Technical Report

Task Assignment Algorithms for Two-type Heterogeneous Multiprocessors

Gurulingesh Raravi

Björn Andersson

Konstantinos Bletsas

Vincent Nelis

HURRAY-TR-111202

Version:

Date: 3/12/2012

Task Assignment Algorithms for Two-type Heterogeneous Multiprocessors

Gurulingesh Raravi, Björn Andersson, Konstantinos Bletsas, Vincent Nelis

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: ghri@isep.ipp.pt, baa@isep.ipp.pt, ksbs@isep.ipp.pt, nelis@isep.ipp.pt

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

Abstract

Consider the problem of assigning real-time tasks on a heterogeneous multiprocessor platform comprising two different types of processors — such a platform is referred to as two-type platform. We present two linearithmic time-complexity algorithms, SA and SA-P, each providing the follow- ing guarantee. For a given two-type platform and a given task set, if there exists a feasible task-to-processor-type assignment such that tasks can be scheduled to meet deadlines by allowing them to migrate only between processors of the same type, then (i) using SA, it is guaranteed to find such a feasible task-to- processor-type assignment where the same restriction on task migration applies but given a platform in which processors are $1+\alpha/2$ times faster and (ii) SA-P succeeds in finding 2 a feasible task-to-processor assignment where tasks are not allowed to migrate between processors but given a platform in which processors are $1+\alpha$ times faster, where $0<\alpha\leq 1$. The parameter α is a property of the task set — it is the maximum utilization of any task which is less than or equal to 1.

Task Assignment Algorithms for Two-type Heterogeneous Multiprocessors

Gurulingesh Raravi*, Björn Andersson†, Konstantinos Bletsas* and Vincent Nélis*
*CISTER-ISEP Research Center, Polytechnic Institute of Porto, Portugal
†Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA
Email: *{ghri, ksbs, nelis}@isep.ipp.pt, †baandersson@sei.cmu.edu

***Abstract*—Consider the problem of assigning implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising two different types of processors — such a platform is referred to as two-type platform. We present two linearithmic time-complexity algorithms, SA and SA-P, each providing the following guarantee. For a given two-type platform and a given task set, if there exists a feasible task-to-processor-type assignment such that tasks can be scheduled to meet deadlines by allowing them to migrate only between processors of the same type, then (i) using SA, it is guaranteed to find such a feasible task-to-processor-type assignment where the same restriction on task migration applies but given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster and (ii) SA-P succeeds in finding a feasible task-to-processor assignment where tasks are not allowed to migrate between processors but given a platform in which processors are $1+\alpha$ times faster, where $0 < \alpha \leq 1$. The parameter α is a property of the task set — it is the maximum utilization of any task which is less than or equal to 1.**

I. Introduction

This paper addresses the problem of assigning a set of implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform comprising processors of two unrelated types: type-1 and type-2. We refer to such a computing platform as *two-type platform*. Our interest in considering such a platform model is motivated by the fact that many chip makers offer chips having two types of processors, both for desktops and embedded devices [1]–[6]. For scheduling tasks on such platforms, we consider three models for migration: *non-migrative*, *intra-migrative* and *fully-migrative*.

In the *non-migrative* model (also referred to as *fully partitioned* model in the literature), every task is statically assigned to a processor at design time and all its jobs must execute only on that processor. The challenge is to find, at design time, a task-to-processor assignment such that, at run time, the given scheduling algorithm meets all the deadlines while scheduling the tasks on their assigned processor. Scheduling the tasks to meet deadlines is a well-understood problem in the non-migrative model. One may use EDF [7] on each processor for example. EDF is an *optimal* scheduling algorithm on uniprocessor systems [7], [8], with the interpretation that it always finds a schedule in which all the deadlines are met, if such a schedule exists. Therefore, assuming that an optimal scheduling algorithm is used on every processor, the challenging part is to find a task-to-processor assignment for which *there exists* a schedule that meets all deadlines — such an assignment is said to be a *feasible* task assignment hereafter. Even in the simpler case of identical multiprocessors, finding a feasible task-to-processor assignment is strongly NP-Complete [9]. In this work, we propose an algorithm, SA-P, for this problem which outperforms state-of-the-art.

In the *intra-migrative* model, every task is statically assigned to a processor type at design 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, scheduling tasks to meet deadlines under the intra-migrative model is well-understood, e.g., one may use an optimal scheduling algorithm (e.g., [10]–[12]) that is designed for identical multiprocessors. Once again, assuming that an optimal algorithm is used for scheduling tasks on processors of each type, the challenging part is to find a *feasible* task-to-processor-type assignment for which there exists a schedule that meets all the deadlines. Even the simpler instance of this problem in which tasks must be assigned to *two* identical processors is NP-Complete [9]. In this work, we propose an algorithm, SA, for this problem (for which no previous algorithm 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, 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 register formats, instruction sets, etc. Hence, this model is not discussed in this work.

Related work. The scheduling problem on heterogeneous multiprocessors has been studied in the past [13]–[19]. The problem considered in [13] was to minimize the *makespan*, i.e., the duration of the schedule, for non-preemptive scheduling of a collection of jobs on heterogeneous multiprocessors. For this problem, [13] proposed an algorithm with an *approximation ratio*[1] of 2. It is well-known that this problem is equivalent to the problem of preemptive scheduling of implicit-deadline sporadic tasks on heterogeneous multiprocessors using EDF on each processor. For this problem, [14], [15] proposed algorithms with

[1]An algorithm with an approximation ratio of A for an optimization problem is an algorithm that, for all instances of the problem, produces a solution whose value is within a factor of A from the optimal value.

