IPP HURRAY!

www.hurray.isep.ipp.pt

Technical Report

A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors

Gurulingesh Raravi

Vincent Nélis

HURRAY-TR-120903

Version:

Date: 09-10-2012

A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors

Gurulingesh Raravi, Vincent Nélis

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Consider the problem of determining a task-to-processor assignment for a given collection of implicit-deadline sporadic tasks upon a multiprocessor platform in which there are two distinct kinds of processors. We propose a polynomial-time approximation scheme (PTAS) for this problem. It offers the following guarantee: for a given task set and a given platform, if there exists a feasible task-to-processor assignment, then given an input parameter, e, our PTAS succeeds, in polynomial time, in finding such a feasible task-to-processor assignment on a platform in which each processor is 1+3e times faster. In the simulations, our PTAS outperforms prior state-of-the-art PTAS and also for the vast majority of task sets, it requires significantly smaller processor speedup than (its upper bound of) 1+3e for successfully determining a feasible task-to-processor assignment.

A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors

Gurulingesh Raravi and Vincent Nélis
CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: ghri@isep.ipp.pt, nelis@isep.ipp.pt

***Abstract*—Consider the problem of determining a task-to-processor assignment for a given collection of implicit-deadline sporadic tasks upon a multiprocessor platform in which there are two distinct kinds of processors. We propose a polynomial-time approximation scheme (PTAS) for this problem. It offers the following guarantee: for a given task set and a given platform, if there exists a feasible task-to-processor assignment, then given an input parameter, ϵ, our PTAS succeeds, in polynomial time, in finding such a feasible task-to-processor assignment on a platform in which each processor is $1+3\epsilon$ times faster. In the simulations, our PTAS outperforms the state-of-the-art PTAS [1] and also for the vast majority of task sets, it requires significantly smaller processor speedup than (its upper bound of) $1+3\epsilon$ for successfully determining a feasible task-to-processor assignment.**

I. Introduction

This paper addresses the problem of finding an assignment of tasks to processors (also referred to as *partitioning*) for a given set of implicit-deadline sporadic tasks (also referred to as *Liu and Layland* (LL) tasks [2]) 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 [3]–[7].

In the partitioning problem, every task must be statically assigned to a processor at design time and all its jobs must execute on that processor at run time. The challenge is to find, at design time, a task-to-processor assignment such that, at run time, an uniprocessor scheduling algorithm running on each processor meets all the deadlines. Scheduling the tasks to meet deadlines on an uniprocessor platform is a well-understood problem. One may use Earliest-Deadline First (EDF) [2], for example. EDF is an *optimal* scheduling algorithm on uniprocessor systems [2], [8], with the interpretation that it always constructs a schedule in which all the deadlines are met, if such a schedule exists. Therefore, assuming that an optimal scheduling algorithm is used on each processor, the challenging part is to find a partitioning for which *there exists* a schedule that meets all the deadlines — such a partitioning is said to be a *feasible* partitioning hereafter. Even in the simpler case of identical multiprocessors, finding a feasible partitioning is strongly NP-Complete [9]. Hence, this result continues to hold for two-type platforms. In this work, we propose a *polynomial-time approximation scheme* (PTAS), for this problem which outperforms the state-of-the-art PTAS [1].

Definition 1 (**PTAS**). *A PTAS takes an instance of an optimization problem (for which exact solutions are intractable) and a parameter $\epsilon > 0$ and, in polynomial time, produces a solution that is within a factor $f(\epsilon)$ of being optimal where function $f()$ is independent of the problem instance.*

Definition 2 (**Approximation ratio**). *An algorithm for solving an optimization problem is said to have an approximation ratio of A if for all instances of the problem, the algorithm produces a solution that is within a factor of A from the optimal value.*

Related work. The partitioning problem on heterogeneous multiprocessors has been studied in the past [10]–[14]. In [10]–[12], the authors proposed algorithms for the problem of partitioning LL task sets on heterogeneous multiprocessors with an approximation ratio of 2. All these approaches [10]–[12] focused on generic heterogeneous multiprocessor platforms with two or more processor types. Due to practical relevance, Andersson et al. [13] considered the partitioning problem on two-type platforms and proposed an algorithm, FF-3C, and couple of its variants based on first-fit heuristic. These had the same performance guarantee as the approaches in [10]–[12] (i.e., requiring processors twice as fast, in the worst-case) but can be implemented efficiently and exhibited better average-case performance than those in [11], [12].

In a recent significant development, Wiese et al. [1] proposed a PTAS (referred to as $\mathrm{PTAS_{LP}}$ since it uses "Linear Programming") for partitioning LL task system on *limited heterogeneous multiprocessors* in which processors are of a relatively small number (≥ 2) of distinct types. The $\mathrm{PTAS_{LP}}$ provides the following guarantee: if there exists a feasible partitioning of a given task set on a limited heterogeneous multiprocessor platform then the $\mathrm{PTAS_{LP}}$ succeeds in partitioning the task set on a platform in which each processor is $\frac{1+\epsilon}{1-\epsilon}$ times faster. This is theoretically a significant result since $\mathrm{PTAS_{LP}}$ partitions the task set in polynomial time, to any desired degree of accuracy. However, its practical significance is severely limited as the algorithm has a very high run-time complexity since it "heavily" relies on solving linear programming formulations. Even on a two-type platform, it has a high run-time complexity which makes its implementation highly inefficient (which is confirmed by the simulations in Section VIII). Therefore, we propose a PTAS for two-type platforms which does not rely on solving linear programs and hence offers a significantly better time-complexity than $\mathrm{PTAS_{LP}}$.

Contribution and significance of this work. We present a PTAS for the problem of partitioning a given LL task set on a two-type platform which offers the following guarantee. If there exists a feasible partitioning of a task set τ on a two-type platform π then given an $\epsilon > 0$, PTAS succeeds, in polynomial time, in finding a feasible partitioning of τ on $\pi^{(1+3\epsilon)}$ where $\pi^{(1+3\epsilon)}$ is a two-type platform in which each processor is $1+3\epsilon$ times faster than the corresponding processor in π.

We believe the significance of this work is as follows. For the problem under consideration, our PTAS has superior performance compared to prior state-of-the-art, i.e., $\mathrm{PTAS_{LP}}$. Specifically, compared to $\mathrm{PTAS_{LP}}$, our PTAS has (i) a much better run-time complexity and (ii) a *competitive* approximation ratio. We evaluate the average-case performance of these algorithms with randomly generated task sets. The evaluation is based on (i) the processor speedup the algorithm needs, for a given task set, so as to succeed, compared to an optimal algorithm and (ii) the running time. Overall, our algorithm outperforms $\mathrm{PTAS_{LP}}$ by requiring much smaller processor speedup and running faster by orders of magnitude. Also, for the vast majority of task sets, it requires significantly smaller processor speedup than its upper bound of $1 + 3\epsilon$.

II. System model

We consider the problem of partitioning a task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n implicit-deadline sporadic tasks (LL tasks) on a two-type heterogeneous multiprocessor platform π 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 an optimal scheduling algorithm such as EDF is used on each processor.

On a two-type platform, the WCET of a task depends on the processor type on which it executes. We denote by C_i^1 and C_i^2 the WCET of a task τ_i on processors of type-1 and type-2 and we denote by $u_i \stackrel{\text{def}}{=} C_i^1/T_i$ and $v_i \stackrel{\text{def}}{=} C_i^2/T_i$ its utilizations on type-1 and type-2 processors, respectively. A task τ_i that cannot be executed on processors of type-1 (resp., type-2) is modeled by setting its $u_i = \infty$ (resp., $v_i = \infty$).

III. An overview of our approach

We now give an overview of our algorithm (referred to as $\mathrm{PTAS_{NF}}$ since it uses "Next-Fit"). Our PTAS takes $\epsilon > 0$ as an input parameter and outputs a feasible partitioning. Let us partition the given task set τ into two subsets as follows:

$$\tau_{\mathrm{hvy}} = \{\tau_i \mid u_i \geq \epsilon \text{ or } v_i \geq \epsilon\} \tag{1}$$

$$\tau_{\mathrm{lgt}} = \tau \setminus \tau_{\mathrm{hvy}} = \{\tau_i \mid u_i < \epsilon \text{ and } v_i < \epsilon\} \tag{2}$$

Intuitively, τ_{hvy} refers to "heavy" and τ_{lgt} refers to "light" tasks. Our PTAS, has the following steps:

Step 1. We first approximate the utilizations of every task in τ_{hvy} to some finite number of pre-computed values. The motivation for doing this is twofold: (i) by restricting the number of pre-computed values to a constant, we ensure polynomial complexity for the algorithm and (ii) by choosing these values cleverly, we ensure the approximation ratio of the algorithm is bounded. Then, we assign the tasks in τ_{hvy} to processors using the algorithm $\mathrm{A_{hvy}}$ described in Section IV-A. In Section IV-E, we show that after using $\mathrm{A_{hvy}}$, the sum of the utilizations of the tasks assigned on processors of type-1 (resp., type-2) does not exceed $(1 + \epsilon) \times m_1$ (resp., $(1 + \epsilon) \times m_2$).

Step 2. Some tasks from τ_{hvy} (with $u_i \geq \epsilon \ \wedge \ v_i < \epsilon$ or $u_i < \epsilon \ \wedge \ v_i \geq \epsilon$) may remain unassigned after using $\mathrm{A_{hvy}}$. These unassigned tasks form the set, τ_{int} ("intermediate" tasks). Now, $\mathrm{A_{int}}$ *fractionally* assigns the tasks (i.e., tasks can be split between processors) with $u_i < \epsilon \ \wedge \ v_i \geq \epsilon$ (resp., $u_i \geq \epsilon \ \wedge \ v_i < \epsilon$) to type-1 (resp., type-2) processors as described in Section V-A. In Section V-B, we show that after using $\mathrm{A_{int}}$, the sum of the utilizations of all the tasks assigned so far on processors of type-1 (resp., type-2) still does not exceed $(1 + \epsilon) \times m_1$ (resp., $(1 + \epsilon) \times m_2$).

