \frac{\alpha}{1+\alpha}$ (see Expression (37)), this re-assignment does not allow the used capacity of m_1^{th} processor to exceed one. Combining this with the fact that the m_2^{th} processor of type-2 is still utilized at most $\frac{1}{1+\alpha}$ of its capacity and with Expression (38) and Expression (39), we obtain:

$$\forall p \in \pi^{(1+\alpha)} : U[p] \leq 1.0 \tag{40}$$

(Analogous reasoning holds for the case when τ_f is integrally assigned to the m_2^{th} processor of type-2.)

Since Expression (40) is a necessary and sufficient feasibility condition for task assignment on a uniprocessor (Liu and Layland, 1973), the non-migrative assignment of τ on $\pi^{(1+\alpha)}$ returned by SA-P is feasible. Hence the proof. □

We now show that the proven approximation ratio of SA-P is a tight bound.

Theorem 7 (Approximation ratio of SA-P is tight)
The proven approximation ratio 2 *of algorithm SA-P is a tight bound.*

Proof In order to show that, the proven approximation ratio is tight for algorithm SA-P, it is sufficient to show that, there exists a (feasible intra-migrative) problem instance for which SA-P needs 2 times faster processors to output a feasible non-migrative assignment. We now show that such a problem instance exists.

Consider a problem instance with a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ comprising n tasks and a two-type platform $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ comprising m processors of which m_1 processors are of type-1 and m_2 processors are of type-2. Also, $n = m_1 + m_2 + 2$. The task set τ can be partitioned into two subsets, τ^1 of $m_1 + 1$

tasks and τ^2 of $m_2 + 1$ tasks, such that:

$$\tau^1 \cup \tau^2 = \tau$$
$$\tau^1 \cap \tau^2 = \emptyset$$
$$\forall \tau_i \in \tau^1 : \quad u_i^1 = \frac{m_1}{m_1+1} \qquad \text{and} \quad u_i^2 = \frac{m_1}{m_1+1} + \frac{1}{(m_1+1)^2}$$
$$\forall \tau_i \in \tau^2 : \quad u_i^1 = \frac{m_2}{m_2+1} + \frac{1}{(m_2+1)^2} \qquad \text{and} \quad u_i^2 = \frac{m_2}{m_2+1}$$

Now consider algorithm SA-P. Initially, the task set τ is partitioned as follows using Expressions (5)–(8): $H12 = \emptyset$, $H1 = \emptyset$, $H2 = \emptyset$ and $L = \{\tau_1, \tau_2, \ldots, \tau_n\}$. As a consequence, it holds that $L = \tau^1 \cup \tau^2$. Since all the tasks in the task set are light, SA-P sorts the tasks in non-increasing order of $\frac{u_i^2}{u_i^1}$. From the utilizations of the tasks, it can be seen that, in such a sorted order, all the tasks from τ^1 precede all the tasks from τ^2 (i.e., all the tasks from τ^1 appear before any task from τ^2 in the list). Since $\forall \tau_i \in \tau^1 : u_i^1 = \frac{m_1}{m_1+1}$ and $|\tau^1| = m_1 + 1$, it can be seen that: $\sum_{\tau_i \in \tau^1} u_i^1 = m_1$. Combining this with the fact that, all the tasks of τ^1 appear before any task of τ^2 in the sorted order and the fact that, there are m_1 processors of type-1, it can be seen that SA-P assigns all the tasks of τ^1 to type-1 processors. Analogously, it can be seen that SA-P assigns all the tasks of τ^2 to type-2 processors. Note that, at this stage, tasks have been assigned to processor types and not to individual processors. Now, the tasks need to be assigned to individual processors.

Consider tasks of τ^1 that are assigned to type-1 processors. We know that $|\tau^1| = m_1 + 1$ and there are m_1 processors of type-1 (i.e., one processor less than the number of tasks). Hence, to obtain a non-migrative assignment (i.e., task-to-processor assignment), SA-P must assign two tasks of τ^1 to at least one processor of type-1. Since, $\forall \tau_i \in \tau^1 : u_i^1 = \frac{m_1}{m_1+1}$, we need to speedup at least one processor of type-1 (which is the processor to which two tasks from τ^1 will be assigned) to $\frac{2m_1}{m_1+1}$. Analogously, we need to speedup at least one processor of type-2 to $\frac{2m_2}{m_2+1}$. By the definition of approximation ratio, we need to speedup every processor by the same factor. Therefore, we need to speedup every processor by a factor of:

$$\max\left\{\frac{2m_1}{m_1+1}, \frac{2m_2}{m_2+1}\right\}$$

Rewriting the above max term gives us: we need to speedup every processor by a factor of:

$$2 \times \max\left\{\frac{m_1}{m_1+1}, \frac{m_2}{m_2+1}\right\}$$

In the above expression, the maximum value that the max term can take is 1 when either m_1 tends to an infinitely large value or when m_2 tends to an infinitely large value. Therefore, we need to speedup every processor by a factor of 2.
Hence the proof. □

Corollary 5 *If there exists a feasible intra-migrative assignment of τ on $\pi(m_1, m_2)$ then SA-P is guaranteed to obtain a feasible non-migrative assignment of τ on $\pi'(2m_1, 2m_2)$.*

Proof We know from Theorem 6 that, after executing Step 1 in SA-P, it holds that:

- the utilization of any task that is assigned to processors of type-1 (resp., type-2) does not exceed α on processors of type-1 (resp., type-2) — see Expression (29) and Expression (30) and
- the utilization of the task split between processors of type-1 and type-2 does not exceed α on both processor types — see Expression (31)

Also, we know from Theorem 6 that, after executing Step 2 in SA-P, it holds that:

- every processor is utilized at most 100% of its capacity (see Expression (32)) and
- at most $m_1 - 1$ (resp., $m_2 - 1$) tasks are *split* between processors of type-1 (resp., type-2) with one task split between each pair of consecutive processors and at most 1 task is split between processors of type-1 and type-2

Hence, if such fractional tasks exist then

- the m_1-1 (resp., m_2-1) tasks that are fractionally assigned between processors of type-1 (resp., type-2) can be integrally assigned to the additional $m_1 - 1$ (resp., $m_2 - 1$) processors of type-1 (resp., type-2) in π'
- the single task that is fractionally assigned between processors of type-1 and type-2 can be integrally assigned to yet another additional processor of either type-1 or type-2 in π' (since only $m_1 - 1$ (resp., $m_2 - 1$) additional processors of type-1 (resp., type-2) were used in the previous step out of m_1 (resp., m_2) additional processors).

From earlier observations about the capacity used on each processor and the utilizations of the tasks assigned on each processor type, it is trivial to see that, the above re-assignment satisfies the uniprocessor feasibility test on every processor in π'. Hence the proof. □

9 Average-case performance evaluation of algorithms

After studying the theoretical bounds of algorithms SA and SA-P (i.e., their approximation ratios), we evaluated their average-case performance using randomly generated task sets and by measuring how well the algorithms perform compared to their theoretical bounds. We assessed their performance by measuring their *minimum required speedup factor* for various task sets. For a given task set and an algorithm $\mathcal{A}$ ($\mathcal{A}$ is either SA or SA-P), we define the minimum required speedup factor, as the *minimum* amount of extra speed of processors that $\mathcal{A}$ needs, so as to succeed, in finding a feasible task assignment (in case of SA, it is task-to-processor-type assignment and in case of SA-P, it is task-to-processor assignment) as compared to an optimal intra-migrative task assignment algorithm. By definition, for any intra-migrative feasible task set, the minimum required speedup factor of SA (resp., SA-P) is upper bounded by its approximation ratio, $1 + \frac{\alpha}{2}$ (resp., $1 + \alpha$). For each task set, we evaluate the performance of both algorithms by comparing the measured *minimum required speedup factor* with the theoretically derived *approximation ratio*. In our evaluations, we observed that, for the vast majority of task sets, our algorithms performed significantly better by succeeding in finding a feasible task assignment with minimum required speedup factors much smaller than the respective approximation ratios. We now discuss these evaluations in detail.