Computing Platform	Adversary	Task Assignment Algorithms			
	Task migration model	Algorithm	Task migration model	Time-complexity	Approximation ratio
t-type[a]	non-migrative	[13]	non-migrative	$O(P)$[c]	2
t-type	non-migrative	[14]	non-migrative	$O(P \cdot 2^m)$	2
t-type	non-migrative	[15]	non-migrative	$O(P)$	2
2-type[b]	non-migrative	[16]	non-migrative	$O(n \cdot \max(\log n, m))$	2
t-type	fully-migrative	[17]	non-migrative	$O(P)$	4
2-type	intra-migrative	SA	intra-migrative	$O(n \log n)$	$1 + \frac{\alpha}{2} \leq 1.5$
2-type	intra-migrative	SA-P	non-migrative	$O(n \log n)$	$1 + \alpha \leq 2$

[a] A heterogeneous multiprocessor platform having two or more processor types.
[b] A heterogeneous multiprocessor platform having only two processor types.
[c] The time-complexity $O(P)$ indicates that the algorithm relies on solving a Linear Program (LP) formulation — note that though a linear program can be solved in polynomial time, the polynomial generally has a higher degree.

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

an approximation ratio of 2. All these approaches [13]–[15] focused on generic heterogeneous multiprocessor platforms, i.e., platforms having two or more processor types. Due to practical relevance, [16] considered the problem of non-migrative scheduling of tasks on two-type platforms and proposed an algorithm, FF-3C, based on first-fit heuristic and couple of variants of this algorithm. These had the same worst-case performance guarantee as [13]–[15] (i.e., requiring processors twice as fast) but can be implemented more efficiently. Also, in simulations, for randomly generated task sets, these algorithms required far smaller processor speedups than their theoretical worst-case estimate and also performed better than [14], [15]. In [17], it is shown that if a task set can be scheduled by an optimal algorithm on a heterogeneous platform with full migrations then an optimal algorithm for scheduling tasks on heterogeneous platform with no migrations needs processors four times as fast. [19] proved that if a task set in which "no utilization can exceed one" can be scheduled to meet deadlines on a heterogeneous platform with full migrations then the algorithm in [14] also succeeds in scheduling the task set on a platform with no migrations in which processors are only twice as fast.

Contributions and significance of this work. We present a linearithmic time-complexity ($O(n \log n)$) task assignment algorithm, called SA, which offers the following guarantee. Consider a two-type platform π and a task set τ in which the utilization of every task (on both processor types) is either no greater than α or is greater than 1, where $0 < \alpha \leq 1$. If there exists a feasible intra-migrative assignment of τ on π (i.e., task-to-processor-type assignment) then using SA it is guaranteed to find such a feasible intra-migrative assignment of τ on $\pi^{(1+\frac{\alpha}{2})}$, where $\pi^{(1+\frac{\alpha}{2})}$ is a two-type platform in which processors are $1+\frac{\alpha}{2}$ times faster than the ones in π. Then, we modify SA to obtain SA-P and show that: if there exists a feasible *intra-migrative* assignment of τ on π then SA-P succeeds in finding a feasible *non-migrative* assignment of τ on $\pi^{(1+\alpha)}$ (i.e., task-to-processor assignment). The state-of-the-art and the contributions of this paper are summarized in Table I.

We believe the significance of this work is two-fold. First, for the problem of intra-migrative task assignment, no previous algorithm exists and hence our algorithm, SA, is the first for this problem[2]. Second, for the problem of non-migrative task assignment, our algorithm, SA-P, has superior performance compared to state-of-the-art. This can be seen from Table I since SA-P has (i) the same approximation ratio as algorithms in [13]–[16] but with a stronger adversary and also a better time-complexity and (ii) among the algorithms with approximation ratio proven against an adversary with a migration model of intra-migration or greater power, SA-P offers the best approximation ratio[3].

The rest of the paper is organized as follows. Section II briefs the system model. Section III presents an optimal *intra-migrative* task assignment algorithm, ILP-Algo, that uses Integer Linear Programming (ILP) formulation. Since ILP is NP-Complete, Section IV presents LP-Algo that relaxes the ILP formulation to LP and derives its approximation ratio. As solving an LP formulation is often time consuming, Section V presents the algorithm SA of time-complexity $O(n \log n)$ that does not rely on solving LP formulation but has the same approximation ratio as LP-Algo, which is shown in Section VI. Section VII extends SA to obtain a *non-migrative* task assignment algorithm, namely SA-P, of time-complexity $O(n \log n)$ and proves its approximation ratio. Section VIII concludes.

II. System model

We consider the problem of scheduling a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n implicit-deadline sporadic tasks on a two-type heterogeneous multiprocessor platform comprising m processors, of which m_1 are of type-1 and m_2 are of type-2. Each task τ_i is characterized by two parameters: a *worst-case execution time* (WCET) and a *period* T_i. Each task τ_i releases a (potentially infinite) sequence of *jobs*, with the first job released at any time during the system execution and subsequent jobs released *at least* T_i time units apart. Each job released by a task τ_i has to complete its execution within T_i time units from its release. We assume that tasks are independent, i.e., they do not share any resources except

[2] Although the approach presented in [13] can be adapted to obtain a solution for the intra-migrative model, it would incur a high time-complexity as it relies on solving a linear program.

[3] We ignore [19] since it applies only to a restricted case where task utilizations cannot exceed one.

processors and do not have any data dependency. We assume that a job can be executing on at most one processor at any given time. We also assume that optimal scheduling algorithms are used in both intra-migrative (e.g., [10]–[12]) and non-migrative (e.g., [7]) models.

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

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

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

$$\begin{aligned} &\forall \tau_i \in \tau: \quad (u_i^1 \leq \alpha) \quad \vee \quad (u_i^1 > 1) \quad \text{and} \\ &\forall \tau_i \in \tau: \quad (u_i^2 \leq \alpha) \quad \vee \quad (u_i^2 > 1) \end{aligned} \tag{2}$$

III. ILP-ALGO: AN OPTIMAL INTRA-MIGRATIVE ASSIGNMENT ALGORITHM

In this section, we provide an *optimal intra-migrative task assignment algorithm*[4] for assigning tasks in τ to processor types on two-type platform π. This algorithm is based on Integer Linear Programming. As described earlier, once the tasks have been assigned to processor types we assume that an optimal scheduling algorithm (e.g., [10]–[12]) is used to schedule them on processors of each type. From the feasibility tests of identical multiprocessor scheduling, the following conditions must hold for $t \in [1, 2]$ in order for the intra-migrative task assignment to be feasible:

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

$$\sum_{\tau_i \in \tau^t} u_i^t \quad \leq \quad m_t \tag{4}$$

where τ^t denotes the tasks assigned to processors of type-t.

