Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors

Gurulingesh Raravi · Björn Andersson · Konstantinos Bletsas

Abstract Consider the problem of partitioned scheduling of an implicit-deadline sporadic task set on heterogeneous multiprocessors to meet all deadlines. Each processor is either of type-1 or type-2. We present a new algorithm, FF-3C, for this problem. FF-3C offers low time-complexity and provably good performance. Specifically, FF-3C offers (i) a time-complexity of $O(n \cdot \max(m, \log n) + m \cdot \log m)$, where n is the number of tasks and m is the number of processors and (ii) the guarantee that if a task set can be scheduled by an optimal partitioned-scheduling algorithm to meet all deadlines then FF-3C meets all deadlines as well if given processors at most $\frac{1}{1-\alpha}$ times as fast (referred to as speed competitive ratio) and tasks are scheduled using EDF; where α is a property of the task set. The parameter α is in the range $(0, 0.5]$ and for each task, it holds that its utilization is no greater than α or greater than $1 - \alpha$ on each processor type. Thus, the speed competitive ratio of FF-3C can never exceed 2.

We also present several extensions to FF-3C; these offer the same performance guarantee and time-complexity but with improved average-case performance. Via simulations, we compare the performance of our new algorithms and two state-of-the-art algorithms (and variations of the latter). We evaluate algorithms based on (i) running time and (ii) the necessary multiplication factor, i.e., the amount of extra speed of processors that the algorithm needs, for a given task set, so as to succeed, compared to an optimal task assignment algorithm. Overall, we observed that our new algorithms perform significantly better than the state-of-the-art. We also observed that our algorithms perform much better in practice, i.e., the necessary multiplication factor of the algorithms is much smaller than their speed competitive ratio. Finally, we also present a clustered version of the new algorithm.

Keywords Bin packing · Heterogeneous multiprocessors · Real-time scheduling

1 Introduction

Designers have been achieving significant speedup of particular tasks by using specialized processing units (e.g., graphics processors for computer graphics or DSPs for signal processing). The advent of heterogeneous multiprocessors on a single chip facilitates this even more. Virtually all major manufacturers offer some kind of heterogeneous multiprocessor implemented on a single chip (AMD Inc 2011a; AMD

Inc 2011b; Freescale Semiconductor 2007; Gschwind et al. 2006; IBM Inc. 2005; IEEE Spectrum 2011; Intel Corporation 2011; Maeda et al. 2005; NVIDIA 2011; Texas Instruments 2011). Their use in embedded systems is non-trivial however because many embedded systems have real-time requirements, whose satisfaction at run-time has to be proven/guaranteed *a priori*; this is a significant challenge for the use of heterogeneous multicores. The way tasks are scheduled significantly influences whether their timing requirements are met. Unfortunately, for heterogeneous multiprocessors, no comprehensive toolbox of real-time scheduling algorithms and analysis techniques exists (unlike e.g., what exists for a uniprocessor).

An algorithm for deciding whether or not a task set can be scheduled on a heterogeneous platform exists (Baruah 2004a) but it assumes that tasks can migrate. This assumption, however, is often unrealistic in practice, since processors with different functionalities typically have different instruction sets, register formats, etc. Thus, the problem of assigning tasks to processors and then scheduling them with a uniprocessor scheduling algorithm (i.e., without migration) is of much greater practical significance. It requires solving two sub-problems: (i) assigning tasks to processors and (ii) once tasks are assigned to processors, performing uniprocessor scheduling on each processor. The latter problem is well-understood, e.g., one may use an *optimal scheduling algorithm*[1] such as EDF (Dertouzos 1974; Liu and Layland 1973)—the difficult part is the task assignment.

Among known task assignment schemes for multiprocessors in general (i.e., not necessarily heterogeneous), (i) bin-packing heuristics (e.g., first-fit), (ii) Integer Linear Programming (ILP) modeling, (iii) Linear Programming (LP) relaxation approaches for ILP and (iv) dynamic programming techniques perform provably well. The task assignment problem on identical multiprocessors can be transformed into a bin-packing problem. Bin-packing heuristics (Coffman et al. 1997) are popular for

[1] An optimal scheduling algorithm is one which always succeeds in finding a schedule in which all the deadlines are met, if such a schedule exists.

task assignment but unfortunately, the proof techniques used on identical multiprocessors do not easily translate to heterogeneous multiprocessors. Traditionally, the literature offered no bin-packing heuristic for assigning real-time tasks on heterogeneous multiprocessors. Instead, task assignment was modeled (Baruah 2004b, 2004c) as Zero-One ILP. Such a formulation can be solved directly but has high computational complexity. In particular, the decision problem ILP is NP-complete and even with knowledge of the structure of the constraints in the modeling of heterogeneous multiprocessor scheduling, no polynomial-time algorithm is known (Garey and Johnson 1979, p. 245). Via relaxation of ILP formulation to LP and certain tricks (Potts 1985), polynomial time-complexity can be attained (Baruah 2004b, 2004c) (provided that polynomial-time LP solver is used to solve the relaxed LP formulation). Neither of the above two algorithms, however, attains low-degree (linear or quadratic) polynomial time-complexity. It is a well-known fact that the problem under consideration is equivalent to the problem of minimizing the *makespan* on unrelated machines. For this problem, when the number of machines is fixed, a *fully polynomial time approximation scheme* (FPTAS)[2] was proposed in Horowitz and Sahni (1976). This scheme required time $O(nm(nm/E)^{m-1})$ and space $O((nm/E)^{m-1})$ where m refers to number of machines, n refers to number of jobs. Later, for the same problem (i.e., when the number of machines is fixed), a polynomial time approximation scheme was proposed in Lenstra et al. (1990). However, the space requirement was significantly improved compared to the algorithm proposed in Horowitz and Sahni (1976) and was bounded by a polynomial which was a function of the input, m, and $\log(1/E)$. The running time of the procedure was bounded by a function that is the product of $(n+1)^{m/E}$ where n is number of jobs. As we can see, neither of the above two algorithms achieves low-degree polynomial time- and space-complexity. Recently, Wiese et al. (2012) proposed a PTAS for the problem of assigning tasks on a computing platform comprising two or more types of processors. However, as any other PTAS, this algorithm also has high degree polynomial time- and space-complexity.

In practice, many heterogeneous multiprocessors only use two types of processors. For example, AMD (AMD Inc 2010), NVIDIA (IEEE Spectrum 2011), Intel (Intel Corporation 2011), FreeScale (Freescale Semiconductor 2007), TI (Texas Instruments 2011) offer such chips. Traditionally, processors of the first type were meant for general purpose computations and processors of the second type were meant for special purpose computations (such as graphics or signal processing), hence task assignment was trivial. Today though, designers (Geer 2005) use processors of the second kind for wide range of computations and this makes task assignment non-trivial. Unfortunately, the literature did not provide any scheduling algorithm that took advantage of this special structure.

Therefore, in Andersson et al. (2010) (the conference version of this paper), we considered the problem of non-migratively scheduling (also referred to as partitioned scheduling) a set of independent implicit-deadline sporadic tasks, to meet

[2] A PTAS is an algorithm which produces a solution that is within a factor $1+E$ of being optimal where $E>0$ is a design parameter. The run time of a PTAS is polynomial in the input size (e.g., number of jobs) and may be exponential in $1/E$. A FPTAS is a PTAS with a running time that is polynomial both in input size and $1/E$.

all deadlines, on a heterogeneous multiprocessor where each processor is either of type-1 or type-2 (with each task having different execution time on each processor type). We presented a new algorithm, FF-3C, for this problem—this algorithm uses a bin-packing heuristic for assigning tasks. FF-3C offered low time-complexity and provably good performance. Specifically, FF-3C offered (i) a time-complexity of $O(n \cdot \max(m, \log n) + m \cdot \log m)$, where n denotes number of tasks and m denotes number of processors and (ii) the guarantee that if a task set can be scheduled by an optimal task assignment algorithm to meet deadlines then FF-3C meets deadlines as well if the given processors are twice as fast (referred to as the *speed competitive ratio*) and tasks are scheduled using preemptive EDF scheduling algorithm. We also presented several extensions to FF-3C; these offered the same time-complexity and performance guarantee but in addition, they offered improved average-case performance. Via experiments with randomly generated task sets, we compared the performance of our algorithms and two established state-of-the-art algorithms (and variations of the latter) (Baruah 2004b, 2004c). We evaluated algorithms based on (i) average running time and (ii) the *necessary multiplication factor*, i.e., the amount of extra speed of processors the algorithm needs, for a task set, in order to succeed as compared to an optimal task assignment algorithm. Overall our new algorithms compared favorably to the state-of-the-art.[3] In particular, in our experimental evaluations, one of our new algorithms, FF-4C-COMB, ran 12000 to 160000 times faster and had significantly smaller necessary multiplication factor than state-of-the-art (Baruah 2004b, 2004c).

It is known in bin-packing that packing small items (i.e., tasks with small utilizations) allows better performance bounds. Our conference paper, however, did not exploit this. Hence, this article extends our conference paper (Andersson et al. 2010) by incorporating this idea among other things. The additional contributions in this paper can be summarized as follows: (i) we improve the analysis of FF-3C by incorporating the task parameters into analysis. Consider a task set in which, for each task, it holds that its utilization is no greater than α or greater than 1-α on a processor of type-1 and its utilization is no greater than α or greater than 1-α on a processor of type-2. For such task sets, we show that FF-3C succeeds in meeting all deadlines if it is possible to meet all deadlines by an optimal task assignment on a computer platform where each processor has the speed 1-α of the corresponding processor that FF-3C uses; (ii) we also prove the performance guarantee as a function of α for the algorithms with improved average-case performance (i.e., FF-4C, FF-4C-NTC and FF-4C-COMB); (iii) we perform additional experiments to show the performance of our algorithms with this new analysis. We have also (re-)run the experiments that were carried out in Andersson et al. (2010) for evaluating the performance of our algorithms with state-of-the-art (Baruah 2004b, 2004c) and (iv) we also present a version of the new algorithm targeted for two-type platform where processors are organized into clusters and task migration is allowed between processors of the same cluster.

[3] For a given problem instance, the *necessary multiplication factor* of an algorithm is upper bounded by its *speed competitive ratio*. If an algorithm has low necessary multiplication factor (compared to its speed competitive ratio) for the vast majority of task sets then it indicates that the algorithm performs well.