The problem instances (number of tasks, their utilizations and the number of processors of each type) were generated randomly. Each problem instance had at most 25 tasks and at most 3 processors of each type. Specifically, for each problem instance, the number of tasks is generated randomly in the range $[1, 25]$ using uniform distribution, the number of type-1 processors is generated randomly in the range $[1, 3]$ using uniform distribution, the number of type-2 processors is generated randomly in the range $[1, 3]$ using uniform distribution and the utilizations of each task on each processor type were generated randomly in the range $(0, 1.0]$ using uniform distribution. We generated 100000 task sets, denoted as $\{\tau^{(1)}, \tau^{(2)}, \ldots, \tau^{(100000)}\}$, which we transformed into "intra-migrative critically feasible task sets". We define an *intra-migrative critically feasible task set* as a task set which is intra-migrative feasible on a given two-type platform but rendered (intra-migrative) infeasible if all the task utilizations (i.e., both u_i^1 and u_i^2 of each task) are increased by an arbitrarily small factor. The intuition behind using critically feasible task sets in our evaluations is that it is "hard" to find a feasible assignment for these task sets since only a few task assignments are feasible among all possible assignments.

To obtain an intra-migrative *critically* feasible task set $\tau_{\text{crit}}^{(k)}$ from a randomly generated task set $\tau^{(k)}$, where $k \in \{1, 2, \ldots, 100000\}$, we perform the task-to-processor-type assignment of $\tau^{(k)}$ by formulating the assignment problem as MILP (as shown in Figure 1 on page 10) and feeding it to an MILP solver (we used IBM ILOG CPLEX (IBM, 2012)) which outputs Z, the utilization of the most utilized processor type. Then, we multiply all the task utilizations by $1/Z$ and repeatedly feed it back to the solver until $0.99 < Z \leq 1$, which gives us $\tau_{\text{crit}}^{(k)}$.

For each intra-migrative critically feasible task set $\tau_{\text{crit}}^{(k)}$ and algorithm $\mathcal{A}$ (where $\mathcal{A}$ is either SA or SA-P), we measure the *minimum required speedup factor* denoted by $\text{MRSF}_{\mathcal{A}}^{(k)}$. We then compare $\text{MRSF}_{\mathcal{A}}^{(k)}$ with the approximation ratio denoted by $\text{AR}_{\mathcal{A}}^{(k)}$. Algorithm 1 shows how we compute $\text{MRSF}_{\mathcal{A}}^{(k)}$ for every intra-migrative critically feasible task set, $\tau_{\text{crit}}^{(k)}$. On line 3, we initially set $\text{MRSF}_{\mathcal{A}}^{(k)}$ to 1.0 as it denotes the speed of processors on which an optimal intra-migrative task assignment algorithm succeeds in finding a feasible intra-migrative task assignment for $\tau_{\text{crit}}^{(k)}$. Then, we input the task set to algorithm $\mathcal{A}$ (on line 5) and if $\mathcal{A}$ cannot find a feasible assignment (which is intra-migrative for SA and non-migrative for SA-P), the minimum speedup factor, $\text{MRSF}_{\mathcal{A}}^{(k)}$, is incremented by a small value (we increment by 0.01 — see line 1 and line 7 in Algorithm 1). Then, the original u_i^1 and u_i^2 of each task in $\tau_{\text{crit}}^{(k)}$ are divided by the new speedup factor (which is equivalent to increasing the speed of all the processors) and this resulting task set is fed back to algorithm $\mathcal{A}$. These steps (speedup factor adjustment and feeding back the derived task set) are repeated until the algorithm $\mathcal{A}$ succeeds in finding a feasible assignment, which gives us the *minimum required speedup factor* of $\mathcal{A}$ for the task set under consideration. This procedure is repeated for 100000 task sets (see line 2).

Recall that we want to evaluate the performance of our algorithms by measuring how well they perform compared to their theoretical bounds. In this regard,

Algorithm 1: Pseudo-code for determining the minimum required speedup factor, $\mathrm{MRSF}_{\mathcal{A}}^{(k)}$, of algorithm $\mathcal{A}$.

```
Input  : Algorithm $\mathcal{A}$
         The critically feasible task sets $\{\tau_{\mathrm{crit}}^{(1)}, \tau_{\mathrm{crit}}^{(2)}, \ldots, \tau_{\mathrm{crit}}^{(100000)}\}$
Output: The minimum required speedup factors
         $\{\mathrm{MRSF}_{\mathcal{A}}^{(1)}, \mathrm{MRSF}_{\mathcal{A}}^{(2)}, \ldots, \mathrm{MRSF}_{\mathcal{A}}^{(100000)}\}$
1  step $\leftarrow$ 0.01 ;
2  for $k = 1$ to 100000 do
3      $\tau \leftarrow \tau_{\mathrm{crit}}^{(k)}$;  $\mathrm{MRSF}_{\mathcal{A}}^{(k)} \leftarrow 1.0$ ;
4      while true do
5          result $\leftarrow$ call $\mathcal{A}(\tau, \mathrm{assignment})$ ;
           // assignment is an output variable which contains the task
              assignment information; A is either SA or SA-P
6          if result $\neq$ SUCCESS then
7              $\mathrm{MRSF}_{\mathcal{A}}^{(k)} \leftarrow \mathrm{MRSF}_{\mathcal{A}}^{(k)} + \mathrm{step}$ ;
8              $\tau \leftarrow \tau_{\mathrm{crit}}^{(k)} \times (1/\mathrm{MRSF}_{\mathcal{A}}^{(k)})$ ;
9          else break ;
10         ;
11     end
12 end
13 return $\{\mathrm{MRSF}_{\mathcal{A}}^{(1)}, \mathrm{MRSF}_{\mathcal{A}}^{(2)}, \ldots, \mathrm{MRSF}_{\mathcal{A}}^{(100000)}\}$ ;
```

for each critically feasible task set, $\tau_{\mathrm{crit}}^{(k)}$, we compute the *performance ratio*, $\mathrm{PR}_{\mathcal{A}}^{(k)}$, (in %) of algorithm $\mathcal{A}$ as follows:

$$\mathrm{PR}_{\mathcal{A}}^{(k)} \overset{\mathrm{def}}{=} \frac{\mathrm{MRSF}_{\mathcal{A}}^{(k)} - 1}{\mathrm{AR}_{\mathcal{A}}^{(k)} - 1} \times 100 \tag{41}$$

Note that both $\mathrm{MRSF}_{\mathcal{A}}^{(k)}$ and $\mathrm{AR}_{\mathcal{A}}^{(k)}$ are numbers that take a value of "$1.x$" where the integral part 1 is the speed of the processors on which an optimal algorithm succeeds to find a feasible intra-migrative task assignment and the fractional part x is the increase in the speed of processors that algorithm $\mathcal{A}$ requires (compared to the optimal algorithm) in order to succeed. Hence, 1 is subtracted from both $\mathrm{MRSF}_{\mathcal{A}}^{(k)}$ and $\mathrm{AR}_{\mathcal{A}}^{(k)}$ in the above expression. The multiplication factor 100 converts the ratio in percentage. This expression enables us to compare the performance of algorithms SA and SA-P for task sets with different values of α on the same scale. For example, for a given task set $\tau_{\mathrm{crit}}^{(k)}$ with $\alpha = 0.1$, if algorithm SA succeeds in finding a feasible intra-migrative task assignment with a minimum required speedup factor, $\mathrm{MRSF}_{SA}^{(k)} = 1.01$, then the value of the above ratio is 20% (since the approximation ratio of SA, $\mathrm{AR}_{SA}^{(k)}$, for this task set is $1 + \frac{\alpha}{2} = 1.05$) indicating that SA required only 20% faster processors than indicated by the theoretical estimate. Similarly, for a given task set in which $\alpha = 0.2$, if SA succeeds in finding a feasible intra-migrative task assignment with $\mathrm{MRSF}_{SA}^{(k)} = 1.02$ then the value of the above ratio is again 20% (since $\mathrm{AR}_{SA}^{(k)}$ of SA for this task set is $1 + \frac{\alpha}{2} = 1.10$) indicating that SA required only 20% faster processors than indicated by the theoretical estimate.

