CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Task assignment algorithms for heterogeneous multiprocessors

Gurulingesh Raravi
Vincent Nélis

CISTER-TR-140510
Version:
Date: 1/1/2014

Task assignment algorithms for heterogeneous multiprocessors

Gurulingesh Raravi, Vincent Nélis

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: guhri@isep.ipp.pt, nelis@isep.ipp.pt

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

Abstract

Consider the problem of assigning implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising a constant number (denoted by 't') of distinct types of processors --- such a platform is referred to as a 't-type platform'. We present two algorithms, LPGim and LPGnm, each providing the following guarantee. For a given t-type platform and a task set, if there exists a task assignment such that tasks can be scheduled to meet their deadlines by allowing them to migrate only between processors of the same type (intra-migrative), then (i) LPGim succeeds in finding such an assignment where the same restriction on task migration applies (intra-migrative) but given a platform in which only one processor of each type is (1 + a*(t-1)/t) times faster and (ii) LPGnm succeeds in finding a task assignment where tasks are not allowed to migrate between processors (non-migrative) but given a platform in which every processor is (1+a) times faster. The parameter 'a' is a property of the task set; it is the maximum of all the task utilizations that are no greater than one. To the best of our knowledge, for t-type heterogeneous multiprocessors, (i) for the problem of intra-migrative task assignment, no previous algorithm exists with a proven bound and hence, our algorithm, LPGim, is the first of its kind and (ii) for the problem of non-migrative task assignment, our algorithm, LPGnm, has superior performance compared to state-of-the-art.

Task assignment algorithms for heterogeneous multiprocessors

(Submitted to Special Issue on Real-Time, Embedded and Cyber-Physical Systems)

Gurulingesh Raravi and Vincent Nélis, Polytechnic Institute of Porto, Portugal

Consider the problem of assigning implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising a constant number (denoted by t) of distinct types of processors — such a platform is referred to as a *t-type platform*. We present two algorithms, $\mathrm{LPG_{IM}}$ and $\mathrm{LPG_{NM}}$, each providing the following guarantee. For a given t-type platform and a task set, *if* there exists a task assignment such that tasks can be scheduled to meet their deadlines by allowing them to migrate only between processors of the same type (intra-migrative), *then* (i) $\mathrm{LPG_{IM}}$ succeeds in finding such an assignment where the same restriction on task migration applies (intra-migrative) but given a platform in which *only one processor of each type* is $1+\alpha\times\frac{t-1}{t}$ times faster and (ii) $\mathrm{LPG_{NM}}$ succeeds in finding a task assignment where tasks are not allowed to migrate between processors (non-migrative) but given a platform in which *every processor* is $1+\alpha$ times faster. The parameter α is a property of the task set; it is the maximum of all the task utilizations that are no greater than one. To the best of our knowledge, for t-type heterogeneous multiprocessors, (i) for the problem of intra-migrative task assignment, no previous algorithm exists with a proven bound and hence, our algorithm, $\mathrm{LPG_{IM}}$, is the first of its kind and (ii) for the problem of non-migrative task assignment, our algorithm, $\mathrm{LPG_{NM}}$, has superior performance compared to state-of-the-art.

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

General Terms: Algorithms, Performance, Theory

Additional Key Words and Phrases: Heterogeneous multiprocessors, Real-time scheduling

1. INTRODUCTION

This paper addresses the problem of assigning a set of real-time tasks on a heterogeneous multiprocessor platform. We consider *implicit-deadline sporadic tasks*, that is, a task generates a (potentially infinite) sequence of jobs where each job has an execution time and a deadline and for each task, the deadline of a job of this task is equal to the minimum time between job arrivals of this task. Such tasks can be used to model a range of applications where the software needs to perform an operation repeatedly on incoming or sampled data, e.g., feedback control systems, signal processing or multimedia playout. We consider a heterogeneous multiprocessor platform comprising a constant number (denoted by t) of distinct types of processors. We refer to such a platform as a *t-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 a constant number of distinct types of processors [Apple Inc. 2013; AMD Inc. 2013; Intel Corp. 2013a; 2013b; 2013c; Nvidia Inc. 2013; Qualcomm Inc 2013; Samsung Inc. 2013; Texas Instruments 2013; Alben 2013]. For scheduling tasks on such platforms, we consider three migration models: *non-migrative*, *intra-migrative* and *fully-migrative*.

In the *non-migrative* model (sometimes referred to as *partition* 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, on each processor, the given scheduling algorithm meets all deadlines of the tasks assigned on that 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. EDF is an *optimal* scheduling algorithm on a uniprocessor system [Liu and Layland 1973; Dertouzos 1974], with the interpretation that, for every valid arrival pattern, if a schedule exists that meets deadlines then EDF constructs a schedule that meets deadlines as well. Therefore, assuming that an optimal scheduling

algorithm is used on every processor, the challenging part is to find a task-to-processor assignment such that, *there exists* a schedule that meets all deadlines — such an assignment is said to be *feasible* assignment hereafter. Even in the simpler case of identical multiprocessors, finding a feasible task-to-processor assignment is NP-Complete in the strong sense [Johnson 1973]. Hence, this result continues to hold for t-type multiprocessors as well. In this work, we propose an algorithm, $\mathrm{LPG_{NM}}$, 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 are assigned, scheduling them to meet all deadlines under the intra-migrative model is well-understood, e.g., one may use an optimal identical multiprocessor scheduling algorithm, such as, ERfair [Anderson and Srinivasan 2000], DP-Fair [Levin et al. 2010] or U-EDF [Nelissen et al. 2012]. 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 such that, *there exists* a schedule that meets all deadlines. Even in the simpler case, in which each processor type has only one processor, finding a feasible task-to-processor-type assignment is NP-Complete in the strong sense (for the reasons discussed earlier). Hence, this result continues to hold for t-type platforms having one or more processors of each type as well. In this work, we propose an algorithm, $\mathrm{LPG_{IM}}$, for this problem, for which no previous algorithm (with a proven bound) exists.

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[1], it is rarely applicable in practice because job migration between processors of different types is hard to achieve as different processor types typically differ in their instruction sets, register formats, etc. Hence, in this work, we decided not to consider this model.

This work relies on the *resource augmentation* framework [Phillips et al. 1997] to characterize the performance of the algorithms. We define the *approximation ratio* $\mathrm{AR_{IM}}$ of an intra-migrative algorithm $\mathcal{A}_{\mathrm{IM}}$ (resp., $\mathrm{AR_{NM}}$ of a non-migrative algorithm $\mathcal{A}_{\mathrm{NM}}$) against an intra-migrative *adversary* as the lowest number such that, for every task set τ and 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 $\mathcal{A}_{\mathrm{IM}}$ (resp., $\mathcal{A}_{\mathrm{NM}}$) outputs an intra-migrative (resp., non-migrative) assignment which meets all the deadlines of τ on a platform $\pi^{(\mathrm{AR_{IM}})}$ (resp., $\pi^{(\mathrm{AR_{NM}})}$) whose processors are $\mathrm{AR_{IM}}$ (resp., $\mathrm{AR_{NM}}$) times faster than the corresponding processors in π. Hence, a low approximation ratio indicates high performance; the best achievable is 1 (which reflects the optimal algorithm for a given problem). Therefore, we aim to design algorithms with finite (and ideally small) approximation ratios.

Related work. The non-migrative task assignment problem on heterogeneous multiprocessors has been studied in the past [Baruah 2004b; 2004a; Raravi et al. 2012; Raravi et al. 2013; Raravi and Nélis 2012; Wiese et al. 2013]. It is a well-known fact that the non-migrative task assignment problem is equivalent to the problem of scheduling a set of non-real-time jobs, arriving at time zero, on unrelated parallel machine, so that they all finish before a specified time. This equivalent problem has been studied in [Horowitz and Sahni 1976; Lenstra et al. 1990; Jansen and Porkolab

[1]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, for a given task set and a computing platform, the set of fully-migrative solutions is a superset of the set of intra-migrative solutions which is a superset of the set of non-migrative solutions.

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

Computing Platform	Adversary Task migration	Task Assignment Algorithms			
		Algorithm	Task migration	Approx. ratio	Complexity
t-type[a]	non-migrative	[Baruah 2004a]	non-migrative	2	polynomial
t-type	non-migrative	[Baruah 2004b]	non-migrative	2	polynomial
t-type	non-migrative	[Lenstra et al. 1990]	non-migrative	2	polynomial
t-type	fully-migrative	[Correa et al. 2012]	non-migrative	4	polynomial
2-type[b]	non-migrative	[Raravi et al. 2013]	non-migrative	2	polynomial
2-type	intra-migrative	[Raravi et al. 2012]	intra-migrative	1.5	polynomial
2-type	intra-migrative	[Raravi et al. 2012]	non-migrative	2	polynomial
2-type	non-migrative	[Raravi and Nélis 2012]	non-migrative	PTAS[c]	exponential in $1/\epsilon$
t-type	non-migrative	[Horowitz and Sahni 1976]	non-migrative	PTAS	exponential in processors
t-type	non-migrative	[Jansen and Porkolab 1999]	non-migrative	PTAS	exponential in processors
t-type	non-migrative	[Wiese et al. 2013]	non-migrative	PTAS	exponential in $1/\epsilon$
t-type	intra-migrative	$\mathrm{LPG_{IM}}$, [This work]	intra-migrative	$1 + \left(\alpha \times \frac{t-1}{t}\right)$[d]	polynomial
t-type	intra-migrative	$\mathrm{LPG_{NM}}$, [This work]	non-migrative	$1 + \alpha \leq 2$	polynomial

[a] A heterogeneous multiprocessor platform having $t \geq 2$ processor types.
[b] A heterogeneous multiprocessor platform having only two processor types.
[c] A PTAS takes an instance of an optimization problem and a parameter $\epsilon > 0$ as inputs and, in time polynomial in the problem size (although not necessarily in the value of ϵ), produces a solution that is within a factor $1+\epsilon$ of being optimal.
[d] The parameter $0 < \alpha \leq 1$ is a property of the task set; it is the maximum of all the utilizations that are no greater than 1.

1999; Correa et al. 2012]. For the problem of assigning implicit-deadline sporadic tasks on heterogeneous multiprocessors, in [Baruah 2004b; 2004a; Lenstra et al. 1990], authors propose non-migrative algorithms with an approximation ratio of 2 against a non-migrative adversary. The approach discussed in [Lenstra et al. 1990] comes closest to our work since it formulates the task assignment problem as a Mixed Integer Linear Program (MILP) and then relaxes it to a Linear Program (LP) and finally uses a rounding technique to obtain non-migrative task assignment. We also follow the same approach in this work; however, by formulating MILP in a different way and using a different rounding technique, we obtain a *better* bound for our non-migrative task assignment algorithm than the one in [Lenstra et al. 1990]. Further, we also provide an algorithm for intra-migrative task assignment problem and prove its bound (whereas authors in [Lenstra et al. 1990] study only the non-migrative assignment problem).

The approaches proposed in [Raravi et al. 2012; Raravi et al. 2013; Raravi and Nélis 2012] are applicable only for two-type platforms (a special case of t-type in which $t = 2$) and hence are not applicable for generic t-type ($t \geq 2$) platforms.

Moving to algorithms whose approximation ratios have been proven against a more powerful adversary, recently, in [Correa et al. 2012], it is shown that if a task set can be scheduled by an optimal algorithm on a heterogeneous platform with full migrations (i.e., jobs *can* migrate between processors of *any type*) then, an optimal algorithm for scheduling tasks on a heterogeneous platform with no migrations (i.e., non-migrative assignment) needs processors four times as fast.

In [Horowitz and Sahni 1976; Jansen and Porkolab 1999; Raravi and Nélis 2012; Wiese et al. 2013], *polynomial-time approximation scheme*s (PTASs) have been proposed for the problem of non-migrative task assignment. A PTAS takes an instance of an optimization problem and a parameter $\epsilon > 0$ as inputs and, in time polynomial in the problem size (although not necessarily in the value of ϵ), produces a solution that is within a factor $1 + \epsilon$ of being optimal. PTAS is theoretically a significant result since such algorithms partition the task set in polynomial time, to any desired degree of accuracy. However, (most often) their practical significance is severely limited due to a very high run-time complexity that they incur.

The state-of-the-art along with the contributions of this paper are summarized in Table I. Each row in the table corresponds to a different algorithm. For example, the first row in the table is read as follows: for a t-type platform, a non-migrative algorithm is proposed in [Baruah 2004a] and it has an approximation ratio of 2 against non-migrative adversary.