In the remainder of this paper, Sect. 2 offers necessary preliminaries. Section 3 presents some previously known and some new results that we use in Sect. 4, where we formulate the new algorithm namely, FF-3C, and prove its performance. Section 5 considers time-complexity. Section 6 describes the enhancements to FF-3C to obtain better average-case performance and proves their performance. Section 7 offers experimental evaluation; Sect. 8 presents a clustered version of FF-3C and finally Sect. 9 concludes.

2 Preliminaries

In a computer platform with two unrelated types of processors, let P^1 be the set of type-1 processors and P^2 be the set of type-2 processors. The workload consists of τ, a set of implicit-deadline sporadic tasks (i.e., for each task, its deadline is equal to its minimum inter-arrival time) each of which releases a (potentially infinite) sequence of jobs.

A task is assigned to a processor and all jobs released by this task must execute there. The utilization of task τ_i depends on the type of processor to which it is assigned. The utilization of task τ_i is u_i^1 if τ_i is assigned to a type-1 processor. Analogously, the utilization of task τ_i is u_i^2 if τ_i is assigned to a type-2 processor. Note that we allow $u_i^1 = \infty$ (resp., $u_i^2 = \infty$) if task τ_i cannot be assigned at all to a type-1 (resp., type-2) processor.

We assume that tasks are assigned unique identifiers. This allows two tasks with the same parameters to be in a set. For example, with $u_i^1 = 0.2$, $u_i^2 = 0.4$ and $u_j^1 = 0.2$, $u_j^2 = 0.4$, we can form the set $\{\tau_i, \tau_j\}$. We also assume that processors are assigned unique identifiers. This assumption is instrumental because if we sort processors in ascending order of their identifiers then we can be sure that, when applying normal bin-packing schemes (e.g., first-fit) repeatedly on the same task set, with tasks ordered in the same way, the bin-packing scheme outputs the same task assignment for each run.

Let $\tau[p]$ denote the set of tasks assigned to a processor p. Earliest-Deadline-First (EDF) is a very popular algorithm in uniprocessor scheduling (Liu and Layland 1973). A slight adaptation of a previously known result (Liu and Layland 1973) gives us:

Lemma 1 *If all tasks in $\tau[p]$ are scheduled under EDF on a processor p (which is of type-z, where $z \in \{1, 2\}$) and $\sum_{\tau_i \in \tau[p]} u_i^z \leq 1$, then all deadlines are met.*

Then the necessary and sufficient set of conditions for schedulability on a partitioned heterogeneous multiprocessor with two types of processor is the following:

$$\forall p \in P^1 : \quad \sum_{\tau_i \in \tau[p]} u_i^1 \leq 1 \tag{1}$$

$$\forall p \in P^2 : \quad \sum_{\tau_i \in \tau[p]} u_i^2 \leq 1 \tag{2}$$

Thus our problem of scheduling tasks on a heterogeneous multiprocessor with two types of processors is reduced to assigning tasks to processors such that the above constraints are satisfied. Yet, even in the special case of identical multiprocessors, this problem is intractable (Baruah 2004b). We therefore aim for a non-optimal algorithm of low-degree polynomial time-complexity which would still offer good performance.

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

The speed competitive ratio CPT_A of a non-migrative algorithm A is defined as the lowest number such that for every task set τ and computing platform Π' it holds that if it is possible for a non-migrative algorithm to meet all deadlines of τ on Π' then algorithm A meets all deadlines of τ on a platform Π whose every processor is CPT_A times faster than the corresponding processor in Π'.[4]

A low speed competitive ratio indicates high performance; the best achievable is 1. If a scheduling algorithm has an infinite speed competitive ratio then a task set exists which could be scheduled (by another algorithm) to meet deadlines but would miss deadlines with the actually used algorithm even if processor speeds were multiplied by an “infinite” factor. Therefore, we aim for an algorithm with finite (ideally small) speed competitive ratio.

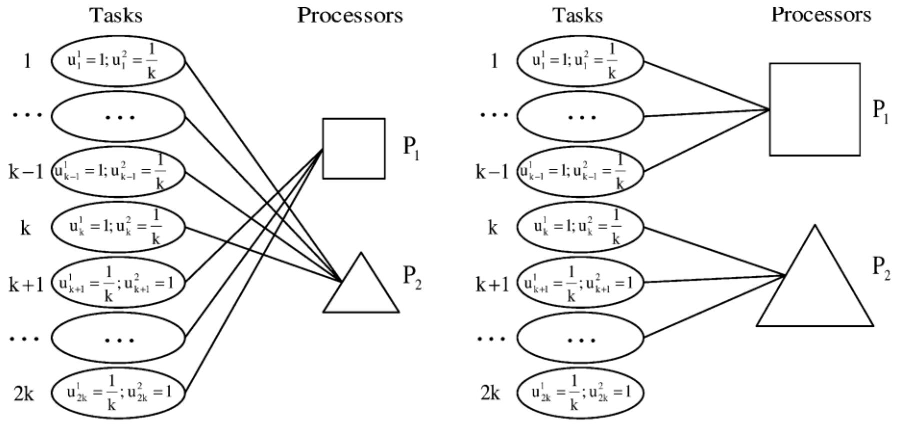
We now introduce few notations that will be used later (from Sect. 4.3 onwards) while proving the speed competitive ratio of our algorithms.

Let $\Pi(|P^1|, |P^2|)$ denote a two-type heterogeneous platform comprising $|P^1|$ processors of type-1 and $|P^2|$ processors of type-2. Let $\Pi(|P^1|, |P^2|) \times (s_1, s_2)$ denote a two-type heterogeneous platform in which the speed of every processor of type-1 is s_1 times the speed of a type-1 processor in $\Pi(|P^1|, |P^2|)$ and the speed of every processor of type-2 is s_2 times the speed of a type-2 processor in $\Pi(|P^1|, |P^2|)$ where s_1 and s_2 are positive real-numbers (i.e., $s_1 > 0$ and $s_2 > 0$).

Let $sched(A, \tau, \Pi(|P^1|, |P^2|) \times (s_1, s_2))$ denote a predicate to signify that a task set τ *meets all its deadlines* when scheduled by an algorithm A on a two-type heterogeneous multiprocessor platform—$\Pi(|P^1|, |P^2|) \times (s_1, s_2)$. The term *meets all its deadlines* in this and other predicates means ‘meets deadlines for every possible arrival of tasks that is valid as per the given parameters of τ’.

We use $sched(\text{nmo-feasible}, \tau, \Pi(|P^1|, |P^2|) \times (s_1, s_2))$ to signify that there exists a *non-migrative-offline-feasible* preemptive schedule which meets all deadlines for the specified system. Here, *non-migrative* schedule refers to a schedule in which all the jobs of a task execute on the same processor on which the task has been assigned

[4] Our notion of speed competitive ratio in this paper is equivalent to that in previous work (Baruah 2004a). It differs from that used in Andersson and Tovar (2007b). Other authors refer to speed competitive ratio as *resource augmentation bound*.


Tasks
Processors
1
$u_1^1 = 1; u_1^2 = \frac{1}{k}$
...
k−1
$u_{k-1}^1 = 1; u_{k-1}^2 = \frac{1}{k}$
k
$u_k^1 = 1; u_k^2 = \frac{1}{k}$
k+1
$u_{k+1}^1 = \frac{1}{k}; u_{k+1}^2 = 1$
2k
$u_{2k}^1 = \frac{1}{k}; u_{2k}^2 = 1$
P_1
P_2

(a) A successful assignment of tasks on processors

(b) The standard first-fit bin packing fails to assign tasks on $k-1$ times faster processors

Fig. 1 The standard first-fit (or any other) bin-packing heuristics does not perform well for assigning tasks on two-type heterogeneous multiprocessor platform

(also referred to as partitioned scheduling). In this predicate and other predicates, the term *offline* (also) encompasses the schedules generated by algorithms which (i) may use inserted idle times and/or (ii) are “clairvoyant” (i.e., use knowledge of future task arrival times).

3 Useful results

Bin-packing heuristics are popular for assigning tasks on identical (Grandpierre et al. 1999) and uniform (Andersson and Tovar 2007a; Hochbaum and Shmoys 1986) multiprocessors (where a processor x times faster executes all tasks x times faster) because they run fast and offer finite speed competitive ratio. Yet, straightforward application of bin-packing heuristics to heterogeneous multiprocessors with two types of processors performs poorly, as illustrated by Example 1.

Example 1 Consider a set of $2k$ tasks and 2 processors (for an integer $k \geq 3$). Processor P_1 is of type-1 and processor P_2 is of type-2. Tasks indexed $1,\ldots,k$ are characterized by $u_i^1 = 1$, $u_i^2 = \frac{1}{k}$ and tasks indexed $k+1,\ldots,2k$ are characterized by $u_i^1 = \frac{1}{k}$, $u_i^2 = 1$.

Tasks can be assigned such that the condition of Lemma 1 is met for both processors, e.g., assigning tasks $1,\ldots,k$ to P_2 and the rest to P_1 as shown in Fig. 1a. Yet, the application of a normal bin-packing algorithm (designed for identical multiprocessors such as First-Fit) causes failure. These algorithms consider tasks in a sequence and each time use the condition of Lemma 1 to decide if the task in consideration can be assigned to a processor. Under First-Fit, τ_1 ends up on P_1 (as processors are considered by order of ascending index). Yet, at most one task of those indexed $1,\ldots,k$ can be assigned there. Thus, the $k-1 \geq 2$ tasks indexed $2,\ldots,k$ will have to be assigned to P_2. Next, the bin-packing scheme tries to assign tasks $k + 1,\ldots, 2k$ to P_2; none fits and the algorithm fails.

Let us now provide the bin-packing algorithm with processors $k - 1$ times faster. Then, tasks indexed $1,\ldots,k - 1$ will be assigned to P_1 and the kth task to P_2 before considering tasks indexed $k + 1,\ldots, 2k$. Of the latter, many can be assigned to P^2 but not all and, since none can be assigned to P^1, the bin-packing algorithm would again fail as shown in Fig. 1b.

This holds for any $k \geq 3$. For $k \rightarrow \infty$, we see that the speed competitive ratio of such bin-packing schemes is infinite.