In general, for a given task set and a given algorithm, the smaller the *performance ratio*, the better the performance of the algorithm. For example, if this

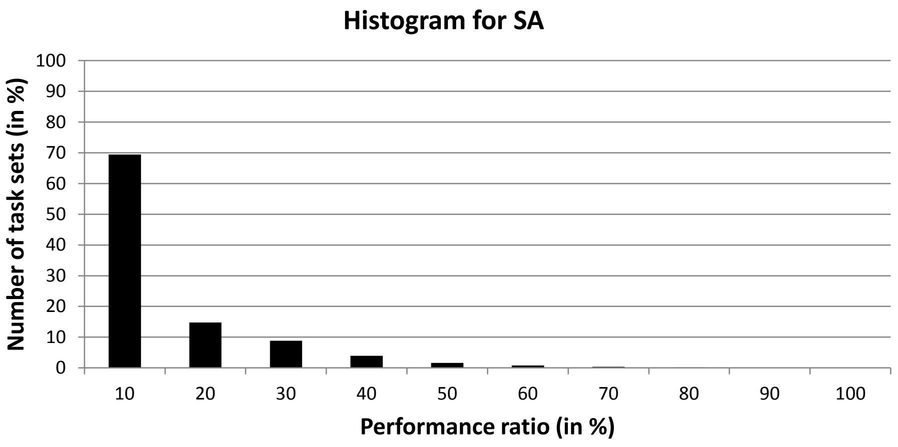

Fig. 3: Performance of algorithm, SA, in terms of *performance ratio* (see Expression 41) for several task sets (if an algorithm has low performance ratio for many task sets then the algorithm is said to perform well).

ratio takes a value of 100% then it implies that the algorithm is not performing any better than what is indicated by its theoretical bound and if this ratio takes a smaller value, say 10%, then it implies that the algorithm is performing much better (to be precise, 90% better) than its theoretical bound. Hence, an algorithm is said to perform better if this ratio is less for many task sets.

We plot the histogram of the performance ratios for both algorithms, SA and SA-P, in Figure 3 and Figure 4, respectively. As we can see from Figure 3, in our evaluations, for approximately 70% of the 100000 intra-migrative critically feasible task sets, SA succeeded in finding a feasible intra-migrative assignment within $(0-10]$% of its theoretical bound, for approximately 15% of the task sets, SA succeeded in finding a feasible intra-migrative assignment within $(10-20]$% of its theoretical bound, and so on. Similarly, as we can see from Figure 4, for approximately 70% of the task sets, SA-P succeeded in finding a feasible intra-migrative assignment within $(0-10]$% of its theoretical bound, for approximately 20% of the task sets, SA-P succeeded in finding a feasible intra-migrative assignment within $(10-20]$% of its theoretical bound, and so on.

To summarize, in our evaluations, for the vast majority of task sets, both algorithms performed significantly better than indicated by their theoretical bounds.

10 Special case: No task utilization can exceed one

In this section, we address the problem of scheduling a set of implicit-deadline sporadic tasks to meet all deadlines on a two-type platform for a *special case* but with even more *powerful adversary*. We consider the *special case* where the maximum utilization of any task on any processor in the given task set is no greater

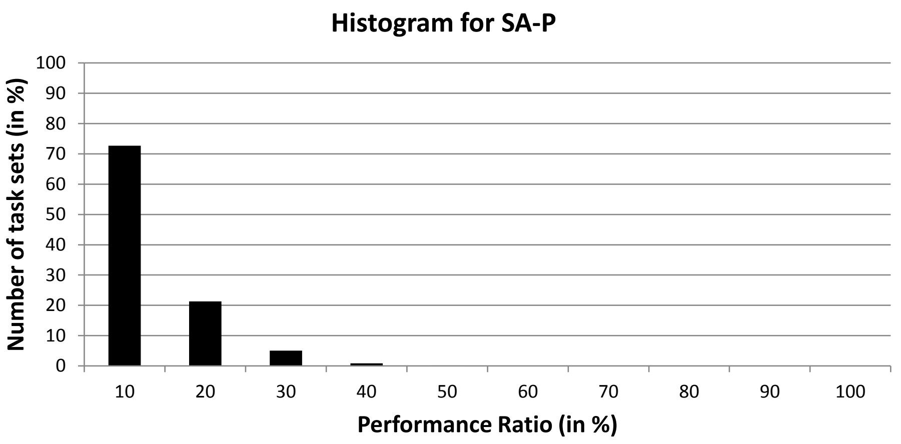

Fig. 4: Performance of algorithm, SA-P, in terms of *performance ratio* (see Expression 41) for several task sets (if an algorithm has low performance ratio for many task sets then the algorithm is said to perform well).

than one, i.e.,

$$\forall \tau_i \in \tau^S : \quad u_i^1 \leq 1 \ \land \ u_i^2 \leq 1 \tag{42}$$

We denote such a task set by τ^S — 'S' refers to "Special case". And we consider a more *powerful adversary* which is *fully-migrative*[5] as opposed to the *intra-migrative* adversary that was considered earlier. We (re-)prove the performance guarantees of both algorithms, SA and SA-P, for this special case and with this more powerful adversary. Specifically, we show that, if the task set τ^S is fully-migrative feasible on platform π then (i) using SA, it is guaranteed to obtain an intra-migrative feasible assignment of τ^S on $\pi^{(1+\frac{\beta}{2})}$ and (ii) SA-P is guaranteed to obtain a non-migrative feasible assignment of τ^S on $\pi^{(1+\beta)}$, where β is a real number such that:

$$\beta \overset{\text{def}}{=} \max_{\forall \tau_i \in \tau^S, t \in \{1,2\}} \{u_i^t\} \tag{43}$$

We show this by formulating the problem of determining the fully-migrative feasibility of a *generic task set*, τ (in which there is no restriction on the maximum utilization of a task), as a linear program and then showing that for the special case under consideration, i.e., for task set τ^S (in which no task utilization can exceed one), this linear program formulation can be transformed into the linear program that was discussed in Section 4 (see Figure 2 on page 12). Since the performance guarantees of both algorithms, SA and SA-P, were proven with the help of this LP formulation (of Figure 2), we conclude that the same performance guarantees continue to hold for this special case against the fully-migrative adversary. We now give the details.

[5] The term "fully-migrative" means that (i) different jobs of a task τ_i may execute on different processors and also (ii) jobs can migrate from any processor to any other processor during their execution.

Minimize Z subject to the following constraints:

C1.	$\sum_{j=1}^{m} x_{i,j} = 1$	$(i = 1, 2, \cdots, n)$
C2.	$\sum_{i=1}^{n} (x_{i,j} \times u_{i,j}) \leq Z$	$(j = 1, 2, \cdots, m)$
C3.	$\sum_{j=1}^{m} (x_{i,j} \times u_{i,j}) \leq 1$	$(i = 1, 2, \cdots, n)$
C4.	Z is a non-negative real number and	
	$x_{i,j}$ is a non-negative real number	$(i = 1, 2, \cdots, n)$;
		$(j = 1, 2, \cdots, m)$

Fig. 5: LP formulation — for determining if a task set τ is fully-migrative feasible on a *generic* heterogeneous multiprocessor platform.

Minimize Z subject to the following constraints:

NC1.	$\sum_{j=1}^{m} x_{i,j} = 1$	$(i = 1, 2, \cdots, n)$
NC2.	$\sum_{i=1}^{n} (x_{i,j} \times u_{i,j}) \leq Z$	$(j = 1, 2, \cdots, m)$
NC3.	Z is a non-negative real number and	
	$x_{i,j}$ is a non-negative real number	$(i = 1, 2, \cdots, n)$;
		$(j = 1, 2, \cdots, m)$

Fig. 6: LP formulation derived from Figure 5 — for determining if a task set τ^S is fully-migrative feasible on a *generic* heterogeneous multiprocessor platform.

For a generic task set, τ, the problem of determining whether a given task set is fully-migrative feasible can be formulated as a linear program as shown in Figure 5 (follows from Baruah (2004a)). One can show (Baruah, 2004a) that determining if a task set τ is fully-migrative feasible on the computer platform π is equivalent to (i) creating a schedule of arbitrarily small duration so that each task makes progress according to its utilization in this schedule and then (ii) repeat this schedule at run-time. In Figure 5, variable $u_{i,j}$ represents the utilization of task τ_i on processor π_j. The variable $x_{i,j}$ indicates the fraction of task τ_i that must be executed on processor π_j, i.e., $x_{i,j} = 1$ implies that τ_i must be entirely executed on processor π_j and $x_{i,j} = 0$ implies that τ_i must not be executed on processor π_j; in addition, $0 < x_{i,j} < 1$ indicates that *fraction* $x_{i,j}$ of τ_i must be executed on processor π_j. The first set of constraints ($C1$) indicates that every task must be entirely executed. The second set of constraints ($C2$) indicates that no processor capacity should be used more than Z. The third set of constraints ($C3$) indicates that the tasks are not allowed to execute in parallel and finally the fourth set of constraints ($C4$) indicates that the indicator variables must be non-negative real numbers. Finally, if $Z \leq 1$ then it implies that the task set is fully-migrative feasible; otherwise, the task set is fully-migrative infeasible.