Step 3. Fractionally assign the tasks in τ_{lgt} to processors using the algorithm $\mathrm{A_{lgt}}$ (which makes use of a fractional knapsack property) described in Section VI-A. In Section VI-B, we show that after using $\mathrm{A_{lgt}}$, the sum of the utilizations of all the tasks assigned so far on processors of type-1 (resp., type-2) does not exceed $(1+2\epsilon) \times m_1$ (resp., $(1+2\epsilon) \times m_2$).

Step 4. Finally, those tasks from τ_{int} and τ_{lgt} that were assigned fractionally by $\mathrm{A_{int}}$ and $\mathrm{A_{lgt}}$ are assigned *integrally* using the algorithm $\mathrm{A_{fract}}$ described in Section VII-A. In Section VII-B, we show that after using $\mathrm{A_{fract}}$, the sum of the utilizations of all the tasks assigned so far on processors of type-1 (resp., type-2) does not exceed $(1+3\epsilon) \times m_1$ (resp., $(1 + 3\epsilon) \times m_2$). Hence, we conclude that if τ has a feasible partitioning on π then $\mathrm{PTAS_{NF}}$ succeeds in finding such a feasible partitioning of τ on $\pi^{(1+3\epsilon)}$.

IV. Assigning the tasks in τ_{hvy} (Step 1)

In this section, we describe the algorithm, $\mathrm{A_{hvy}}$, for integrally assigning (a subset of) the tasks in τ_{hvy} to processors and also analyze its returned assignment.

A. Description of the algorithm $\mathrm{A_{hvy}}$

It consists of three steps described in the next three sections:

Step 1.1. It defines a finite set $S(\epsilon)$ of utilization values, based on the value of the input parameter, ϵ. Then, it computes the "rounded-down utilizations" u_i^{rd} and v_i^{rd} of every task $\tau_i \in \tau$ by rounding *down* u_i and v_i to one of the quantized values in $S(\epsilon)$. We will denote by $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ the set of tasks obtained by rounding down the utilizations of the tasks of τ_{hvy}.

Step 1.2. It uses dynamic programming to determine, in polynomial time, (i) all the subsets of $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ that can be partitioned upon m_1 processors of type-1 and (ii) all the subsets that can be partitioned upon m_2 processors of type-2.

Step 1.3. It exhaustively considers each pair of subsets such that one subset can be partitioned on m_1 processors of type-1 and the other subset can be partitioned on m_2 processors of type-2. Using the ordered pair of subsets under consideration, it integrally assigns (a subset of) the tasks from τ_{hvy} to processors (at least all the tasks with $u_i \geq \epsilon \wedge v_i \geq \epsilon$).

B. Rounding-down the utilizations of the tasks (Step 1.1)

We compute the set $S(\epsilon)$ of all real numbers ≤ 1 that are of the form $\epsilon(1+\epsilon)^k$, for all integers $k \geq 0$. Then, we compute the rounded-down utilizations u_i^{rd} and v_i^{rd} of every task $\tau_i \in \tau$ by rounding down each of its utilizations (u_i and v_i) to the nearest value present in the set $S(\epsilon)$. For tasks with $u_i < \epsilon$ (resp., $v_i < \epsilon$), we set $u_i^{\mathrm{rd}} = 0$ (resp., $v_i^{\mathrm{rd}} = 0$) and for tasks with $u_i = \infty$ (resp., $v_i <= \infty$), we set $u_i^{\mathrm{rd}} = \infty$ (resp., $v_i^{\mathrm{rd}} = \infty$). The definition of $S(\epsilon)$ leads to the following property.

Property 1. *For a task τ_i, if $\epsilon \le u_i \le 1$ then there exists k such that $\epsilon(1+\epsilon)^k \le u_i < \epsilon(1+\epsilon)^{k+1}$ and thus*

$$\frac{u_i}{u_i^{\mathrm{rd}}} = \frac{u_i}{\epsilon(1+\epsilon)^k} < \frac{\epsilon(1+\epsilon)^{k+1}}{\epsilon(1+\epsilon)^k} = (1+\epsilon) \tag{3}$$

The same holds for v_i.

Therefore, if the utilizations of each task is reduced by this maximal factor, it follows that any collection of tasks with their reduced utilizations summing to ≤ 1 would have their original utilizations summing to $\le (1+\epsilon)$.

Let us now determine the number L of distinct values in $S(\epsilon)$. Since only values with $\epsilon(1+\epsilon)^k \le 1$ are included in $S(\epsilon)$, it holds that $k\log(1+\epsilon) \le \log(1/\epsilon)$ and thus, $k \le \frac{\log(1/\epsilon)}{\log(1+\epsilon)}$. Then we conclude that $L = \left\lfloor \frac{\log(1/\epsilon)}{\log(1+\epsilon)} \right\rfloor + 1$.

For each ℓ, $0 \le \ell < L$, we denote by X_ℓ (resp., Y_ℓ) the number of tasks in $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ with u_i^{rd} (resp., v_i^{rd}) equal to $\epsilon(1+\epsilon)^\ell \in S(\epsilon)$. The task set $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ can thus be represented by $2 \times L$ non-negative integers $X_0, X_1, \ldots, X_{L-1}, Y_0, Y_1, Y_{L-1}$. Note that each X_ℓ and each Y_ℓ is no greater than $|\tau_{\mathrm{hvy}}|$.

C. Generating the feasible configurations (Step 1.2)

The rounding down of the utilizations described in the previous section ensures that the utilizations of the tasks in τ_{hvy} may only take one of the values in $S(\epsilon)$, providing the set $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$. In this section, using dynamic programming, we determine, in polynomial time, all the subsets of $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ that can be partitioned upon m_1 processors of type-1 (resp., m_2 processors of type-2). Once all the feasible subsets (also referred to as *feasible configurations*) are determined, we use this information to assign a subset of tasks from τ_{hvy} on type-1 and type-2 processors (described in Section IV-D).

Definition 3 (feasible configurations). *Consider any L-tuple $T = (x_0, x_1, \ldots, x_{L-1})$ where $x_\ell \ge 0, \forall \ell \in [0, L-1]$, and let $\tau_{(T)}$ denote a task set containing exactly x_ℓ tasks τ_i of utilization $u_i = \epsilon(1+\epsilon)^\ell$ for each ℓ. The L-tuple T is said to be a feasible configuration on m_1 processors of type-1 if and only if there exists a feasible partitioning for the corresponding task set $\tau_{(T)}$ on m_1 processors of type-1. Analogously, we define an L-tuple $(y_0, y_1, \ldots, y_{L-1})$ with v_i values that is a feasible configuration on m_2 processors of type-2.*

The algorithm $\mathrm{A}_{\mathrm{hvy}}$ uses the same approach as the one presented in [14] to determine all the configurations $(x_0, x_1, \ldots, x_{L-1})$ of tasks in $\tau_{\mathrm{hvy}}^{\mathrm{rd}}$ (resp., $(y_0, y_1, \ldots, y_{L-1})$) that are *feasible* on m_1 processors of type-1 (resp., m_2 processors of type-2), in which $x_\ell \le X_\ell \le |\tau_{\mathrm{hvy}}|$ (resp., $y_\ell \le Y_\ell \le |\tau_{\mathrm{hvy}}|$) for each ℓ, $0 \le \ell < L$. This approach [14] is summarized below. As there are no more than $\Pi_{\ell=0}^{L-1}(1+X_\ell) \le \Pi_{\ell=0}^{L-1}(1+|\tau_{\mathrm{hvy}}|) = O(n^L)$ such feasible configurations on type-1 processors (and the same holds for type-2 processors) and since L is a constant for a given value of ϵ, the time to determine all the feasible configurations is polynomial in n.

Summary of the approach in [14]: It constructs two separate tables: one table each for storing the information about all the configurations on processors of each type. The table for type-1 processors has m_1 rows and $\Pi_{\ell=0}^{L-1}(1+X_\ell)$ columns. Each column corresponds to a different configuration and each cell has a value $\in \{\text{yes}, \text{no}\}$. A cell in the i'th row and the j'th column is a "yes" if the corresponding configuration is feasible on i processors of type-1. This table is filled row-wise starting with the first row. Filling in the first row is straightforward for all the configurations: it is a "yes" if the corresponding configuration, say $(x_0, x_1, \ldots, x_{L-1})$, is feasible on a single processor, i.e., if $\sum_{\ell=0}^{L-1} x_\ell \times \epsilon(1+\epsilon)^\ell \le 1$, it is a "no" otherwise. The i'th row is filled in by using the entries of the $(i-1)$'th row. Specifically, for the configuration corresponding to the j'th column, say $(x_0, x_1, \ldots, x_{L-1})$, the cell at the i'th row is a "yes" if and only if there exists two configurations $(x'_0, x'_1, \ldots, x'_{L-1})$ and $(x''_0, x''_1, \ldots, x''_{L-1})$ such that

1) $(x'_0, x'_1, \ldots, x'_{L-1})$ is a feasible configuration on $(i-1)$ processors of type-1;
2) $(x''_0, x''_1, \ldots, x''_{L-1})$ is a feasible configuration on one processor of type-1; and
3) $x_\ell = x'_\ell + x''_\ell$, for all $0 \le \ell < L$.

For each cell in the i'th row, there are polynomially many possible candidates for the role of $(x'_0, x'_1, \ldots, x'_{L-1})$; hence, each cell in the i'th row can be filled in polynomial time. Similarly, the second table for type-2 processors is constructed.
Note: By using standard dynamic programming tricks which require storing additional information [14], we can obtain a task-to-processor assignment from the feasible configurations.