It can be seen that the cause of low performance of such a bin-packing scheme is that, by considering tasks one by one, it lacks a "global view" of the problem, hence may assign a task to a processor where it executes slowly. It seems a good idea to try to assign each task to the processor where it executes faster. We will use this idea; let us thus introduce the following definitions:

P^1 is the set of type-1 processors and P^2 is the set of type-2 processors. The task set τ is viewed as two disjoint subsets, τ^1 and τ^2. The set τ^1 consists of those tasks which run at least as fast on a type-1 processor as on a type-2 processor; τ^2 consists of all other tasks. In notation:

$$\tau = \tau^1 \cup \tau^2 \qquad (3)$$

$$\forall \tau_i \in \tau^1 : \quad u_i^1 \leq u_i^2 \qquad (4)$$

$$\forall \tau_i \in \tau^2 : \quad u_i^1 > u_i^2 \qquad (5)$$

We now list two useful observations along with their proofs.

Lemma 2 *If there is a task τ_i in τ^1 such that $1 < u^1_i$ it is then impossible to meet all deadlines with partitioning. Likewise for a task τ_i in τ^2 with $1 < u^2_i$.*

Proof Intuitively, if the execution time of τ_i exceeds its deadline on processor type where it runs fastest, it cannot be assigned anywhere to meet deadlines. D

Lemma 3 *It is impossible to meet all deadlines if*

$$\sum_{\tau_i \in \tau^1} u_i^1 + \sum_{\tau_i \in \tau^2} u_i^2 > |P^1| + |P^2| \qquad (6)$$

Proof

The proof is by contradiction. Let τ be a task set for which Inequality (6) holds and for which a feasible partitioning exists. Given that τ is feasible, the set of constraints expressed by Inequalities (1) and (2) must hold. Then, respectively from those inequalities, we have:

$$\forall p \in P^1: \quad \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^1 + \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^1 \leq 1 \tag{7}$$

$$\forall p \in P^2: \quad \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^2 + \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^2 \leq 1 \tag{8}$$

However, from Inequalities (5) and (4) respectively:

$$(5) \quad \Rightarrow \quad \forall \tau_i \in \tau^2: \quad u_i^1 > u_i^2 \quad \text{and} \tag{9}$$

$$(4) \quad \Rightarrow \quad \forall \tau_i \in \tau^1: \quad u_i^1 \leq u_i^2 \tag{10}$$

Then, respectively:

$$(7) \quad \overset{(9)}{\Longrightarrow} \quad \forall p \in P^1: \quad \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^1 + \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^2 < 1 \tag{11}$$

$$(8) \quad \overset{(10)}{\Longrightarrow} \quad \forall p \in P^2: \quad \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^1 + \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^2 \leq 1 \tag{12}$$

We can combine Inequalities (11) and (12) into:

$$\forall p: \quad \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^1 + \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^2 \leq 1 \tag{13}$$

Via summation of Inequality (13) over all p we obtain

$$\sum_p \sum_{\tau_i \in \tau[p] \cap \tau^1} u_i^1 + \sum_p \sum_{\tau_i \in \tau[p] \cap \tau^2} u_i^2 \leq \sum_p 1$$

$$\Rightarrow \sum_{\tau_i \in \tau^1} u_i^1 + \sum_{\tau_i \in \tau^2} u_i^2 \leq |P^1| + |P^2| \tag{14}$$

This contradicts Inequality (6). D

We next highlight how the problem in consideration is related to fractional knapsack problem, to help with proofs later. If you read this paper for the first time, you may want to skip this section now and revisit it later.

Fractional Knapsack Problem: A vector x has n elements. The problem instance is represented by vectors v and w of real numbers, arranged such that $\frac{v_i}{w_i} \geq \frac{v_{i+1}}{w_{i+1}}$ $\forall i \in \{1, 2, \ldots, n-1\}$. (Intuitively, v_i and w_i may be thought of as, respectively, the “value” and “weight” of an element.) Consider the problem of assigning values to the elements in vector x so as to maximize $\sum_{i=1}^{n} x_i \cdot v_i$ subject to $\sum_{i=1}^{n} x_i \cdot w_i \leq \text{CAP}$ where x_i is a real number such that $0 \leq x_i \leq 1$ and CAP is a given upper bound.

(Intuitively, determine how much of each item to use such that cumulative value is maximized, subject to cumulative weight not exceeding some bound.)

Lemma 4 *An optimal solution to the Fractional Knapsack Problem is obtained by Algorithm* 1.

Proof This is found in textbooks (Chap. 16.2 (Cormen et al. 2001)). D

Algorithm 1: For fractional knapsack problem

```
re-index tuples $\{v_i, w_i\}$ by order of descending $v_i/w_i$
for $i=1$ to $n$ do $x_i := 0$ end
$i := 1$; SUMWEIGHT := 0; SUMVALUE := 0
while (SUMWEIGHT + $w_i \leq$ CAP) $\wedge$ $(i \leq n)$ do
    $x_i := 1$
    SUMWEIGHT := SUMWEIGHT + $w_i$
    SUMVALUE := SUMVALUE + $v_i$
    i:=i+1
end
if $i \leq n$ then
    $x_i := ($CAP $-$ SUMWEIGHT$)/w_i$
    SUMWEIGHT := SUMWEIGHT + $w_i \cdot x_i$
    SUMVALUE := SUMVALUE + $v_i \cdot x_i$
end
```

For a given problem instance in our scheduling problem, we can create an instance of a fractional knapsack problem as follows: (i) for each task in our scheduling problem, create a corresponding item in the fractional knapsack problem, (ii) the weight of an item in the fractional knapsack problem is the utilization of the corresponding task where the utilization here is taken for the processor on which the task executes fast and (iii) the value of an item in the fractional knapsack problem is how much lower the utilization of its corresponding task is when the task is assigned to the processor on which it executes fast as compared to its utilization if assigned to the processor on which it executes slowly. Informally speaking, we can see that if tasks could be split, then solving the fractional knapsack problem is equivalent to assigning tasks to processors so that the cumulative utilization of tasks is minimized. Again, informally speaking, we can then show that a task assignment minimizes the cumulative utilization of tasks assuming that (i) the cumulative utilization of tasks that are assigned to the processors on which they execute fast is sufficiently high and (ii) the tasks that are assigned to the processors where they execute fast has a higher ratio (u_i^2/u_i^1) than the ones that are not. Lemmas 5 and 6 expresses this formally and proves it.

Lemma 5 *Consider n tasks and a heterogeneous multiprocessor conforming to the system model (and notation) of Sect.* 2. *Let x denote a number such that* $0 \leq x \leq |P^1| \cdot (1 - y)$ *where* $0 < y \leq \frac{1}{2}$. *Let* $A1$ *denote a subset of* τ^1 *such that*

$$\sum_{\tau_i \in A1} u_i^1 > |P^1| \cdot (1-y) - x \tag{15}$$

$^1 \cdot (1 - y) - x$ (15)

and for every pair of tasks $\tau_i \in A1$ *and* $\tau_j \in \tau^1 \setminus A1$ *it holds that* $\frac{u_i^2}{u_i^1} - 1 \geq \frac{u_j^2}{u_j^1} - 1$.

Let $A2$ *denote* $\tau^1 \setminus A1$.

Let $B1$ *denote a subset of* τ^1 *such that*

$$\sum_{\tau_i \in B1} u_i^1 \leq \left| P^1 \right| \cdot (1 - y) - x \qquad (16)$$

$$_{\tau_i \in B1} u_i^1 \leq \; P^1 \; . (1 - y) - x \qquad (16)$$

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

$$\sum_{\tau_i \in A1} u_i^1 + \sum_{\tau_i \in A2} u_i^2 + \sum_{\tau_i \in \tau^2} u_i^2 \leq \sum_{\tau_i \in B1} u_i^1 + \sum_{\tau_i \in B2} u_i^2 \tag{17}$$

1

Proof Let us arbitrarily choose $A1$, $B1$ as defined. We will prove that this implies Inequality (17). Using Inequalities (15) and (16) we clearly get:

$$\sum_{\tau_i \in A1} u_i^1 > \sum_{\tau_i \in B1} u_i^1 \tag{18}$$

With this choice of $A1$ and $B1$, let us consider different instances of the fractional knapsack problem:

Instance1: CAP = left-hand side of Inequality (18).
For each $\tau_i \in \tau$, create an item i with $v_i = u_i^2 - u_i^1$ and $w_i = u_i^1$
SUMVALUE$_1$ = value of variable SUMVALUE when Algorithm 1 terminates with Instance1 as input.

Instance2: CAP = left-hand side of Inequality (18).
For each $\tau_i \in A1$, create an item i with $v_i = u_i^2 - u_i^1$ and $w_i = u_i^1$
SUMVALUE$_2$ = value of variable SUMVALUE when Algorithm 1 terminates with Instance2 as input.

Instance3: CAP = right-hand side of Inequality (18).
For each $\tau_i \in B1$, create an item i with $v_i = u_i^2 - u_i^1$ and $w_i = u_i^1$
SUMVALUE$_3$ = value of variable SUMVALUE when Algorithm 1 terminates with Instance3 as input.

Instance4: CAP = right-hand side of Inequality (18).
For each $\tau_i \in \tau$, create an item i with $v_i = u_i^2 - u_i^1$ and $w_i = u_i^1$
SUMVALUE$_4$ = value of variable SUMVALUE when Algorithm 1 terminates with Instance4 as input.

Observe that:

O1: In all four instances, it holds for each element that $\frac{v_i}{w_i} = \frac{u_i^2}{u_i^1} - 1$.
O2: Instance1 and Instance2 have the same capacity.
O3: Although Instance2 has a subset of the elements of Instance1, this subset is the subset of those elements with the largest v_i/w_i—follows from definition of $A1$.
O4: CAP in Instance2 is exactly the sum of the weights of the elements in $A1$.
O5: From O1,O2,O3 and O4: SUMVALUE$_2$ = SUMVALUE$_1$.
O6: Instance3 and Instance4 have the same capacity.
O7: Instance3 has a subset of the elements of Instance4.
O8: From O6 and O7: SUMVALUE$_3$ ≤ SUMVALUE$_4$.
O9: Instance4 has smaller capacity than Instance1.
O10: Instance4 has the same elements as Instance1.