Now, observe that, for the special case under consideration in which $\forall \tau_i \in \tau^S, \forall \pi_j \in \pi : u_{i,j} \leq 1$, for any solution returned by the LP solver, the third set of constraints (C3), are never violated, provided the first set of constraints (C1) are satisfied. Hence, removing these redundant constraints gives us the LP formulation shown in Figure 6.

Note that this LP formulation for determining the fully-migrative feasibility of a task set, upon solving, gives us the information about how much fraction of every task must be executed on each *processor*. Now, without loss of generality, let us convert it into an LP formulation (for determining the fully-migrative feasibility)

Minimize Z subject to the following constraints:

RC1.	$\forall \tau_i \in \tau^S$: $x_i^1 + x_i^2 = 1$
RC2.	$\sum_{\tau_i \in \tau^S} x_i^1 \times u_i^1 \leq Z \times m_1$
RC3.	$\sum_{\tau_i \in \tau^S} x_i^2 \times u_i^2 \leq Z \times m_2$
RC4.	Z is a non-negative real number and
	$\forall \tau_i \in \tau^S$: x_i^1, x_i^2 are non-negative real numbers

Fig. 7: LP formulation — for determining if a task set τ^S is fully-migrative feasible on a *two-type* heterogeneous multiprocessor platform.

Minimize Z subject to the following constraints:

RC1.	$\forall \tau_i \in \tau^S$: $x_i^1 + x_i^2 = 1$
RC2.	$U^1 + \sum_{\tau_i \in \tau^S} x_i^1 \times u_i^1 \leq Z \times m_1$
RC3.	$U^2 + \sum_{\tau_i \in \tau^S} x_i^2 \times u_i^2 \leq Z \times m_2$
RC5.	Z is a non-negative real number and
	$\forall \tau_i \in \tau^S$: x_i^1, x_i^2 are non-negative real numbers

Fig. 8: Relaxed LP formulation obtained from Figure 7 — for determining if a task set τ^S is fully-migrative feasible on a *two-type* heterogeneous multiprocessor platform.

which upon solving, gives the information about how much fraction of every task must be executed on each *processor type*:

- We know that every processor in π is either of type-1 or type-2. Hence, for $t \in \{1, 2\}$, it must hold that: for any task, $\tau_i \in \tau^S$:

$$\forall \pi_j, \pi_k \in \text{ type-t of } \pi : u_{i,j} = u_{i,k}$$

  Hence, let us represent the utilization of a task, $\tau_i \in \tau^S$, on any processor of type-t by u_i^t.
- Let us substitute $\sum_{\pi_j \in \text{ type-t of } \pi} x_{i,j}$ with x_i^t to obtain the information about how much fraction of every task must be executed on each processor type.
- Then, sum all the $C2$ constraints corresponding to type-1 (resp., type-2) processors — this will reduce the $C2$ set of constraints (m in total) to only two constraints.

Performing these operations gives us the LP formulation shown in Figure 7. The set of constrains, RC1 and RC4, in Figure 7 are derived from the corresponding set of constraints, NC1 and NC3, of Figure 6, respectively. And the constraints, RC2 and RC3, are derived from the second set of constraints, NC2.

Adding two dummy constants, $U^1 = 0$ and $U^2 = 0$ to the left hand side terms in constraints, RC2 and RC3, respectively, gives us the LP formulation shown in Figure 8.

For task sets in which no task utilization can exceed one, it holds that, $\beta = \alpha$ (see Expression (1) on page 8 and Expression (43) on page 36). Also, upon partitioning τ^S into H12, H1, H2 and L using Expressions (6)–(8), we obtain: H12 $= \emptyset$, H1 $= \emptyset$ (which implies that $U^1 = 0$ — see Expression (9) on page 10), H2 $= \emptyset$ (which implies that $U^2 = 0$ — see Expression (10) on page 10) and L $= \tau^S$. Using this information, we can observe that the linear program formulation shown in Figure 8 is identical to the one shown in Figure 2 of Section 4. Also, recall that the algorithm, SA, is designed such that, it obtains a solution which is similar to the vertex solution for the optimization problem of Figure 2. The approximation ratios of both SA and SA-P are derived using this property. From the equivalence of these two optimization problems, it is easy to see that the results of our algorithms, SA and SA-P, continue to hold for this special case against fully-migrative adversary.

Theorem 8 *If task set τ^S is fully-migrative feasible on platform π then, using SA, it is guaranteed to obtain a feasible intra-migrative task assignment of τ^S on $\pi^{(1+\frac{\beta}{2})}$.*

Proof It follows from the proof of Theorem 4. □

Remark 5 Although Theorem 8 states that, for a fully-migrative feasible task set, SA needs a platform in which *every processor* is $1 + \frac{\alpha}{2}$ times faster, in order to output a schedulable intra-type task assignment, it is trivial to see that a platform in which only *one processor* is $1+\frac{\alpha}{2}$ times faster is sufficient (to which the fractional task can be integrally assigned).

Corollary 6 *If task set τ^S is fully-migrative feasible on platform $\pi(m_1, m_2)$ then, using SA, it is guaranteed to obtain a feasible intra-migrative assignment of τ^S on $\pi'(m_1 + 1, m_2)$.*

Proof It follows from the proof of Corollary 4. □

Remark 6 It is trivial to see that Corollary 6 holds true if SA is given a platform $\pi'(m_1, m_2 + 1)$.

Theorem 9 *If task set τ^S is fully-migrative feasible on platform π then, SA-P is guaranteed to obtain a feasible non-migrative assignment of τ^S on $\pi^{(1+\beta)}$.*

Proof It follows from the proof of Theorem 6. □

Corollary 7 *If task set τ^S is fully-migrative feasible on platform $\pi(m_1, m_2)$ then SA-P is guaranteed to obtain a feasible non-migrative assignment of τ^S on platform $\pi'(2m_1, 2m_2)$.*

Proof It follows from the proof of Corollary 5. □

11 Average-case performance evaluation for the special case

For this special case in which no task utilization in the given task set can exceed one, we evaluate the average-case performance of our algorithm, SA-P, and compare it with prior state-of-the-art algorithm, $\mathrm{LP_{EE}}$ (Baruah, 2004c; Raravi et al, 2011). It was shown (in Raravi et al, 2011) that, for task sets in which no task utilization can exceed one, the non-migrative algorithm, $\mathrm{LP_{EE}}$ (originally proposed by Baruah (2004c)), has an approximation ratio of 2 against a fully-migrative adversary. For this purpose, we look at the following aspects: (i) how much faster processors SA-P needs for determining a feasible non-migrative task assignment (which is upper bounded by its approximation ratio) compared to LP_{EE} and (ii) how fast SA-P runs compared to LP_{EE}. We now discuss the experiments in detail[6].

The algorithm, LP_{EE}, is a two-step algorithm, for obtaining a non-migrative task assignment on a heterogeneous multiprocessor platform and works as follows:

1. The non-migrative task assignment problem is formulated as MILP and then relaxed to LP (more details in Raravi et al (2011)). The LP formulation is solved using an LP solver (e.g., GUROBI Optimizer (Gurobi Optimization Inc., 2012), IBM ILOG CPLEX (IBM, 2012)). Tasks are then assigned to processors according to the values of the respective indicator variables in the solution. 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 is the number of processors.
2. The remaining at most $m-1$ tasks are integrally assigned on the remaining capacity of the processors using "exhaustive enumeration".

LP_{EE} is named so because it solves "**L**inear **P**rogram" and uses "**E**xhaustive **E**numeration" technique.

We implemented the algorithms in C on an Intel Core2 (2.80 GHz) machine. For LP_{EE}, we used a state-of-the-art LP solver, IBM ILOG CPLEX.

For LP_{EE}, we implemented two versions — the original version, referred to as LP_{EE}, and its efficient version, referred to as $\text{LP}_{\text{EE-EFF}}$, which gives a better average-case performance (as shown in Raravi et al (2013) for generic task sets in which there is no restriction on the maximum task utilization). In LP_{EE}, while integrally assigning the at most $m-1$ fractional tasks (in the second step), the utilization of the task under consideration is compared against the remaining capacity of any processor (after solving LP formulation), which is given by $1-Z$, for assignment decisions *on any processor*, where the value of the variable Z (returned by the LP solver) is the maximum utilized fraction of any processor — LP_{EE} implements this (pessimistic) rule. Since the actual remaining capacity of each processor[7] can easily be computed from the LP solver solution, the improved version of LP_{EE}, namely $\text{LP}_{\text{EE-EFF}}$, uses that value instead of $1-Z$, for checking whether a fractionally assigned task can be integrally assigned on a processor without violating the uniprocessor feasibility condition, for a better average-case performance.