D. Determining the partitioning (Step 1.3)

Using the two configuration tables that were constructed in the previous step, we now determine a partitioning for (a subset of) the heavy tasks. The main idea is as follows. Suppose that the task set τ can indeed be partitioned on the given platform and let $\mathcal{H}_{\mathrm{feas}}$ denote (one of) the feasible partitioning. For each ℓ, $0 \le \ell < L$, let x_ℓ^{feas} denote the number of tasks τ_i satisfying $\epsilon(1+\epsilon)^\ell \le u_i < \epsilon(1+\epsilon)^{\ell+1}$ that are assigned to type-1 processors in $\mathcal{H}_{\mathrm{feas}}$. Since $\mathcal{H}_{\mathrm{feas}}$ is a feasible partitioning, the configuration $(x_0^{\mathrm{feas}}, x_1^{\mathrm{feas}}, \ldots, x_{L-1}^{\mathrm{feas}})$ must appear in the table constructed in the previous step for type-1 processors and the cell at the m_1'th row of the corresponding column must contain "yes". Analogously, the configuration $(y_0^{\mathrm{feas}}, y_1^{\mathrm{feas}}, \ldots, y_{L-1}^{\mathrm{feas}})$ must appear in the table constructed for type-2 processors and the cell at the m_2'th row of the corresponding column must contain "yes". However, since we do not know which of the feasible configurations in our tables correspond to $\mathcal{H}_{\mathrm{feas}}$, we consider every ordered pair of configurations that are feasible on m_1 and m_2 processors of type-1 and type-2 respectively. Since there are only polynomially (i.e., $O(n^L)$) many distinct feasible configurations in each table, it follows that there are at most polynomially many such ordered pairs of feasible configurations to consider.

For each considered ordered pair of configurations, by assuming that they are the ones corresponding to $\mathcal{H}_{\mathrm{feas}}$, we attempt to construct a *similar* task-to-processor assignment for the tasks in τ_{hvy} as that of $\mathcal{H}_{\text{feas}}$. The assignment obtained will be *similar* to $\mathcal{H}_{\text{feas}}$ in the following sense: although the tasks assigned in both the assignments may not be the same, it holds that (as we show later), the sum of utilizations of the tasks assigned by our algorithm on each processor type does not exceed that of $\mathcal{H}_{\text{feas}}$ by a factor of $1+\epsilon$.

Let $\{(x_0, x_1, \ldots, x_{L-1}), (y_0, y_1, \ldots, y_{L-1})\}$ denote the currently considered ordered pair of feasible configurations on m_1 and m_2 processors of type-1 and type-2, respectively. The algorithm A_{hvy} to determine the corresponding task-to-processor assignment of tasks from τ_{hvy} is as follows.

Step 1.3.1. For each ℓ, $0 \le \ell \le L-1$, A_{hvy} assigns *exactly* x_ℓ tasks τ_i satisfying $u_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ to type-1 processors. Specifically, for each ℓ,

1) If there are *fewer* than x_ℓ such tasks in τ_{hvy}, then A_{hvy} declares failure with respect to this particular ordered pair of feasible configurations, and moves on to the next ordered pair of feasible configurations.
2) If there are *exactly* x_ℓ such tasks then A_{hvy} assigns all of them to type-1 processors.
3) If there are *more* than x_ℓ such tasks, it assigns x_ℓ of them to type-1 processors by favoring those with larger v_i.

Step 1.3.2. After assigning tasks to processors of type-1, A_{hvy} assigns the remaining tasks to processors of type-2 as follows. For each ℓ, starting with $\ell = L-1$ and repeatedly decreasing ℓ by one until ℓ equals 0,

1) If there are *less* than y_ℓ unassigned tasks τ_i satisfying $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ (say, n_1 tasks), then A_{hvy} assigns these n_1 tasks to type-2 processors. Then, A_{hvy} assigns $y_\ell - n_1$ other (unassigned) tasks τ_j with smaller utilization on type-2 processors (i.e., $v_j^{\text{rd}} < \epsilon(1+\epsilon)^\ell$), by favoring those with larger v_j and within these tasks that are favored, those with larger u_i are favored.
2) If there are *exactly* y_ℓ unassigned tasks τ_i satisfying $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ then all of them are assigned to type-2 processors.
3) If there are *more* than y_ℓ unassigned tasks τ_i satisfying both (i) $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ and (ii) $u_i^{\text{rd}} > 0$, then A_{hvy} declares failure with respect to this particular ordered pair of feasible configurations and moves on to the next ordered pair of feasible configurations.
4) If there are more than y_ℓ unassigned tasks τ_i satisfying $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ but not more than y_ℓ of these tasks have $u_i^{\text{rd}} > 0$, then A_{hvy} assigns y_ℓ of these tasks by favoring those with larger u_i.

Step 1.3.3. If any task τ_i remains unassigned with both $u_i^{\text{rd}} > 0$ and $v_i^{\text{rd}} > 0$, A_{hvy} declares failure with respect to this particular ordered pair of feasible configurations, and moves on to the next ordered pair of feasible configurations.

If A_{hvy} did not declare failure in any of the above steps, implying that *all* the tasks with $u_i \ge \epsilon \wedge v_i \ge \epsilon$ are assigned (and may be *few* other tasks from τ_{hvy} with $u_i \ge \epsilon \wedge v_i < \epsilon$ or $u_i < \epsilon \wedge v_i \ge \epsilon$) then algorithm A_{int} is called with the ordered pair of feasible configurations under consideration. This algorithm, A_{int}, is presented in Section V.

E. Assignment analysis

Let $\mathcal{H}_{\text{hvy}}$ denote the assignment of the heavy tasks returned by A_{hvy}. In this section, we show that in $\mathcal{H}_{\text{hvy}}$, the subset of tasks assigned to each processor consumes no more than $(1+\epsilon)$ of the capacity of that processor.

Definition 4 (The subsets Γ^1_{hvy} and Γ^2_{hvy}). *We denote by $\Gamma^1_{\text{hvy}}, \Gamma^2_{\text{hvy}} \subseteq \tau_{\text{hvy}}$ the subsets of tasks assigned to the processors of type-1 (resp., type-2) in the assignment $\mathcal{H}_{\text{hvy}}$ returned by the algorithm, A_{hvy}.*

Note: Hereafter, we use the notation τ for the subsets of tasks that we explicitly define (like τ_{hvy} and τ_{lgt}, for example), Γ for the subsets of tasks returned by the different steps of our PTAS and Φ for the subsets of tasks assigned in $\mathcal{H}_{\text{feas}}$.

We know that the ordered pair of feasible configurations $\{(x_0^{\text{feas}}, x_1^{\text{feas}}, \ldots, x_{L-1}^{\text{feas}}), (y_0^{\text{feas}}, y_1^{\text{feas}}, \ldots, y_{L-1}^{\text{feas}})\}$ corresponding to the feasible partitioning $\mathcal{H}_{\text{feas}}$ must be present in the tables constructed in Step 1.2 (in Section IV-C). Therefore, this particular ordered pair of feasible configurations (denoted by P^{feas} hereafter) will come to be considered by A_{hvy}.

Lemma 1. *If P^{feas} is the ordered pair of feasible configurations currently under consideration by A_{hvy}, then A_{hvy} successfully terminates (i.e., without declaring failure) and it holds that every task $\tau_i \in \Gamma^1_{\text{hvy}}$ can be 1:1 mapped to exactly one task τ_k that is assigned to a type-1 processor in $\mathcal{H}_{\text{feas}}$ such that $u_i \le (1+\epsilon)u_k$. An analogous property holds for the tasks in Γ^2_{hvy} (such that $v_i \le (1+\epsilon)v_k$).*

Proof: First, let us focus on the tasks in Γ^1_{hvy}. In Step 1.3.1, for each $\ell \in [0, L-1]$, it is straightforward (from the fact that we consider the ordered pair P^{feas}) to see that A_{hvy} successfully assigns *exactly* x_ℓ^{feas} tasks τ_i satisfying $\epsilon(1+\epsilon)^\ell \le u_i < \epsilon(1+\epsilon)^{\ell+1}$ to type-1 processors (through either case 1.3.1.2 or 1.3.1.3). While these may not be the same tasks as those that are assigned to these processors in $\mathcal{H}_{\text{feas}}$, the utilization of each task does not exceed that of the corresponding task assigned in $\mathcal{H}_{\text{feas}}$ by more than a factor of $(1+\epsilon)$. Hence the lemma holds for the heavy tasks in Γ^1_{hvy}.

Now, let us focus on Step 1.3.2, i.e., on the tasks in Γ^2_{hvy}. If A_{hvy} terminates without declaring failure then it means that for each $\ell \in [0, L-1]$, A_{hvy} went through either case 1.3.2.1, 1.3.2.2 or 1.3.2.4 and it is trivial to see that the lemma holds for all these cases. Indeed, for each task τ_i with $\epsilon(1+\epsilon)^\ell \le v_i < \epsilon(1+\epsilon)^{\ell+1}$ that is assigned to processors of type-2 through one of these cases, there is a task, say τ_k, also with $\epsilon(1+\epsilon)^\ell \le v_k < \epsilon(1+\epsilon)^{\ell+1}$ which is also assigned to processors of type-2 in $\mathcal{H}_{\text{feas}}$ (since we consider the ordered pair P^{feas}).

Since we have shown that the lemma holds as long as A_{hvy} does not declare failure, we now show that A_{hvy} cannot fail while considering the ordered pair P^{feas} of feasible configurations. For a failure to occur, it is necessary for A_{hvy} to go through case 1.3.2.3, i.e., there must be some $\ell \in [0, L-1]$ such that there are strictly more than y_ℓ^{feas} tasks τ_i yet unassigned, that satisfy both $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ and $u_i^{\text{rd}} > 0$. Let us consider the largest such ℓ and denote by $n_1 > y_\ell^{\text{feas}}$ the number of tasks satisfying both the aforementioned conditions.

Recall that in $\mathcal{H}_{\text{feas}}$, y_ℓ^{feas} tasks τ_i with $v_i^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ are assigned to type-2 processors. Therefore, it must be the case that in $\mathcal{H}_{\text{feas}}$, some of the $n_1 - y_\ell^{\text{feas}}$ "additional" tasks were assigned to type-1 processors. Let τ_j denote one of these additional tasks, thus satisfying $v_j^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ and $u_j^{\text{rd}} = \epsilon(1+\epsilon)^x > 0$, for some $x \in [0, L-1]$. Since this task τ_j has not been assigned yet by A_{hvy}, we know that at the time A_{hvy} was assigning tasks in Step 1.3.1 with $\ell = x$, it went through case 1.3.1.3 and instead of choosing to assign τ_j, it chose to assign another task $\tau_k \neq \tau_j$, also with $u_k^{\text{rd}} = \epsilon(1+\epsilon)^x$, that is assigned to type-2 processors in $\mathcal{H}_{\text{feas}}$. Furthermore, according to case 1.3.1.3, it must hold that $v_k^{\text{rd}} \geq v_j^{\text{rd}} = \epsilon(1+\epsilon)^\ell$. Now, two cases may arise.