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

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

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

A task in H12 cannot be scheduled to meet its deadline unless it executes in parallel, which is forbidden in our system model. Hence, we assume this set to be empty hereafter.

[4]A task assignment algorithm is said to be *optimal* if it always succeeds in finding a feasible assignment, provided such an assignment exists.

Minimize Z subject to the following constraints:

I1.	$\forall \tau_i \in \text{L}$: $x_i^1 + x_i^2 = 1$
I2.	$U^1 + \sum_{\tau_i \in \text{L}} x_i^1 \times u_i^1 \leq Z \times m_1$
I3.	$U^2 + \sum_{\tau_i \in \text{L}} x_i^2 \times u_i^2 \leq Z \times m_2$
I4.	$\forall \tau_i \in \text{L}$: x_i^1, x_i^2 are non-negative integers

Figure 1: ILP formulation – ILP-Feas(L, π, U^1, U^2).

H1 is the set of tasks that must be assigned to type-1 processors as their utilization on type-2 exceeds one, i.e.,

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

H2 is the set of tasks that must be assigned to type-2 processors as their utilization on type-1 exceeds one, i.e.,

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

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

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

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

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

First, it assigns tasks in H1 to type-1 (resp., H2 to type-2) processors. Let U^1 refer to the capacity consumed on type-1 processors after assigning H1 tasks, i.e., $U^1 = \sum_{\tau_i \in \text{H1}} u_i^1$. Analogously, let U^2 refer to the capacity consumed on type-2 processors after assigning H2 tasks, i.e., $U^2 = \sum_{\tau_i \in \text{H2}} u_i^2$. If $U^1 > m_1$ or $U^2 > m_2$ then it declares failure as this violates the feasibility condition shown in Inequality (4).

Second, it solves the ILP formulation shown in Figure 1 for assigning tasks in L. In this formulation, Z denotes the average used capacity of either type-1 or type-2 processors, whichever is greater, and is set as the objective function to be minimized. Each variable x_i^t ($t \in [1, 2]$) indicates the assignment of task τ_i to type-t processors. The first constraint specifies that every task must be assigned to a processor type. The second (resp., third) constraint asserts that at most $Z \times m_1$ of type-1 (resp., $Z \times m_2$ of type-2) processors' capacity can be used. The fourth constraint asserts that each task must be assigned entirely to either processors of type-1 or type-2. Using the solution of this ILP formulation, it assigns the tasks in L to processor types as follows: for each $\tau_i \in \text{L}$, τ_i is assigned to type-t processors if and only if $x_i^t = 1$. If $Z > 1$ then it declares failure as the feasibility condition in Inequality (4) is violated.

Lemma 1. *If there exists a feasible intra-migrative assignment of τ on π then ILP-Algo is guaranteed to return such a feasible intra-migrative assignment, i.e., ILP-Algo is an optimal intra-migrative task assignment algorithm.*

Since ILP-Algo relies on solving ILP formulation which is often time consuming, we now present a *sub-optimal*

Minimize Z subject to the following constraints:

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

Figure 2: Relaxed LP formulation – LP-Feas(L, π, U^1, U^2).

polynomial-time algorithm by relaxing the ILP formulation to an LP formulation.

IV. LP-ALGO: AN INTRA-MIGRATIVE ASSIGNMENT ALGORITHM

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

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

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

1) assigns tasks in H1 to type-1 (resp., H2 to type-2) processors. Let U^1 and U^2 denote the same entities as before. If $U^1 > m_1$ or $U^2 > m_2$ then it declares failure.
2) assigns tasks in L by solving the LP formulation shown in Figure 2. In the returned solution, if $x_i^t = 1$ (for $t \in [1, 2]$) then the corresponding task τ_i is *integrally* assigned to processors of type-t. If $0 < x_i^t < 1$ then a fraction x_i^t of τ_i is assigned to processors of type-t; we say that such tasks are *fractionally* assigned and are referred to as *fractional* tasks in the rest of the paper. If $Z > 1$ then it declares failure.

Among all the optimal solutions to an LP problem, at least one solution lies at a *vertex* of the *feasible region*[5](see pp. 117 in [20]). We are interested in such a solution, as we show below that it leads to a task assignment with at most one fractional task. For ease of discussion, we use index $1, 2, \ldots, \ell$ to refer to tasks in subset L hereafter.

Lemma 3. *For any optimal solution* $S = \{x_1^1, x_1^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ *to the LP formulation shown in Figure 2, if* S *lies at a vertex of the feasible region then there exists at most one task from* L *which is fractionally assigned to both processor types (and the rest are integrally assigned to processors of type-1 and type-2) in the task assignment that* S *reflects, i.e., there exists at most one index* $f \in [1, \ell]$ *such that* $0 < x_f^1 < 1$ *and* $0 < x_f^2 < 1$.

Proof: The proof is based on Fact 2 in [14]: *"consider a linear program on* n *variables* $x_1, x_2, \ldots, x_n$*, in which each variable* x_i *is subject to the non-negativity constraint, i.e.,* $x_i \geq 0$*. Suppose that there are further* m *linear constraints. If* $m < n$*, then at each vertex of the feasible region (including the basic solution), at most* m *of the variables have non-zero values"*. Clearly, the LP formulation of Figure 2 is a linear program on $n' = 2\ell+1$ variables (i.e., 2ℓ variables x_i^t, plus variable Z), all subject to non-negativity constraint, and $m' = \ell + 2$ further linear constraints (ℓ constraints on C1 plus constraints C2 and C3). As $m' < n'$ (we assume $\ell > 1$; otherwise the problem becomes trivial), we know from the above fact that in every optimal solution at the vertex of the feasible region, it holds that at most $m' = \ell+2$ variables take non-zero values. Since Z is certain to be non-zero, at most $\ell + 1$ variables x_i^t can be non-zero.