O11: From O9 and O10: SUMVALUE$_4$ ≤ SUMVALUE$_1$.
O12: From O8 and O11: SUMVALUE$_3$ ≤ SUMVALUE$_1$.
13: From O12 and O5: SUMVALUE$_3$ ≤ SUMVALUE$_2$.

$$\underset{\tau_i \in B1}{} u_i^1 \leq P^1 \,.\, (1-y) - x \qquad (16)$$

Using O13 and the definitions of the instances of $A1$ and $B1$ and observing that the capacity of Instance2 and Instance3 are set such that all elements in either instance will fit into the respective “knapsack”, we obtain:

$$\sum_{\tau_i \in B1} \left(u_i^2 - u_i^1\right) \leq \sum_{\tau_i \in A1} \left(u_i^2 - u_i^1\right) \qquad (19)$$

Now, observing that $\tau = \tau^1 \cup \tau^2 = B1 \cup B2$ gives us:

$$\sum_{\tau_i \in \tau^1} u_i^2 + \sum_{\tau_i \in \tau^2} u_i^2 = \sum_{\tau_i \in B1} u_i^2 + \sum_{\tau_i \in B2} u_i^2 \qquad (20)$$

Combining Expression (19) and (20) gives us:

$$\begin{aligned} &\sum_{\tau_i \in \tau^1} u_i^2 + \sum_{\tau_i \in \tau^2} u_i^2 - \left(\sum_{\tau_i \in A1} u_i^2 - \sum_{\tau_i \in A1} u_i^1\right) \\ &\leq \sum_{\tau_i \in B1} u_i^2 + \sum_{\tau_i \in B2} u_i^2 - \left(\sum_{\tau_i \in B1} u_i^2 - \sum_{\tau_i \in B1} u_i^1\right) \qquad (21) \end{aligned}$$

Rearranging terms and exploiting $A2 = \tau^1 \backslash A1$ yields:

1

$$\sum_{\tau_i \in A1} u_i^1 + \sum_{\tau_i \in A2} u_i^2 + \sum_{\tau_i \in \tau^2} u_i^2 \leq \sum_{\tau_i \in B1} u_i^1 + \sum_{\tau_i \in B2} u_i^2$$

This is the statement of the lemma. D

Lemma 5 considers τ . We can however apply this on only a subset of τ . Let us assume that $H1$ and $H2$ are two disjoint subsets of τ . By applying Lemma 5 on $\tau \setminus (H1 \cup H2)$ and then adding the same sum to both sides of Inequality (17), we get:

Lemma 6 *Consider n tasks and a heterogeneous multiprocessor conforming to the system model (and notation) of Sect.* 2. *Let x denote a number such that* $0 \leq x \leq |P^1| \cdot (1 - y)$ *where* $0 < y \leq {}_2^1$. *Let* A1 *denote a subset of* $(\tau^1 \setminus (H1 \cup H2))$ *such that*

u^1

$$\sum_{\tau_i \in A1} u_i^1 > \left|P^1\right| \cdot (1 - y) - x \qquad (22)$$

and for every pair of tasks $\tau_i \in A1$ *and* $\tau_j \in (\tau^1 \setminus (H1 \cup H2)) \setminus A1$ *it holds that* $\frac{u_i^2}{u_i^1} - 1 \geq \frac{u_i}{u_j^1} - 1$. *Let* $A2$ *denote* $(\tau^1 \setminus (H1 \cup H2)) \setminus A1$.

Let $B1$ *denote a subset of* $\tau^1 \setminus (H1 \cup H2)$ *such that*

$$\sum_{\tau_i \in B1} u_i^1 \leq \left|P^1\right| \cdot (1 - y) - x \qquad (23)$$

–

$$\underset{\tau_i \in B1}{} u_i^1 \leq P^1 \,.\, (1 - y) - x \qquad (23)$$

*Let B*2 *denote* $(\tau \setminus (H1 \cup H2))\overset{2}{\setminus} B1$. *It then holds that*:

$$\sum_{\tau_i \in H1} u_i^1 + \sum_{\tau_i \in H2} u_i^2 + \sum_{\tau_i \in A1} u_i^1 + \sum_{\tau_i \in A2} u_i^2 + \sum_{\tau_i \in \tau^2 \setminus (H1 \cup H2)} u_i^2$$

$$\leq \sum_{\tau_i \in H1} u_i^1 + \sum_{\tau_i \in H2} u_i^2 + \sum_{\tau_i \in B1} u_i^1 + \sum_{\tau_i \in B2} u_i^2$$

Lemma 6 is used while proving the performance of our new algorithm.

4 The FF-3C algorithm and its speed competitive ratio

4.1 The FF-3C algorithm

The new algorithm, FF-3C, is based on two ideas.

Idea1: A task should ideally be assigned to the processor type where it runs faster (termed “favorite” type).

Idea2: A task with utilization above $\frac{1}{2}$ on its non-favorite type of processor must be assigned to its favorite type of processor. This special case of Idea1 is stated separately because this facilitates creating an algorithm with the desired speed competitive ratio (which is upper bounded by 2): Since we will compare the performance of our new algorithm versus every other algorithm that uses processors of at most $\frac{1}{2}$ the speed, following Idea2 ensures that each of those tasks is assigned to the same corresponding processor type as under every other successful assignment algorithm.

Based on these ideas and the concepts of τ^1 and τ^2 (defined in Sect. 2), we also define the following disjoint sets:

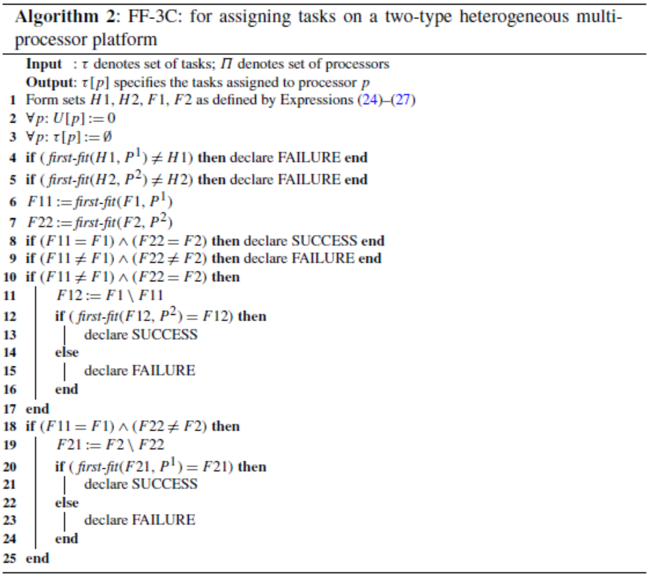
$$H1 = \left\{\tau_i \in \tau^1 : u_i^2 > \frac{1}{2}\right\} \tag{24}$$

$$H2 = \left\{\tau_i \in \tau^2 : u_i^1 > \frac{1}{2}\right\} \tag{25}$$

$$F1 = \tau^1 \setminus H1 \tag{26}$$

$$F2 = \tau^2 \setminus H2 \tag{27}$$

A task is termed to be *heavy on type-1 processors* (resp., type-2 processors) if its utilization on that processor type strictly exceeds $\frac{1}{2}$. Intuitively, $H1$ and $H2$ identify those tasks which should be assigned based on Idea2. $H1$ stands for “Set of tasks with type-1 processors as favorite and are heavy if they are assigned to their non-favorite processor type (type-2)”. Analogous for $H2$. (Obviously, a task in $H1$ or $H2$ might also be heavy on its favorite processor type.) Also, intuitively, $F1$ and $F2$ identify those tasks which should be assigned based on Idea1. $F1$ stands for “Set of tasks that have type-1 processors as their favorite and are not heavy on either processor type”.



Algorithm 2: FF-3C: for assigning tasks on a two-type heterogeneous multiprocessor platform

```
   Input  : $\tau$ denotes set of tasks; $\Pi$ denotes set of processors
   Output: $\tau[p]$ specifies the tasks assigned to processor $p$
1  Form sets $H1$, $H2$, $F1$, $F2$ as defined by Expressions (24)–(27)
2  $\forall p$: $U[p] := 0$
3  $\forall p$: $\tau[p] := \emptyset$
4  if ( first-fit($H1$, $P^1$) $\neq H1$) then declare FAILURE end
5  if ( first-fit($H2$, $P^2$) $\neq H2$) then declare FAILURE end
6  $F11 :=$ first-fit($F1$, $P^1$)
7  $F22 :=$ first-fit($F2$, $P^2$)
8  if ($F11 = F1$) $\wedge$ ($F22 = F2$) then declare SUCCESS end
9  if ($F11 \neq F1$) $\wedge$ ($F22 \neq F2$) then declare FAILURE end
10 if ($F11 \neq F1$) $\wedge$ ($F22 = F2$) then
11     $F12 := F1 \setminus F11$
12     if ( first-fit($F12$, $P^2$) $= F12$) then
13         declare SUCCESS
14     else
15         declare FAILURE
16     end
17 end
18 if ($F11 = F1$) $\wedge$ ($F22 \neq F2$) then
19     $F21 := F2 \setminus F22$
20     if ( first-fit($F21$, $P^1$) $= F21$) then
21         declare SUCCESS
22     else
23         declare FAILURE
24     end
25 end
```

Analogous for $F2$. From the definitions of $H1$, $H2$, $F1$, $F2$ (and Inequalities (4) and (5)), we have:

$$\tau_i \in H1 \quad \Rightarrow \quad u_i^2 > \frac{1}{2} \tag{28}$$

$$\tau_i \in H2 \quad \Rightarrow \quad u_i^1 > \frac{1}{2} \tag{29}$$

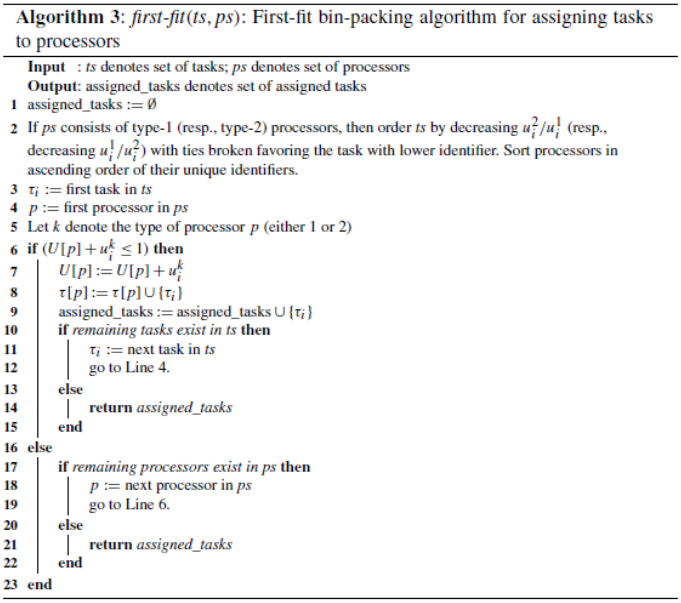
$$\tau_i \in F1 \quad \Rightarrow \quad u_i^1 \leq \frac{1}{2} \wedge u_i^2 \leq \frac{1}{2} \tag{30}$$

$$\tau_i \in F2 \quad \Rightarrow \quad u_i^1 \leq \frac{1}{2} \wedge u_i^2 \leq \frac{1}{2} \tag{31}$$

Algorithm 2 shows the pseudo-code of the new algorithm FF-3C. The intuition behind the design of FF-3C is that first we assign tasks to their favorite processors which would be heavy on other processor type (Lines 4–5). Then we assign the non-heavy tasks to their favorite processors (Lines 6–7). Then, if there are remaining non-heavy tasks, these have to be assigned to processors that are not their favorite (Lines 12 and 20).