1) If $v_k^{\text{rd}} = v_j^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ then τ_k is one of the y_ℓ^{feas} tasks assigned to type-2 processors in $\mathcal{H}_{\text{feas}}$ and, since A_{hvy} assigned τ_k to type-1 processors, there is a free "slot" on type-2 processors in which τ_j can fit. This contradicts our assumption that τ_j is unassigned at this time instant.
2) If $v_k^{\text{rd}} > v_j^{\text{rd}} = \epsilon(1+\epsilon)^\ell$ then τ_k is one of the y_r^{feas} tasks (with $r > \ell$) assigned to type-2 processors in $\mathcal{H}_{\text{feas}}$ and, since A_{hvy} assigned τ_k to type-1 processors, there *was* a free slot on type-2 processors in Step 1.3.2, when ℓ was equal to r. At this moment, when $\ell = r$, A_{hvy} necessarily went through case 1.3.2.1 and since this case allows tasks with smaller utilization on type-2 processors to be accommodated in unused slots that were reserved for tasks with larger utilization, τ_j must have been assigned at that moment. This contradicts our assumption that τ_j is unassigned at this time instant.

Hence, we can conclude that A_{hvy} does not declare failure for the ordered pair P^{feas} of feasible configurations and the lemma holds for every task in $\Gamma_{\text{hvy}}^1 \cup \Gamma_{\text{hvy}}^2$. ■

Definition 5 (The corresponding sets Φ_{hvy}^1 and Φ_{hvy}^2). *We define by Φ_{hvy}^1 the set of tasks assigned to type-1 processors in $\mathcal{H}_{\text{feas}}$ such that each task $\tau_k \in \Phi_{\text{hvy}}^1$ can be mapped to exactly one task $\tau_i \in \Gamma_{\text{hvy}}^1$ (bijective relation, implying $|\Phi_{\text{hvy}}^1| = |\Gamma_{\text{hvy}}^1|$) and for which $u_i \leq (1+\epsilon)u_k$. The set Φ_{hvy}^2 is defined analogously (for which $v_i \leq (1+\epsilon)v_k$)*[1].

Lemma 2. *After assigning the tasks in τ_{hvy}, we have*

$$\textstyle\sum_{\tau_i \in \Gamma_{\text{hvy}}^1} u_i \leq (1+\epsilon)m_1 \tag{4}$$

$$\textit{and} \quad \textstyle\sum_{\tau_i \in \Gamma_{\text{hvy}}^2} v_i \leq (1+\epsilon)m_2 \tag{5}$$

Proof: We show only the proof of Expression (4), as the proof of Expression (5) is quite similar. The proof is a direct consequence of Lemma 1. We know from Lemma 1 and Definition 5 that there exists a 1:1 mapping between every task τ_i in Γ_{hvy}^1 and every task $\tau_k \in \Phi_{\text{hvy}}^1$ such that $u_i \leq (1+\epsilon)u_k$. Therefore, since $|\Phi_{\text{hvy}}^1| = |\Gamma_{\text{hvy}}^1|$ (from the bijective relation between the two sets), we have:

$$\sum_{\tau_i \in \Gamma_{\text{hvy}}^1} u_i \leq (1+\epsilon) \sum_{\tau_k \in \Phi_{\text{hvy}}^1} u_k \tag{6}$$

Finally, we know from the feasibility of $\mathcal{H}_{\text{feas}}$ that $\sum_{k \in \Phi_{\text{hvy}}^1} u_k \leq m_1$ and hence $\sum_{\tau_i \in \Gamma_{\text{hvy}}^1} u_i \leq (1+\epsilon)m_1$. ■

[1]Note that Lemma 1 showed that such task sets Φ_{hvy}^1 and Φ_{hvy}^2 exist.

V. Assigning the tasks in τ_{int} (Step 2)

The tasks from τ_{hvy} that were not assigned by algorithm A_{hvy} form the set τ_{int}, i.e., $\tau_{\text{int}} = \tau_{\text{hvy}} \setminus \{\Gamma_{\text{hvy}}^1 \cup \Gamma_{\text{hvy}}^2\}$. Let us partition τ_{int} into two subsets τ_{int}^1 and τ_{int}^2 as follows:

$$\tau_{\text{int}}^1 = \{\tau_i \in \tau_{\text{int}} \mid u_i < \epsilon \text{ and } v_i \geq \epsilon\} \tag{7}$$

$$\tau_{\text{int}}^2 = \{\tau_i \in \tau_{\text{int}} \mid u_i \geq \epsilon \text{ and } v_i < \epsilon\} \tag{8}$$

A. The description of the algorithm A_{int}

The algorithm A_{int} to assign the tasks in τ_{int} is as follows:

1) Assign all the tasks in τ_{int}^1 to type-1 processors using the *wrap-around* technique. This technique works as follows. Take the first processor of type-1 and assign as many of the tasks as possible from τ_{int}^1 "integrally" onto that processor. When a task fails to be assigned integrally, assign that task "fractionally" such that the current processor is filled completely and the remaining fraction is assigned to the next processor of type-1, continue this procedure until all the tasks from τ_{int}^1 are assigned to type-1 processors.
2) Analogously, assign all the tasks in τ_{int}^2 to type-2 processors using the *wrap-around* technique.

B. Assignment analysis

We now show that for a task set τ that is feasible on a platform π, A_{int} always succeeds in assigning all the tasks in τ_{int}^1 to type-1 processors on a platform $\pi^{(1+\epsilon)}$. That is, if Γ_{int}^1 and Γ_{int}^2 denote the set of tasks assigned to type-1 and type-2 processors by A_{int}, we have $\Gamma_{\text{int}}^1 = \tau_{\text{int}}^1$ and $\Gamma_{\text{int}}^2 = \tau_{\text{int}}^2$.

In the following lemma, we make use of the fact that the two sets of tasks Γ_{hvy}^1 and Γ_{hvy}^2 have been obtained by algorithm A_{hvy}, using the ordered pair P^{feas} of feasible configurations.

Lemma 3. *After assigning all the tasks in τ_{int} using the ordered pair of feasible configuration , we have:*

$$\textstyle\sum_{\tau_i \in \Gamma_{\text{hvy}}^1} u_i + \sum_{\tau_i \in \tau_{\text{int}}^1} u_i \leq (1+\epsilon)m_1 \tag{9}$$

$$\textit{and} \quad \textstyle\sum_{\tau_i \in \Gamma_{\text{hvy}}^2} v_i + \sum_{\tau_i \in \tau_{\text{int}}^2} v_i \leq (1+\epsilon)m_2 \tag{10}$$

Proof: In the feasible assignment $\mathcal{H}_{\text{feas}}$, $|\tau_{\text{int}}^1|$ number of tasks with $u_i < \epsilon \wedge v_i \geq \epsilon$ must have been assigned to type-1 processors. This is a consequence of the fact that P^{feas} contains exactly the same number of tasks with utilization $\geq \epsilon$ on the processor that they are assigned to, as in $\mathcal{H}_{\text{feas}}$. Let Φ_{int}^1 denote the set of tasks with $u_i < \epsilon \wedge v_i \geq \epsilon$ that are assigned to type-1 processors in $\mathcal{H}_{\text{feas}}$. Since $\mathcal{H}_{\text{feas}}$ is a feasible assignment, it holds that,

$$\sum_{\tau_i \in \Phi_{\text{hvy}}^1} u_i + \sum_{\tau_i \in \Phi_{\text{int}}^1} u_i \leq m_1 \tag{11}$$

Since the number of tasks with $u_i < \epsilon \wedge v_i \geq \epsilon$ that have been assigned to type-1 processors is same in both $\mathcal{H}_{\text{feas}}$ and the assignment computed by our algorithm, we have $|\tau_{\text{int}}^1| = |\Phi_{\text{int}}^1| = |\Gamma_{\text{int}}^1|$. Here, it is worth recalling Step 1.3.1.3 and Step 1.3.2.4 of algorithm A_{hvy}. In these steps, while assigning the tasks to processors of type-1 (resp., type-2), when A_{hvy} has to choose few tasks to assign from the available set of tasks, it always chooses those tasks that have a larger utilization on type-2 (resp., type-1) processors (leaving "easier" tasks for $\mathrm{A}_{\mathrm{int}}$ to assign). Now coming back to algorithm $\mathrm{A}_{\mathrm{int}}$, although the tasks (with $u_i < \epsilon \wedge v_i \geq \epsilon$) assigned by $\mathrm{A}_{\mathrm{int}}$ to type-1 processors may not be the *same* as those assigned by $\mathcal{H}_{\mathrm{feas}}$, we can infer that:

$$\sum_{\tau_i \,\in\, \tau^1_{\mathrm{int}}} u_i \leq \sum_{\tau_i \,\in\, \Phi^1_{\mathrm{int}}} u_i \tag{12}$$

Applying Inequality (6) and (12) on (11) and then performing some arithmetic manipulations (see [15] for details), we get:

$$\sum_{\tau_i \,\in\, \Gamma^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \,\in\, \tau^1_{\mathrm{int}}} u_i \leq (1+\epsilon) \times m_1$$

Using similar reasoning as above, we can show that Expression (10) holds as well. Hence the proof. ■

Corollary 1. *After assigning the tasks in τ_{int}, we have:*

$$\sum_{\tau_i \,\in\, \Gamma^1_{\mathrm{hvy}} \cup \Gamma^1_{\mathrm{int}}} u_i \leq (1+\epsilon) \sum_{\tau_i \,\in\, \Phi^1_{\mathrm{hvy}} \cup \Phi^1_{\mathrm{int}}} u_i \tag{13}$$

and

$$\sum_{\tau_i \,\in\, \Gamma^2_{\mathrm{hvy}} \cup \Gamma^2_{\mathrm{int}}} v_i \leq (1+\epsilon) \sum_{\tau_i \,\in\, \Phi^2_{\mathrm{hvy}} \cup \Phi^2_{\mathrm{int}}} v_i \tag{14}$$