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

Lemma 4. *Any solution,* S_f^{LP}*, to the LP formulation (Figure 2) with at most one fractional task and* $Z_f^{\mathrm{LP}} \leq 1$*, can be converted to a solution,* $S_{\mathrm{nf}}^{\mathrm{LP}}$*, with no fractional task and*

$$Z_{\mathrm{nf}}^{\mathrm{LP}} \leq Z_f^{\mathrm{LP}} + \frac{\alpha}{2} \leq 1 + \frac{\alpha}{2} \tag{9}$$

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

$$\forall i \in [1, \ell], i \neq f : x_i^{1'} \leftarrow x_i^1 \ \wedge \ x_i^{2'} \leftarrow x_i^2 \tag{10}$$

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

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

[5]The feasible region in n-dimensional space is the region over which all the constraints hold.

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

$$Z_{\text{nf}}^{\text{LP}} = Z_f^{\text{LP}} + \frac{x_f^1 \cdot u_f^2}{m_2}$$

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

$$Z_{\text{nf}}^{\text{LP}} = \min\left(Z_f^{\text{LP}} + \frac{x_f^2 \cdot u_f^1}{m_1},\ Z_f^{\text{LP}} + \frac{x_f^1 \cdot u_f^2}{m_2}\right)$$

Since $m_1 \geq 1$ and $m_2 \geq 1$ and $u_f^1 \leq \alpha$ and $u_f^2 \leq \alpha$ and $x_f^2 = 1 - x_f^1$, the above expression yields:

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

The maximum values that Z_f^{LP} and the $\min$ term can take are 1.0 and 0.5 respectively. Hence, the term becomes: $Z_{\text{nf}}^{\text{LP}} \leq Z_f^{\text{LP}} + \frac{\alpha}{2} = 1 + \frac{\alpha}{2}$.

Thus, we showed that this transformed solution $S_{\text{nf}}^{\text{LP}} = \{x_1^{1'}, x_1^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z_{\text{nf}}^{\text{LP}}\}$ has no fractional tasks (i.e., indicator variables with fractional values) and satisfies Expression (9) and all the constraints of LP formulation. ■

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

Proof: We know that LP-Algo assigns tasks in H1 and H2 in the same way as an optimal intra-migrative task assignment algorithm does (as there is no other way to assign those tasks to meet deadlines). It then uses LP formulation to assign tasks in L. Combining Lemma 1, 2 and 3 gives us: if there exists a feasible intra-migrative task assignment of τ on π then LP-Algo returns an assignment of τ on π in which at most one task from L is fractionally assigned and the rest are integrally assigned to either type-1 or type-2 processors. Then, it follows from Lemma 4 that this fractional task can be assigned integrally to one of the processor types if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster. ■

It is well known that the assignment techniques that rely on LP solvers take considerable amount of time to output a solution compared to techniques that do not use LP solvers [16]. So, we now propose an algorithm, namely SA, that has the same approximation ratio as LP-Algo but does not rely on LP solvers and instead uses a simple assignment technique.

V. SA: An intra-migrative assignment algorithm

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

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

SA is named so because we "Sort and Assign" the tasks in L. Its time-complexity is at most:

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

VI. Performance analysis of algorithm SA

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

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

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

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

The remainder of this section is organized as follows. In subsection VI-A, we describe a method to transform any feasible solution for the LP formulation into a feasible swap solution (Lemma 5). Then, in subsection VI-B, we show that the solution returned by SA for assigning tasks in L is similar to the *swap solution* in the respect that at most one

task is assigned fractionally to both processor types and the rest are integrally assigned to type-1 and type-2 processors (Theorem 1). Finally, we show that this fractional task can be integrally assigned to a processor type if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster (Theorem 2). Considering that SA assigns tasks in H1 and H2 in a same way as an optimal intra-migrative task assignment algorithm does, we establish that its approximation ratio is $1 + \frac{\alpha}{2}$.

A. The swapping method

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

Lemma 5. *Any feasible solution* $S = \{x_1^1, x_1^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ *to the LP formulation of Figure 2 can be transformed into a feasible swap solution* $S' = \{x_1^{1'}, x_1^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z'\}$ *for which* $Z' = Z$.

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

$$\tau_p \neq \tau_q \text{ and } \frac{u_p^2}{u_p^1} \geq \frac{u_q^2}{u_q^1} \text{ and } x_p^1 < 1 \ \wedge \ x_q^2 < 1 \qquad (11)$$

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

P1. $\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$: $x_i^{1'} = x_i^1$ and $x_i^{2'} = x_i^2$

P2. $x_p^{1'} = 1$ or $x_q^{2'} = 1$

P3. Constraints C1-C4 of LP formulation hold and $Z' = Z$

The steps involved in this transforming a solution S into S' are described below. Performing those steps iteratively as long as such a pair $\tau_p, \tau_q \in \mathrm{L}$ fulfilling Expression (11) exists, will ultimately lead to a feasible swap solution S' with Z' equal to Z. Property P1 and P2 ensure that, with each iteration, the solution is moving closer towards the swap solution and P3 ensures that this (intermediate) solution is feasible. At each iteration, we denote by $S = \{x_1^1, x_1^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ the feasible solution computed in the previous iteration (in the first iteration, this solution is the given one) and by $S' = \{x_1^{1'}, x_1^{2'}, \ldots, x_\ell^{1'}, x_\ell^{2'}, Z'\}$ the modified feasible solution after the current iteration. The solution obtained after the final iteration is the feasible swap solution. Each iteration is performed as follows:
$\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$:

$$x_i^{1'} \leftarrow x_i^1 \qquad (12)$$
$$x_i^{2'} \leftarrow x_i^2 \qquad (13)$$

and

$$x_p^{1'} \leftarrow x_p^1 + \delta_1 \qquad (14)$$
$$x_p^{2'} \leftarrow x_p^2 - \delta_1 \qquad (15)$$
$$x_q^{1'} \leftarrow x_q^1 - \delta_2 \qquad (16)$$
$$x_q^{2'} \leftarrow x_q^2 + \delta_2 \qquad (17)$$

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

Proof of P1. From Expressions (12) and (13), it is trivial to see that Property **P1** holds.