Table 1 An example task set to illustrate the working of FF-3C algorithm

	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_9
u_i^1	0.60	0.70	0.14	0.35	0.98	0.10	0.25	0.60	0.15
u_i^2	0.80	0.06	0.48	0.25	0.75	0.15	0.85	0.20	0.10



Algorithm 3: *first-fit*(ts, ps): First-fit bin-packing algorithm for assigning tasks to processors

```
   Input  : $ts$ denotes set of tasks; $ps$ denotes set of processors
   Output: assigned_tasks denotes set of assigned tasks
1  assigned_tasks := $\emptyset$
2  If $ps$ consists of type-1 (resp., type-2) processors, then order $ts$ by decreasing $u_i^2/u_i^1$ (resp.,
   decreasing $u_i^1/u_i^2$) with ties broken favoring the task with lower identifier. Sort processors in
   ascending order of their unique identifiers.
3  $\tau_i$ := first task in $ts$
4  $p$ := first processor in $ps$
5  Let $k$ denote the type of processor $p$ (either 1 or 2)
6  if $(U[p] + u_i^k \leq 1)$ then
7      $U[p] := U[p] + u_i^k$
8      $\tau[p] := \tau[p] \cup \{\tau_i\}$
9      assigned_tasks := assigned_tasks $\cup \{\tau_i\}$
10     if remaining tasks exist in $ts$ then
11         $\tau_i$ := next task in $ts$
12         go to Line 4.
13     else
14         return assigned_tasks
15     end
16 else
17     if remaining processors exist in $ps$ then
18         $p$ := next processor in $ps$
19         go to Line 6.
20     else
21         return assigned_tasks
22     end
23 end
```

FF-3C is named after the fact that each task has three chances to be assigned using first-fit: (i) according to Idea2 (to avoid making a task heavy), (ii) assignment to its favorite and (iii) assignment to its non-favorite processor type.

As already mentioned, the FF-3C algorithm performs several passes with first-fit bin-packing. It uses the subroutine `first-fit` (see Algorithm 3 for pseudo-code) which takes two parameters, a set of tasks to be assigned using first-fit bin-packing and a set of processors to assign these tasks, and it returns the set of successfully assigned tasks. FF-3C keeps track of processor utilizations in a global vector U, initialized to zero (Line 2).

4.2 An example to illustrate the working of FF-3C

In this section, we illustrate the working of FF-3C with an example.

Example 2 Consider a two-type heterogeneous multiprocessor platform with one processor of type-1 (namely, P_1) and two processors of type-2 (namely, P_2 and P_3) and a task set as shown in Table 1. Let us see how FF-3C assigns the task set to processors. The task set τ is partitioned as follows: $\tau^1 = \{\tau_1, \tau_3, \tau_6, \tau_7\}$ and $\tau^2 = \{\tau_2, \tau_4, \tau_5, \tau_8, \tau_9\}$—see Inequalities (4). On Line 1, FF-3C (pseudo-code shown in Algorithm 2) forms sets $H1, H2, F1$ and $F2$ (as defined by Inequalities (24)–(27)) as follows: $H1 = \{\tau_1, \tau_7\}$, $H2 = \{\tau_2, \tau_5, \tau_8\}$, $F1 = \{\tau_3, \tau_6\}$ and $F2 = \{\tau_4, \tau_9\}$.

On Line 4, FF-3C calls first-fit sub-routine (shown in Algorithm 3) to assign tasks in $H1 = \{\tau_1, \tau_7\}$ on processor P_1 (of type-1). The first-fit sub-routine (on Line 2 in Algorithm 3) sorts the tasks in $H1$ in descending order of u_i^2/u_i^1, i.e., (τ_7, τ_1). The sub-routine successfully assigns both the tasks in $H1$ to processor P_1. After assigning $H1$ tasks, the remaining utilization of processor P_1 is 0.15.

On Line 5, FF-3C calls first-fit sub-routine to assign tasks in $H2 = \{\tau_2, \tau_5, \tau_8\}$ on processor P_2 and P_3 (of type-2). The first-fit sub-routine sorts the tasks in $H2$ in ascending order of u_i^2/u_i^1, i.e., (τ_2, τ_8, τ_5). The sub-routine successfully assigns τ_2 and τ_8 to processor P_2 (but fails to assign τ_5 to P_2) and τ_5 to processor P_3. After assigning $H2$ tasks, the remaining utilization of processor P_2 is 0.74 and the remaining utilization of processor P_3 is 0.25.

On Line 6, FF-3C calls first-fit sub-routine to assign tasks in $F1 = \{\tau_3, \tau_6\}$ on processor P_1 (of type-1). The first-fit sub-routine sorts the tasks in $F1$ in descending order of u_i^2/u_i^1, i.e., (τ_3, τ_6). The sub-routine successfully assigns the task τ_3 to P_1 but fails to assign τ_6 to P_1 as there is not enough capacity left in P_1. After assigning τ_3, the remaining utilization of processor P_1 is 0.01. Hence, when first-fit returns on Line 6, we have: $F11 = \{\tau_3\}$.

On Line 7, FF-3C calls first-fit sub-routine to assign tasks in $F2 = \{\tau_4, \tau_9\}$ on processors P_2 and P_3 (of type-2). The first-fit sub-routine (on Line 2 in Algorithm 3) sorts the tasks in $F2$ in ascending order of u_i^2/u_i^1, i.e., (τ_9, τ_4). The sub-routine successfully assigns both the tasks in $F2$ to processor P_2. After assigning τ_9 and τ_4, the remaining utilization of processor P_2 is 0.39. Hence, when first-fit returns on Line 7, we have: $F22 = \{\tau_4, \tau_9\}$.

The condition on Line 10, i.e., $(F11 /= F1) \wedge (F22 = F2)$ is TRUE and hence, new task set $F12$ is formed on Line 11, i.e., $F12 = \{\tau_6\}$. FF-3C on Line 12 calls first-fit sub-routine to assign tasks in $F12$ to processors P_2 and P_3 (of type-2). The first-fit sub-routine successfully assigns the single task τ_6 of $F12$ to processors P_2. The remaining utilization of processor P_2 is 0.24. Since the sub-routine managed to assign all tasks in $F12$ to type-2 processors, FF-3C declares SUCCESS on Line 13.

So, the final assignment of tasks to processors looks as follows: τ_1, τ_3 and τ_7 are assigned to processor P_1 (of type-1), $\tau_2, \tau_4, \tau_6, \tau_8$ and τ_9 are assigned to processor P_2 (of type-2) and τ_5 is assigned to processor P_3 (of type-2).

4.3 The speed competitive ratio of FF-3C

In this section, we will prove the speed competitive ratio of FF-3C. We will derive its speed competitive ratio in terms of a task set parameter, namely α. The parameter α is a property of the task set on which FF-3C is applied and it reflects the values that the task utilizations on either processor types range over. Specifically, $0 < \alpha \leq 0.5$ is the smallest number such that, for each task (in the task set on which FF-3C is applied), it holds that its utilization is no greater than α or greater than $1 - \alpha$ on a processor of type-1 and its utilization is no greater than α or greater than $1 - \alpha$ on a processor of type-2.

Lemma 7 *Let α denote a real number*:

$$0 < \alpha \leq \frac{1}{2} \tag{32}$$

Let us derive a new task set τ' *from the task set* τ *as follows*:

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

If *for* τ *, it holds that*:

$$\begin{aligned} \forall \tau_i \in \tau : \quad & \left(u_i^1 \leq \alpha\right) \vee \left(1 - \alpha < u_i^1\right) \quad \textit{and} \\ \forall \tau_i \in \tau : \quad & \left(u_i^2 \leq \alpha\right) \vee \left(1 - \alpha < u_i^2\right) \end{aligned} \tag{34}$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-3C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Proof An equivalent claim is that if a task set τ is not schedulable under FF-3C over a computing platform Π then the task set τ' would likewise be unschedulable, using any algorithm, over platform Π. We will prove this by contradiction.

Combining the definitions of H1-F2 (Inequalities (28)–(31)), the definition of α (Inequality (32) in Lemma 7) and the assumptions of task set τ (Inequality (34) in Lemma 7), we obtain:

$$\tau_i \in H1 \quad \overset{(28)}{\Rightarrow} \quad u_i^2 > \frac{1}{2} \quad \overset{(32)}{\Rightarrow} \quad u_i^2 \not\leq \alpha \quad \overset{(34)}{\Rightarrow} \quad u_i^2 > 1 - \alpha \tag{35}$$

$$\tau_i \in H2 \quad \overset{(29)}{\Rightarrow} \quad u_i^1 > \frac{1}{2} \quad \overset{(32)}{\Rightarrow} \quad u_i^1 \not\leq \alpha \quad \overset{(34)}{\Rightarrow} \quad u_i^1 > 1 - \alpha \tag{36}$$

$$\begin{aligned} \tau_i \in F1 \quad & \overset{(30)}{\Rightarrow} \quad u_i^1 \leq \frac{1}{2} \wedge u_i^2 \leq \frac{1}{2} \quad \overset{(32)}{\Rightarrow} \quad u_i^1 \not> 1 - \alpha \wedge u_i^2 \not> 1 - \alpha \\ & \overset{(34)}{\Rightarrow} \quad u_i^1 \leq \alpha \wedge u_i^2 \leq \alpha \end{aligned} \tag{37}$$

$$\begin{aligned} \tau_i \in F2 \quad & \overset{(31)}{\Rightarrow} \quad u_i^1 \leq \frac{1}{2} \wedge u_i^2 \leq \frac{1}{2} \quad \overset{(32)}{\Rightarrow} \quad u_i^1 \not> 1 - \alpha \wedge u_i^2 \not> 1 - \alpha \\ & \overset{(34)}{\Rightarrow} \quad u_i^1 \leq \alpha \wedge u_i^2 \leq \alpha \end{aligned} \tag{38}$$

Assume that FF-3C failed to assign τ on Π but it is possible (using an algorithm OPT) to assign τ' on Π. Since FF-3C failed to assign τ on Π, it must have declared FAILURE. We explore all possibilities for the failure of FF-3C to occur:

Failure on Line 4 in FF-3C: It has been shown (López et al. 2004) that if first-fit (or any other *reasonable allocation algorithm*[5]) is used and

$$\sum_{\tau_i \in \tau} u_i \leq m - (m-1)u_{max}$$

then the task set is successfully assigned on an identical multiprocessor platform; where m is the number of processors and u_{max} is the maximum utilization of any task in the given task set.