Proof: Inequality (13) follows from Expressions (6) and (12) (since $\Gamma^1_{\mathrm{int}} = \tau^1_{\mathrm{int}}$) and Inequality (14) can be inferred from analogous expressions for type-2 processors. ■

VI. Assigning the tasks in τ_{lgt} (Step 3)

Let us partition τ_{lgt} into τ^1_{lgt} and τ^2_{lgt} as follows:

$$\tau^1_{\mathrm{lgt}} = \{\tau_i \in \tau_{\mathrm{lgt}} \mid u_i \leq v_i\} \tag{15}$$

$$\tau^2_{\mathrm{lgt}} = \{\tau_i \in \tau_{\mathrm{lgt}} \mid u_i > v_i\} \tag{16}$$

A. The description of the algorithm $\mathrm{A}_{\mathrm{lgt}}$

The pseudo-code for assigning tasks in τ_{lgt} is shown in Algorithm 1 (which uses the `fract-next-fit` subroutine shown in Algorithm 2). The intuition behind the design of this algorithm is that, assuming a platform, $\pi^{(1+2\epsilon)}$, first we assign tasks to processors on which they have a smaller utilization (lines 1 and 2). Then, if there are remaining tasks, these have to be assigned to processors on which they have a larger utilizations (lines 7 and 15).

B. Assignment analysis

First, we present some useful result in Lemma 4, obtained by relating the problem under consideration to the *fractional knapsack problem* (see Chapter 16.2 in [16]). This result will be used in Lemma 5. The relation between the fractional knapsack problem and the problem under consideration was explored in [13]. Lemma 4 is an adaptation of Lemma 5 in [13]. Hence, we only state the lemma here. The detailed description of the fractional knapsack problem, its relation with the task assignment problem and the proof of Lemma 4 can be found in Appendix A in [15].

[2]While assigning tasks to type-1 processors, if a task cannot be assigned integrally on m_1'th processor (the last processor of type-1), then assign a fraction of that task such that m_1'th processor is fully utilized and assign the rest of the fraction to m_2'th processor (the last processor of type-2). This task is denoted by τ_f later in the proofs — in Section VII. This is not shown in the pseudo-code explicitly for ease of representation.

Algorithm 1: $\mathrm{A}_{\mathrm{lgt}}$: An algorithm to assign τ_{lgt} tasks

1 $\Gamma^1_{\mathrm{lgt}^1}$:= *fract-next-fit*$(\tau^1_{\mathrm{lgt}}, \mathrm{m}_1)$
2 $\Gamma^2_{\mathrm{lgt}^2}$:= *fract-next-fit*$(\tau^2_{\mathrm{lgt}}, \mathrm{m}_2)$
3 **if** $(\Gamma^1_{\mathrm{lgt}^1} = \tau^1_{\mathrm{lgt}} \wedge \Gamma^2_{\mathrm{lgt}^2} = \tau^2_{\mathrm{lgt}})$ **then** declare SUCCESS
4 **if** $(\Gamma^1_{\mathrm{lgt}^1} \neq \tau^1_{\mathrm{lgt}} \wedge \Gamma^2_{\mathrm{lgt}^2} \neq \tau^2_{\mathrm{lgt}})$ **then** declare FAILURE
5 **if** $(\Gamma^1_{\mathrm{lgt}^1} \neq \tau^1_{\mathrm{lgt}} \wedge \Gamma^2_{\mathrm{lgt}^2} = \tau^2_{\mathrm{lgt}})$ **then**
6     $\Gamma^2_{\mathrm{lgt}^1} := \tau^1_{\mathrm{lgt}} \setminus \Gamma^1_{\mathrm{lgt}^1}$
7   **if** (*fract-next-fit*$(\Gamma^2_{\mathrm{lgt}^1}, \mathrm{m}_2) = \Gamma^2_{\mathrm{lgt}^1}$) **then**
8   declare SUCCESS
9   **else**
10   declare FAILURE
11   **end**
12 **end**
13 **if** $(\Gamma^1_{\mathrm{lgt}^1} = \tau^1_{\mathrm{lgt}} \wedge \Gamma^2_{\mathrm{lgt}^2} \neq \tau^2_{\mathrm{lgt}})$ **then**
14     $\Gamma^1_{\mathrm{lgt}^2} := \tau^2_{\mathrm{lgt}} \setminus \Gamma^2_{\mathrm{lgt}^2}$
15   **if** (*fract-next-fit*$(\Gamma^1_{\mathrm{lgt}^2}, \mathrm{m}_1) = \Gamma^1_{\mathrm{lgt}^2}$) **then**
16   declare SUCCESS
17   **else**
18   declare FAILURE
19   **end**
20 **end**

Algorithm 2: fract-next-fit(ts, ps): Next-fit bin-packing with fractional assignment of tasks

Input : ts: set of tasks; ps: set of processors
Output: set of tasks that were assigned successfully
If ps consists of type-1 (resp., type-2) processors, then sort ts by decreasing v_i/u_i (resp., increasing v_i/u_i). Use any order for processors ps, but maintain it during the execution of *fract-next-fit*.
Assign tasks using *wrap-around* technique[2].
Return the set of successfully assigned tasks.

Lemma 4. *Consider a task set T and a two-type platform conforming to the system model of Section II. Let us partition T into two disjoint subsets, T^1 and T^2 as follows:*

$$T = T^1 \cup T^2 \tag{17}$$

$$\forall \tau_i \in T^1 : u_i \leq v_i \tag{18}$$

$$\forall \tau_i \in T^2 : u_i > v_i \tag{19}$$

Let x denote a real number such that $0 \leq x \leq m_1$.

Let $A1$ denote a subset of T^1 such that $\sum_{\tau_i \in A1} u_i > m_1 - x$ and for every pair of tasks $\tau_i \in A1$ and $\tau_j \in T^1 \setminus A1$ it holds that $\frac{v_i}{u_i} - 1 \geq \frac{v_j}{u_j} - 1$.

Let $A2$ denote $T^1 \setminus A1$.

Let $B1$ denote a subset of T^1 such that $\sum_{\tau_i \in B1} u_i \leq m_1 - x$.

Let $B2$ denote $\tau \setminus B1$. It then holds that:

$$\sum_{\tau_i \,\in\, A1} u_i + \sum_{\tau_i \,\in\, A2} v_i + \sum_{\tau_i \,\in\, T^2} v_i \leq \sum_{\tau_i \,\in\, B1} u_i + \sum_{\tau_i \,\in\, B2} v_i \tag{20}$$

Lemma 5. *Let Γ^1_{lgt} and Γ^2_{lgt} be the subset of tasks from τ_{lgt} that are assigned by $\mathrm{A}_{\mathrm{lgt}}$ to type-1 and type-2 processors, respectively. After assigning all the tasks from τ_{lgt}, we have:*

$$\textstyle\sum_{\tau_i \in \Gamma^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Gamma^1_{\mathrm{int}}} u_i + \sum_{\tau_i \in \Gamma^1_{\mathrm{lgt}}} u_i \leq (1+2\epsilon) m_1 \tag{21}$$

and

$$\textstyle\sum_{\tau_i \in \Gamma^2_{\mathrm{hvy}}} v_i + \sum_{\tau_i \in \Gamma^2_{\mathrm{int}}} v_i + \sum_{\tau_i \in \Gamma^2_{\mathrm{lgt}}} v_i \leq (1+2\epsilon) m_2 \tag{22}$$

where

1) *all the tasks in $\tau_{\mathrm{hvy}} \setminus \tau_{\mathrm{int}}$ are assigned integrally*

2) *some tasks in* τ_{int} *are assigned fractionally and the rest are assigned integrally*
3) *some tasks in* τ_{lgt} *are assigned fractionally and the rest are assigned integrally*

Proof: Informally, the claim can be written as follows: if there exists a feasible partitioning for a task set τ on a two-type platform π then algorithms A_{hvy}, A_{int} and A_{lgt} succeed in assigning the tasks in τ on a platform $\pi^{(1+2\epsilon)}$, with some tasks assigned fractionally. We already know from Lemma 3 that after assigning the tasks in $\tau_{\text{hvy}} \setminus \tau_{\text{int}}$ and τ_{int} using algorithms A_{hvy} and A_{int}, respectively, the sum of the utilizations of the tasks assigned on type-1 (resp., type-2) processors does not exceed $(1+\epsilon)m_1$ (resp., $(1+\epsilon)m_2$).

Therefore, we need to show that after assigning the tasks in τ_{lgt} by using algorithm A_{lgt}, the sum of the utilizations of the tasks assigned on processors of type-1 (resp., type-2) does not exceed $(1+2\epsilon)m_1$ (resp., $(1+2\epsilon)m_2$). An equivalent claim is that, after assigning tasks in $\tau_{\text{hvy}} \setminus \tau_{\text{int}}$ and τ_{int} by using algorithms A_{hvy} and A_{int} respectively, if A_{lgt} fails to assign the tasks of τ_{lgt} (with fractional assignment of tasks allowed) on platform $\pi^{(1+2\epsilon)}$ then there does not exist a feasible partitioning of the tasks in τ on platform π. Here, we prove this equivalent claim by contradiction. Assume that there exists a feasible assignment $\mathcal{H}_{\text{feas}}$ of τ on π but A_{lgt} fails to assign the tasks in τ_{lgt} on $\pi^{(1+2\epsilon)}$ (after A_{hvy} and A_{int} successfully assigned the tasks of $\tau_{\text{hvy}} \setminus \tau_{\text{int}}$ and τ_{int}). Since A_{lgt} failed to assign these tasks, it must have declared FAILURE and we explore all possibilities for this to occur:

Failure on line 4 in Algorithm 1: From the case, we have $\Gamma^1_{\text{lgt}^1} \subset \tau^1_{\text{lgt}}$ and $\Gamma^2_{\text{lgt}^2} \subset \tau^2_{\text{lgt}}$. Therefore, when executing line 1 in A_{lgt} there was a task $\tau_{f_1} \in \tau^1_{\text{lgt}} \setminus \Gamma^1_{\text{lgt}^1}$ which could not be assigned to type-1 processors and similarly, when executing line 2 in A_{lgt} there was a task $\tau_{f_2} \in \tau^2_{\text{lgt}} \setminus \Gamma^2_{\text{lgt}^2}$ which could not be assigned to type-2 processors. Hence, we have:

$$\sum_{p \in P^1} U[p] + u_{f_1} > m_1(1+2\epsilon) = m_1 + 2m_1\epsilon \tag{23}$$

$$\text{and} \quad \sum_{p \in P^2} U[p] + v_{f_2} > m_2(1+2\epsilon) = m_2 + 2m_2\epsilon \tag{24}$$

where P^1 and P^2 denote the set of type-1 and type-2 processors respectively and $U[p]$ denotes the sum of the utilization of the tasks assigned on processor p.