Contributions and significance of this work. Consider a t-type platform π and an implicit-deadline sporadic task set τ in which, it holds that: $\forall k \in \{1, 2, \ldots, t\}$, for every task in τ, utilization of each task on a type-k processor is either no greater than α or is equal to ∞, where $0 < \alpha \leq 1$. We first present an intra-migrative algorithm, $\mathrm{LPG_{IM}}$, which offers the following guarantee. If there exists a feasible intra-migrative assignment of τ on π then $\mathrm{LPG_{IM}}$ succeeds in finding such a feasible intra-migrative assignment of τ but on π' in which *only one processor of each type* is $1 + \alpha \times \frac{t-1}{t}$ times faster than the corresponding processor in π (for defining its approximation ratio, we say that $\mathrm{LPG_{IM}}$ needs a platform $\pi^{(1+\alpha\times\frac{t-1}{t})}$ in which every processor is $1+\alpha\times\frac{t-1}{t}$ times faster). Then, we modify $\mathrm{LPG_{IM}}$ to obtain $\mathrm{LPG_{NM}}$, a non-migrative algorithm which offers the following guarantee. If there exists a feasible *intra-migrative* assignment of τ on π then $\mathrm{LPG_{NM}}$ succeeds in finding a feasible *non-migrative* assignment of τ but on $\pi^{(1+\alpha)}$ in which *every processor* is $1 + \alpha$ times faster.

We believe that the significance of this work is two-fold. First, for the problem of finding an intra-migrative assignment of real-time tasks on t-type platforms, no previous algorithm with a proven approximation ratio exists and hence our algorithm, $\mathrm{LPG_{IM}}$, is the first for this problem. Second, for the problem of non-migrative task assignment on t-type platforms, our algorithm, $\mathrm{LPG_{NM}}$, has superior performance compared to state-of-the-art. This can be seen from Table I since (i) $\mathrm{LPG_{NM}}$ has a tighter bound (i.e., its approximation ratio, $1 + \alpha \leq 2$, is quantified using the parameter, α, which is a characteristic of the task set) and that too against a more powerful *intra-migrative* adversary when compared to the bounds of algorithms in [Baruah 2004a; 2004b; Lenstra et al. 1990] (whose approximation ratio, 2, is a constant against non-migrative adversary), (ii) among algorithms with approximation ratio proven against an adversary with a migration model of intra-migrative or greater power [Correa et al. 2012], $\mathrm{LPG_{NM}}$ offers the best approximation ratio and (iii) compared to PTAS algorithms [Horowitz and Sahni 1976; Jansen and Porkolab 1999; Wiese et al. 2013] whose practical significance is severely limited as they incur a very high time-complexity (exponential in processors or exponential in $1/\epsilon$), our algorithm offers a lower (i.e., polynomial) time-complexity.

2. SYSTEM MODEL

We consider the problem of scheduling a task set τ of n independent implicit-deadline sporadic tasks on a t-type heterogeneous multiprocessor platform π comprising m processors. In platform π, the set of m_k processors of type-k is denoted by $\pi^k = \{p_1, p_2 \ldots, p_{m_k}\}$, where $1 \leq k \leq t$ and p_j denotes a processor of type-k, where $1 \leq j \leq m_k$. It then holds that: $\bigcup_{k=1}^{t} \pi^k = \pi$ and $\bigcap_{k=1}^{t} \pi^k = \emptyset$ and finally $\sum_{k=1}^{t} m_k = m$.

Each task $\tau_i \in \tau$ is characterized by a *worst-case execution time* (WCET) 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 τ_i has to complete its execution within T_i time units (also referred to as *deadline*) from its release. On a t-type platform, the WCET of a task depends on the type of the processor on which the task executes. We denote by C_i^k the WCET of a task τ_i when executed on a type-k processor, where $k \in \{1, 2, \ldots, t\}$. We denote by $u_i^k \stackrel{\text{def}}{=} C_i^k/T_i$ the utilization of task τ_i on a type-k processor and u_i^k is a real number in $[0, 1] \cup \{\infty\}$ — if τ_i cannot be executed on

Minimize Z subject to the following constraints:

I1.	$\forall \tau_i \in \tau : \sum_{k \in \{1,2,\ldots,t\}} x_i^k = 1$
I2.	$\forall k \in \{1,2,\ldots,t\} : \sum_{\tau_i \in \tau} x_i^k \times u_i^k \leq \mathrm{Z} \times m_k$
I3.	$\forall \tau_i \in \tau, \forall k \in \{1,2,\ldots,t\} : x_i^k \in \{0,1\}$ are integers

Fig. 1: MILP-Feas(τ, π) — MILP formulation for assigning tasks in τ to processor types in π.

a type-k processor then u_i^k is set to ∞. Let α be a real number defined as follows:

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

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

$$\forall k \in \{1,2,\ldots,t\}, \forall \tau_i \in \tau : (u_i^k \leq \alpha) \quad \vee \quad (u_i^k = \infty) \tag{2}$$

3. MILP-ALGO: AN OPTIMAL INTRA-MIGRATIVE ALGORITHM

In this section, an *optimal intra-migrative algorithm* is presented for assigning tasks in τ to processor types in π, that is, for each task set, it succeeds in finding a feasible assignment, if such an assignment exists. The proposed algorithm is based on solving a Mixed Integer Linear Programming (MILP) formulation. As described in Section 1, once the tasks are assigned to processor types, we assume that, an optimal identical multiprocessor scheduler (e.g., [Anderson and Srinivasan 2000; Levin et al. 2010; Nelissen et al. 2012]) is used to schedule the tasks on processors of each type. From the feasibility tests of identical multiprocessor scheduling, the following necessary and sufficient set of conditions must hold for intra-migrative assignment to be feasible:

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

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

where τ^k denotes the set of tasks that are assigned to processors of type-k. The first condition is essential since the system model does not allow *parallel* execution of any job. The second condition is essential as it ensures that the computing *workload* does not exceed the processing *capacity* [Horn 1974].

Given these necessary and sufficient feasibility conditions, we now propose an optimal intra-migrative task assignment algorithm, MILP-Algo, which works as follows.

First, solve the MILP formulation, MILP-Feas(τ, π), shown in Fig. 1. In this formulation, variable Z is the objective function to be minimized and it denotes the maximum capacity that is used on any processor type (which is given by $\max_{k \in \{1,2,\ldots,t\}} \sum_{\tau_i \in \tau^k} x_i^k \times u_i^k$). Each variable x_i^k indicates whether a task τ_i is assigned to a processor type-k or not. The first set of constraints specifies that every task must be entirely assigned. The second set of constraints asserts that at most $\mathrm{Z} \times m_k$ capacity of type-k processors can be used. The third set of constraints asserts that each task must be *integrally* assigned to one of the t processor types.

Second, using the solution of this MILP formulation, assign the tasks to processor types as follows. If $\mathrm{Z} > 1$ then declare failure as this indicates that the feasibility condition shown in Eq. (4) is violated (implying that the task set is not intra-migrative feasible). Otherwise, for each task $\tau_i \in \tau$, assign τ_i to type-k processors only if $x_i^k = 1$. We now show that the MILP-Algo is an optimal intra-migrative task assignment algorithm.

LEMMA 3.1 (**MILP-ALGO IS OPTIMAL**). *If there exists a feasible intra-migrative assignment of τ on π then MILP-Algo succeeds in finding such a feasible intra-migrative assignment.*

PROOF. Suppose that the task set τ is intra-migrative feasible on platform π and let $\mathcal{X}$ denote a feasible assignment. It can be seen that, $\forall \tau_i \in \tau$, by assigning values to x_i^k variables of MILP formulation, MILP-Feas(τ, π), of Fig. 1 as:

$$\text{if } \mathcal{X}(i) = k \text{ then } \quad x_i^k \leftarrow 1$$
$$x_i^j \leftarrow 0, \forall j \in \{1, 2, \ldots, t\} \wedge j \neq k$$

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

Now, suppose that there is a (feasible) solution with $\mathrm{Z} \leq 1$ to the MILP formulation, MILP-Feas(τ, π), of Fig. 1. Using this solution, define the assignment of tasks to processor types as follows:

$$\forall \tau_i \in \tau \ : \ \mathcal{X}(i) \leftarrow k, \ \text{ if } x_i^k = 1$$

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

In general, solving an MILP formulation has high computational complexity. In particular, the decision problem MILP is NP-complete and even with knowledge of the structure of the constraints in the modeling of heterogeneous multiprocessor scheduling, no polynomial-time algorithm is known (p. 245 in [Garey and Johnson 1979]). Hence, we now propose a polynomial time-complexity (but non-optimal) intra-migrative algorithm, $\mathrm{LPG_{IM}}$, by relaxing the MILP formulation to LP (which can be solved in polynomial time [Karmakar 1984]) and using graph theory techniques.

4. AN OVERVIEW OF OUR NEW INTRA-MIGRATIVE ALGORITHM $\mathrm{LPG_{IM}}$

We now give an overview of our intra-migrative algorithm, $\mathrm{LPG_{IM}}$. It has the following four steps:

Step 1. We first relax the MILP formulation of Fig. 1 to LP formulation by allowing all the x_i^k variables to take *real* values in the range $[0, 1]$ instead of binary values $\{0, 1\}$ and then solve it. In the solution returned by the LP solver, some tasks will be *integrally* assigned to a processor type and the rest will be *fractionally* assigned to more than one processor type. We show that, for this LP formulation, there exists a *(vertex) solution* in which *at most* $t - 1$ tasks are fractionally assigned and such a solution is of interest to us. This step is discussed in Section 5.

Step 2. From such a solution, we construct a bi-partite graph with (i) a set of nodes corresponding to fractional tasks, (ii) another set of nodes corresponding to those processor types to which these fractional tasks are assigned and (iii) a set of edges which connect these task nodes and processor type nodes depending on the values of the x_i^k variables (which also represent the *weights* of these edges). The solution (returned by the LP solver) might be such that, upon representing it with a bi-partite graph, the graph may contain a few *circuits*. This step is discussed in detail in Section 6 along with the relevant graph theory terminology.

Step 3. The circuits in the graph, if any, are detected and broken, one by one. A circuit is broken by re-adjusting the weights of the edges such that the weight of at least one edge in the circuit becomes zero which is then deleted. While re-adjusting the weights, it is ensured that, for each processor type, its used capacity after re-adjusting the weights does not exceed its used capacity before re-adjusting. This step (discussed

Minimize Z subject to the following constraints:

R1.	$\forall \tau_i \in \tau : \sum_{k \in \{1,2,\ldots,t\}} x_i^k = 1$
R2.	$\forall k \in \{1, 2, \ldots, t\} : \sum_{\tau_i \in \tau} x_i^k \times u_i^k \leq \mathrm{Z} \times m_k$
R3.	$\forall \tau_i \in \tau, \forall k \in \{1, 2, \ldots, t\} : x_i^k \geq 0$ are real numbers

Fig. 2: LP-Feas(τ, π) — Relaxed LP formulation for assigning tasks in τ to processor types in π.

in Section 7) reduces the complexity of the problem when assigning the at most $t-1$ fractional tasks integrally to processor types, in the final step.

Step 4. The at most $t-1$ fractional tasks are assigned integrally to processor types. We show that, in order to do this, the algorithm needs a platform in which *only one processor of each type* is $1 + \alpha \times \frac{t-1}{t}$ times faster. This step is discussed in Section 8 along with the proof of approximation ratio of this four step intra-migrative algorithm, $\mathrm{LPG_{IM}}$.

5. STEP 1 OF $\mathrm{LPG_{IM}}$: SOLVING THE LP FORMULATION

First, we relax the MILP formulation, MILP-Feas(τ, π), to an LP formulation, LP-Feas(τ, π), as shown in Fig. 2. In this LP formulation, all the variables have the same meaning as in the MILP formulation and the first two sets of constraints are the same as well. Only the third set of constraints is different (i.e., *relaxed*) and it now asserts that a task can either be *integrally* assigned or *fractionally* assigned to processor types. We then solve the LP formulation using standard LP solvers (e.g., IBM ILOG CPLEX [IBM 2013]). Since the LP formulation is less constrained than the MILP, the following lemma trivially holds.