Clearly, from trivial arithmetic, we have $m(1 - u_{max}) \leq m - (m-1)u_{max}$ and this gives us the following: if first-fit (or any other reasonable allocation algorithm) is used and

$$\sum_{\tau_i \in \tau} u_i \leq m(1 - u_{max})$$

then the task set is successfully assigned on an identical multiprocessor platform.

Applying the above expression to the tasks in $H1$ for which it holds that $\forall \tau_i \in H1 : u_i^1 \leq \alpha$ (shown later in the proof, immediately after Expression (54)) and to the type-1 processors, we obtain:

$$\text{If} \quad \sum_{\tau_i \in H1} u_i \leq \left|P^1\right|(1-\alpha) \quad \text{then first-fit succeeds.}$$

Since FF-3C failed (because first-fit failed), it must hold that

$$\sum_{\tau_i \in H1} u_i^1 > \left|P^1\right| \cdot (1-\alpha) \quad \overset{(33)}{\Rightarrow} \quad \sum_{\tau_i \in H1} u_i^{1'} > \left|P^1\right|$$

Therefore, OPT cannot assign all tasks in $H1$ to P^1. Hence, it assigns at least one task $\tau_i \in H1$ to P^2. From Expression (33) and (35) we get $u_i^{2'} > 1$, hence (from Lemma 2) OPT produces an infeasible assignment—a contradiction.

Failure on Line 5 in FF-3C: This results in contradiction (symmetric to the case above).

Failure on Line 9 in FF-3C: From the case, we obtain that $F11 \subset F1$ and $F22 \subset F2$. Therefore, when executing Line 6 in FF-3C, there was a task $\tau_{failed1} \in F1$ which could not be assigned on any processor in P^1 and when executing Line 7 in FF-3C there was a task $\tau_{failed2} \in F2$ which could not be assigned on any processor in P^2. Hence:

$$\forall p \in P^1 : \quad U[p] + u^1_{failed1} > 1 \quad \text{and} \tag{39}$$

$$\forall p \in P^2 : \quad U[p] + u^2_{failed2} > 1 \tag{40}$$

where $U[p]$ is the current utilization of a processor p.

[5] A reasonable allocation algorithm is an algorithm that fails to assign a task *only* when there is no processor in the system that can hold the task (López et al. 2004). Allocation algorithms such as first-fit and best-fit are two examples of reasonable allocation algorithms.

We know from Expression (37) that $u^1_{failed1} \leq \alpha$ and from Expression (38) that $u^2_{failed2} \leq \alpha$. Using these on Inequalities (39) and (40) gives:

$$\forall p \in P^1 : \quad U[p] > 1 - \alpha \quad \text{and} \tag{41}$$

$$\forall p \in P^2 : \quad U[p] > 1 - \alpha \tag{42}$$

Observing that tasks assigned on processors in P^1 are a subset of τ^1 and using Inequality (41) gives us:

$$u^1 \qquad \sum_{\tau_i \in \tau^1} u_i^1 > \left|P^1\right| \cdot (1 - \alpha) \tag{43}$$

With analogous reasoning, Inequality (42) gives us:

$$u^2 \qquad \sum_{\tau_i \in \tau^2} u_i^2 > \left|P^2\right| \cdot (1 - \alpha) \tag{44}$$

Applying Expression (33) on Inequalities (43) and (44), we obtain:

$$u^{1'} \qquad \sum_{\tau_i \in \tau^1} u_i^{1'} > \left|P^1\right| \quad \text{and} \tag{45}$$

$$, \qquad \sum_{\tau_i \in \tau^2} u_i^{2'} > \left|P^2\right| \tag{46}$$

Observing these two inequalities and Lemma 3 gives us that OPT fails to assign τ' on Π. This is a contradiction.

Failure on Line 15 in FF-3C: From the case, we obtain that $F11 \subset F1$ and $F22 = F2$. Therefore, when executing Line 12 there was a task $\tau_{failed} \in (F1 \setminus F11)$ for which an assignment attempt was made on each of the processors in P^2. But all of these attempts failed. Therefore:

$$\forall p \in P^2 : \quad U[p] + u^2_{failed} > 1 \tag{47}$$

We can add these inequalities together and get: (48)

$$\sum_{p \in P^2} U[p] > \left|P^2\right| \cdot \left(1 - u^2_{failed}\right) \qquad (48) \qquad)$$

We know that the tasks assigned to processors in P^2 are $H2 \cup F22 \cup \tau^{F12assigned}$ where $\tau^{F12assigned}$ is the set of tasks that were assigned when executing Line 12 of FF-3C. We also know that $\tau^{F12assigned} \subset F12$. Hence, Inequality (48) becomes:

$$\underset{\tau_i \in (H2 \cup F22 \cup F12)}{} u_i^2 > P^2 \cdot \left(1 - u_{failed}^2\right)$$

From Expression (37), we obtain $u^2_{failed} \leq \alpha$. Thus, the above inequality becomes:

$$\sum_{\tau_i \in (H2 \cup F22 \cup F12)} u_i^2 > \left|P^2\right| \cdot (1-\alpha) \qquad (49)$$

We also know that FF-3C has executed Line 6 and when it performed first-fit bin-packing, there must have been a task $\tau_{failed1} \in (F1 \setminus F11)$ which was attempted to each of the processors in P^1. But all of them failed. Note that this task $\tau_{failed1}$ may be the same as τ_{failed} or it may be different. Because it was not possible to assign $\tau_{failed1}$ on any of the processors in P^1, we have:

$$\forall p \in P^1 : \quad U[p] + u^1_{failed1} > 1 \qquad (50)$$

Adding these inequalities together gives us: (51)

$$\sum_{p \in P^1} U[p] > \left|P^1\right| \cdot \left(1 - u^1_{failed1}\right) \qquad (51)$$

We know that the tasks assigned to processors in P^1 just after executing Line 6 in FF-3C are $H1 \cup F11$. Also, we know from Expression (37) that $u^1_{failed1} \leq \alpha$. Therefore, we have:

$$\sum_{\tau_i \in (H1 \cup F11)} u_i^1 > \left|P^1\right| \cdot (1-\alpha) \qquad (52)$$

Let us now discuss OPT, the algorithm which succeeds in assigning the task set τ' on platform Π. Let us discuss tasks in $H1$. From Expression (35), we know that:

$$\forall \tau_i \in H1 : \quad u_i^2 > 1 - \alpha \qquad (53)$$

Using Expression (33) gives us:

$$\forall \tau_i \in H1 : \quad u_i^{2'} > 1 \qquad (54)$$

If $\exists \tau_i \in H1 : u^1_i > 1 - \alpha$, then $\exists \tau_i \in H1 : u^{1'}_i > 1$ and using $\tau_i \in H1$ and Inequality (4) gives us $\exists \tau_i \in H1 \subseteq \tau^1 : u_i^{2'} > 1$. Hence such a task cannot be assigned by OPT on any processor of Π (of any type) and this is a contradiction. Hence we can assume that $\forall \tau_i \in H1 : u^1_i \leq 1 - \alpha$, to be precise, $\forall \tau_i \in H1 : u^1_i \leq \alpha$—see Expression (34). Combining this and Expression (33), we get:

$$\forall \tau_i \in H1 : \quad u_i^{1'} \leq 1 \qquad (55)$$

Using Inequalities (54) and (55) yields that every task in H1 is assigned to processors in P^1 by OPT. With analogous reasoning, we have that every task in $H2$ is assigned to a processor in P^2. Let τ^{OPT1} denote the tasks (except those from $H1$) assigned to processors in P^1 by OPT. Analogously, let τ^{OPT2} denote the tasks (except those from $H2$) assigned to processors in P^2 by OPT. Therefore (using Inequalities (1) and (2)), we know that:

$$\sum_{\tau_i \in (H1 \cup \tau^{OPT1})} u_i^{1'} \leq |P^1| \quad \text{and} \tag{56}$$

$$\sum_{\tau_i \in (H2 \cup \tau^{OPT2})} u_i^{2'} \leq |P^2| \tag{57}$$

Using Expression (33) gives us:

$$\sum_{\tau_i \in (H1 \cup \tau^{OPT1})} u_i^1 \leq |P^1| \cdot (1-\alpha) \quad \text{and} \tag{58}$$

$$\sum_{\tau_i \in (H2 \cup \tau^{OPT2})} u_i^2 \leq |P^2| \cdot (1-\alpha) \tag{59}$$

We can now reason about the inequalities we obtained about the assignments of FF-3C and OPT. Rewriting Inequalities (52) and (58) respectively yields:

$$\sum_{\tau_i \in F11} u_i^1 > |P^1| \cdot (1-\alpha) - \sum_{\tau_i \in H1} u_i^1 \tag{60}$$

$$\sum_{\tau_i \in \tau^{OPT1}} u_i^1 \leq |P^1| \cdot (1-\alpha) - \sum_{\tau_i \in H1} u_i^1 \tag{61}$$

We can see that Inequalities (60) and (61) with $x = \text{J,}_{\tau_i \in H1} u_i^1$ and $y = \alpha$ ensure that the assumptions of Lemma 6 are true, given also the ordering of $F1$ during assignment over P^1 (Line 2 in Algorithm 3), which ensures that $\forall \tau_i \in F11, \forall \tau_j \in F12$: $\frac{u_i^2}{u_i^1} \geq \frac{u_j^2}{u_j^1}$. Using Lemma 6 gives us:

$$\sum_{\tau_i \in H1} u_i^1 + \sum_{\tau_i \in H2} u_i^2 + \sum_{\tau_i \in F11} u_i^1 + \sum_{\tau_i \in F12} u_i^2 + \sum_{\tau_i \in F22} u_i^2$$
$$\leq \sum_{\tau_i \in H1} u_i^1 + \sum_{\tau_i \in H2} u_i^2 + \sum_{\tau_i \in \tau^{OPT1}} u_i^1 + \sum_{\tau_i \in \tau^{OPT2}} u_i^2$$

Applying Inequalities (58) and (59) to the inequality above gives us:

$$\sum_{\tau_i \in H1} u_i^1 + \sum_{\tau_i \in H2} u_i^2 + \sum_{\tau_i \in F11} u_i^1 + \sum_{\tau_i \in F12} u_i^2 + \sum_{\tau_i \in F22} u_i^2$$
$$\leq |P^1| \cdot (1-\alpha) + |P^2| \cdot (1-\alpha) \tag{62}$$

Applying Inequalities (49) and (52) to left-hand side of Inequality (62) gives us:

$<$

$$\begin{aligned}&\left|P^1\right| \cdot (1-\alpha) + \left|P^2\right| \cdot (1-\alpha)\\ &\quad < \left|P^1\right| \cdot (1-\alpha) + \left|P^2\right| \cdot (1-\alpha)\end{aligned} \tag{63}$$

This is a contradiction.