For a given task set, we define the *minimum required speedup factor*, $\text{MRSF}_{\text{SA-P}}$, of SA-P as the *minimum* amount of extra speed of processors that SA-P needs, so as to succeed in finding a feasible non-migrative task assignment as compared to an optimal fully-migrative algorithm. We define $\text{MRSF}_{\text{LP}_{\text{EE}}}$ of LP_{EE} and $\text{MRSF}_{\text{LP}_{\text{EE-EFF}}}$ of $\text{LP}_{\text{EE-EFF}}$, analogously. Observe that, the approximation

[6] For this special case, the algorithm, SA, exhibited a similar average-case performance as discussed in Section 9. Hence, we do not discuss it here.

[7] The actual remaining capacity on processor p is given by $1-\sum_{i:x_{i,p}=1} u_{i,p}$, where $u_{i,p}$ represents the utilization of τ_i on processor p (Baruah, 2004c). The symbol $x_{i,p}$ represents the indicator variable and the value of $0 \le x_{i,p} \le 1$ indicates how much fraction of task τ_i must be assigned to processor p. The term $1-\sum_{i:x_{i,p}=1} u_{i,p}$ gives an accurate estimation of the remaining capacity on processor p as it ignores the fractionally assigned tasks on that processor whereas Z is pessimistic since it includes those tasks as well.

ratio of $\mathrm{LP}_{\mathrm{EE}}$ (and of $\mathrm{LP}_{\mathrm{EE\text{-}EFF}}$) is 2 which is a constant, whereas the approximation ratio of SA-P is $1 + \beta \leq 2$ which is a function of the utilizations of the given task set. For ease of comparison, we consider that the approximation ratio of SA-P is also a constant and is given by its upper bound of 2. With this, we assess the average-case performance of these algorithms by measuring their (i) minimum required speedup factors and (ii) running times, for a large number of task sets.

The problem instances were generated randomly (using uniform distribution as described earlier in Section 9). Each problem instance had at most 25 tasks and at most 3 processors of each type. We generated 25000 task sets, denoted as $\{\tau^{S(1)}, \tau^{S(2)}, \ldots, \tau^{S(25000)}\}$, which we transformed into "fully-migrative critically feasible task sets". We define a fully-migrative critically feasible task set as a task set which is fully-migrative feasible on a given two-type platform but rendered (fully-migrative) infeasible, if all the task utilizations (i.e., both u_i^1 and u_i^2 of each task) are increased by an arbitrarily small factor (without exceeding one).

To obtain a fully-migrative critically feasible task set, $\tau^{S(k)}_{\mathrm{crit}}$, from a randomly generated task set, $\tau^{S(k)}$, where $k \in \{1, 2, \ldots, 25000\}$, we formulate the problem (of obtaining a fully-migrative feasible task set) as a Linear Program shown in Figure 6 (on page 37) for task set $\tau^{S(k)}$ and feed it to the CPLEX solver which outputs Z. Then, if $Z > 1$, we multiply all the task utilizations with $1/Z$ else, we increase the utilizations of "some tasks" by a small factor (of 0.01). The tasks whose utilizations must be increased are chosen such that, for a given task, upon increasing its u_i^1 (resp., u_i^2), the new utilization value must not exceed 1.0 (since no task utilization in the given task set can exceed one). We then feed this derived task set to the solver. These steps (modifying the utilizations and feeding it back to the solver) are repeated until $0.99 < Z \leq 1$, which gives us the fully migrative critically feasible task set, $\tau^{S(k)}_{\mathrm{crit}}$. Note that, the procedure discussed here to obtain a *fully-migrative* critically feasible task set is different from the one described in Section 9 (to obtain an *intra-migrative* critically feasible task set) because of the additional restriction that no task utilization in the given task set can exceed one.

We ran SA-P, $\mathrm{LP}_{\mathrm{EE}}$ and $\mathrm{LP}_{\mathrm{EE\text{-}EFF}}$ on 25000 fully-migrative critically feasible task sets and for each task set, we obtained $\mathrm{MRSF}_{\mathrm{SA\text{-}P}}$, $\mathrm{MRSF}_{\mathrm{LP}_{\mathrm{EE}}}$ and $\mathrm{MRSF}_{\mathrm{LP}_{\mathrm{EE\text{-}EFF}}}$ as follows. We initially set the speedup factor to 1.0 and input the task set to the algorithm. If the algorithm cannot find a feasible non-migrative task assignment, we increment the speedup factor by a small value, i.e., by 0.01, and divide the original utilizations, u_i^1 and u_i^2, of each task by the new speedup factor (which is equivalent to increasing the speed of every processor by a factor of 0.01) and feed the resulting task set to the algorithm. These steps (adjust the speedup factor and feed back the derived task set) are repeated till the algorithm succeeds, which gives us the MRSF of the algorithm for the given task set. This entire procedure is repeated for 25000 critically feasible task sets.

With this procedure, we obtain the histograms of MRSFs for these algorithms which is shown in Figure 9. As can be seen, the $\mathrm{MRSF}_{\mathrm{LP}_{\mathrm{EE\text{-}EFF}}}$ never exceeded 1.60, whereas $\mathrm{MRSF}_{\mathrm{LP}_{\mathrm{EE}}}$ is as high as 2.0. Hence, $\mathrm{LP}_{\mathrm{EE\text{-}EFF}}$ gives a better average-case performance than $\mathrm{LP}_{\mathrm{EE}}$ (in accordance with the observations made for these algorithms for generic task sets in Raravi et al (2013)). So, comparing SA-P with $\mathrm{LP}_{\mathrm{EE\text{-}EFF}}$, we can see that, for a large number of fully-migrative critically feasible task sets, $\mathrm{MRSF}_{\mathrm{SA\text{-}P}}$ is much better (i.e., smaller) than $\mathrm{MRSF}_{\mathrm{LP}_{\mathrm{EE\text{-}EFF}}}$. Therefore,

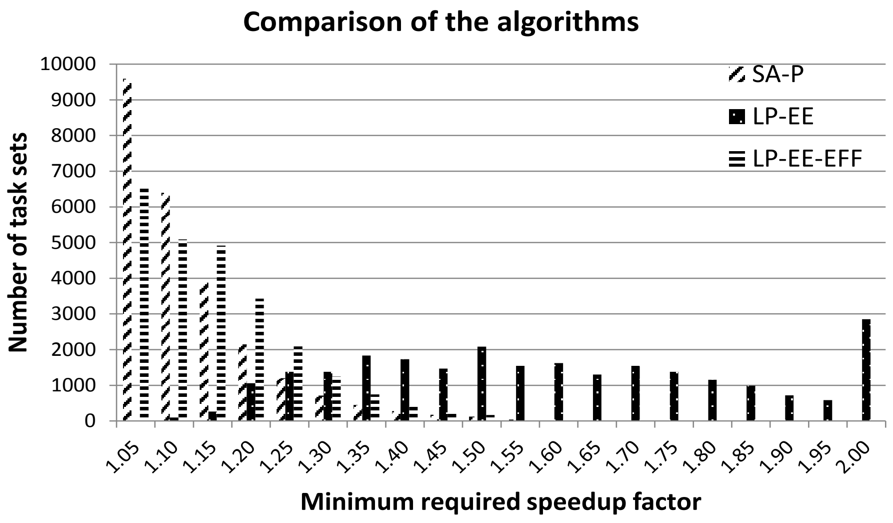

Fig. 9: Comparison of *minimum required speedup factor* of algorithms SA-P, LP_{EE} and $LP_{EE\text{-}EFF}$ (if an algorithm has low minimum required speedup factor for many task sets then the algorithm is said to perform well).

	Measured average run-time (μs)		
Value of MRSF	SA-P	LP_{EE}	$LP_{EE\text{-}EFF}$
1.00	3.82	17307.68	17537.37
1.25	3.78	17440.61	17502.96
1.50	3.73	18022.68	17687.59
1.75	3.70	17543.52	17665.57
2.00	3.65	17387.41	17673.16

Table 8: Comparison of average running times of algorithms SA-P, LP_{EE} and $LP_{EE\text{-}EFF}$ (in μs).

on an average, SA-P requires significantly smaller processor speedup compared to LP_{EE} for finding a feasible non-migrative task assignment.