LEMMA 5.1. *Let* $\mathrm{Z_{MILP}}$ *and* $\mathrm{Z_{LP}}$ *be the values of the objective functions that any MILP solver and LP solver would return by solving MILP-Feas*(τ, π) *and LP-Feas*(τ, π)*, respectively. It then holds that,* $\mathrm{Z_{LP}} \leq \mathrm{Z_{MILP}}$.

Among all the optimal solutions to an LP formulation, at least one solution lies at a *vertex* of the *feasible region*[2] (see, pp. 117 in [Luenberger and Ye 2008]). We are interested in such a solution, as it reflects a task assignment in which at most $t-1$ tasks are fractionally assigned between different processor types (referred to as *fractional tasks*, hereafter) — see Lemma 5.2 below. We would like to mention that, if the solution returned by the solver is not a vertex solution then it can always be converted into a vertex solution [Baruah 2004b].

LEMMA 5.2. *Consider an optimal solution for LP-Feas*(τ, π)*, that lies at the vertex of the feasible region. For such a solution, it holds that, at most* $t-1$ *tasks are fractionally assigned.*

PROOF. The proof is based on Fact 2 in [Baruah 2004b]: *"consider a linear program on* n *variables, 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 basic solution), at most* m *of the variables have non-zero values".* Clearly, the LP formulation of Fig. 2 is a linear program on $n' = n \times t + 1$ variables (i.e., $n \times t$ x_i^k variables and one Z variable), all subject to non-negativity constraint, and $m' = n + t$ further linear constraints (n constraints due to R1 plus t constraints due to R2). As $m' < n'$ (we assume $n \geq 2 \wedge t \geq 2$; otherwise the problem becomes trivial), we know from the above fact that in every optimal solution

[2]The *feasible region* of an LP in n-dimensional space is the region over which all the constraints are satisfied. Further, in general, LP solvers (such as CPLEX [IBM 2013]) always return optimal vertex solution.

at the vertex of the feasible region, it holds that, at most $m' = n + t$ variables take non-zero values. Since Z is certain to be non-zero, it holds that:

$$\text{the number of non-zero } x_i^k \text{ variables is } \textit{at most } n + t - 1 \tag{5}$$

We know that, for each task $\tau_i \in \tau$, there exists at least one $k \in \{1, 2, \ldots, t\}$ such that $x_i^k > 0$. Let num denote the number of tasks for which there exists at least two k such that $x_i^k > 0$. It follows from the definition of num that the total number of non-zero variables is *at least* $\mathrm{num} \times 2 + (n - \mathrm{num})$ which can be rewritten as *at least* $n + \mathrm{num}$. If $\mathrm{num} \geq t$ then:

$$\text{the number of non-zero } x_i^k \text{ variables is } \textit{at least } n + t \tag{6}$$

This contradicts Eq. (5). Hence, $\mathrm{num} < t$, which implies that the number of tasks fractionally assigned between different processor types is at most $t - 1$. □

The remaining three steps focus on assigning these (at most) $t - 1$ fractional tasks integrally to processor types.

6. STEP 2 OF $\mathrm{LPG_{IM}}$: FORMING THE BI-PARTITE GRAPH

In this step, using the vertex solution, in which at most $t - 1$ tasks are fractionally assigned, we construct a bi-partite graph[3]. The graph is constructed with only (i) *fractional tasks* and (ii) those processor types to which at least one fractional task is assigned (referred to as *fractional processor types*). Hence, while forming the graph, we ignore all the tasks that are integrally assigned and all the processor types to which no fractional task is assigned. Let $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ denote such a bi-partite graph and it is formed as follows:

— each *fractional task* $\tau_i \in \tau$, is represented by a *task node* $\tau_{\mathrm{i}} \in \mathrm{A}$ defined by a one-to-one mapping.
— each *fractional processor type-k*, $k \in \{1, 2, \ldots, t\}$, is represented by a *processor type node* $\pi^{\mathrm{k}} \in \mathrm{B}$ defined by a one-to-one mapping.
— a task node $\tau_{\mathrm{i}} \in \mathrm{A}$ is connected by an edge $\mathrm{e_i^k} \in \mathrm{E}$ to a processor type node $\pi^{\mathrm{k}} \in \mathrm{B}$ if and only if $0 < x_i^k < 1$. Each edge $\mathrm{e_i^k} \in \mathrm{E}$ has a weight set to x_i^k.

Observe that, since the bi-partite graph is constructed only with fractional tasks and fractional processor types, the graph may contain a few *circuits* (defined below).

Definition 6.1 (***Circuit***). A circuit $C = \{n_1 \to n_2 \to \cdots \to n_s \to n_1\}$ in a graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ is a path in which each node is visited exactly once except one node which is visited twice, i.e., both at the start and at the end. Each circuit C can also be denoted by a corresponding subgraph $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right) \subseteq \mathrm{G}$ containing only those nodes and edges that are in C.

For convenience, we use C and $\mathrm{G^C}$ interchangeably, in the rest of the paper. The following lemma states that a circuit in a bi-partite graph is always an even circuit.

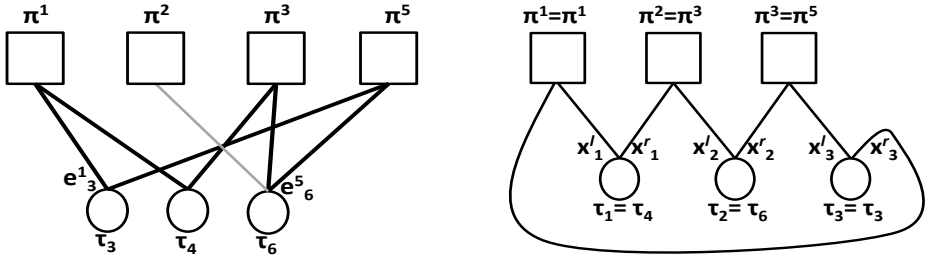
LEMMA 6.2 (**FROM THEOREM 1.2.18 IN [WEST 2000]**). *Any circuit* $C = \{n_1 \to n_2 \to \cdots \to n_{s=2N_c} \to n_1\}$*, where* $N_c > 0$ *is a positive integer, in a bi-partite graph* $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$*, always has an even number of distinct nodes, with half the number of nodes from the set* A *and the other half from the set* B.

PROOF. In cycle, $C = \{n_1 \to n_2 \to \ldots \to n_s \to n_1\}$, let the node n_1 be in set A (abbreviated $n_1 \in \mathrm{A}$). If $n_1 \in \mathrm{A}$ then by definition of bi-partite graph, it must be that $n_2 \in \mathrm{B}$,

[3]A bi-partite graph is a graph with two *disjoint* sets of vertices such that every edge connects a vertex in one set to a vertex in the other set.

Table II: Values of x_i^k variables output by the LP solver.

Tasks	Values of indicator variables				
	x_i^1	x_i^2	x_i^3	x_i^4	x_i^5
τ_1	1	0	0	0	0
τ_2	0	0	0	1	0
τ_3	0.7	0	0	0	0.3
τ_4	0.5	0	0.5	0	0
τ_5	0	0	0	1	0
τ_6	0	0.1	0.5	0	0.4
τ_7	0	0	0	0	1



(a) The bi-partite graph constructed from Table II; circuit is indicated by black edges.

(b) The circuit of Fig. 3a (with $N_c = 3$ task and $N_c = 3$ processor type nodes) after re-arranging and re-indexing.

Fig. 3: An example to illustrate the bi-partite graph (formed from fractional tasks and processor types) and the concept of a circuit.

$n_3 \in \mathrm{A}$, $n_4 \in \mathrm{B}$ and so on. In general, it holds that, $n_{2j+1} \in \mathrm{A}$ and $n_{2j} \in \mathrm{B}$. Since C is a cycle, $n_s \in \mathrm{B}$ so that $s = 2N_c$ for some positive integer N_c. Therefore, cycle C has even number of nodes (and half the nodes in circuit C are from set A and the other half are from set B). Hence the proof. □

PROPERTY 1 (**FOLLOWS FROM LEMMA 6.2**). *In a circuit* $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right)$*, it holds that,* $\left|\mathrm{A^C}\right| = \left|\mathrm{B^C}\right| = N_c$*, where* $N_c > 0$ *is a positive integer.*

We now illustrate these concepts with an example.

Example 6.3. Consider a task set τ of 7 tasks and a t-type platform π with $t = 5$. Let the solution output by the LP solver be as shown in Table II. The bi-partite graph constructed from this solution using the fractional tasks (τ_3, τ_4 and τ_6) and the fractional processor types (type-1, type-2, type-3 and type-5), is shown in Fig. 3a. As can be seen, there is a circuit $C = \left\{\tau_3 \rightarrow \pi^1 \rightarrow \tau_4 \rightarrow \pi^3 \rightarrow \tau_6 \rightarrow \pi^5 \rightarrow \tau_3\right\}$ in the graph, with 6 distinct nodes in which $N_c = 3$ nodes each are from the set A and the set B. The graph corresponding to this circuit is given by $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right)$ where $\mathrm{A^C} = \{\tau_3, \tau_4, \tau_6\}$, $\mathrm{B^C} = \left\{\pi^1, \pi^3, \pi^5\right\}$ and $\mathrm{E^C} = \left\{\mathrm{e}_3^1, \mathrm{e}_4^1, \mathrm{e}_4^3, \mathrm{e}_6^3, \mathrm{e}_6^5, \mathrm{e}_3^5\right\}$.

Definition 6.4 (***shared processor type node***). A fractional processor type node $\pi^{\mathrm{k}} \in \mathrm{B}$ in a graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ is said to be shared only if it is connected to at least two task nodes $\tau_{\mathrm{i1}} \in \mathrm{A}$ and $\tau_{\mathrm{i2}} \in \mathrm{A}$. Otherwise, it is said to be non-shared.

For example, in Fig. 3a, although all the four nodes, π^1, π^2, π^3 and π^5, are *fractional processor type nodes*, only π^1, π^3 and π^5, are *shared processor type nodes*.

LEMMA 6.5. *If there is no circuit in a graph* $G = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ *then there exists at least one task node in* A *that is connected to at most one shared processor type node in* B.

Further, since this task is fractional, we know that, it is also connected to at least one non-shared processor type node in B.

PROOF. From Definition 6.1 and 6.4, it holds that, in a circuit, each task node is connected to exactly two shared processor type nodes. Thus, it can be easily proven that, if every task node in a graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ is connected to at least two *shared* processor type nodes then there exists at least one circuit in G. Hence, by contraposition, it holds that, if there is no circuit in graph G then not every task node is connected to at least two shared processor type nodes. This implies that, if there is no circuit in graph G then there exists at least one task node $\tau_{\mathrm{i}} \in \mathrm{A}$ that is connected to at most one shared processor type node. Since all the task nodes in G are fractional, the task node τ_{i} must be connected to at least two processor type nodes and hence to at least one *non-shared* processor type node. Hence the proof. □

The circuit shown in Fig. 3a can be re-arranged as shown in Fig. 3b. Note in Fig. 3b that, the nodes are re-indexed. For ease of explanation, we use this notion of re-arranged graph in the next step.

Finally, we define the capacity used on a processor type in a circuit C by the tasks in that circuit as follows.

Definition 6.6 (***Capacity used on a processor type in a circuit***). Consider a circuit $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right)$ in a graph G. The capacity $\mathcal{C}^j_{\mathrm{C}}$ used on a processor type-j node, $\pi^{\mathrm{j}} \in \mathrm{B^C}$, by the task nodes $\forall\, \tau_{\mathrm{i}} \in \mathrm{A^C}$, is given by:

$$\mathcal{C}^j_{\mathrm{C}} \stackrel{\mathrm{def}}{=} \sum_{\tau_{\mathrm{i}} \in \mathrm{A^C}:\ \mathrm{x}^{\mathrm{j}}_{\mathrm{i}} > 0} \mathrm{x}^{\mathrm{j}}_{\mathrm{i}} \times \mathrm{u}^{\mathrm{j}}_{\mathrm{i}} \tag{7}$$

Remark about notation. In Eq. (7), index j is used for processor type instead of (the earlier notation) k. This is to avoid any confusion since the processor type nodes are re-indexed in the circuit (as shown in Fig. 3b).