Since $\tau_{f_1} \in \tau^1_{\text{lgt}} \overset{(15)}{\Rightarrow} \tau_{f_1} \in \tau_{\text{lgt}} \overset{(2)}{\Rightarrow} u_{f_1} < \epsilon \leq m_1\epsilon$ and analogously since $\tau_{f_2} \in \tau^2_{\text{lgt}}$, we know that $v_{f_2} < \epsilon \leq m_2\epsilon$. Using these on Expressions (23) and (24), we get

$$\sum_{p \in P^1} U[p] > m_1(1+\epsilon) \tag{25}$$

$$\text{and} \quad \sum_{p \in P^2} U[p] > m_2(1+\epsilon) \tag{26}$$

Observe that (i) the set of tasks that has been assigned on type-1 processors so far is $\Gamma^1_{\text{hvy}} \cup \Gamma^1_{\text{int}}$ and a strict subset of τ^1_{lgt}, and (ii) the set of tasks assigned on type-2 processors is $\Gamma^2_{\text{hvy}} \cup \Gamma^2_{\text{int}}$ and a strict subset of τ^2_{lgt}. Therefore, it holds from Expression (25) and (26) that:

$$\sum_{\tau_i \in \Gamma^1_{\text{hvy}} \cup \Gamma^1_{\text{int}}} u_i + \sum_{\tau_i \in \tau^1_{\text{lgt}}} u_i > m_1(1+\epsilon) \tag{27}$$

$$\sum_{\tau_i \in \Gamma^2_{\text{hvy}} \cup \Gamma^2_{\text{int}}} v_i + \sum_{\tau_i \in \tau^2_{\text{lgt}}} v_i > m_2(1+\epsilon) \tag{28}$$

Applying Expression (13) and (14) on Expression (27) and (28) respectively, performing some arithmetic manipulations and summing the resulting expressions (see [15] for details) yields:

$$\sum_{\tau_i \in \Phi^1_{\text{hvy}} \cup \Phi^1_{\text{int}} \cup \tau^1_{\text{lgt}}} u_i + \sum_{\tau_i \in \Phi^2_{\text{hvy}} \cup \Phi^2_{\text{int}} \cup \tau^2_{\text{lgt}}} v_i > m_1 + m_2 \tag{29}$$

It is trivial to see that assigning all the tasks of τ^1_{lgt} and τ^2_{lgt} to type-1 and type-2 processors, respectively (as in the above expression), requires the minimum processing capacity. Hence, Expression (29) continues to hold for any other assignment of these tasks, implying that $\mathcal{H}_{\text{feas}}$ cannot be a feasible assignment, which leads to a contradiction.

Failure on line 10 in Algorithm 1: From the case, we have $\Gamma^1_{\text{lgt}^1} \subset \tau^1_{\text{lgt}}$ and $\Gamma^2_{\text{lgt}^2} = \tau^2_{\text{lgt}}$. Therefore, when executing line 7 in A_{lgt} there was a task $\tau_f \in \tau^1_{\text{lgt}} \setminus \Gamma^1_{\text{lgt}^1}$ which was attempted on type-2 processors but failed. Hence, we have:

$$\sum_{p \in P^2} U[p] + v_f > m_2(1+2\epsilon) \tag{30}$$

We know that the tasks assigned to type-2 processors at this stage are $\Gamma^2_{\text{hvy}} \cup \Gamma^2_{\text{int}} \cup \Gamma^2_{\text{lgt}^2}$ and a strict subset of tasks from $\Gamma^2_{\text{lgt}^1}$ (line 7). Therefore, we can rewrite Expression (30) as:

$$\sum_{\tau_i \in \Gamma^2_{\text{hvy}} \cup \Gamma^2_{\text{int}} \cup \Gamma^2_{\text{lgt}^2} \cup \Gamma^2_{\text{lgt}^1}} v_i > m_2(1+2\epsilon) - v_f \tag{31}$$

Since $\tau_f \in \tau^1_{\text{lgt}} \setminus \Gamma^1_{\text{lgt}^1}$, we know that $v_f < \epsilon \leq m_2\epsilon$. Using this on Expression (31), then applying Expression (14) and finally performing some arithmetic manipulations (see [15] for details) gives us:

$$\sum_{\tau_i \in \Phi^2_{\text{hvy}} \cup \Phi^2_{\text{int}}} v_i + \sum_{\tau_i \in \Gamma^2_{\text{lgt}^2} \cup \Gamma^2_{\text{lgt}^1}} v_i > m_2 \tag{32}$$

We also know that, when A_{lgt} executed line 1 (where it performed *fract-next-fit*), there must have been a task $\tau_{f_1} \in \tau^1_{\text{lgt}} \setminus \Gamma^1_{\text{lgt}^1}$ which was attempted on type-1 processors but failed to be assigned. Note that this task τ_{f_1} may be the same as τ_f mentioned above or it may be different. Because it was not possible to assign τ_{f_1} on type-1 processors, we know that:

$$\sum_{p \in P^1} U[p] + u_{f_1} > m_1(1+2\epsilon) \tag{33}$$

We know that the tasks assigned to type-1 processors are $\Gamma^1_{\text{hvy}} \cup \Gamma^1_{\text{int}} \cup \Gamma^1_{\text{lgt}^1}$ and thus, we rewrite Expression (33) as:

$$\sum_{\tau_i \in \Gamma^1_{\text{hvy}} \cup \Gamma^1_{\text{int}} \cup \Gamma^1_{\text{lgt}^1}} u_i > m_1(1+2\epsilon) - u_{f_1} \tag{34}$$

Since $\tau_{f_1} \in \tau^1_{\text{lgt}} \setminus \Gamma^1_{\text{lgt}^1}$, we have $u_{f_1} < \epsilon \leq 2\epsilon$. Using this on Expression (34), then applying Expression (13) and finally performing some arithmetic manipulations (see [15] for details) gives us:

$$\sum_{\tau_i \in \Phi^1_{\text{hvy}} \cup \Phi^1_{\text{int}}} u_i + \sum_{\tau_i \in \Gamma^1_{\text{lgt}^1}} u_i > m_1 \tag{35}$$

Finally, Expression (35) can be rewritten as:

$$\sum_{\tau_i \in \Gamma^1_{\mathrm{lgt}^1}} u_i > m_1 - \left(\sum_{\tau_i \in \Phi^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{int}}} u_i \right) \tag{36}$$

Let us now discuss the feasible assignment $\mathcal{H}_{\mathrm{feas}}$. Let Φ^1_{lgt} denote the set of tasks assigned to type-1 processors in $\mathcal{H}_{\mathrm{feas}}$, excluding those in $\Phi^1_{\mathrm{hvy}} \cup \Phi^1_{\mathrm{int}}$. Similarly, let Φ^2_{lgt} denote the set of tasks assigned to type-2 processors in $\mathcal{H}_{\mathrm{feas}}$, excluding those in $\Phi^2_{\mathrm{hvy}} \cup \Phi^2_{\mathrm{int}}$. Since, by assumption, $\mathcal{H}_{\mathrm{feas}}$ succeeds in assigning all the tasks in τ to the processors, it holds that:

$$\sum_{\tau_i \in \Phi^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{int}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{lgt}}} u_i \leq m_1 \tag{37}$$

$$\text{and} \sum_{\tau_i \in \Phi^2_{\mathrm{hvy}}} v_i + \sum_{\tau_i \in \Phi^2_{\mathrm{int}}} v_i + \sum_{\tau_i \in \Phi^2_{\mathrm{lgt}}} v_i \leq m_2 \tag{38}$$

Expression (37) can be rewritten as:

$$\sum_{\tau_i \in \Phi^1_{\mathrm{lgt}}} u_i \leq m_1 - \left(\sum_{\tau_i \in \Phi^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{int}}} u_i \right) \tag{39}$$

We can now reason about the inequalities we obtained about the assignment $\mathcal{H}_{\mathrm{feas}}$ and the one constructed by $\mathrm{A}_{\mathrm{lgt}}$. We can see that Expressions (36) and (39), with $x = \sum_{\tau_i \in \Phi^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{int}}} u_i$, ensure that the assumptions of Lemma 4 are true, given the ordering of tasks in τ^1_{lgt} during assignment over type-1 processors (line 1 in Algorithm 2), which ensures that $\forall \tau_i \in \Gamma^1_{\mathrm{lgt}^1}, \forall \tau_j \in \Gamma^2_{\mathrm{lgt}^1}$: $\frac{v_i}{u_i} \geq \frac{v_j}{u_j}$. By applying Lemma 4 with the following input:

- $T = \tau \setminus (\Phi^1_{\mathrm{hvy}} \cup \Phi^2_{\mathrm{hvy}} \cup \Phi^1_{\mathrm{int}} \cup \Phi^2_{\mathrm{int}})$,
- $T^1 = \tau^1_{\mathrm{lgt}}$, $T^2 = \tau^2_{\mathrm{lgt}} = \Gamma^2_{\mathrm{lgt}^2}$,
- $x = \sum_{\tau_i \in \Phi^1_{\mathrm{hvy}}} u_i + \sum_{\tau_i \in \Phi^1_{\mathrm{int}}} u_i$,
- $A1$ is $\Gamma^1_{\mathrm{lgt}^1}$; $\overset{(36)}{\Rightarrow} \sum_{\tau_i \in A1} u_i > m_1 - x$,
- $A2$ is $\Gamma^2_{\mathrm{lgt}^1}$; Note that for every pair of tasks $\tau_i \in A1$ and $\tau_j \in A2$ it holds that $\frac{v_i}{u_i} - 1 \geq \frac{v_j}{u_j} - 1$,
- $B1$ is Φ^1_{lgt}; $\overset{(39)}{\Rightarrow} \sum_{\tau_i \in B1} u_i \leq m_1 - x$,
- $B2$ is Φ^2_{lgt}.

we get:

$$\sum_{\tau_i \in \Gamma^1_{\mathrm{lgt}^1}} u_i + \sum_{\tau_i \in \Gamma^2_{\mathrm{lgt}^1}} v_i + \sum_{\tau_i \in \Gamma^2_{\mathrm{lgt}^2}} v_i \leq \sum_{\tau_i \in \Phi^1_{\mathrm{lgt}}} u_i + \sum_{\tau_i \in \Phi^2_{\mathrm{lgt}}} v_i$$

Adding $\sum_{\tau_i \in \Phi^1_{\mathrm{hvy}} \cup \Phi^1_{\mathrm{int}}} u_i + \sum_{\tau_i \in \Phi^2_{\mathrm{hvy}} \cup \Phi^2_{\mathrm{int}}} v_i$ to both the sides in the above inequality, then applying Expressions (37) and (38) to the right-hand side and then applying Expressions (32) and (35) to the left-hand side yields:

$$m_1 + m_2 < m_1 + m_2$$

This is a contradiction.

Failure on line 18 in Algorithm 1: A contradiction results — proof analogous to the previous case.

We showed that all the cases where $\mathrm{A}_{\mathrm{lgt}}$ declares FAILURE lead to a contradiction. Hence, the lemma holds. ■

VII. Integral assignment of τ_{int} and τ_{lgt} (Step 4)

We now discuss how to integrally assign the tasks from τ_{int} and τ_{lgt} that were fractionally assigned by algorithms $\mathrm{A}_{\mathrm{int}}$ and $\mathrm{A}_{\mathrm{lgt}}$, respectively. We also show that, if there is a feasible partitioning of the given task set on a given two-type platform then our PTAS succeeds in finding such a feasible partitioning on a platform in which each processor is $1 + 3\epsilon$ times faster.

A. The description of the algorithm $\mathrm{A}_{\mathrm{fract}}$

The algorithm, $\mathrm{A}_{\mathrm{fract}}$, works as follows:

1) copy the assignment (made by $\mathrm{A}_{\mathrm{hvy}}$, $\mathrm{A}_{\mathrm{int}}$ and $\mathrm{A}_{\mathrm{lgt}}$ on $\pi^{(1+2\epsilon)}$) onto a faster platform $\pi^{(1+3\epsilon)}$.
2) on this platform, $\pi^{(1+3\epsilon)}$, assign the task split between any two processors p_1 and p_1+1 of type-1 entirely on to processor p_1, where $1 \leq p_1 < m_1$; similarly, assign the task split between any two processors p_2 and $p_2 + 1$ of type-2 entirely on to processor p_2, where $1 \leq p_2 < m_2$.
3) assign the task split between m_1'th processor of type-1 and m_2'th processor of type-2 to any of these processors.

B. Assignment analysis

Theorem 1. *If there exists a feasible partitioning of τ on π then our PTAS algorithm,* $\mathrm{PTAS}_{\mathrm{NF}}$*, (which uses $\mathrm{A}_{\mathrm{hvy}}$, $\mathrm{A}_{\mathrm{int}}$, $\mathrm{A}_{\mathrm{lgt}}$ and $\mathrm{A}_{\mathrm{fract}}$ in sequence) succeeds in finding a feasible partitioning of τ on $\pi^{(1+3\epsilon)}$.*

Proof: We know from Lemma 5 that if there exists a feasible partitioning of τ on π then the three algorithms $\mathrm{A}_{\mathrm{hvy}}$, $\mathrm{A}_{\mathrm{int}}$ and $\mathrm{A}_{\mathrm{lgt}}$ described in Sections IV-VI succeed in assigning tasks in τ (with a subset of tasks from τ_{int} and τ_{lgt} fractionally assigned) on $\pi^{(1+2\epsilon)}$. As a consequence, we have:

$$\forall p \in \pi^{(1+2\epsilon)} : U[p] \leq 1 + 2\epsilon \tag{40}$$

We also know that in such an assignment (as a consequence of using the wrap-around technique in $\mathrm{A}_{\mathrm{int}}$ and $\mathrm{A}_{\mathrm{lgt}}$):

- at most $m_1 - 1$ tasks are *split* between processors of type-1 with one task split between each pair of consecutive processors; let the set $\Gamma^1_{\mathrm{split}}$ denote these fractional tasks.
- at most $m_2 - 1$ tasks are *split* between processors of type-2 with one task split between each pair of consecutive processors; let the set $\Gamma^2_{\mathrm{split}}$ denote these fractional tasks.
- at most one task (from τ_{lgt}) is *split* between processors of type-1 and type-2; let $\tau_f \in \tau_{\mathrm{lgt}}$ denote this task that must be split between the m_1'th processor of type-1 and the m_2'th processor of type-2.
- the rest of the tasks are integrally assigned to either type-1 or type-2 processors.

Let $\tau^1_{p_1,p_1+1} \in \Gamma^1_{\mathrm{split}}$ denote the task split between the p_1'th and the (p_1+1)'th processors of type-1 where $1 \leq p_1 < m_1$. Analogously, let $\tau^2_{p_2,p_2+1} \in \Gamma^2_{\mathrm{split}}$ denote the task split between the p_2'th and the (p_2+1)'th processors of type-2 where $1 \leq p_2 < m_2$.

To prove the theorem, we need to show that $\mathrm{A}_{\mathrm{fract}}$ succeeds in integrally assigning all the fractional tasks on $\pi^{(1+3\epsilon)}$.

On Step 1, $\mathrm{A}_{\mathrm{fract}}$ copies the assignment from $\pi^{(1+2\epsilon)}$ onto a faster platform $\pi^{(1+3\epsilon)}$. After this step,

$$\forall p \in \pi^{(1+3\epsilon)} : U[p] \leq 1 + 2\epsilon \tag{41}$$

Since $\Gamma^1_{\text{split}} \subseteq \{\tau^1_{\text{int}} \cup \tau_{\text{lgt}}\}$, $\Gamma^2_{\text{split}} \subseteq \{\tau^2_{\text{int}} \cup \tau_{\text{lgt}}\}$, we have:

$$(2),(7) \Rightarrow \forall \tau_i \in \Gamma^1_{\text{split}} : u_i < \epsilon \quad (42)$$
$$(2),(8) \Rightarrow \forall \tau_i \in \Gamma^2_{\text{split}} : v_i < \epsilon \quad (43)$$

On Step 2, $\mathrm{A}_{\text{fract}}$ assigns the split tasks integrally. So, $\forall p_1 \in$ type-1 of $\pi^{(1+3\epsilon)}$, it moves the fraction of the 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 (41) and (42) that:

$$\forall p_1 \in \text{type-1 of } \pi^{(1+3\epsilon)} \ \wedge \ p_1 \neq m_1 : \quad U[p_1] \leq 1+3\epsilon \quad (44)$$
$$\text{if } p_1 \in \text{type-1 of } \pi^{(1+3\epsilon)} \ \wedge \ p_1 = m_1 : \quad U[p_1] \leq 1+2\epsilon \quad (45)$$

Analogously, $\forall p_2 \in$ type-2 of $\pi^{(1+3\epsilon)}$, it moves the fraction of the 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 (41) and (43) that:

$$\forall p_2 \in \text{type-2 of } \pi^{(1+3\epsilon)} \ \wedge \ p_2 \neq m_2 : \quad U[p_2] \leq 1+3\epsilon \quad (46)$$
$$\text{if } p_2 \in \text{type-2 of } \pi^{(1+3\epsilon)} \ \wedge \ p_2 = m_2 : \quad U[p_2] \leq 1+2\epsilon \quad (47)$$

Finally, the task τ_f that is split between the m_1'th processor of type-1 and the m_2'th processor of type-2 remains to be integrally assigned. Since $\tau_f \in \tau_{\text{lgt}}$, it holds that $u_f < \epsilon$ and $v_f < \epsilon$. From Expression (45) and (47), it follows that task τ_f can be *integrally* assigned to either m_1'th or m_2'th processor. Hence, after integrally assigning this task, we obtain:

$$\forall p \in \pi^{(1+3\epsilon)} : U[p] \leq 1+3\epsilon \quad (48)$$

Since Expression (48) is a necessary and sufficient schedulability condition for EDF on a uniprocessor of capacity $1+3\epsilon$, the assignment of τ on $\pi^{(1+3\epsilon)}$ returned by our algorithm, $\mathrm{PTAS_{NF}}$, is a feasible assignment. Hence, the proof. ■

VIII. Experimental setup and results

After studying the theoretical (worst-case) bound, i.e., the approximation ratio of our algorithm, $\mathrm{PTAS_{NF}}$, we evaluate its average-case performance and compare it with prior state-of-the-art, i.e., $\mathrm{PTAS_{LP}}$. For this purpose, we look at the following aspects: (i) how much faster processors our algorithm needs *in practice* in order to successfully partition a task set compared to $\mathrm{PTAS_{LP}}$? and (ii) how fast our algorithm runs compared to $\mathrm{PTAS_{LP}}$? Also, we look at (iii) how much pessimism is there in our theoretically derived performance bound? In order to answer these questions, we performed two sets of experiments. The first set of experiments described in Section VIII-A addresses (i) and (ii) and the second set of experiments described in Section VIII-B addresses (iii).