We also measure the average running times of the algorithms. Table 8 shows the running times of these algorithms for different values of minimum required speedup factor. Note that, we are measuring the time it takes for an algorithm to complete a *single run* wherein either it outputs a feasible non-migrative task assignment or indicates that, with the given speedup factor, it cannot find a feasible non-migrative task assignment for the given task set. This ensures that the experiments to measure the running times are not biased to give advantage to any of the algorithms (especially to SA-P, which on an average, requires smaller speedup factor, as discussed earlier). In our evaluations with 25000 task sets, as can be seen in Table 8, both LP_{EE} and $LP_{EE\text{-}EFF}$ have approximately same running times. This is expected as LP_{EE} and $LP_{EE\text{-}EFF}$ only differ in the feasibility test that they use while trying to assign a task. However, as can be seen from the table, SA-P runs at least 4500 times faster than these algorithms. The high running times of LP_{EE} and $\text{LP}_{\text{EE-EFF}}$ can be attributed to the fact that both rely on solving linear programs.

To summarize, for the special case under consideration in which no task utilization in the given task set can exceed one, SA-P, exhibits a better average-case performance by requiring significantly smaller processor speedup for finding a feasible non-migrative task assignment and by running orders of magnitude faster compared to LP_{EE}. Overall, our algorithm outperforms prior state-of-the-art.

12 Conclusions

We proposed two low degree polynomial time-complexity algorithms, namely SA and SA-P, for assigning implicit-deadline sporadic tasks on two-type heterogeneous multiprocessors. We also showed that they provide the following guarantee. If there exists a feasible intra-migrative assignment of a task set on a two-type platform then (i) using SA, it is guaranteed to find such a feasible intra-migrative task assignment, if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster and (ii) SA-P is guaranteed to find a feasible non-migrative task assignment, if given a platform in which processors are $1+\alpha$ times faster. In the average-case performance evaluations with randomly generated task sets, for the vast majority of these task sets, our algorithms required significantly smaller processor speedup than indicated by their theoretical bounds. We also extended the analysis of our algorithms to a case, in which, no task utilization in the given task set can exceed one and showed that, for this case, changing the adversary to a more powerful one, namely fully-migrative, does not deteriorate the performance guarantees of both the algorithms. For this special case, in the average-case performance evaluations, our algorithm, SA-P, outperformed prior state-of-the-art algorithm.

Acknowledgments

This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects ref. FCOMP-01-0124-FEDER-022701 (CISTER) and ref. FCOMP-01-0124-FEDER-020447 (REGAIN); by National Funds through FCT and by the EU ARTEMIS JU funding, within grant nr. 333053 (CONCERTO); by ERDF, through ON2 – North Portugal Regional Operational Programme, under the National Strategic Reference Framework (NSRF), within project ref. NORTE-07-0124-FEDER-000063 (BEST-CASE).

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.

Appendix A: The hardness of the task assignment problems

In this section, we show that both intra-migrative and non-migrative task assignment problems on a two-type heterogeneous multiprocessors are intractable.

A.1 Intra-migrative task assignment problem is NP-Complete

In this section, we show that the problem of intra-migrative task assignment on two-type heterogeneous multiprocessor platform is NP-Complete. We denote this problem as HET2-INTRA-ASSIGN and is stated in Figure 10. In order to show this, we will first consider a

HET2-INTRA-ASSIGN PROBLEM	
Instance	A task set τ of n implicit-deadline sporadic tasks and a two-type platform π of m processors of which m_1 processors are of type-1 and m_2 processors are of type-2. The utilization of a task τ_i on a processor of type-t is given by u_i^t where $i \in \{1, 2, \ldots, n\}$ and $t \in \{1, 2\}$.
Problem	Find an assignment $f : \{1, 2, \ldots, n\} \rightarrow \{1, 2\}$ such that, $\forall t \in \{1, 2\}$, it holds that: $\left(\sum_{i:f(i)=t} u_i^t \leq m_t\right) \wedge \left(\forall i \in \{1, 2, \ldots, n\} \text{ such that } f(i) = t : u_i^t \leq 1\right)$.

Fig. 10: The intra-migrative task assignment problem on a two-type heterogeneous multiprocessor platform

restricted version of this problem which is denoted as HET2-INTRA-ASSIGN-SPEC-CASE — see Figure 11. We will show that this problem is NP-complete. It then follows that the HET2-INTRA-ASSIGN problem is NP-complete as well.

HET2-INTRA-ASSIGN-SPEC-CASE PROBLEM	
Instance	A task set τ of n implicit-deadline sporadic tasks and a two-type platform π of m processors of which m_1 processors are of type-1 and m_2 processors are of type-2. The utilization of a task τ_i on a processor of type-t is given by u_i^t where $i \in \{1, 2, \ldots, n\}$ and $t \in \{1, 2\}$. Assume that: $\forall \tau_i \in \tau : u_i^1 = u_i^2$ and $m_1 = 1$ and $m_2 = 1$
Problem	Find an assignment $f : \{1, 2, \ldots, n\} \rightarrow \{1, 2\}$ such that, $\forall t \in \{1, 2\}$, it holds that: $\left(\sum_{i:f(i)=t} u_i^t \leq m_t\right) \wedge \left(\forall i \in \{1, 2, \ldots, n\} \text{ such that } f(i) = t : u_i^t \leq 1\right)$.

Fig. 11: A restricted version of the intra-migrative task assignment problem on a two-type heterogeneous multiprocessor platform

For showing that the HET2-INTRA-ASSIGN-SPEC-CASE problem is NP-Complete, we make use of the PARTITION problem. The PARTITION problem is shown in Figure 12 and it is well-known that this problem is NP-Complete (Corollary 15.28 in Korte and Vygen (2006), p. 365).

PARTITION PROBLEM	
Instance	A list of n natural numbers $c_1, c_2, \ldots, c_n$.
Question	Is there a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$.

Fig. 12: The partitioning problem, which is known to be NP-Complete (Korte and Vygen, 2006)

Lemma 6 *The HET2-INTRA-ASSIGN-SPEC-CASE problem is NP-Complete.*

Proof In order to show that a problem is NP-Complete, we need to: (1) show that the problem is in NP, (2) transform an NP-Complete problem to the problem under consideration and (3) show that the transformation (of Step (2)) can be done in polynomial time. We now show these for HET2-INTRA-ASSIGN-SPEC-CASE problem.

1. It is straightforward to see that the problem belongs to NP. To show that the problem is in NP, we should be able to verify, in polynomial time, the given *certificate* for a *yes*-instance of the problem. As a *certificate*, we take the assignment on each processor type. To check whether the given assignment in fact satisfies, for all $t \in \{1, 2\}$: $\left(\sum_{i:f(i)=t} u_i^t \leq m_t\right) \wedge \left(\forall i \in \{1, 2, \ldots, n\} \text{ such that } f(i) = t : u_i^t \leq 1\right)$, is obviously possible in polynomial time; specifically, the time complexity of this step is $O(n)$.
2. We now transform the PARTITION problem (which is NP-Complete) to the above decision problem. Given an instance $c_1, c_2, \ldots, c_n \in \mathbb{N}$ of the PARTITION problem, transform it into an instance of HET2-INTRA-ASSIGN-SPEC-CASE problem with n tasks and compute utilizations of tasks as follows:

$$\forall \tau_i \in \tau, \forall t \in \{1, 2\} : \quad u_i^t = \frac{2c_i}{\sum_{k=1}^{n} c_k} \in (0, 1] \tag{44}$$

   We now show that (intra-migrative) assignment of these n tasks on two processor types is possible *if and only if* there is a set $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$. We do so by first showing, in (a), some results we will use and then showing, in (b), the implication in one direction and finally showing, in (c), the implication in the other direction.

   (a) *Results we will use:*

   (a.1) It is trivial to see that $(a = b) \Rightarrow \left(a = b = \frac{a+b}{2}\right)$. This gives us:

$$\left(\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j\right) \Rightarrow$$
$$\left(\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j = \frac{\sum_{j \in \{1,2,\ldots,n\}} c_j}{2}\right)$$

   (a.2) It is also trivial to see that $\left(\left(a \leq \frac{a+b}{2}\right) \wedge \left(b \leq \frac{a+b}{2}\right)\right) \Rightarrow (a = b)$. This gives us:

$$\left(\left(\sum_{j \in S} c_j \leq \frac{\sum_{k=1}^{n} c_k}{2}\right) \wedge \left(\sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j \leq \frac{\sum_{k=1}^{n} c_k}{2}\right)\right) \Rightarrow$$
$$\left(\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j\right)$$

   (a.3) Let us introduce g that maps an element in $\{1, 2, \ldots, n\}$ to a processor type. It is defined as follows:

$$i \in S \Leftrightarrow g(i) = 1$$
$$i \in (\{1, 2, \ldots, n\} \setminus S) \Leftrightarrow g(i) = 2$$

   (b) *Implication in one direction:* We now show (using g) that *if* there is a set $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$ *then* intra-migrative assignment of these n tasks on two processor types is possible.