7. STEP 3 OF $\mathrm{LPG_{IM}}$: DETECTING AND REMOVING THE CIRCUITS IN THE GRAPH

In the graph constructed as described in the previous section, if there are any circuits then we break all such circuits, in this step. Each circuit is broken by re-adjusting the weights of the edges (x^j_i) within the circuit such that the weight of at least one edge becomes zero, which breaks the circuit. The edge whose weight becomes zero is removed from the graph. While manipulating the weights of edges in a circuit $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right)$, it is ensured that, for each (shared) processor type $\pi^{\mathrm{j}} \in \mathrm{B^C}$, its capacity used by the tasks in the circuit after re-adjusting the weights (denoted by $\mathcal{C}^{j'}_{\mathrm{C}}$) does not exceed its original used capacity, i.e., before re-adjusting the weights (denoted by $\mathcal{C}^j_{\mathrm{C}}$). Breaking all the circuits reduces the complexity of the problem when assigning the (at most) $t-1$ fractional tasks integrally to processor types, which is discussed in Section 8. We now discuss, in detail, how to detect and remove circuits from the graph.

A circuit in a graph can be detected using *Depth First Search* (DFS) algorithm, generally found in textbooks (e.g., see Chap. 22.3 in [Cormen et al. 2001]). Hence, we mainly focus on removing the detected circuits in our graph. The following lemma shows how to remove at least one edge in the given circuit without increasing the capacity used on any of the shared processor types that are in the circuit.

LEMMA 7.1. *Consider a circuit* $\mathrm{G^C} = \left(\mathrm{A^C}, \mathrm{B^C}, \mathrm{E^C}\right)$ *(with* N_c *task and* N_c *processor type nodes — see Property 1) arranged as shown in Fig. 3b. Let* x^ℓ_i *and* x^{r}_i *denote the fraction of task* τ_i *(*$\forall i \in \{1, 2, \ldots, N_c\}$*) that is assigned to the shared processor type which is on* τ_i*'s "left" and* τ_i*'s "right", respectively. From Fig. 3b and Definition* (1)*,*

$\mathcal{C}_{\mathrm{C}}^{j}, \forall j \in \{1, 2, \ldots, N_c\}$*, can be re-defined as:*

$$\mathcal{C}_{\mathrm{C}}^{j} = \begin{cases} (x_{N_c}^{\mathrm{r}} \times u_{N_c}^{\mathrm{r}}) + (x_1^{\ell} \times u_1^{\ell}) & \textit{if } j = 1 \\ (x_{j-1}^{\mathrm{r}} \times u_{j-1}^{\mathrm{r}}) + (x_j^{\ell} \times u_j^{\ell}) & \textit{if } j \in \{2, \ldots, N_c\} \end{cases} \tag{8}$$

If it holds that

$$\prod_{g=1}^{N_c} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \geq 1 \tag{9}$$

then after updating the fractional assignments as follows:

$$x_i^{\mathrm{r}'} = x_i^{\mathrm{r}} - \frac{\epsilon}{u_i^{\ell}} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \tag{10}$$

$$x_i^{\ell'} = x_i^{\ell} + \frac{\epsilon}{u_i^{\ell}} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \tag{11}$$

where $\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}}$ *is assumed to be* 1 *for* $i = 1$ *and where* $\epsilon > 0$ *denotes a real number such that*

$$\epsilon = \min_{z \in [1,2,\ldots,N_c]} \left\{ \frac{x_z^{\mathrm{r}} \times u_z^{\ell}}{\prod_{g=1}^{z-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}}} \right\} \tag{12}$$

the following properties are satisfied:

P1.. $\forall j \in \{1, 2, \ldots, N_c\} : \mathcal{C}_{\mathrm{C}}^{j'} \leq \mathcal{C}_{\mathrm{C}}^{j}$*, where* $\mathcal{C}_{\mathrm{C}}^{j'}$ *denotes the capacity used on shared processor type* j*, after updating the fractional assignments.*
P2.. $\forall i \in \{1, 2, \ldots, N_c\} : x_i^{\ell'} + x_i^{\mathrm{r}'} = x_i^{\ell} + x_i^{\mathrm{r}}$.
P3.. $\forall i \in \{1, 2, \ldots, N_c\} : x_i^{\ell'} \geq 0$ *and* $x_i^{\mathrm{r}'} \geq 0$.
P4.. $\exists i \in \{1, 2, \ldots, N_c\} : x_i^{\mathrm{r}'} = 0$.

PROOF. We now prove each of these four properties.
Proof of P1. This will be shown separately for processor type $j = 1$ and $\forall j \in \{2, 3, \ldots, N_c\}$.
Case 1: $j = 1$. From Eq. (8) and (10), we have:

$$\text{from Eq. (8):} \quad \mathcal{C}_{\mathrm{C}}^{1'} = (x_{N_c}^{\mathrm{r}'} \times u_{N_c}^{\mathrm{r}}) + (x_1^{\ell'} \times u_1^{\ell}) \tag{13}$$

$$\text{from Eq. (10):} \quad x_{N_c}^{\mathrm{r}'} = x_{N_c}^{\mathrm{r}} - \frac{\epsilon}{u_{N_c}^{\ell}} \times \prod_{g=1}^{N_c-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \tag{14}$$

From Eq. (11) and from the assumption that $\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} = 1$ for $i = 1$, we have:

$$x_1^{\ell'} = x_1^{\ell} + \frac{\epsilon}{u_1^{\ell}} \tag{15}$$

Thus, by substituting Eq. (14) and (15) in Eq. (13) yields:

$$\begin{aligned}
\mathcal{C}_{\mathrm{C}}^{1'} &= \left(x^{\mathrm{r}}_{N_c} - \frac{\epsilon}{u^{\ell}_{N_c}} \times \prod_{g=1}^{N_c-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \times u^{\mathrm{r}}_{N_c} + \left(x^{\ell}_1 + \frac{\epsilon}{u^{\ell}_1}\right) \times u^{\ell}_1 \\
&= (x^{\mathrm{r}}_{N_c} \times u^{\mathrm{r}}_{N_c}) - \epsilon \times \prod_{g=1}^{N_c} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g} + \left(x^{\ell}_1 \times u^{\ell}_1\right) + \epsilon \\
&\overset{\text{from (8)}}{=} \mathcal{C}^1_{\mathrm{C}} + \epsilon \times \left(1 - \prod_{g=1}^{N_c} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \overset{\text{from (9)}}{\leq} \mathcal{C}^1_{\mathrm{C}}
\end{aligned} \tag{16}$$

Case 2: $j \in \{2, \ldots, N_c\}$. From Eq. (8) and from Eq. (10) and (11), we have:

$$\mathcal{C}_{\mathrm{C}}^{j'} = (x^{\mathrm{r}'}_{j-1} \times u^{\mathrm{r}}_{j-1}) + (x^{\ell'}_j \times u^{\ell}_j) \tag{17}$$

$$x^{\mathrm{r}'}_{j-1} = x^{\mathrm{r}}_{j-1} - \frac{\epsilon}{u^{\ell}_{j-1}} \times \prod_{g=1}^{j-2} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g} \tag{18}$$

$$x^{\ell'}_j = x^{\ell}_j + \frac{\epsilon}{u^{\ell}_j} \times \prod_{g=1}^{j-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g} \tag{19}$$

Thus, by substituting Eq. (18) and (19) in Eq. (17) yields:

$$\begin{aligned}
\mathcal{C}_{\mathrm{C}}^{j'} &= \left(x^{\mathrm{r}}_{j-1} - \frac{\epsilon}{u^{\ell}_{j-1}} \times \prod_{g=1}^{j-2} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \times u^{\mathrm{r}}_{j-1} + \left(x^{\ell}_j + \frac{\epsilon}{u^{\ell}_j} \times \prod_{g=1}^{j-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \times u^{\ell}_j \\
&= (x^{\mathrm{r}}_{j-1} \times u^{\mathrm{r}}_{j-1}) + (x^{\ell}_j \times u^{\ell}_j) - \left(\epsilon \times \frac{u^{\mathrm{r}}_{j-1}}{u^{\ell}_{j-1}} \times \prod_{g=1}^{j-2} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) + \left(\epsilon \times \prod_{g=1}^{j-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \\
&= (x^{\mathrm{r}}_{j-1} \times u^{\mathrm{r}}_{j-1}) + (x^{\ell}_j \times u^{\ell}_j) - \left(\epsilon \times \prod_{g=1}^{j-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) + \left(\epsilon \times \prod_{g=1}^{j-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}\right) \\
&= (x^{\mathrm{r}}_{j-1} \times u^{\mathrm{r}}_{j-1}) + (x^{\ell}_j \times u^{\ell}_j) \\
&\overset{\text{from (9)}}{=} \mathcal{C}^j_{\mathrm{C}}
\end{aligned} \tag{20}$$

From Eq. (16) and (20), it can be seen that, performing operations shown in Eq. (10) and (11) satisfies property **P1**.

Proof of P2. For every $i \in \{1, \ldots, N_c\}$, it can be seen that adding Eq. (10) and (11) results in $x^{\ell'}_i + x^{\mathrm{r}'}_i = x^{\ell}_i + x^{\mathrm{r}}_i$, and hence the property immediately follows.

Proof of P3. Since $\epsilon > 0$, it is trivial from Eq. (11) that $\forall i \in \{1, \ldots, N_c\}$: $x^{\ell'}_i > x^{\ell}_i > 0$. Then, from Eq. (10), any $x^{\mathrm{r}'}_i$ will be negative if and only if

$$\begin{aligned}
x^{\mathrm{r}}_i &< \frac{\epsilon}{u^{\ell}_i} \times \prod_{g=1}^{i-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g} \\
&\overset{\text{from (12)}}{<} \min_{z \in \{1,2,\ldots,N_c\}} \left\{ \frac{x^{\mathrm{r}}_z \times u^{\ell}_z}{\prod_{g=1}^{z-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}} \right\} \times \frac{1}{u^{\ell}_i} \times \prod_{g=1}^{i-1} \frac{u^{\mathrm{r}}_g}{u^{\ell}_g}
\end{aligned}$$

Since the $\min$ term evaluates to $\leq \frac{x_i^{\mathrm{r}} \times u_i^{\ell}}{\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}}}$, we have:

$$\begin{aligned} x_i^{\mathrm{r}} &< \frac{x_i^{\mathrm{r}} \times u_i^{\ell}}{\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}}} \times \frac{1}{u_i^{\ell}} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} < \frac{x_i^{\mathrm{r}} \times u_i^{\ell}}{u_i^{\ell}} \\ &< x_i^{\mathrm{r}} \end{aligned}$$

which is impossible. Hence $x_i^{\mathrm{r}'} \geq 0$.

Proof of P4. From Eq. (12), it holds that:

$$\exists i \in \{1, 2, \ldots, N_c\} : \epsilon = \frac{x_i^{\mathrm{r}} \times u_i^{\ell}}{\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}}} \tag{21}$$

For such i, Eq. (10) can be re-written as:

$$x_i^{\mathrm{r}'} = x_i^{\mathrm{r}} - \frac{\epsilon}{u_i^{\ell}} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \tag{22}$$

Substituting the value of ϵ, we obtain, $\exists i \in \{1, 2, \ldots, N_c\} : x_i^{\mathrm{r}'} = 0$. Hence the property holds.

As a conclusion, we showed that modifying the fractional assignments of the tasks according to Eq. (10) and (11) ensures that all the four properties P1, P2, P3 and P4 are satisfied. Hence the proof. □

Lemma 7.1 showed that, in a circuit with N_c task nodes, if $\prod_{g=1}^{N_c} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} \geq 1$ then transferring the fractions from "right to left" within the circuit will (i) delete an edge (as its weight becomes zero, by P4) so that the circuit breaks and (ii) ensures that $\forall j \in \{1, 2, \ldots, N_c\} : \mathcal{C}_{\mathrm{C}}^{j'} \leq \mathcal{C}_{\mathrm{C}}^{j}$. Since no fraction was moved to/from those processor types that are in set B but not in circuit C, their capacities remain unaffected. Hence, $\forall \pi^j \in B : \mathcal{C}_{\mathrm{C}}^{j'} \leq \mathcal{C}_{\mathrm{C}}^{j}$. Analogously, it can be shown that if $\prod_{g=1}^{N_c} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} < 1$ then transferring the fractions from "left to right" within the circuit will also yield the same result. The claim is presented formally below in Lemma 7.2 but the proof which is very similar to the proof of Lemma 7.1, is omitted due to space constraint.