Failure on Line 23 in FF-3C: A contradiction results—proof analogous to previous case.

We see that all cases where FF-3C declares FAILURE lead to contradiction. Hence, the lemma holds. D

Note: The value of α must depend on the utilization of tasks in the task set on which FF-3C is applied. To apply the above result for a task assignment problem, α must be assigned the smallest value so that Expression (34) holds for the task set. As the value of α increases, the speed competitive ratio of FF-3C also increases.

In Lemma 7, we used α to denote a bound on the utilization of a task set (τ) on which we apply FF-3C and we stated a relation between the utilization of one task set (τ) used for FF-3C and another task set (τ') used for an optimal task assignment algorithm. It is sometimes convenient to express similar relationship but with an expression of a bound on the utilization of a task set on which we apply the optimal algorithm. For this purpose, we use α' to denote a bound on the utilization of a task set (τ') on which the optimal algorithm is applied. Let $\alpha' = \frac{\alpha}{1-\alpha}$. Algebraic rewriting gives us $\alpha = \frac{\alpha'}{1+\alpha'}$. With this α', note that the expression $u_i^{1'} = \frac{u_i^1}{1-\alpha}$ can be rewritten as: $u_i^1 = u_i^{1'} \times (1 - \frac{\alpha'}{1+\alpha'})$ which can be rewritten as: $u_i^1 = \frac{u_i^{1'}}{1+\alpha'}$. Also, with this α', the expression $u_i^1 \leq \alpha$ can be rewritten as: $u_i^{1'} \leq \alpha'$. Applying this on Lemma 7 gives us:

Lemma 8 *Let α' denote a real number*:

$$0 < \alpha' \leq 1 \tag{64}$$

Let us derive a new task set τ from the task set τ' as follows:

$$\forall \tau_i \in \tau' : \quad u_i^1 = \frac{u_i^{1'}}{1+\alpha'} \wedge u_i^2 = \frac{u_i^{2'}}{1+\alpha'} \tag{65}$$

***If** for τ', it holds that*:

$$\begin{aligned}&\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}\\ &\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)\end{aligned} \tag{66}$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-3C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Proof The proof follows from the discussion above. D

The above result can also be expressed in terms of the additional processor speed required by FF-3C as compared to that of an optimal algorithm for scheduling a given task set.

Theorem 1 *Let* α' *denote a real number*: $0 < \alpha' \leq 1$.

***If** for a task set* τ'*, it holds that*:

$$\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}$$

$$\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)$$

then

$$\begin{aligned} &\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \\ &\quad \Rightarrow \textit{sched}\left(\text{FF-3C}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right) \times \langle 1+\alpha', 1+\alpha'\rangle\right) \end{aligned}$$

Proof The theorem directly follows from Lemma 8. D

Note: The value of α' must depend on the utilization of tasks in the task set (τ ') on which the optimal algorithm is applied. To apply the above results for a task assignment problem, α' must be assigned the smallest value so that Expression (66) holds for a given task set.

Theorem 2 *The speed competitive ratio of FF*-3*C is at most* 2.

Proof The proof follows from applying $\alpha' = 1$ in Theorem 1. D

Note: Our results continue to hold if we replace first-fit with any *reasonable allocation algorithm* that has a resource augmentation bound of $1 - \alpha$. Another example of such an algorithm is best-fit. We have used first-fit for ease of explanation.

5 Time-complexity of FF-3C

We show that the time-complexity of FF-3C is a low-degree polynomial function of the number of tasks (n) and processors (m). By inspection of the pseudo-code for FF-3C (Algorithm 2), the function *first-fit* is invoked at most 5 times. Within each of those invocations:

- Sorting is performed over a subset of τ (i.e., at most n tasks). The time-complexity of this operation is $O(n \cdot \log n)$ e.g., using Heapsort.
- Sorting is performed over all processors, (i.e., m processors). The time complexity of this operation is $O(m \cdot \log m)$.
- First-fit bin-packing is performed whose time complexity is $O(n \cdot m)$.

Thus the time-complexity of the algorithm is at most

$$5 \cdot \big(\underbrace{O(n \cdot \log n)}_{\text{sort tasks}} + \underbrace{O(m \cdot \log m)}_{\text{sort processors}} + \underbrace{O(n \cdot m)}_{\text{bin-packing}}\big) = O\big(n \cdot \max(m, \log n) + m \cdot \log m\big)$$

6 Extensions to FF-3C

We now discuss how to enhance FF-3C to attain better average-case performance.

6.1 The FF-4C algorithm

One drawback of FF-3C is the early declaration of failure while trying to assign heavy tasks. If heavy tasks could not be assigned to their favorite processor type then FF-3C declares failure (on Lines 4 and 5 in Algorithm 2) without even trying to assign them on their non-favorite processor type. In an extreme case, FF-3C would fail with a system composed of (i) a heavy task of type H1 (resp., of type H2) that could fit on a processor of type-2 (resp., type-1) and (ii) zero processors of type-1 (resp., type-2) and infinite processors of type-2 (resp., type-1). FF-4C, an enhanced version of FF-3C, overcomes this drawback and gives better average-case performance than FF-3C. The algorithm FF-4C, upon failing to assign tasks in H1 (resp., H2) on processors of type-1 (resp., type-2), tries to assign those unassigned tasks onto their non-favorite processors of type-2 (resp., type-1).

The pseudo-code of FF-4C is shown in Algorithm 4. Lines 1–3 of FF-4C are the same as that of Lines 1–3 of FF-3C (shown in Algorithm 2) and Lines 21–40 of FF-4C are same as that of Lines 6–25 of FF-3C. Lines 4–5 of FF-3C are replaced as shown in Lines 4–20 of FF-4C.

6.1.1 The speed competitive ratio of FF-4C

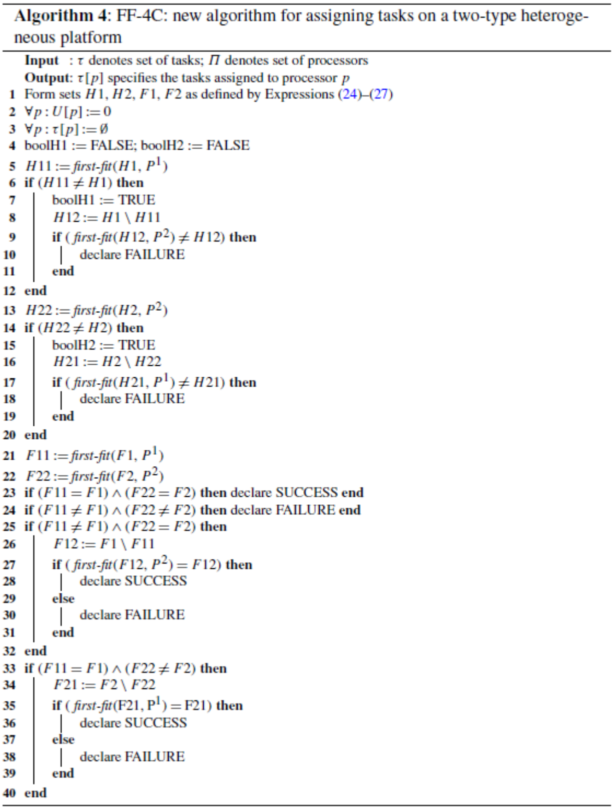
We first prove the superiority of FF-4C in terms of set of tasks that it can successfully schedule as compared to that of FF-3C and then we prove the speed competitive ratio of FF-4C.

Theorem 3 *The task sets that are schedulable by FF-4C are a strict superset of those that are schedulable by FF-3C.*

Proof To prove that the claim is true, we need to show that:

1. whenever FF-4C fails, FF-3C would also fail and
2. there is at least one task set τ for which FF-3C fails to assign τ on Π whereas FF-4C succeeds in assigning τ on Π.

The intuition for proving 1. is that if $H11 = H1$ and $H22 = H2$ (i.e., the code between Lines 7–11 and 15–19 are not executed in FF-4C) then the behavior of FF-4C is exactly the same as that of FF-3C. For proving 1., we consider all the cases where



Algorithm 4: FF-4C: new algorithm for assigning tasks on a two-type heterogeneous platform

```
   Input  : $\tau$ denotes set of tasks; $\Pi$ denotes set of processors
   Output: $\tau[p]$ specifies the tasks assigned to processor $p$
1  Form sets $H1$, $H2$, $F1$, $F2$ as defined by Expressions (24)–(27)
2  $\forall p : U[p] := 0$
3  $\forall p : \tau[p] := \emptyset$
4  boolH1 := FALSE; boolH2 := FALSE
5  $H11 :=$ first-fit($H1$, $P^1$)
6  if ($H11 \neq H1$) then
7      boolH1 := TRUE
8      $H12 := H1 \setminus H11$
9      if ( first-fit($H12$, $P^2$) $\neq H12$) then
10         declare FAILURE
11     end
12 end
13 $H22 :=$ first-fit($H2$, $P^2$)
14 if ($H22 \neq H2$) then
15     boolH2 := TRUE
16     $H21 := H2 \setminus H22$
17     if ( first-fit($H21$, $P^1$) $\neq H21$) then
18         declare FAILURE
19     end
20 end
21 $F11 :=$ first-fit($F1$, $P^1$)
22 $F22 :=$ first-fit($F2$, $P^2$)
23 if ($F11 = F1$) $\wedge$ ($F22 = F2$) then declare SUCCESS end
24 if ($F11 \neq F1$) $\wedge$ ($F22 \neq F2$) then declare FAILURE end
25 if ($F11 \neq F1$) $\wedge$ ($F22 = F2$) then
26     $F12 := F1 \setminus F11$
27     if ( first-fit($F12$, $P^2$) $= F12$) then
28         declare SUCCESS
29     else
30         declare FAILURE
31     end
32 end
33 if ($F11 = F1$) $\wedge$ ($F22 \neq F2$) then
34     $F21 := F2 \setminus F22$
35     if ( first-fit(F21, $\mathrm{P}^1$) = F21) then
36         declare SUCCESS
37     else
38         declare FAILURE
39     end
40 end
```

FF-4C declares FAILURE and show that FF-3C will also declare FAILURE in each of those cases.