Proof of P2. We have to consider two cases:

1) $\frac{x_p^2}{u_q^1} \leq \frac{x_q^1}{u_p^1}$. In this case, $\delta_1 = x_p^2$ and $\delta_2 = u_p^1 \times \frac{x_p^2}{u_q^1}$. Substituting the value of δ_1 in Expression (14) gives: $x_p^{1'} \leftarrow x_p^1 + x_p^2$. Since we know that $x_p^1 + x_p^2 = 1$ (it is true in the initial solution S and it holds true in all the subsequent iterations as well, as shown in **Proof of P3**), we get $x_p^{1'} \leftarrow 1$ and hence Property **P2** is satisfied.
2) $\frac{x_p^2}{u_q^1} > \frac{x_q^1}{u_p^1}$. Analogous to the above case, it can be shown that, we get $x_q^{2'} \leftarrow 1$ thereby satisfying Property **P2**.

Proof of P3. Since the initial solution S is feasible, constraint C1 holds by definition, i.e., $\forall \tau_i \in \mathrm{L} : x_i^1 + x_i^2 = 1$. Let us see whether this holds in solution S' which is obtained from S with the help of Expressions (12)-(17). Let us consider the following two cases:

Case (i): $\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$. Adding Expressions (12) and (13), we get: $x_i^{1'} + x_i^{2'} = x_i^1 + x_i^2$. Since we know that $\forall \tau_i \in \mathrm{L} : x_i^1 + x_i^2 = 1$, we obtain: $x_i^{1'} + x_i^{2'} = 1$. Recall that, in the next iteration, this solution S' acts as S while computing another S'. Hence, this holds in that iteration and all subsequent iterations. Hence constraint C1 holds true.

Case (ii): $\tau_i = \tau_p$ **or** $\tau_i = \tau_q$. Analogous to the previous case, adding Expressions (14) and (15), gives: $x_p^{1'} + x_p^{2'} = 1$ and adding Expressions (16) and (17), gives: $x_q^{1'} + x_q^{2'} = 1$. This holds true in all the iterations for the reasons stated in the previous case. Hence, constraint C1 holds true.

Now, we show that constraint C2 holds. From Equations (12)–(17), we have:

$$\begin{aligned}\sum_{i=1}^{\ell}(x_i^{1'} \times u_i^1) \quad &= \quad \sum_{\substack{i=1 \\ i\neq p, i\neq q}}^{\ell} (x_i^1 \times u_i^1) \\ &+ \quad \left(x_p^1 + u_q^1 \times \min\left(\frac{x_p^2}{u_q^1}, \frac{x_q^1}{u_p^1}\right)\right) \times u_p^1 \\ &+ \quad \left(x_q^1 - u_p^1 \times \min\left(\frac{x_p^2}{u_q^1}, \frac{x_q^1}{u_p^1}\right)\right) \times u_q^1 \qquad (18)\end{aligned}$$

In both the cases (i.e., $\frac{x_p^2}{u_q^1} \leq \frac{x_q^1}{u_p^1}$ and $\frac{x_p^2}{u_q^1} > \frac{x_q^1}{u_p^1}$), terms in Expression (18) cancel out and hence it simplifies to:

$$\sum_{i=1}^{\ell}(x_i^{1'} \times u_i^1) = \sum_{i=1}^{\ell}(x_i^1 \times u_i^1) \leq Z \times m_1 \qquad (19)$$

Hence, Constraint C2 is not violated.

With analogous reasoning, we can show that Constraint C3 is also not violated.

Now let us consider constraint C4. We know by definition that in solution S, $\forall \tau_i \in \mathrm{L}$, it holds that $x_i^1 \geq 0$ and $x_i^2 \geq 0$. Hence, from Expressions (12) and (13), in solution S', $\forall \tau_i \in \mathrm{L}, \tau_i \neq \tau_p, \tau_i \neq \tau_q$, it holds that $x_i^{1'} \geq 0$ and $x_i^{2'} \geq 0$. Now for $\tau_i = \tau_p$ or $\tau_i = \tau_q$, we have two cases:

Case (i): $\frac{x_p^2}{u_q^1} \leq \frac{x_q^1}{u_p^1}$. In this case, we have $\delta_1 = x_p^2$ and $\delta_2 = u_p^1 \times \frac{x_p^2}{u_q^1}$. Since we have shown that constraint C1 holds, substituting the value of δ_1 in Expression (14) and (15), we get $x_p^{1'} = 1$ and $x_p^{2'} = 0$ respectively. From the case, we

have: $x_q^1 \geq u_p^1 \times \frac{x_p^2}{u_q^1} > 0$. So, substituting the value of δ_2 in Expression (16) and (17) gives us $x_q^{1'} \geq 0$ and $x_q^{2'} > 0$ respectively. Hence, constraint C4 holds in this case.

Case (ii): $\frac{x_p^2}{u_q^1} > \frac{x_q^1}{u_p^1}$. Analogous to previous case, it can be shown that constraint C4 holds in this case well.