LEMMA 7.2. *Consider a circuit* $\mathrm{G}^{\mathrm{C}} = \left(\mathrm{A}^{\mathrm{C}}, \mathrm{B}^{\mathrm{C}}, \mathrm{E}^{\mathrm{C}}\right)$ *(with* N_c *task and* N_c *processor type nodes — see Property 1) arranged as shown in Fig. 3b. Let* x_i^{ℓ} *and* x_i^{r} *denote the fraction of task* τ_i *(*$\forall i \in \{1, 2, \ldots, N_c\}$*) that is assigned to the shared processor type which is on* τ_i*'s "left" and* τ_i*'s "right", respectively. From Fig. 3b and Definition* (1)*,* $\mathcal{C}_{\mathrm{C}}^{j}, \forall j \in \{1, 2, \ldots, N_c\}$*, can be re-defined as:*

$$\mathcal{C}_{\mathrm{C}}^{j} = \begin{cases} (x_{N_c}^{\mathrm{r}} \times u_{N_c}^{\mathrm{r}}) + (x_1^{\ell} \times u_1^{\ell}) & \textit{if } j = 1 \\ (x_{j-1}^{\mathrm{r}} \times u_{j-1}^{\mathrm{r}}) + (x_j^{\ell} \times u_j^{\ell}) & \textit{if } j \in \{2, \ldots, N_c\} \end{cases}$$

If it holds that

$$\prod_{g=1}^{N_c} \frac{u_g^{\mathrm{r}}}{u_g^{\ell}} < 1$$

then after updating the fractional assignments as follows:

$$x_i^{\mathrm{r}'} = x_i^{\mathrm{r}} + \frac{\epsilon}{u_i^\ell} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^\ell} \qquad \textit{and} \qquad x_i^{\ell'} = x_i^\ell - \frac{\epsilon}{u_i^\ell} \times \prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^\ell}$$

where $\prod_{g=1}^{i-1} \frac{u_g^{\mathrm{r}}}{u_g^\ell}$ *is assumed to be* 1 *for* $i = 1$ *and where* $\epsilon > 0$ *denotes a real number such that*

$$\epsilon = \min_{z \in [1,2,\ldots,N_c]} \left\{ \frac{x_z^{\mathrm{r}} \times u_z^\ell}{\prod_{g=1}^{z-1} \frac{u_g^{\mathrm{r}}}{u_g^\ell}} \right\}$$

the following properties are satisfied:

P1.. $\forall j \in \{1, 2, \ldots, N_c\} : \mathcal{C}_{\mathrm{C}}^{j'} \leq \mathcal{C}_{\mathrm{C}}^{j}$*, where* $\mathcal{C}_{\mathrm{C}}^{j'}$ *denotes the capacity used on shared processor type* j*, after updating the fractional assignments.*
P2.. $\forall i \in \{1, 2, \ldots, N_c\} : x_i^{\ell'} + x_i^{\mathrm{r}'} = x_i^\ell + x_i^{\mathrm{r}}$.
P3.. $\forall i \in \{1, 2, \ldots, N_c\} : x_i^{\ell'} \geq 0$ *and* $x_i^{\mathrm{r}'} \geq 0$.
P4.. $\exists i \in \{1, 2, \ldots, N_c\} : x_i^{\ell'} = 0$.

Thus, each circuit identified in the graph (for example, using DFS [Cormen et al. 2001]) can be broken using the procedure described above (i.e., either using Lemma 7.1 or Lemma 7.2). Observe that, while removing the circuits, zero or more fractional tasks may get integrally assigned to processor types but for all practical purposes, it is sufficient for us to know that, at the end of this step, (i) there are at most $t - 1$ fractional tasks (by Lemma 5.2) and (ii) there are no circuits in the graph anymore. In the final step, we integrally assign these (at most) $t - 1$ fractional tasks to processor types.

8. STEP 4 OF $\mathrm{LPG_{IM}}$: INTEGRALLY ASSIGNING THE FRACTIONAL TASKS

In this section, we describe how to assign the fractional tasks *integrally* to processor types. This fourth step takes as input the output of the previous step, i.e, *a graph* $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ *with no circuits* and with *at most* $t - 1$ fractional tasks, and works iteratively on this input graph. In each iteration y, our algorithm chooses *one* fractional task $\tau_i \in \mathrm{A}$ and assigns it integrally to one of the processor types in B. Then, it removes that fractional task node from A, deletes all the edges incident on τ_i and removes from B all the *non-shared* processor type nodes to which τ_i was fractionally assigned. This procedure of integrally assigning a task and then deleting a few nodes and edges is repeated until the graph becomes empty, which implies that all the fractional tasks have been integrally assigned to processor types.

We now introduce two additional sets of notations that we will use extensively in the rest of the section while describing the working of this fourth step and proving its correctness. The first set of notations can be seen as *"global" with respect to the input graph* G while the second set of notations can be seen as *"local" with respect to each task in the graph*.

Global notations w.r.t. the graph. Recall that, in this step, we use the circuit-free graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ output by the previous step. Since this graph contains only fractional tasks and fractional processor types (see Section 6), this step deals with only these tasks and processor types. For the purpose of this section, we re-index the fractional tasks in A and the fractional processor types in B as follows. In graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$, let τ_i denote the i'th task (node) in A and let π^j denote the j'th processor type (node) in B. Since this step works iteratively, let y denote the iteration counter. During this step, assigning a fractional task integrally to one of the processor types comes at the cost of additional computing capacity required on the processor type for

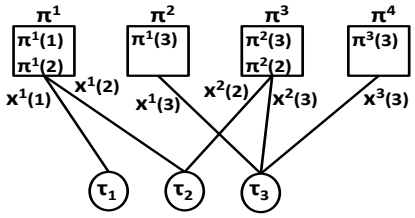


Fig. 4: The graph of Fig. 3a after breaking the circuit (as described in Section 7) and re-indexing the nodes.

accommodating this task *entirely*. We denote by $\mathcal{C}_{+}^{j}[\mathrm{y}]$ the *cumulative* extra capacity required on processor type $\pi^j \in \mathrm{B}$ from iteration 1 until the beginning of iteration y. Since some of the processor type nodes are deleted from the graph at the end of each iteration, let $\mathrm{P}^{\mathrm{in}}[\mathrm{y}]$ denote the set of processor type nodes that are still in the graph at the beginning of iteration y and let $\mathrm{P}^{\mathrm{out}}[\mathrm{y}]$ denote the set of all the processor types that have been removed from the graph from iteration 1 till the beginning of iteration y. It holds by definition that, $\mathrm{P}^{\mathrm{in}}[1] = \mathrm{B}$ and $\mathrm{P}^{\mathrm{out}}[1] = \phi$.

For example, in the previous section, let the circuit in the graph shown in Fig. 3a be broken by removing the edge e_3^5. In that case, the graph output by the previous step, after re-indexing the task and processor types, is shown in Fig. 4. In Fig. 4, the re-indexed task nodes τ_1, τ_2 and τ_3 denote the original task nodes τ_3, τ_4 and τ_6 of Fig. 3a, respectively. Analogously, the re-indexed processor type nodes π^1, π^2, π^3 and π^4 denote the original processor type nodes π^1, π^2, π^3 and π^5 of Fig. 3a, respectively.

Local notations w.r.t. a task in the graph. Since this fourth step of $\mathrm{LPG}_{\mathrm{IM}}$ considers one fractional task $\tau_i \in \mathrm{A}$ in each iteration and assigns it integrally to one of the processor types to which it is *fractionally* assigned, we also define some notations with respect to task τ_i. Let $\pi(i) = \left\{\pi^1(i), \pi^2(i), \ldots, \pi^{|\pi(i)|}(i)\right\}$ denote the set of processor types to which task $\tau_i \in \mathrm{A}$ is assigned in G, where $\forall j \in \{1, 2, \ldots, |\pi(i)|\}$, $\pi^j(i) \in \pi(i)$ denote the j'th processor type to which task τ_i is assigned. Let $X(i) = \left\{x^1(i), x^2(i), \ldots, x^{|\pi(i)|}(i)\right\}$ denote the set of fractional assignments of task $\tau_i \in \mathrm{A}$, where $\forall j \in \{1, 2, \ldots, |\pi(i)|\}$, $x^j(i) \in X(i)$ denotes the fractional assignment of task τ_i to *its* j'th processor type, i.e., its fractional assignment to $\pi^j(i)$. Let $\mathcal{C}_{+}^{j}(i)[\mathrm{y}]$ denote the cumulative extra capacity required on processor type $\pi^j(i)$ from iteration 1 to iteration y.

Note that these two sets of notations, i.e., global and local, can be used to refer to the same processor type node. For example, in Fig. 4, it can be seen that: $\pi(1) = \left\{\pi^1(1) = \pi^1\right\}$, $\pi(2) = \left\{\pi^1(2) = \pi^1,\ \pi^2(2) = \pi^3\right\}$ and $\pi(3) = \left\{\pi^1(3) = \pi^2,\ \pi^2(3) = \pi^3,\ \pi^3(3) = \pi^4\right\}$.

Finally, since G is formed using only fractional tasks and fractional processor types (see Section 6), observe that:

$$\forall \tau_i \in \mathrm{A} : \sum_{j=1}^{|\pi(i)|} x^j(i) = 1 \tag{23}$$

With these new notations, we now describe the working of this fourth step of $\mathrm{LPG}_{\mathrm{IM}}$ algorithm. The pseudo-code of this step is provided in Algorithm 1 and it can be summarized as follows. As long as there is a task node in the graph, Algorithm 1 chooses a task τ_i which is connected to *only* non-shared processor type nodes (line 3–4). If there is no such task then it chooses a task which is connected to *exactly* one shared processor type node (line 5–6) — we will prove in Lemma 8.2 that there always exists such a task. Then, Algorithm 1 *tries* to integrally assign the chosen task τ_i to one of its non-shared

ALGORITHM 1: Step 4 of $\mathrm{LPG_{IM}}$ algorithm for assigning the fractional tasks integrally to processor types.

Input : $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$: A graph output by Step 3 of $\mathrm{LPG_{IM}}$ representing task assignment with no circuits and at most $t-1$ fractional tasks

1 $\mathrm{y} \leftarrow 1$, $\mathrm{P^{in}[y]} \leftarrow \mathrm{B}$, $\mathrm{P^{out}[y]} \leftarrow \phi$;
2 **while** A *is not empty* **do**
3 **if** $\exists \tau_\ell \in \mathrm{A}$ *connected to only non-shared processor types* **then**
4 $\tau_i \leftarrow \tau_\ell$;
5 **else**
6 $\tau_i \leftarrow$ a task in A that is connected to exactly one shared processor type ;
7 **end**
8 **foreach** *non-shared processor type* $\pi^\ell(i) \in \pi(i)$ **do**
9 $\mathrm{newCap} \leftarrow \mathcal{C}^\ell_+(i)[\mathrm{y}] + \sum_{j=1, j\neq\ell}^{|\pi(i)|} x^j(i) \times u^j_i$;
10 **if** $\mathrm{newCap} \leq \alpha \times \frac{t-1}{t}$ **then**
11 assign τ_i to $\pi^\ell(i)$;
12 $\mathcal{C}^\ell_+(i)[\mathrm{y}] \leftarrow \mathrm{newCap}$;
13 break the foreach-loop ;
14 **end**
15 **end**
16 **if** τ_i *is not assigned* **then**
17 assign τ_i to the only shared processor type, say $\pi^z(i)$, to which it is connected ;
18 $\mathcal{C}^z_+(i)[\mathrm{y}] \leftarrow \mathcal{C}^z_+(i)[\mathrm{y}] + \sum_{j=1, j\neq z}^{|\pi(i)|} x^j(i) \times u^j_i$;
19 **end**
  // remove (i) the task τ_i from A and (ii) all the non-shared processor types that are connected to τ_i from B (and the edges connecting τ_i to these processor types
20 $\mathrm{y} \leftarrow \mathrm{y}+1$;
21 $\mathrm{A} \leftarrow \mathrm{A} \setminus \{\tau_\mathrm{i}\}$;
22 $\mathrm{delpt} \leftarrow \{\pi^\mathrm{k} \in \mathrm{B} \mid \exists x^k_i > 0$ and π^k is non-shared$\}$;
23 $\mathrm{B} \leftarrow \mathrm{B} \setminus \mathrm{delpt}$;
24 $\mathrm{P^{in}[y]} \leftarrow \mathrm{P^{in}[y]} \setminus \mathrm{delpt}$;
25 $\mathrm{P^{out}[y]} \leftarrow \mathrm{P^{out}[y]} \cup \mathrm{delpt}$;
26 $\mathrm{E} \leftarrow \mathrm{E} \setminus \{\mathrm{e^k_i} \mid \pi^k \in \pi(i)$ and π^k is non-shared$\}$;
27 **end**

processor types. We say that it *fails* to assign τ_i to a processor type $\pi^\ell(i) \in \pi(i)$ if the (cumulative) extra capacity required on $\pi^\ell(i)$, after assigning τ_i to it, exceeds $\alpha \times \frac{t-1}{t}$. If the extra capacity does not exceed that threshold on any one of the non-shared processor types then τ_i is integrally assigned to it (lines 8–15). Otherwise, τ_i is assigned to its (sole) shared processor type (lines 16–19), and we also show in Lemma 8.2 that this assignment cannot fail. Finally, the algorithm removes τ_i from the graph, as well as all its non-shared processor type nodes and all the edges connected to τ_i (lines 21–26), and iterates with another task until the graph becomes empty.