We will do so by assuming that the if-condition of (b) is true and then show that this implies that the then-condition of (b) must also be true. We know that $\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$. Using (a.1) on this gives us:

$$\sum_{j \in S} c_j = \frac{\sum_{j \in \{1,2,\ldots,n\}} c_j}{2}$$
$$\sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j = \frac{\sum_{j \in \{1,2,\ldots,n\}} c_j}{2}$$

Multiplying each side by $\frac{2}{\sum_{k=1}^{n} c_k}$ and applying the definition of u_i^t on the left hand side and using the definition of g gives us:

$$\sum_{j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1} u_j^1 = 1$$
$$\sum_{j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1} u_j^2 = 1$$
$$\sum_{j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2} u_j^1 = 1$$
$$\sum_{j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2} u_j^2 = 1$$

It obviously holds that, for a set of non-negative numbers, each element cannot be greater than the sum of all numbers in the set. Using this observation on the above gives us:

$$\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1: \; u_j^1 \le 1$$
$$\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1: \; u_j^2 \le 1$$
$$\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2: \; u_j^1 \le 1$$
$$\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2: \; u_j^2 \le 1$$

Hence, we have shown that g is an assignment of tasks to processor types that satisfies the constraints stated in HET2-INTRA-ASSIGN-SPEC-CASE problem.

(c) *Implication in the other direction:* We now show (using g) that *if* intra-migrative assignment of these n tasks on two processor types is possible *then* there is a set $S \subseteq \{1,2,\ldots,n\}$ such that $\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$.
We will do so by assuming that the if-condition of (c) is true and then show that this implies that the then-condition of (c) must also be true. We know that an intra-migrative assignment of these n tasks is possible. Using the function g to express this gives us:

$$\left(\sum_{\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1} u_j^1 \le 1 \right) \wedge$$
$$\left(\sum_{\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2} u_j^2 \le 1 \right) \wedge$$
$$\big(\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=1: \; u_j^1 \le 1 \big) \wedge$$
$$\big(\forall j \in \{1,2,\ldots,n\} \text{ such that } g(j)=2: \; u_j^2 \le 1 \big)$$

HET2-NON-ASSIGN PROBLEM	
Instance	A task set τ of n implicit-deadline sporadic tasks and a two-type platform π of m processors of which m_1 processors are of type-1 and m_2 processors are of type-2. The utilization of a task τ_i on a processor of type-t is given by u_i^t where $i \in \{1, 2, \ldots, n\}$ and $t \in \{1, 2\}$.
Problem	Find an assignment $f : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, m\}$ such that $\forall j \in$ type-t of π, it holds that: $\sum_{i:f(i)=j} u_i^t \leq 1$, where $t \in \{1, 2\}$.

Fig. 13: The non-migrative task assignment problem on a two-type heterogeneous multiprocessor platform

Using the definition of u_i^t and the mapping g and multiplying each side by $\frac{\sum_{k=1}^{n} c_k}{2}$ gives us:

$$\left(\sum_{\forall j \in S} c_j \leq \frac{\sum_{k=1}^{n} c_k}{2} \right) \wedge$$
$$\left(\sum_{\forall j \in (\{1,2,\ldots,n\} \setminus S)} c_j \leq \frac{\sum_{k=1}^{n} c_k}{2} \right) \wedge$$
$$\left(\forall j \in S : c_j \leq \frac{\sum_{k=1}^{n} c_k}{2} \right) \wedge$$
$$\left(\forall j \in (\{1,2,\ldots,n\} \setminus S) : c_j \leq \frac{\sum_{k=1}^{n} c_k}{2} \right)$$

Observing the first two expressions and using (a.2) gives us:

$$\sum_{j \in S} c_j = \sum_{j \in (\{1,2,\ldots,n\} \setminus S)} c_j$$

This satisfies the constraints of the PARTITION problem.

3. Finally, it can be easily seen that the transformation from PARTITION to HET2-INTRA-ASSIGN-SPEC-CASE using Expression (44) is possible in polynomial time; specifically, the time complexity is $O(n)$.

Hence the proof. □

Theorem 10 *The HET2-INTRA-ASSIGN problem is NP-Complete.*

Proof Follows from Lemma 6 and the fact that HET2-INTRA-ASSIGN-SPEC-CASE problem is a restricted form of HET2-INTRA-ASSIGN problem. □

A.2 Non-migrative task assignment problem is NP-Complete in the strong sense

In this section, we show that the problem of non-migrative task assignment on a two-type heterogeneous multiprocessor platform is NP-Complete in the strong sense. We denote this problem as HET2-NON-ASSIGN and is stated in Figure 13. In order to show this, we will first consider a *restricted version* of this problem which is denoted as HET2-NON-ASSIGN-SPEC-CASE — see Figure 14. We will show that this problem is NP-complete in the strong sense. It then follows that the HET2-NON-ASSIGN problem is NP-complete in the strong sense as well.

For showing that the HET2-NON-ASSIGN-SPEC-CASE problem is NP-Complete in the strong sense, we make use of the 3-PARTITION problem. The 3-PARTITION problem is shown in Figure 15 and it is well-known that this problem is NP-Complete in the strong sense (see, e.g., Garey and Johnson, 1978).

HET2-NON-ASSIGN-SPEC-CASE PROBLEM	
Instance	A task set τ of n implicit-deadline sporadic tasks and a two-type platform π of m processors of which m_1 processors are of type-1 and m_2 processors are of type-2. The utilization of a task τ_i on a processor of type-t is given by u_i^t where $i \in \{1, 2, \ldots, n\}$ and $t \in \{1, 2\}$. Assume that: $\forall \tau_i \in \tau : u_i^1 = u_i^2$ and $\forall \tau_i \in \tau, \forall t \in \{1, 2\} : \frac{1}{4} < u_i^t < \frac{1}{2}$ and $\frac{1}{m} \times \left(\sum_{i \in \{1,2,\ldots,n\}} u_i^1\right) = \frac{1}{m} \times \left(\sum_{i \in \{1,2,\ldots,n\}} u_i^2\right) = 1$
Problem	Find an assignment $f : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, m\}$ such that $\forall j \in$ type-t of π, it holds that $\sum_{i:f(i)=j} u_i^t \leq 1$, where $t \in \{1, 2\}$.

Fig. 14: A restricted version of the non-migrative task assignment problem on a two-type heterogeneous multiprocessor platform

3-PARTITION PROBLEM	
Instance	A list of $3m$ integers $I = \{c_1, c_2, \ldots, c_{3m}\}$ where $\forall i : c_i \geq 2$ and a bound B such that $\sum_{i=1}^{3m} c_i = mB$ and $\forall i : B/4 < c_i < B/2$.
Question	Can I be partitioned into m subsets $I_1, I_2, \ldots, I_m$ such that $\forall j : \sum_{i \in I_j} c_i = B$.

Fig. 15: The 3-partitioning problem, which is known to be NP-Complete in the strong sense (Garey and Johnson, 1978)

Lemma 7 *The HET2-NON-ASSIGN-SPEC-CASE problem is NP-Complete in the strong sense.*

Proof In order to show that a problem is NP-Complete in the strong sense, we need to: (1) show that the problem is in NP, (2) transform a problem which is NP-Complete in the strong sense to the problem under consideration and (3) show that the transformation (of Step (2)) can be done in polynomial time. We now show these for HET2-NON-ASSIGN-SPEC-CASE problem.