Since none of the constraints, C1-C4, of LP formulation are violated, the transformed solution remains feasible, and from Expression (19) (and analogous expression for type-2 processors), we can conclude that $Z' = Z$. Hence, applying the transformation shown in Expressions (12)-(17) iteratively, we obtain a feasible swap solution. ■

Lemma 6. *For any feasible swap solution* $S = \{x_1^1, x_1^2, \ldots, x_\ell^1, x_\ell^2, Z\}$ *to the LP formulation, we can re-index tasks in* L *such that* $\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1}$ *(with ties broken favoring the task with lower index before re-indexing) and with this order, there is an index* f *such that:*

$$\begin{aligned} &\forall i < f: \quad x_i^1 = 1 \quad \textit{and} \\ &\forall i > f: \quad x_i^2 = 1 \end{aligned}$$

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

$$\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1} \tag{20}$$

with ties broken as described in the claim. We now prove that there exists f such that $\forall \tau_i \in \mathrm{L}$, if $i < f$ then $x_i^1 = 1$ and if $i > f$ then $x_i^2 = 1$. Three cases may arise: (1) all the tasks in L are assigned to the same processor type or (2) tasks in L are assigned to both processor types and there is one fractional task or (3) tasks in L are assigned to both processor types and there is no fractional task.

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

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

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

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

Case (3): The tasks in L are assigned to both processor types and there is no fractional task. In this case, let f be the index of the first task in the sorted order (of tasks in L as shown in Expression (20)) that is integrally assigned to type-2 processors. By definition of τ_f, we know that all the tasks $\tau_i \in \mathrm{L}$ with $i < f$ must be integrally assigned to type-1 processors. Now consider any task $\tau_i \in \mathrm{L}$ with $i > f$. Since $\frac{u_f^2}{u_f^1} \geq \frac{u_i^2}{u_i^1}$, we know from Definition 1 that $x_f^1 = 1$ or $x_i^2 = 1$. But, we know that $x_f^1 = 0$, so it must hold that $x_i^2 = 1$. Hence, all tasks $\tau_i \in \mathrm{L}$ with $i > f$ are integrally assigned to type-2 processors.

We showed that the claim holds for all the cases, i.e., there exists a task in the sorted order (of tasks in L as shown in Expression (20)) such that all the tasks to its *left* are assigned to type-1 processors and all the tasks to its *right* are assigned to type-2 processors. Hence the proof. ■

B. The approximation ratio of SA

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

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

Proof: We know from Lemma 1 that if τ is intra-migrative feasible on π then ILP-Algo succeeds in finding such an assignment. This implies that there exists a feasible solution to the ILP formulation of Figure 1 with $Z_{\mathrm{ILP}} \leq 1$. Then, we know from Lemma 2 that, since there exists a solution to the ILP formulation with $Z_{\mathrm{ILP}} \leq 1$, there also exists a feasible solution to the LP formulation of Figure 2 with $Z_{\mathrm{LP}} \leq 1$. We know from Lemma 5 that such a solution can be converted into a feasible swap solution in which at most one task from L is fractionally assigned. Finally, we know from Lemma 6 that in this feasible swap solution, tasks in L can be re-indexed such that $\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1}$ (with ties broken, during re-indexing favoring the task with lower index before re-indexing) and with this order, there is an index f such that:

$$\begin{aligned} &\forall i < f: \quad x_i^1 = 1 \quad \text{and} \\ &\forall i > f: \quad x_i^2 = 1 \end{aligned}$$

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

We prove the theorem by contradiction. Let us assume that the task set τ is intra-migrative feasible on π but SA fails to find an assignment of τ on π in which at most one

task from L is fractionally assigned. We consider all the scenarios and show that it is impossible for this to happen.

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

$$\frac{u_1^2}{u_1^1} \geq \frac{u_2^2}{u_2^1} \geq \cdots \geq \frac{u_\ell^2}{u_\ell^1} \tag{21}$$

with ties broken as described in Lemma 6, during re-indexing. It considers tasks one by one from the left-hand side in the sorted order (as shown in Inequality (21)) and starts assigning them to type-1 processors. It stops assigning tasks to type-1 processors upon failing to assign a task say, τ_x, integrally on type-1 processors or all the tasks are successfully assigned thereby resulting in a successful assignment — whichever happens first. If it stops at τ_x then it considers tasks one by one from the right-hand side in the sorted order and starts assigning them to type-2 processors. It stops assigning tasks to processors of type-2 as soon as it fails to assign a task integrally (if τ is intra-migrative feasible on π then this task can be none other than τ_x as shown later in the theorem) or it successfully assigns τ_x integrally to a type-2 processor thereby resulting in a successful assignment – whichever happens first. If it stopped because it could not assign τ_x integrally to type-2 processor then it fractionally assigns τ_x to type-1 and type-2 processors.

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

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

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

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

$$U^1 + \sum_{i=1}^{x} u_i^1 \leq m_1 \tag{22}$$

i.e., tasks $\{\tau_1, \tau_2, \ldots, \tau_x\} \in$ L have been integrally assigned to processors of type-1 in S. However, we know that SA failed to integrally assign those tasks $\{\tau_1, \tau_2, \ldots, \tau_x\}$ to type-1 processors, which means that $U^1 + \sum_{i=1}^{x} u_i^1 > m_1$, in contradiction with Inequality (22).

Case 1.2 ($x > f$): This case is symmetrical to Case 1.1 and also leads to contradiction.

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

$$U^1 + \sum_{i=1}^{x-1} u_i^1 + x_f^{1,S} \cdot u_x^1 \leq m_1 \tag{23}$$

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

$$U^1 + \sum_{i=1}^{x-1} u_i^1 + x_x^{1,\text{SA}} \cdot u_x^1 = m_1 \tag{24}$$

From Expression (23) and (24), we then observe that $x_f^{1,S} \leq x_x^{1,\text{SA}}$. As a first conclusion, SA is thus able to integrally assign to type-1 processors all the tasks in τ that are integrally assigned to type-1 processors in solution S, plus (at least) the same fraction of task τ_x as that of task τ_f assigned to type-1 processor in S. Also, $x_f^{1,S} \leq x_x^{1,\text{SA}}$ implies that $x_f^{2,S} \geq x_x^{2,\text{SA}}$, which in turn yields:

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

The left-hand (resp., right-hand) side of the above inequality denotes the utilization of the tasks, including the fractional assignment of τ_f (which is same task as τ_x), assigned to type-2 processors in the solution S (resp., SA). As a second conclusion, SA is thus able to integrally assign to type-2 processors all the tasks in τ that are integrally assigned to type-2 processors in solution S, and assign no greater fraction of the task τ_x (which is same task as τ_f) to type-2 processor than in solution S. This provides the contradiction (as SA succeeds in assigning all the tasks).