Now, we prove the approximation ratio of the intra-migrative algorithm, $\mathrm{LPG_{IM}}$, with the help of Property 2 and the intermediate Lemma 8.1.

PROPERTY 2. *It holds, from lines 21–26 of Algorithm 1, that at each iteration* y*:*

$$\mathrm{P^{in}[y]} \cup \mathrm{P^{out}[y]} = \mathrm{B} \quad \textit{and} \quad \mathrm{P^{in}[y]} \cap \mathrm{P^{out}[y]} = \emptyset \tag{24}$$

LEMMA 8.1. $\forall \tau_i \in \mathrm{A}$, $\exists \pi^j(i) \in \pi(i)$ *such that* $x^j(i) \geq \frac{1}{|\pi(i)|}$.

PROOF. The proof is by contradiction. If $\forall \pi^j(i) \in \pi(i)$, if $x^j(i) < \frac{1}{|\pi(i)|}$ then $\sum_{j=1}^{|\pi(i)|} x^j(i) < |\pi(i)| \times \frac{1}{|\pi(i)|} < 1$, which contradicts Eq. (23). Hence the proof. □

LEMMA 8.2. *Consider a task set τ which is intra-migrative feasible on a platform π. After running steps 1 to 3 of $\mathrm{LPG_{IM}}$, if the graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ (with no circuits and at most $t-1$ fractional tasks) that was output by step 3, is given as input to Algorithm 1 (step 4 of $\mathrm{LPG_{IM}}$) then Algorithm 1 succeeds to integrally assign the at most $t-1$ fractional tasks in A to the processor types in B and in order to succeed it only requires that each processor type in B are provided with an additional capacity of $\alpha \times \frac{t-1}{t}$.*

PROOF. The proof is split into three parts where we show:
Part 1. At lines 3–7, there always exists, at each iteration y, a task τ_i assigned to *at most* one shared processor type.
Part 2. At the beginning of the first iteration ($\mathrm{y} = 1$), it holds that $\sum_{\pi^j \in \mathrm{P^{in}}[1]} \mathcal{C}^j_+[1] \leq \alpha \times \frac{|\mathrm{P^{out}}[1]|}{t}$.
Part 3. At the beginning of each iteration $\mathrm{y} \geq 1$, if it holds that

$$\sum_{\pi^j \in \mathrm{P^{in}}[\mathrm{y}]} \mathcal{C}^j_+[\mathrm{y}] \leq \frac{|\mathrm{P^{out}}[\mathrm{y}]|}{t} \times \alpha \tag{25}$$

then the task τ_i chosen on line 4 (or line 6) can be assigned *integrally* to one of its non-shared processor types on lines 8–15 (or, to its (sole) shared processor type on lines 16–19). Then, after assigning τ_i integrally, Eq. (25) remains satisfied at the beginning of the next iteration $\mathrm{y}+1$.
Proof of Part 1. Here we show that, at each iteration y, there always exists a task τ_i assigned to *at most* one shared processor type. Since the input graph $\mathrm{G} = (\mathrm{A}, \mathrm{B}, \mathrm{E})$ does not contain any circuit, we know from Lemma 6.5 that, at the first iteration of Algorithm 1, there is a task $\tau_i \in \mathrm{A}$ which is assigned to *at most* one shared processor type. Then, at the end of each iteration $\mathrm{y} \geq 1$ one task is deleted from the graph (line 21) and all the non-shared processor types connected to that task are also deleted (lines 22–25). Since removing nodes and edges from the graph cannot create a new circuit, the graph will always be circuit-free in all the subsequent iterations of Algorithm 1. Hence, from Lemma 6.5, at every iteration $\mathrm{y} \geq 1$ there is always a task $\tau_i \in \mathrm{A}$ assigned to *at most* one shared processor type.
Proof of Part 2. Here we show that at the beginning of the first iteration ($\mathrm{y} = 1$), it holds that $\sum_{\pi^j \in \mathrm{P^{in}}[1]} \mathcal{C}^j_+[1] \leq \frac{|\mathrm{P^{out}}[1]|}{t} \times \alpha$. At the beginning of the first iteration, no fractional task in G has been integrally assigned to a processor type yet. Hence, the extra capacity needed on each processor type to accommodate the tasks of G that have been already assigned is trivially zero, i.e. $\mathcal{C}^j_+[1] = 0$, $\forall \pi^j \in \mathrm{B}$. Besides, we have $\mathrm{P^{out}}[1] = \phi$ and thus it holds that $\sum_{\pi^j \in \mathrm{P^{in}}[1]} \mathcal{C}^j_+[1] = 0 \leq \frac{|\mathrm{P^{out}}[1]|}{t} \times \alpha = 0$.
Proof of Part 3. Here we show that at each iteration y, as long as Eq. (25) holds (and we have shown above that it holds for $\mathrm{y} = 1$), the fractional task τ_i chosen at line 4 (or line 6) can be integrally assigned to one of the processor types connected to it. For this, we need to investigate three cases:
Case 3.1. Task τ_i is *not* assigned to a shared processor type (chosen on line 4). In this case, we need to show that τ_i can be integrally assigned to at least one of its non-shared processor type (on lines 8–15) and Eq. (25) holds true at the beginning of the next iteration $\mathrm{y}+1$.
Case 3.2. Task τ_i is assigned to exactly one shared processor types (chosen on line 6) and is successfully assigned integrally to (one of) its non-shared processor types on

line 8–15. In this case, we only have to show that Eq. (25) holds true at the beginning of the next iteration $\mathrm{y}+1$.

Case 3.3. Task τ_i is assigned to exactly one shared processor type (chosen on line 6) and fails to be assigned to any of its non-shared processor types. In this case, we need to show that Algorithm 1 succeeds in integrally assigning τ_i to its shared processor type on lines 16–19 and Eq. (25) holds true at the beginning of the next iteration $\mathrm{y}+1$.

In the three cases proven below, we assume that Eq. (25) holds true at the beginning of iteration y and then show that it also holds at the beginning of iteration $\mathrm{y}+1$.

Proof of Case 3.1. We prove this case by contradiction, i.e., we assume that Algorithm 1 tried to integrally assign the task τ_i to every non-shared processor type (to which τ_i is fractionally assigned) but failed to do so and then we show that it is impossible for this to happen. From the case, τ_i failed to be integrally assigned to its non-shared processor types, which means that for *every* processor type node $\pi^j(i) \in \pi(i)$, migrating all the fractional assignments of τ_i to $\pi^j(i)$ requires an extra capacity on that processor type j greater than $\alpha \times \frac{t-1}{t}$, i.e., the following $|\pi(i)|$ inequalities hold:

$$\forall \ell \in [1, |\pi(i)|] : \sum_{\substack{j=1 \\ j \neq \ell}}^{|\pi(i)|} \left(x^j(i) \times u_i^\ell\right) + \mathcal{C}_+^\ell(i)[\mathrm{y}] \;>\; \alpha \times \tfrac{t-1}{t}$$

$$\overset{\text{re-writing}}{\Leftrightarrow} \forall \ell \in [1, |\pi(i)|] : \sum_{\substack{j=1 \\ j \neq \ell}}^{|\pi(i)|} \left(x^j(i) \times u_i^\ell\right) \;>\; \alpha \times \tfrac{t-1}{t} - \mathcal{C}_+^\ell(i)[\mathrm{y}]$$

By summing these $|\pi(i)|$ inequalities, we get

$$\sum_{\ell=1}^{|\pi(i)|} \sum_{\substack{j=1 \\ j \neq \ell}}^{|\pi(i)|} \left(x^j(i) \times u_i^\ell\right) \;>\; \left(|\pi(i)| \times \alpha \times \frac{t-1}{t}\right) - \sum_{\ell=1}^{|\pi(i)|} \mathcal{C}_+^\ell[\mathrm{y}] \tag{26}$$

In the left-hand side of Eq. (26), each $x^j(i)$ appears $(|\pi(i)|-1)$ times and since $\forall \ell$, $u_i^\ell \leq \alpha$ (from Eq. (2)), for the left-hand side of Eq. (26), we have:

$$\sum_{\ell=1}^{|\pi(i)|} \sum_{\substack{j=1 \\ j \neq \ell}}^{|\pi(i)|} x^j(i) \times u_i^\ell \;\leq\; \alpha \times (|\pi(i)| - 1) \times \sum_{j=1}^{|\pi(i)|} x^j(i)$$

$$\overset{\text{from (23)}}{=} \alpha \times (|\pi(i)| - 1) \tag{27}$$

Regarding the right-hand side of Eq. (26), since we know that $\pi(i) \subseteq \mathrm{P}^{\mathrm{in}}[\mathrm{y}]$, we have

$$\sum_{\ell=1}^{|\pi(i)|} \mathcal{C}_+^\ell(i)[\mathrm{y}] \;\leq\; \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}]} \mathcal{C}_+^j[\mathrm{y}]$$

$$\overset{\text{from (25)}}{\leq} \alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t}$$

Therefore, for the RHS of Eq. (26), we have:

$$\left(|\pi(i)| \times \alpha \times \frac{t-1}{t}\right) - \left(\sum_{\ell=1}^{|\pi(i)|} \mathcal{C}_+^\ell(i)[\mathrm{y}]\right) \;\geq\; \left(|\pi(i)| \times \alpha \times \frac{t-1}{t}\right) - \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t}\right) \tag{28}$$

By combining Eq. (26), (27) and (28), we obtain:

$$\alpha \times (|\pi(i)| - 1) \;>\; \left(|\pi(i)| \times \alpha \times \frac{t-1}{t}\right) - \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t}\right) \tag{29}$$

Then, since $\pi(i) \subseteq \mathrm{P}^{\mathrm{in}}[\mathrm{y}]$, we have $\left|\mathrm{P}^{\mathrm{in}}[\mathrm{y}]\right| \geq |\pi(i)|$ and thus $|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]| \leq t - |\pi(i)|$. Using this, Eq. (29) is re-written as:

$$\begin{aligned} \alpha \times (|\pi(i)| - 1) \quad &> \left(|\pi(i)| \times \alpha \times \frac{t-1}{t}\right) - \left(\alpha \times \frac{t - |\pi(i)|}{t}\right) \\ \Leftrightarrow\ |\pi(i)| - \left(|\pi(i)| \times \tfrac{t-1}{t}\right) &> \frac{|\pi(i)|}{t} \quad \Leftrightarrow \quad \frac{1}{t} > \frac{1}{t} \end{aligned}$$

which is impossible and contradicts the assumption that τ_i could not be integrally assigned to any of its non-shared processor types and hence Algorithm 1 succeeds in doing so. This concludes Case 3.1.

Proof of Case 3.2. Task τ_i is assigned to exactly one shared processor type and is successfully assigned integrally on lines 8–15 to (one of) its non-shared processor types in $\pi(i)$. Here, we only need to show that Eq. (25) holds true at the beginning of the next iteration $\mathrm{y}+1$. The proof is somewhat similar to that of Case 3.1. Let us assume, without loss of generality, that $\pi^1(i)$ is the shared processor type in $\pi(i)$. After assigning τ_i to (one of) its *non-shared* processor type, we have,

$$| \mathrm{P}^{\mathrm{out}}[\mathrm{y}+1] | \ = \ | \mathrm{P}^{\mathrm{out}}[\mathrm{y}] | + |\pi(i)| - 1 \tag{30}$$

$$| \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] | \ = \ | \mathrm{P}^{\mathrm{in}}[\mathrm{y}] | - |\pi(i)| + 1 \tag{31}$$

The "-1" and "+1" is the shared processor type node $\pi^1(i)$ which is *not* removed from the graph. Hence, $\pi^1(i)$ remains in $\mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]$ and is not added to $\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]$. As explained in the proof of Case 3.1, since τ_i is integrally assigned to (one of) its non-shared processor type, say $\pi^\ell(i)$, and since $\pi^\ell(i) \notin \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]$ as $\pi^\ell(i)$ is removed from graph, we have

$$\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}^j_+[\mathrm{y}+1] = \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}^j_+[\mathrm{y}] \tag{32}$$

and since $\mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] \subset \mathrm{P}^{\mathrm{in}}[\mathrm{y}]$, we can rewrite Eq. (32) as:

$$\begin{aligned} \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}^j_+[\mathrm{y}+1] \quad &\leq \quad \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}]} \mathcal{C}^j_+[\mathrm{y}] \\ &\overset{\text{from (25)}}{\leq} \quad \alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t} \\ &\overset{\text{from (30)}}{=} \quad \alpha \times \left(\frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]|}{t} - \frac{|\pi(i)| - 1}{t}\right) \\ &< \quad \alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]|}{t} \text{ (since } |\pi(i)| \geq 2) \end{aligned}$$

This concludes Case 3.2.