1. It is straightforward to see that the problem belongs to NP. As a *certificate*, we take the assignment on each processor. To check whether the given assignment in fact satisfies $\sum_{i:f(i)=j} u_i^t \leq 1$ for every processor $j \in$ type-t of π (where $t \in \{1, 2\}$) is obviously possible in polynomial time; specifically the time complexity is $O(n)$.
2. We now transform the 3-PARTITION problem (which is NP-Complete in the strong sense) to the above decision problem. Given an instance $c_1, c_2, \ldots, c_{n=3m}$ and B of the 3-PARTITION problem, transform it into an instance of HET2-NON-ASSIGN-SPEC-CASE problem with $n = 3m$ tasks by computing utilizations of tasks as follows:

$$\forall \tau_i \in \tau, \forall t \in \{1, 2\} : \quad u_i^t = \frac{c_i}{B} \tag{45}$$

We now show that (non-migrative) assignment of these $3m$ tasks on m processors is possible *if and only if* $c_1, c_2, \ldots, c_{n=3m}$ can be partitioned into m subsets $I_1, I_2, \ldots, I_m$ such that $\forall j \in \{1, 2, \ldots, m\} : \sum_{i \in I_j} c_i = B$. We do so by first showing, in (a), some results we will use and then showing, in (b), the implication in one direction and finally showing, in (c), the implication in the other direction.

(a) *Results we will use:*

(a.1) Let us introduce g that maps an element in $\{1, 2, \ldots, 3m\}$ to a processor. It is defined as follows:

$$i \in I_j \Leftrightarrow g(i) = j$$

(b) *Implication in one direction:* We now show (using g) that *if* $c_1, c_2, \ldots, c_{3m}$ can be partitioned into m subsets $I_1, I_2, \ldots, I_m$ such that $\forall j \in \{1, 2, \ldots, m\} : \sum_{i \in I_j} c_i = B$ *then* there is an assignment of these $3m$ tasks on m processors.

We will do so by assuming that the if-condition of (b) is true and then show that this implies that the then-condition of (b) must also be true. We know that $c_1, c_2, \ldots, c_{3m}$ can be partitioned into m subsets $I_1, I_2, \ldots, I_m$ such that $\forall j \in \{1, 2, \ldots, m\} : \sum_{i \in I_j} c_i =$

B. Multiplying each side by $\frac{1}{B}$ and applying the definition of u_i^t on the left hand side and using the definition of g gives us:

$$\forall j \in \{1,2,\ldots,m\} : \sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^1 = 1$$
$$\forall j \in \{1,2,\ldots,m\} : \sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^2 = 1$$

Hence, we have shown that g is an assignment of tasks to processors that satisfies the constraints stated in HET2-NON-ASSIGN-SPEC-CASE problem.

(c) *Implication in the other direction:* We now show (using g) that *if* non-migrative assignment of these n tasks on m processors is possible *then* $c_1, c_2, \ldots, c_{3m}$ can be partitioned into m subsets $I_1, I_2, \ldots, I_m$ such that $\forall j \in \{1,2,\ldots,m\} : \sum_{i \in I_j} c_i = B$. We will do so by assuming that the if-condition of (c) is true and then show that this implies that the then-condition of (c) must also be true. We know that a non-migrative assignment of these n tasks is possible. Using the function g to express this gives us:

$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^1 \leq 1 \right) \wedge$$
$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^2 \leq 1 \right)$$

Since it is a non-migrative assignment, it also holds that (from one of the assumptions of HET2-NON-ASSIGN-SPEC-CASE problem):

$$\frac{1}{m} \times \left(\sum_{i \in \{1,2,\ldots,n\}} u_i^1 \right) = \frac{1}{m} \times \left(\sum_{i \in \{1,2,\ldots,n\}} u_i^2 \right) = 1$$

Applying this on the earlier expression gives:

$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^1 = 1 \right) \wedge$$
$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} u_i^2 = 1 \right)$$

Multiply both sides by B and using the definition of u_i^t gives us:

$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} c_i = B \right) \wedge$$
$$\forall j \in \{1,2,\ldots,m\} : \left(\sum_{\forall i \in \{1,2,\ldots,n\} \text{ such that } g(i)=j} c_i = B \right)$$

Note that these two expressions state the same thing so only one is needed. Also, we form the partitioning as follows. Let I_j be the set of all integers such that $i \in \{1,2,\ldots,n\}$ and $g(i) = j$. This gives us:

$$\forall j \in \{1,2,\ldots,m\} : \sum_{\forall i \in I_j} c_i = B$$

This satisfies the constraints of the 3-PARTITION problem.

3. Finally, it can be easily seen that the transformation from 3-PARTITION to HET2-NON-ASSIGN-SPEC-CASE using Expression (45) is possible in polynomial time; specifically, the time complexity is $O(n)$.

Hence the proof. □

Theorem 11 *The HET2-NON-ASSIGN problem is NP-Complete in the strong sense.*

Proof Follows from Lemma 7 and the fact that HET2-NON-ASSIGN-SPEC-CASE problem is a restricted form of HET2-NON-ASSIGN problem. □

References

AMD Inc (2013) AMD Accelerated Processing Units. http://fusion.amd.com

Anderson J, Srinivasan A (2000) Early-release fair scheduling. In: Proceedings of the 12^{th} Euromicro conference on Real-time systems, pp 35–43

Andersson B, Bletsas K (2008) Sporadic Multiprocessor Scheduling with Few Preemptions. In: 20th Euromicro Conference on Real-Time Systems, pp 243–252

Andersson B, Tovar E (2007) Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors. In: Proceedings of the 15^{th} International Workshop on Parallel and Distributed Real-Time Systems, pp 1–8

Andersson B, Baruah S, Jonsson J (2001) Static-Priority Scheduling on Multiprocessors. In: Proceedings of the 22^{nd} IEEE Real-Time Systems Symposium, pp 193–202

Apple Inc (2013) 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: 33^{rd} 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, Fisher N (2007) The Partitioned Dynamic-priority Scheduling of Sporadic Task Systems. Real-Time Systems 36(3):199–226

Chen JJ, Chakraborty S (2011) Resource Augmentation Bounds for Approximate Demand Bound Functions. In: Proceedings of the 32nd IEEE Real-Time Systems Symposium, pp 272–281

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

Davis R, Burns A (2011) A survey of hard real-time scheduling for multiprocessor systems. ACM Comput Surv 43(4):1–44

Dertouzos M (1974) Control robotics: The procedural control of physical processes. In: Proceedings of IFIP Congress (IFIP'74)

DeVuyst M, Venkat A, Tullsen D (2012) Execution migration in a heterogeneous-ISA chip multiprocessor. In: Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems, pp 261–272

Garey M, Johnson D (1978) "Strong" NP-Completeness results: Motivation, examples, and implications. Journal of ACM 25(3):499–508

Gurobi Optimization Inc (2012) Gurobi optimizer reference manual. http://www.gurobi.com

Horn A (1974) Some simple scheduling algorithms. Naval Research Logistics Quarterly 21(1):177–185

IBM (2012) CPLEX Optimizer: High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming. http://www.ibm.com/software/commerce/optimization/cplex-optimizer

Intel Corporation (2013a) Intel Atom Processor. http://www.intel.com/atom

Intel Corporation (2013b) Intel Core Processor Family. http://www.intel.com/core

Johnson D (1973) Near-optimal bin packing algorithm. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology

Korte B, Vygen J (2006) Combinatorial Optimization: Theory and Algorithms, 3rd edn. Springer

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

Levin G, Funk S, Sadowskin C, Pye I, Brandt S (2010) DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In: Proceedings of the 22^{nd} Euromicro Conference on Real-Time Systems, pp 3–13

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

Luenberger DG, Ye Y (2008) Linear and Nonlinear Programming, 3rd Ed. International Series in Operations Research & Management Science

Nvidia Inc (2013) Tegra 4: Mobility at the speed of life. http://www.nvidia.com/object/tegra.html

Papadimitriou C (1994) Computational Complexity. Addison-Wesley

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 Applied Mathematics 10:155–164

Qualcomm Inc (2013) Snapdragon Processors: All-in-one Mobile Processor. http://www.qualcomm.com/snapdragon

Raravi G, Andersson B, Bletsas K (2011) Provably good task assignment on heterogeneous multiprocessor platforms for a restricted case but with a stronger adversary. SIGBED Review 8:19–22

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

Raravi G, Andersson B, Bletsas K, Nélis V (2012) Task Assignment Algorithms for Two-Type Heterogeneous Multiprocessors. In: 24^{th} Euromicro Conference on Real-Time Systems, pp 34–43

Samsung Inc (2013) Exynos 5 OCTA Processor. www.samsung.com/exynos/

ST Ericsson (2013) NOVATHOR - Smartphone and Tablet Platforms. http://www.stericsson.com/products/smartphone-platforms.jsp

Texas Instruments (2013) OMAP Technologies – OMAP Applications Processors. http://www.ti.com/omap