As shown in above three cases, SA always succeeds in finding a task assignment similar to swap solution S when S has a fractional task, hence leading to a contradiction.

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

Case 2.1 ($x < f$) and Case 2.2 ($x > f$) These are similar to Cases 1.1 and 1.2 above and lead to contradiction.

Case 2.3 ($f = x$): This indicates that SA was able to integrally assign tasks $\tau_1, \ldots, \tau_{x-1} \in L$ to type-1 processors as in S. However, it failed to integrally assign tasks $\tau_x, \ldots, \tau_\ell \in L$ to type-2 processors that are integrally

assigned in S. This means $U^2 + \sum_{i=x}^{\ell} u_i^2 > m_2$ whereas $U^2 + \sum_{i=f}^{\ell} u_i^2 \leq m_2$. From the case (i.e., $f = x$), this is a contradiction and hence SA would also succeed in assigning those tasks to type-2 processors.

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

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

Proof: We know from Theorem 1 that if τ is intra-migrative feasible on π then SA succeeds in returning a feasible assignment of τ on π in which at most one task from L is fractionally assigned and the rest are integrally assigned to type-1 and type-2 processors. It follows from Lemma 4 that this fractional task can also be assigned integrally to one of the processor types if given a platform in which processors are $1 + \frac{\alpha}{2}$ times faster. Hence the proof. ■

VII. SA-P: A NON-MIGRATIVE ASSIGNMENT ALGORITHM

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

The new algorithm, SA-P, for assigning tasks to processors, works as follows: It

1) assigns tasks in τ to processor types on π using SA.
   - SA assigns tasks to only processor types (and not to processors) – let τ^1 (resp., τ^2) be the subset of tasks assigned to type-1 (resp., type-2) processors.
   - SA guarantees that, for an intra-migrative feasible task set, at most one task is fractionally assigned to both the processor types — let τ_f be this task and let fraction x_f^1 of τ_f be assigned to type-1 and $x_f^2 = 1 - x_f^1$ be assigned to type-2.
2) (Index the processors in some way and maintain this indexing throughout the algorithm.) assigns tasks from τ^1 (resp., τ^2) to individual processors of type-1 (resp., type-2) using first-fit but allowing *splitting* of tasks between consecutive processors (also referred to as "wrap-around" assignment in literature). It assigns fraction x_f^1 of τ_f to the last processor (i.e., m_1^{th} processor) of type-1 and fraction x_f^2 to the last processor (i.e., m_2^{th} processor) of type-2. It is trivial to see that such an assignment ensures following things:
   - at most $m_1 - 1$ tasks are *split* between processors of type-1 with one task split between each pair of consecutive processors
   - at most $m_2 - 1$ tasks are *split* between processors of type-2 with one task split between each pair of consecutive processors and
   - at most 1 task is fractionally assigned between processors of type-1 and type-2; τ_f split between m_1^{th} processor of type-1 and m_2^{th} processor of type-2
3) copies this assignment of tasks onto a faster platform π' (we show in Theorem 3 that a platform that is $1+\alpha$ times as fast as π is sufficient).
4) on platform π', assigns a task split between processor p and $p+1$ of type-1 to processor p, where $1 \leq p < m_1$; similarly, it assigns a task split between processor q and $q+1$ of type-2 to processor q, where $1 \leq q < m_2$. Finally, it assigns task τ_f to processor m_1 (or, m_2).

SA-P is named so because it is the *Partitioned* (i.e., non-migrative) version of SA. Its time-complexity is at most:

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

Theorem 3. *If there exists a feasible intra-migrative assignment of τ on π then SA-P is guaranteed to find a feasible non-migrative assignment of τ on $\pi^{(1+\alpha)}$.*

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

Let us consider the assignment of tasks in τ to processor types on π returned by SA with at most one fractional task. We know that SA assigns tasks to only processor types (and not to processors) – let τ^1 be the subset of tasks assigned to type-1 processors and τ^2 to type-2 processors. Let τ_f be the task fractionally assigned to both the processor types — fraction x_f^1 to type-1 and x_f^2 to type-2. We know that:

$$\forall \tau_i \in \tau^1 : u_i^1 \leq \alpha \tag{25}$$

$$\forall \tau_i \in \tau^2 : u_i^2 \leq \alpha \tag{26}$$

$$u_f^1 \leq \alpha \quad \wedge \quad u_f^2 \leq \alpha \tag{27}$$

SA-P uses this assignment information and assigns tasks to individual processors (with splitting allowed using "wrap-around" technique) as described earlier in Step 2 of SA-P algorithm. After this step,

$$\forall p \in \pi : U[p] \leq 1 \tag{28}$$

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

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

$$\forall \tau_i \in \tau : \frac{u_i^{2'}}{u_i^2} = \frac{u_i^{1'}}{u_i^1} = \frac{1}{1+\alpha} \tag{29}$$

Combining Expression (28) and (29) gives us:

$$\forall p \in \pi^{(1+\alpha)} : U[p] \leq \frac{1}{1+\alpha} \quad (30)$$

Also, combining Expressions (25)-(27) and (29), we get:

$$\forall \tau_i \in \tau^1 : u_i^{1'} \leq \frac{\alpha}{1+\alpha} \quad (31)$$

$$\forall \tau_i \in \tau^2 : u_i^{2'} \leq \frac{\alpha}{1+\alpha} \quad (32)$$

$$u_f^{1'} \leq \frac{\alpha}{1+\alpha} \quad \wedge \quad u_f^{2'} \leq \frac{\alpha}{1+\alpha} \quad (33)$$