Proof of Case 3.3. Task τ_i is assigned to exactly one shared processor type and fails to be integrally assigned on lines 8–15 to any of its non-shared processor types in $\pi(i)$. In this case, we need to show that Algorithm 1 succeeds in integrally assigning τ_i to its (sole) shared processor type on lines 16–19 and Eq. (25) holds true at the beginning of the next iteration $\mathrm{y}+1$. As in the previous case, let us assume, without loss of generality, that $\pi^1(i) \in \pi(i)$ is the shared processor type connected to τ_i. We prove by contradiction that the integral assignment of τ_i to $\pi^1(i)$ cannot fail, i.e. by contradiction, we assume that it does fail and then show that it is impossible for this to happen.

From the case, task τ_i also failed to be assigned to *all* its non-shared processor types $\pi^j(i) \in \pi(i) \ \wedge \ j \neq 1$ (on lines 8–15), which means that for *every* processor type node $\pi^j(i) \in \pi(i)$, migrating all the fractional assignments of τ_i to that node $\pi^j(i)$ requires

an extra capacity on that processor type j that exceeds $\alpha \times \frac{t-1}{t}$. This scenario is same as Case 3.1 and thus it leads to contradiction. Hence, the assumption that Algorithm 1 fails to integrally assign τ_i to its only shared processor type $\pi^1(i)$ is not true and therefore, Algorithm 1 must succeed in doing so.

Now, we assume that τ_i is integrally assigned to $\pi^1(i)$ and show that Eq. (25) still holds at the beginning of the iteration $\mathrm{y}+1$. Assigning τ_i integrally to $\pi^1(i)$ gives

$$\mathcal{C}^1_+(i)[\mathrm{y}+1] = \mathcal{C}^1_+(i)[\mathrm{y}] + \sum_{j=2}^{|\pi(i)|} \left(x^j(i) \times u^1_i\right) \tag{33}$$

As explained earlier, since the algorithm failed to assign τ_i integrally to each of the *non-shared* processor types, it holds $\forall \pi^\ell(i) \in \pi(i)$, $\pi^\ell(i) \neq \pi^1(i)$ that,

$$\sum_{\substack{j=1 \\ j\neq\ell}}^{|\pi(i)|} \left(x^j(i) \times u^\ell_i\right) \quad > \quad \left(\alpha \times \frac{t-1}{t}\right) - \mathcal{C}^\ell_+(i)[\mathrm{y}] \tag{34}$$

Since we know from Eq. (23) that, $\sum_{j=1}^{|\pi(i)|} x^j(i) = 1$, we have $\forall \ell \in [1, |\pi(i)|]$: $\sum_{\substack{j=1 \\ j\neq\ell}}^{|\pi(i)|} x^j(i) = 1 - x^\ell(i)$ and Eq. (34) can be re-written as: $\forall \pi^\ell(i) \in \pi(i)$, $\pi^\ell(i) \neq \pi^1(i)$, it holds that:

$$\begin{aligned}
& (1 - x^\ell(i)) \times u^\ell_i && > \left(\alpha \times \frac{t-1}{t}\right) - \mathcal{C}^\ell_+(i)[\mathrm{y}] \\
\overset{\text{from (2)}}{\Leftrightarrow} \quad & \alpha \times \left(1 - x^\ell(i)\right) && > \left(\alpha \times \frac{t-1}{t}\right) - \mathcal{C}^\ell_+(i)[\mathrm{y}] \\
\overset{\text{re-writing}}{\Leftrightarrow} \quad & \alpha \times x^\ell(i) && < \alpha - \left(\alpha \times \frac{t-1}{t}\right) + \mathcal{C}^\ell_+(i)[\mathrm{y}] \\
\overset{\text{re-writing}}{\Leftrightarrow} \quad & x^\ell(i) && < \frac{\frac{\alpha}{t} + \mathcal{C}^\ell_+(i)[\mathrm{y}]}{\alpha}
\end{aligned} \tag{35}$$

By using Eq. (35) in Eq. (33), we get

$$\begin{aligned}
\mathcal{C}^1_+(i)[\mathrm{y}+1] \quad & \leq \quad \mathcal{C}^1_+(i)[\mathrm{y}] + \sum_{j=2}^{|\pi(i)|} \left(\frac{\frac{\alpha}{t} + \mathcal{C}^\ell_+(i)[\mathrm{y}]}{\alpha}\right) \times u^1_i \\
& \overset{\text{from (2)}}{\leq} \quad \mathcal{C}^1_+(i)[\mathrm{y}] + \sum_{j=2}^{|\pi(i)|} \left(\frac{\alpha}{t} + \mathcal{C}^\ell_+(i)[\mathrm{y}]\right) \\
& \leq \quad \mathcal{C}^1_+(i)[\mathrm{y}] + \left(\alpha \times \frac{|\pi(i)|-1}{t}\right) + \sum_{j=2}^{|\pi(i)|} \mathcal{C}^\ell_+(i)[\mathrm{y}]
\end{aligned} \tag{36}$$

Now, let us focus on the term $\sum_{j=2}^{|\pi(i)|} \mathcal{C}^j_+(i)[\mathrm{y}]$ from the right-hand side of the above inequality. Since we know that:

$\pi(i) \setminus \{\pi^1(i)\} = \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus (\mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)) \setminus \pi^1(i)$, we can write:

$$\begin{aligned}\sum_{j=2}^{|\pi(i)|} \mathcal{C}_+^j(i)[\mathrm{y}] &= \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}]} \mathcal{C}_+^j[\mathrm{y}] - \left(\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}] \right) - \mathcal{C}_+^1(i)[\mathrm{y}] \\ &\overset{\text{from (25)}}{\leq} \alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t} - \left(\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j(i)[\mathrm{y}] \right) - \mathcal{C}_+^1(i)[\mathrm{y}] \qquad (37)\end{aligned}$$

By using Inequalities (36) and (37) together, we get

$$\begin{aligned}\mathcal{C}_+^1(i)[\mathrm{y}+1] &\leq \mathcal{C}_+^1(i)[\mathrm{y}] + \left(\alpha \times \frac{|\pi(i)|-1}{t}\right) + \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t}\right) - \left(\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}] \right) - \mathcal{C}_+^1(i)[\mathrm{y}] \\ &\leq \left(\alpha \times \frac{|\pi(i)|-1}{t}\right) + \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}]|}{t}\right) - \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}]\end{aligned}$$

Here, we can re-use Eq. (30) since all the processor type nodes connected to τ_i, except $\pi^1(i)$, are deleted from the graph on line 21 (this case is similar to Case 3.2 in that regard). So, the above equation can be re-written as:

$$\begin{aligned}\mathcal{C}_+^1(i)[\mathrm{y}+1] &\leq \left(\alpha \times \frac{|\pi(i)|-1}{t}\right) + \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]| - (|\pi(i)|-1)}{t}\right) - \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}] \\ &\leq \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]|}{t}\right) - \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}] \qquad (38)\end{aligned}$$

Now, let us look at the term $\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}_+^j[\mathrm{y}+1]$:

$$\begin{aligned}\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}_+^j[\mathrm{y}+1] &= \left(\sum_{\substack{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] \\ \pi^j \neq \pi^1(i)}} \mathcal{C}_+^j[\mathrm{y}+1] \right) + \mathcal{C}_+^1(i)[\mathrm{y}+1] \\ &\overset{\text{from (38)}}{\leq} \left(\sum_{\substack{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] \\ \pi^j \neq \pi^1(i)}} \mathcal{C}_+^j[\mathrm{y}+1] \right) + \left(\alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]|}{t}\right) - \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}] \qquad (39)\end{aligned}$$

From the case, we have $\mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] = \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i) \cup \{\pi^1(i)\}$, and thus $\mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] \setminus \{\pi^1(i)\} = \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)$. Hence,

$$\sum_{\substack{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1] \\ \pi^j \neq \pi^1(i)}} \mathcal{C}_+^j[\mathrm{y}+1] = \sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}] \setminus \pi(i)} \mathcal{C}_+^j[\mathrm{y}]$$

Using this on Eq. (39) leads to:

$$\sum_{\pi^j \in \mathrm{P}^{\mathrm{in}}[\mathrm{y}+1]} \mathcal{C}_+^j[\mathrm{y}+1] \leq \alpha \times \frac{|\mathrm{P}^{\mathrm{out}}[\mathrm{y}+1]|}{t}$$

This concludes Case 3.3.

Hence the proof. □

COROLLARY 8.3. *If there exists a feasible intra-migrative assignment of τ on π then* $\mathrm{LPG_{IM}}$ *succeeds in finding such a feasible intra-migrative assignment of τ but on a platform π' in which only one processor of each type is $1 + \alpha \times \frac{t-1}{t}$ times faster.*

PROOF. This follows from Lemma 8.2. From Lemma 8.2, we have, if there exists a feasible intra-migrative assignment of τ on π then $\mathrm{LPG_{IM}}$ succeeds in finding such a feasible intra-migrative assignment of τ but on a platform π'' in which each *fractional processor type* (i.e., processor type in the graph to which a fractional task is assigned after step 3 of $\mathrm{LPG_{IM}}$) has an additional capacity $\alpha \times \frac{t-1}{t}$ than the corresponding processor type in π. Also, for those processor types that are not in the graph, $\mathrm{LPG_{IM}}$ does not require any additional capacity on those processors. However, increasing the capacity of those processors does not affect the performance guarantee (i.e., Lemma 8.2) of $\mathrm{LPG_{IM}}$. Further, since there was no restriction was placed by step 4 of $\mathrm{LPG_{IM}}$ algorithm on how to distribute this additional required capacity among the processors of each type, adding the entire $\alpha \times \frac{t-1}{t}$ capacity to *only one processor of each type* satisfies Lemma 8.2. Hence the proof. □

THEOREM 8.4 (**APPROXIMATION RATIO OF** $\mathrm{LPG_{IM}}$). *If there exists a feasible intra-migrative assignment of τ on π then* $\mathrm{LPG_{IM}}$ *succeeds in finding such a feasible intra-migrative assignment of τ but on $\pi^{\left(1+\alpha\times\frac{t-1}{t}\right)}$ in which every processor is $1+\alpha\times\frac{t-1}{t}$ times faster than the corresponding processor in π.*

PROOF. This trivially follows from Corollary 8.3. □

9. THE NON-MIGRATIVE ALGORITHM $\mathrm{LPG_{NM}}$

We now present a *non-migrative* algorithm, $\mathrm{LPG_{NM}}$, an enhanced version of $\mathrm{LPG_{IM}}$, for assigning tasks to *individual* processors on a t-type platform. We also evaluate its performance against (a more powerful) intra-migrative adversary. The non-migrative algorithm, $\mathrm{LPG_{NM}}$, works as follows.

Step 1. Assign tasks in τ to processor types in π' using $\mathrm{LPG_{IM}}$ algorithm; in π', *only one processor of each type* is $1 + \alpha \times \frac{t-1}{t}$ times faster compared to π. Recall that $\mathrm{LPG_{IM}}$ assigns tasks to processor types and not to processors.

Step 2. Assign the tasks, that are assigned to type-k processors, to individual processors of type-k ($\forall k \in \{1, 2, \ldots, t\}$), using next-fit but allowing *splitting* of tasks between consecutive processors. Such an assignment ensures that [Levin et al. 2010]: at most $m_k - 1$ tasks are *split* between processors of type-k with at most one task split between each pair of consecutive processors.