A. Comparison with prior state-of-the-art

We compare the average-case performance of $\mathrm{PTAS_{NF}}$ with $\mathrm{PTAS_{LP}}$. We implemented both the algorithms in C on an Intel Core2 (2.80 GHz) machine. For $\mathrm{PTAS_{LP}}$, we used a state-of-the-art LP solver, IBM ILOG CPLEX [17].

For a given task set, we define the *minimum required speedup factor*, $\mathrm{MRSF_{NF}}$, of $\mathrm{PTAS_{NF}}$ as the *minimum* amount of extra speed of processors that $\mathrm{PTAS_{NF}}$ needs, so as to succeed in finding a feasible partitioning as compared to an optimal algorithm. We define $\mathrm{MRSF_{LP}}$ of $\mathrm{PTAS_{LP}}$ analogously. For different values of ϵ, we assess the average-case performance of these algorithms by measuring their (i) minimum required speedup factors and (ii) running times.

The problem instances (number of tasks, their utilizations and number of processors of each type) were generated randomly. Each problem instance had at most 10 tasks and at most 2 processors of each type. For performing fair evaluation, we convert the randomly generated task sets into *critically feasible* task sets — more details in Appendix B in [15]. A task set is termed critically feasible if it is feasible on a given two-type platform but rendered infeasible if all u_i and v_i are increased by an arbitrarily small factor. The intuition behind using critically feasible task sets in our simulations is that it is "hard" to find a feasible partitioning for these task sets since only a few assignments are feasible among all the possible assignments. Hence, by using such task sets, we believe our evaluations have been fair and unbiased.

We ran $\mathrm{PTAS_{NF}}$ and $\mathrm{PTAS_{LP}}$ on 5000 critically feasible task sets and for each task set, we obtain $\mathrm{MRSF_{NF}}$ and $\mathrm{MRSF_{LP}}$ as follows. We initially set the speedup factor to 1.0 and input the task set to the algorithm. If the algorithm cannot find a feasible mapping, we increment the speedup factor by a small value, i.e., by 0.01, and divide the original utilizations, u_i and v_i, of each task by the new speedup factor and feed the resulting task set to the algorithm. These steps (adjust the speedup factor and feed back the derived task set) are repeated till the algorithm succeeds, which gives us the MRSF of the algorithm for the task set. This entire procedure is repeated for 5000 critically feasible task sets.

With this procedure, we obtain the histograms of MRSFs for both the algorithms for different values of ϵ. Figure 1 shows the histogram for $\epsilon = 0.2$ (note that the y-axis is in log scale). As can be seen, the $\mathrm{MRSF_{NF}}$ never exceeded 1.12 which is 20% from the optimal value of 1.0 compared to its upper bound of $1+3\epsilon = 1.60$, i.e., $\frac{1.12-1.0}{1.6-1.0} \times 100 = 20\%$, whereas $\mathrm{MRSF_{LP}}$ is as high as 1.28 which is 56% from the optimal value of 1.0 compared to its upper bound of $\frac{1+\epsilon}{1+\epsilon} = 1.50$, i.e., $\frac{1.28-1.0}{1.5-1.0} \times 100 = 56\%$. Therefore, $\mathrm{PTAS_{NF}}$ requires much smaller processor speedup on an average than $\mathrm{PTAS_{LP}}$ in order to find a feasible partitioning. The observations for other values of ϵ follow the same trend — see Appendix B in [15].

We also measure the average running times of both the algorithms for different values of ϵ. In these experiments, the speedup factor is set to $1+3\epsilon$ for $\mathrm{PTAS_{NF}}$ and to $\frac{1+\epsilon}{1+\epsilon}$ for $\mathrm{PTAS_{LP}}$. This ensures that both the algorithms succeed in finding a feasible partitioning for a given task set in a *single run* and hence the experiments are not biased to give advantage to any of them. In our experiments with 5000 critically feasible task sets, as can be seen in Table I, for $\epsilon = 0.1$, $\mathrm{PTAS_{NF}}$ runs approximately 50000 times faster compared to $\mathrm{PTAS_{LP}}$. This factor is even higher for other values of ϵ.

To summarize, our algorithm exhibits a better average-case performance by requiring significantly smaller processor speedup for finding a feasible partitioning and by running orders of magnitude faster compared to $\mathrm{PTAS_{LP}}$. Overall, $\mathrm{PTAS_{NF}}$ outperforms prior state-of-the-art, $\mathrm{PTAS_{LP}}$.

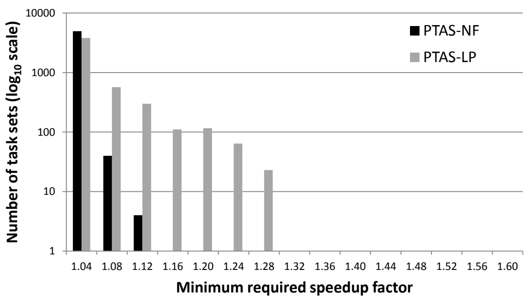

Fig. 1: Comparison of *minimum required speedup factor* of $\mathrm{PTAS_{NF}}$ and $\mathrm{PTAS_{LP}}$ for $\epsilon = 0.2$ (if an algorithm has low MRSF for many task sets then the algorithm is said to perform well).

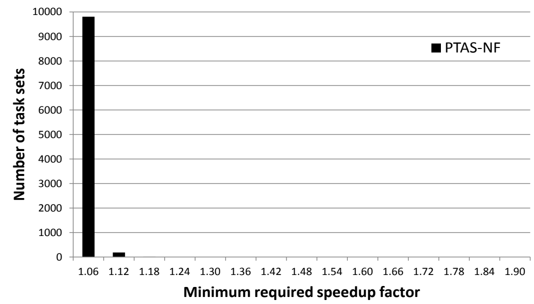

Fig. 2: Performance evaluation of $\mathrm{PTAS_{NF}}$ for $\epsilon = 0.3$ in terms of the minimum required speedup factor.

	Measured avg. run-time		Ratio of avg. run-time
Value of ϵ	$\mathrm{PTAS_{NF}}$	**PTAS of [1]**	
0.10	128.57	6583384.71	51204
0.15	45.43	6914127.72	152192
0.20	18.40	4449061.29	241796
0.25	10.48	1564060.39	149242
0.30	7.17	465894.09	64978

TABLE I: Comparison of run times of $\mathrm{PTAS_{NF}}$ and $\mathrm{PTAS_{LP}}$ (μs).

B. Evaluation of $\mathrm{PTAS_{NF}}$ for different values of ϵ

In order to understand how much pessimism is there in the analysis of $\mathrm{PTAS_{NF}}$, we evaluated its performance for different values of ϵ. In this set of experiments, we chose larger number of problem instances with each problem instance being more complex[3]. We generated 10000 critically feasible task sets where each task set had at most 25 tasks and at most 3 processors of each type. Then, for different values of ϵ, we ran $\mathrm{PTAS_{NF}}$ on these 10000 critically feasible task sets and obtained the histograms of $\mathrm{MRSF_{NF}}$. Figure 2 shows the histogram for $\epsilon = 0.3$. As can be seen, for almost 98% of the task sets, the $\mathrm{MRSF_{NF}}$ did not exceed 1.06, i.e., approximately 7% of its theoretical bound (i.e., $1+3\epsilon = 1.90$), for the remaining 2% of the task sets, the factor did not exceed 1.12, i.e., approximately 13% of its theoretical bound. Thus, in the simulations, for the vast majority of task sets, our algorithm requires much smaller processor speedup than indicated by its approximation ratio. The observations for other values of ϵ follow the same trend — see Appendix B in [15].

Hence, $\mathrm{PTAS_{NF}}$ performs significantly better in simulations than indicated by its theoretical bound.

IX. Conclusions

A polynomial-time approximation scheme was proposed for the problem of partitioning a given collection of implicit-deadline sporadic tasks upon a multiprocessor platform in which there are two distinct kinds of processors. It provides the following guarantee: if a task set has a feasible partitioning on a two-type platform then given an input, $\epsilon > 0$, our PTAS succeeds in finding such a feasible partitioning for the task set on a two-type platform in which each processor is $1 + 3\epsilon$ times faster. In simulations, our algorithm outperforms prior state-of-the-art PTAS [1] and also performs significantly better than indicated by its theoretical bound.

[3]Since we do not run $\mathrm{PTAS_{LP}}$ (which takes much longer to output the solution) in this batch of experiments, we could increase the problem instances and size of each problem compared to previous set of experiments.

Acknowledgment

This work was partially supported by FCT(Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within REHEAT project, ref. FCOMP-01-0124-FEDER-010045; by FCT and the EU ARTEMIS JU funding, within RECOMP project, ref. ARTEMIS/0202/2009, JU grant nr. 100202; by FCT and ESF through POPH, under PhD grant SFRH/BD/66771/2009.

References

[1] A. Wiese, V. Bonifaci, and S. Baruah, "Partitioned EDF scheduling on a few types of unrelated multiprocessors," The University of North Carolina, Tech. Rep., 2012, http://www.cs.unc.edu/ baruah/Submitted/2012-k-unrelated.pdf.

[2] 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.

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

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

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

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

[7] Samsung Inc., "Samsung exynos 4 quad application processor," www.samsung.com/exynos/.

[8] M. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings of IFIP Congress*, 1974, pp. 807–813.

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

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

[11] 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.

[12] ——, "Partitioning real-time tasks among heterogeneous multiprocessors," in *Proceedings of the 33rd International Conference on Parallel Processing*, 2004, pp. 467–474.

[13] 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.

[14] S. Baruah, "Task assignment on two unrelated types of processors," in *19th International Conference on Real-Time and Network Systems*, 2011, pp. 69–78.

[15] G. Raravi and V. Nélis, "A PTAS for assigning sporadic tasks on two-type heterogeneous multiprocessors," CISTER/INESC-TEC, ISEP, Porto, Tech. Rep., 2012, http://www.hurray.isep.ipp.pt/docs/.

[16] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Ed.* McGraw-Hill, 2009.

[17] IBM, "IBM ILOG CPLEX Optimizer," http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/.