On step 4, SA-P assigns the split tasks integrally. So, $\forall p_1 \in$ type-1 of $\pi^{(1+\alpha)}$, it moves the fraction of task $\tau^1_{p_1,p_1+1}$ that is assigned to $(p_1+1)^{th}$ processor of type-1 to p_1^{th} processor of type-1. After this re-assignment, it follows from Expressions (30) and (31) that:

$$\forall p_1 \in \text{type-1 of } \pi^{(1+\alpha)} \ \wedge \ p_1 \neq m_1 : \ U[p_1] \leq 1.0 \quad (34)$$

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

Analogously, $\forall p_2 \in$ type-2 of $\pi^{(1+\alpha)}$, it moves the fraction of task $\tau^2_{p_2,p_2+1}$ that is assigned to $(p_2+1)^{th}$ processor of type-2 to p_2^{th} processor of type-2. After this re-assignment, it follows from Expressions (30) and (32) that:

$$\forall p_2 \in \text{type-2 of } \pi^{(1+\alpha)} \ \wedge \ p_2 \neq m_2 : \ U[p_2] \leq 1.0 \quad (35)$$

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

Finally, the task τ_f (split between m_1^{th} processor of type-1 and m_2^{th} processor of type-2) remains to be integrally assigned. It turns out that this task can be entirely assigned to either m_1^{th} processor of type-1 or m_2^{th} processor of type-2. Consider the case that it is integrally assigned to m_1^{th} processor of type-1. Since, m_1^{th} processor is used at most $\frac{1}{1+\alpha}$ of its capacity and $u_f^{1'} \leq \frac{\alpha}{1+\alpha}$, this re-assignment does not allow the used capacity of m_1^{th} processor to exceed one. Combining this with the fact that m_2^{th} processor of type-2 is still utilized at most $\frac{1}{1+\alpha}$ of its capacity and with Expressions (34) and (35), we obtain:

$$\forall p \in \pi^{(1+\alpha)} : U[p] \leq 1.0 \quad (36)$$

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

Since Expression (36) is a necessary feasibility condition for task assignment on a uni-processor, the non-migrative assignment of τ on $\pi^{(1+\alpha)}$ returned by SA-P is feasible. ■

VIII. Conclusions

We proposed two linearithmic time-complexity algorithms, namely SA and SA-P, for assigning implicit-deadline sporadic tasks on two-type heterogeneous multiprocessors. We also showed that they provide the following guarantee. If there exists a feasible intra-migrative assignment of a task set on a two-type platform then (i) using SA, it is guaranteed to find such a feasible intra-migrative assignment if given a platform in which processors are $1+\frac{\alpha}{2}$ times faster and (ii) SA-P is guaranteed to find a feasible non-migrative assignment if given a platform in which processors are $1+\alpha$ times faster.

Acknowledgment

This work was supported by the REHEAT project, ref. FCOMP-01-0124-FEDER-010045, co-funded by FCT-MCTES (Portuguese Foundation for Science and Technology) and ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'); the RECOMP project, ref. ARTEMIS/0202/2009, co-funded by FCT-MCTES, and by EU funds through the ARTEMIS Joint Undertaking, under grant nr. 100202.

References

[1] AMD Inc., "The AMD fusion family of APUs," http://sites.amd.com/us/fusion/apu/Pages/fusion.aspx.

[2] Intel Corporation, "The second generation Intel core processor," http://www.intel.com/consumer/products/processors/core-family.htm.

[3] Nvidia Inc., "NVIDIA Tegra: The world's first mobile super chip," http://www.nvidia.com/object/tegra.html.

[4] TI Inc., "OMAP application processors: OMAP 5 platform," http://www.ti.com/ww/en/omap/omap5/omap5-platform.html.

[5] Qualcomm Inc, "Quad-core for next generation devices," http://www.qualcomm.com/snapdragon/specs.

[6] Samsung Inc., "Samsung exynos processor," www.samsung.com/exynos/.

[7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, 1973.

[8] M. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings of IFIP Congress (IFIP'74)*, 1974.

[9] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 3rd ed. Springer, 2006.

[10] B. Andersson and K. Bletsas, "Sporadic Multiprocessor Scheduling with Few Preemptions," in *20th Euromicro Conference on Real-Time Systems*, 2008, pp. 243–252.

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

[12] J. Anderson and A. Srinivasan, "Early-release fair scheduling," in *Proceedings of the* 12^{th} *Euromicro conference on Real-time systems*, 2000, pp. 35–43.

[13] J. Lenstra, D. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Math. Program.*, vol. 46, pp. 259–271, 1990.

[14] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proceedings of the 10th IEEE International Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 536–543.

[15] ——, "Partitioning real-time tasks among heterogeneous multiprocessors," in 33^{rd} *International Conference on Parallel Processing*, 2004, pp. 467–474.

[16] B. Andersson, G. Raravi, and K. Bletsas, "Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors," in *Proceedings of the 31st IEEE International Real-Time Systems Symposium*, 2010, pp. 239–248.

[17] J. Correa, M. Skutella, and J. Verschae, "The power of preemption on unrelated machines and applications to scheduling orders," *Mathematics of Operations Research*, 2011.

[18] S. Baruah, "Task assignment on two unrelated types of processors," in *International Conference on Real-Time and Network Systems (RTNS)*, 2011, pp. 69–78.

[19] G. Raravi, B. Andersson, and K. Bletsas, "Provably good task assignment on heterogeneous multiprocessor platforms for a restricted case but with a stronger adversary," *SIGBED Review*, vol. 8, pp. 19–22, 2011.

[20] D. G. Luenberger and Y. Ye, *Linear and Nonlinear Programming, 3rd Ed.* International Series in Operations Research & Management Science, 2008.