Step 3. Copy this assignment onto a faster platform $\pi^{(1+\alpha)}$ in which every processor is $1 + \alpha$ times faster than π.

Step 4. On platform $\pi^{(1+\alpha)}$, $\forall k \in \{1, 2, \ldots, t\}$, assign a task split between consecutive processors, say p and $p + 1$, of type-k, to processor p, where $p_1 \leq p < p_{m_k}$.

With this description of $\mathrm{LPG_{NM}}$ algorithm, we now derive its approximation ratio.

THEOREM 9.1 (**APPROXIMATION RATIO OF** $\mathrm{LPG_{NM}}$**.**). *If there exists a feasible intra-migrative task assignment of τ on π then* $\mathrm{LPG_{NM}}$ *succeeds in finding a feasible non-migrative task assignment of τ on $\pi^{(1+\alpha)}$.*

PROOF. We know from Corollary 8.3 that, if τ is intra-migrative feasible on π then $\mathrm{LPG_{IM}}$ algorithm outputs a feasible intra-migrative assignment of τ on π', in which *only one processor of each type* is $1 + \alpha \times \frac{t-1}{t}$ times faster and the remaining processors are of the same speed as the corresponding processors in π. Let p_{m_k} denote the processor of type-k ($\forall k \in \{1, 2, \ldots, t\}$) whose speed is $1 + \alpha \times \frac{t-1}{t}$ times faster. So, in platform

π', before assigning any tasks, it holds by definition that, $\forall k \in \{1, 2, \ldots, t\}$ of π':

$$\forall p \in \text{ type-k } \wedge p \neq p_{m_k} : \mathcal{FC}[p] = 1 \quad \text{and} \tag{40}$$

$$p \in \text{ type-k } \wedge p = p_{m_k} : \mathcal{FC}[p] = 1 + \alpha \times \frac{t-1}{t} \tag{41}$$

where $\mathcal{FC}[p]$ denotes the current available/free capacity on processor p. Since τ is intra-migrative feasible on π, after Step 1 of $\mathrm{LPG_{NM}}$, it holds (Corollary 8.3) that, $\forall k \in \{1, 2, \ldots, t\}$ of π':

$$\sum_{\tau_i \in \tau^k} u_i^k \;\; \leq \;\; m_k + \left(\alpha \times \frac{t-1}{t}\right) \tag{42}$$

where τ^k denotes the set of tasks assigned to type-k processors (i.e., to processor types and not to individual processors). We also know from Eq. (2) and (3) that:

$$\forall k \in \{1, 2, \ldots, t\} : \tau_i \in \tau^k : u_i^k \leq \alpha \tag{43}$$

In Step 2, $\mathrm{LPG_{NM}}$ assigns tasks to individual processors using "wrap-around" technique, which allows splitting of tasks between processors of same type. Combining such an assignment with Eq. (40)–(42), it holds that,
$\forall k \in \{1, 2, \ldots, t\}$ of π':

$$\forall p \in \text{ type-k } \wedge p \neq p_{m_k} : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^k \leq 1 \tag{44}$$

$$p \in \text{ type-k } \wedge p = p_{m_k} : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^k \;\; \leq \;\; 1 + \left(\alpha \times \frac{t-1}{t}\right) \tag{45}$$

$$\forall p \in \text{ type-k } : \mathcal{FC}[p] \geq 0 \quad \text{and} \tag{46}$$

at most $m_k - 1$ tasks are fractionally assigned between type-k processors with each task split between consecutive processors (47)

where $\tau[p]$ and $\mathcal{UC}[p]$ denote the set of tasks assigned on processor p and the capacity currently used on processor p, respectively.

On step 3, $\mathrm{LPG_{NM}}$ copies this assignment onto the faster platform $\pi^{(1+\alpha)}$. In platform $\pi^{(1+\alpha)}$, before assigning any tasks, it holds by definition that, $\forall k \in \{1, 2, \ldots, t\}$ of $\pi^{(1+\alpha)}$:

$$\forall p \in \text{ type-k } : \mathcal{FC}[p] = 1 + \alpha \tag{48}$$

From Eq. (44)–(48) and since the assignment is "copied" on $\pi^{(1+\alpha)}$, we have, $\forall k \in \{1, 2, \ldots, t\}$ of $\pi^{(1+\alpha)}$:

$$\forall p \in \text{ type-k } \wedge p \neq p_{m_k} : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^k \leq 1 \tag{49}$$

$$p \in \text{ type-k } \wedge p = p_{m_k} : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^k \;\; \leq \;\; 1 + \left(\alpha \times \frac{t-1}{t}\right) \tag{50}$$

$$\forall p \in \text{ type-k } \wedge p \neq p_{m_k} : \mathcal{FC}[p] \geq \alpha \tag{51}$$

$$p \in \text{ type-k } \wedge p = p_{m_k} : \mathcal{FC}[p] \geq \alpha/t \tag{52}$$

at most $m_k - 1$ tasks are fractionally assigned between type-k processors with each task split between consecutive processors (53)

From Eq. (51), (53) and (43), it can be seen that, each of the at most $m_k - 1$ fractional tasks can be integrally assigned to each of the $m_k - 1$ processors of type-k (i.e., $\forall p \in$

type-k $\wedge p \neq p_{m_k}$) in platform $\pi^{(1+\alpha)}$ in their respective free capacities. Combining this with Eq. (50) yields: $\forall k \in \{1, 2, \ldots, t\}$ of $\pi^{(1+\alpha)}$:

$$\forall p \in \text{ type-k } : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^k \leq 1 + \alpha \tag{54}$$

Observe that u_i^k is the utilization of a task τ_i on a processor of type-k on platform π. Let $u_i^{k'}$ denote the utilization of task τ_i on a processor of type-k on platform $\pi^{(1+\alpha)}$. Then it holds (by definition of these platforms) that: $\forall \tau_i \in \tau : \frac{u_i^{k'}}{u_i^k} = \frac{1}{1+\alpha}$. Applying this on Eq. (54) yields: $\forall k \in \{1, 2, \ldots, t\}$ of $\pi^{(1+\alpha)}$:

$$\forall p \in \text{ type-k } : \mathcal{UC}[p] = \sum_{\tau_i \in \tau[p]} u_i^{k'} \leq 1 \tag{55}$$

Since Eq. (55) 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 $\mathrm{LPG_{NM}}$ is feasible. □

10. CONCLUSIONS

The heterogeneous multiprocessor model is more generic than identical or uniform multiprocessor model, in terms of the systems that it can accommodate. Hence, it is interesting to study heterogeneous multiprocessors since a solution designed for such systems can also be applied to identical and uniform multiprocessors. Further, heterogeneous multiprocessors comprising a constant number of distinct types of processors, are increasingly becoming relevant [Apple Inc. 2013; AMD Inc. 2013; Intel Corp. 2013a; 2013b; 2013c; Nvidia Inc. 2013; Qualcomm Inc 2013; Samsung Inc. 2013; Texas Instruments 2013; Alben 2013]. Generally, this called for designing algorithms for such multiprocessors with provably good performance.

In this work, we considered the problem of finding a feasible assignment of implicit-deadline sporadic tasks on t-type heterogeneous multiprocessors. For this problem, we proposed two algorithms, $\mathrm{LPG_{IM}}$ and $\mathrm{LPG_{NM}}$, and showed that they provide the following guarantee. For a given t-type platform and a task set, if there exists a feasible *intra-migrative* task assignment then (i) $\mathrm{LPG_{IM}}$ succeeds in finding such a feasible *intra-migrative* assignment but given a platform in which *only one processor of each type* is $1 + \alpha \times \frac{t-1}{t}$ times faster and (ii) $\mathrm{LPG_{NM}}$ succeeds in finding a feasible *non-migrative* task assignment but given a platform in which *every processor* is $1 + \alpha$ times faster, where α is a property of the task set; it is the maximum of all the task utilizations that are no greater than one. To the best of our knowledge, for t-type platforms, (i) for the problem of intra-migrative task assignment, no previous algorithm with a proven approximation ratio exists and hence, our algorithm, $\mathrm{LPG_{IM}}$, is the first of its kind and (ii) for the problem of non-migrative task assignment, our algorithm, $\mathrm{LPG_{NM}}$, outperforms the state-of-the-art.

REFERENCES

Jonah Alben. 2013. NVIDIA Brings Kepler, World's Most Advanced Graphics Architecture, to Mobile Devices. http://blogs.nvidia.com/blog/2013/07/24/kepler-to-mobile/. (2013).

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

James Anderson and Anand Srinivasan. 2000. Early-release fair scheduling. In *Proceedings of the* 12^{th} *Euromicro conference on Real-time systems*.

Apple Inc. 2013. Apple A5X: Dual-core CPU and Quad-core GPU. http://www.apple.com/ipad/specs/. (2013).

Sanjoy Baruah. 2004a. Partitioning Real-Time Tasks Among Heterogeneous Multiprocessors. In 33^{rd} *International Conference on Parallel Processing*.

Sanjoy Baruah. 2004b. Task partitioning upon heterogeneous multiprocessor platforms. In *Proceedings of the 10th IEEE International Real-Time and Embedded Technology and Applications Symposium*. 536–543.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms, 2nd Ed.* McGraw-Hill.

José Correa, Martin Skutella, and José Verschae. 2012. The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. *Math. Oper. Res.* 37, 2 (May 2012), 379–398.

Michael Dertouzos. 1974. Control Robotics: The Procedural Control of Physical Processes. In *Proceedings of IFIP Congress (IFIP'74)*.

Michael Garey and David Johnson. 1979. *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman & Co.

W. Horn. 1974. Some simple scheduling algorithms. *Naval Research Logistics Quarterly* 21, 1 (1974), 177–185.

Ellis Horowitz and Sartaj Sahni. 1976. Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *Journal of the ACM* 23 (April 1976), 317–327. Issue 2.

IBM. 2013. CPLEX Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/. (2013).

Intel Corp. 2013a. Bay Trail: Multicore SoC Family for Mobile Devices. http://www.intel.com/newsroom/kits/idf/2013_fall/pdfs/bay_trail_fact_sheet.pdf. (2013).

Intel Corp. 2013b. Intel Atom Processor. http://www.intel.com/atom. (2013).

Intel Corp. 2013c. The 4th Generation Core i7 Processors. http://ark.intel.com/products/family/75023. (2013).

Klaus Jansen and Lorant Porkolab. 1999. Improved approximation schemes for scheduling unrelated parallel machines. In *Proceedings of the* 31^{st} *annual ACM symposium on Theory of computing*. 408–417.

David Johnson. 1973. *Near-optimal Bin Packing Algorithm*. Ph.D. Dissertation. Department of Mathematics, MIT, USA.

Narendra Karmakar. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* 4, 4 (1984), 373–395.

Jan Lenstra, David Shmoys, and Eva Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Math. Program.* 46 (1990), 259–271. Issue 3.

Greg Levin, Shelby Funk, Caitlin Sadowskin, Ian Pye, and Scott Brandt. 2010. DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In *Proceedings of the* 22^{nd} *Euromicro Conference on Real-Time Systems*. 3–13.

Chang L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM* 20 (1973), 46–61.

David Luenberger and Yinyu Ye. 2008. *Linear and Nonlinear Programming* (3rd ed.). International Series in Operations Research Management Science.

Geoffrey Nelissen, Vandy Berten, Vincent Nélis, Joël Goossens, and Milojevic Milojevic. 2012. U-EDF: An Unfair But Optimal Multiprocessor Scheduling Algorithm for Sporadic Tasks. In *24th Euromicro Conference on Real-Time Systems*. 13–23.

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

Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. 1997. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing*. 140–149.

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

Gurulingesh Raravi, Björn Andersson, and Konstantinos Bletsas. 2013. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. *Real-Time Systems* 49, 1 (2013), 29–72.

Gurulingesh Raravi, Björn Andersson, Konstantinos Bletsas, and Vincent Nélis. 2012. Task Assignment Algorithms for Two-Type Heterogeneous Multiprocessors. In *24th Euromicro Conference on Real-Time Systems*. 34–43.

Gurulingesh Raravi and Vincent Nélis. 2012. A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium*. 117–126.

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

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

Douglas B. West. 2000. *Introduction to Graph Theory* (2nd ed.). Prentice Hall.

Andreas Wiese, Vincenzo Bonifaci, and Sanjoy Baruah. 2013. Partitioned EDF scheduling on a few types of unrelated multiprocessors. *Real-Time Systems* 49, 2 (2013), 219–238.