Failure on Line 10 in FF-4C: This implies that FF-4C could not assign all the tasks in H 1 to their favorite processor type P ^{1} and hence only few tasks (H 11) were assigned to P ^{1} and the rest were attempted to be assigned to their non-favorite processors P^2 and failed. In such a case, FF-3C would have declared failure on Line 4 (in Algorithm 2) itself as it would also fail to assign all the tasks in $H1$ to P^1 since it also uses the same *first-fit* algorithm (of Algorithm 3) that is used by FF-4C.

Failure on Line 18 in FF-4C: When the algorithm fails here, there are two scenarios that need to be considered with respect to the assignment of tasks in $H1$ (earlier in the algorithm): (i) all the tasks in $H1$ were successfully assigned to P^1 (indicated by $boolH1 = FALSE$, i.e., Lines 7–11 were not executed at all) and (ii) only few tasks from $H1$ could be assigned to P^1 and hence the rest were assigned to P^2 (indicated by $boolH1 = TRUE$). For the first scenario, the proof is *symmetric* to the previous case (i.e., proof given for 'Failure on Line 10 in FF-4C'; FF-3C would have declared FAILURE on Line 5 itself as it would also fail to assign all the tasks in $H2$ to processors in P^2). For the second scenario, the proof is analogous to the previous case as FF-3C would have declared FAILURE on Line 4 (in Algorithm 2) itself as soon as a task from H1 was failed to be assigned to P^1.

Failure on Lines 24, 30 and 38 in FF-4C: When the algorithm fails on one of these lines, our proof depends on the assignment of tasks in $H1$ (resp., $H2$) earlier in the algorithm, i.e., whether all the tasks of $H1$ (resp., $H2$) have been successfully assigned to their favorite processors, i.e., P^1 (resp., P^2) or only few tasks could be assigned to their favorite processors and the rest to the non-favorite processors, i.e., $H11$ on P^1 and $H12$ on P^2 (resp., $H22$ on P^2 and $H21$ on P^1). In the algorithms, this information is captured using a boolean variable, $boolH1$ (resp., $boolH2$). For example, $boolH1 = FALSE$ indicates that all the tasks of $H1$ were assigned on their favorite processors P^1 and $boolH1 = TRUE$ implies that only few tasks from $H1$, i.e., $H11$ could be assigned on their favorite processors P^1 and the rest, i.e., $H12$, were assigned on their non-favorite processors P^2. Analogous explanation holds for boolean variable $boolH2$. Hence, with the help of these two boolean variables we have captured all the possible scenarios for FF-4C to fail (on one of the Lines 24, 30 or 38) in Table 2 along with the corresponding proof to look for in the paper (as the proofs provided earlier in the paper can be reused to reason about these scenarios).

For proving 2., we illustrate the superiority of FF-4C over FF-3C with an example task set. D

Example 3 Consider a platform comprising a processor P_1 of type-1 and a processor P_2 of type-2, a task set $\tau = \{\tau_1, \tau_2, \tau_3\}$ shown in Table 3.

It is trivial to observe that a schedulable assignment exists for this task set on the given platform: assign τ_1 and τ_3 to P_1 and τ_2 to P_2. We now simulate the behavior of FF-4C and FF-3C for this task set on the given platform and show that FF-4C succeeds whereas FF-3C fails.

First, let us look at FF-4C. On Line 1 (see Algorithm 4), FF-4C groups the tasks as follows: $H1 = \{\tau_1, \tau_2\}$ and $F1 = \{\tau_3\}$.

On Line 5, FF-4C calls first-fit sub-routine to assign tasks in $H1$ to processor P_1 of type-1. The sub-routine succeeds in assigning task τ_1 to P_1 and fails to assign the other task τ_2 to P_1 as there is not enough capacity left on P_1. Hence, after executing Line 5 of FF-4C, we have: $H11 = \{\tau_1\}$. After assigning τ_1 to P_1, the remaining utilization on P_1 is $\frac{1}{2} - E$.

Table 2 Summary of proof of speed competitive ratio of FF-4C for different cases

boolH1	boolH2	Explanation of the scenario	Use the reasoning of
FALSE	FALSE	All the tasks of $H1$ and $H2$ were assigned to their favorite processors P^1 and P^2 respectively. This indicates that the behavior of FF-4C is same as that of FF-3C in this case (i.e., code on Lines 7–11 and 15–19 of FF-4C is not executed). Hence, the reason for failure of FF-4C on Line 24, 30 and 38 is same as that of failure of FF-3C on Line 9, 15 and 23	Proof of Lemma 7, 'Failure on Line 9, 15 and 23' respectively
FALSE	TRUE	Only few tasks of $H2$ ($H22$) could be assigned on P^2 and the rest ($H21$) were assigned to P^1. In such a case, FF-3C would have failed on Line 5 itself during the assignment of $H2$ on P^2 as it fails to assign all the tasks from $H2$ on P^2 and does not even try to assign the failed tasks of $H2$ on P^1	Proof of Theorem 3, 'Failure on Line 18 in FF-4C'
TRUE	FALSE	This case is analogous to the previous case where only few tasks of $H1$ ($H11$) could be assigned to P^1 and rest ($H12$) were assigned to P^2. In this case, FF-3C would have failed on Line 4 itself during the assignment of $H1$ on P^1 as it fails to assign all the tasks from $H1$ on P^1 and does not even try to assign the failed tasks of $H1$ on P^2	Proof of Theorem 3, 'Failure on Line 10 in FF-4C'
TRUE	TRUE	This case is similar to one of the two previous cases, i.e., boolH1 = FALSE ∧ boolH2 = TRUE and boolH1 = TRUE ∧ boolH2 = FALSE	Proof of Theorem 3, 'Failure on Line 10 in FF-4C' and 'Failure on Line 18 in FF-4C' respectively

Table 3 An example task set schedulable by FF-4C but not by FF-3C

τ_i	u_i^1	u_i^2	belongs to
τ_1	$\frac{1}{2}+E$	$\frac{1}{2}+2E$	H1
τ_2	$\frac{1}{2}+E$	$\frac{1}{2}+2E$	H1
τ_3	$\frac{1}{2}-E$	$\frac{1}{2}$	F1

On Line 8, it creates $H12 = \{\tau_2\}$.

On Line 9, it successfully assigns τ_2 to processor P_2 using first-fit sub-routine. After assigning τ_2 to processor P_2; the remaining utilization on P_2 is $\frac{1}{2} - 2E$.

On Line 21, it successfully assigns τ_3 (of $F1$) to processor P_1 using first-fit sub-routine. After assigning τ_3 to P_1, the remaining utilization on P_1 is 0.

So, the final assignment of tasks is as follows: τ_1 and τ_3 are assigned to P_1 and τ_2 is assigned to P_2—hence, FF-4C succeeds.

Now let us look at FF-3C. FF-3C groups the tasks as follows: $H1 = \{\tau_1, \tau_2\}$ and $F1 = \{\tau_3\}$. FF-3C fails to assign both the tasks in $H1$ to processor P_1 of type-1 since the sum of their utilization $((\frac{1}{2} + E) + (\frac{1}{2} + E) = 1 + 2E)$ exceeds 1.0.

Hence FF-3C declares FAILURE on Line 4 (see Algorithm 2).

Thus, we showed that: 1. whenever FF-4C fails, FF-3C also fails and 2. there is at least one task set τ for which FF-3C fails to assign τ on Π whereas FF-4C succeeds in assigning τ on Π . Hence the theorem holds. D

Now we prove the speed competitive ratio of FF-4C.

Lemma 9 *Let α denote a real number*: $0 < \alpha \leq \frac{1}{2}$.

Let us derive a new task set τ' from the task set τ as follows:

$$\forall \tau_i \in \tau: \quad u_i^{1'} = \frac{u_i^1}{1-\alpha} \wedge u_i^{2'} = \frac{u_i^2}{1-\alpha}$$

***If** for τ, it holds that*:

$$\forall \tau_i \in \tau: \quad \left(u_i^1 \leq \alpha\right) \vee \left(1-\alpha < u_i^1\right) \quad \textit{and}$$
$$\forall \tau_i \in \tau: \quad \left(u_i^2 \leq \alpha\right) \vee \left(1-\alpha < u_i^2\right)$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-4C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Proof We know from Lemma 7 that

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-3C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \quad (67)$$

Also, from Theorem 3 we know that if FF-3C succeeds to assign a task set τ on a computing platform $\Pi\ (|P^1|, |P^2|)$ then FF-4C succeeds as well (on the same platform). Formally, this can be stated as:

$$\textit{sched}\left(\text{FF-3C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-4C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \qquad (68)$$

Combining Expression (67) and (68) gives us:

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-4C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Hence, the proof. D

Similar to Lemma 7, the above lemma uses α to denote a bound on the utilization of a task set (τ) on which we apply FF-4C and states a relation between the utilization of one task set (τ) used for FF-4C and another task set (τ') used for an optimal task assignment algorithm. Now, similar to Lemma 8, let us express this relationship with α', an expression of a bound on the utilization of a task set (τ') on which we apply the optimal algorithm.

Lemma 10 *Let α' denote a real number*: $0 < \alpha' \leq 1$.

Let us derive a new task set τ from the task set τ' as follows:

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

If *for τ', it holdsthat*:

$$\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}$$
$$\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-4C}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Proof The reasoning is analogous to the proof of Lemma 8. D

Now, we express the above result in terms of the additional processor speed required by FF-4C as compared to that of an optimal algorithm for scheduling a given task set.

Theorem 4 *Let α' denote a real number* $0 < \alpha' \leq 1$.
If *for a task set τ', it holds that*:

$$\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}$$
$$\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$
$$\Rightarrow \textit{sched}\left(\text{FF-4C}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right) \times \langle 1+\alpha', 1+\alpha' \rangle\right)$$

Proof The theorem directly follows from Lemma 10. D

Theorem 5 *The speed competitive ratio of FF-4C is at most* 2.

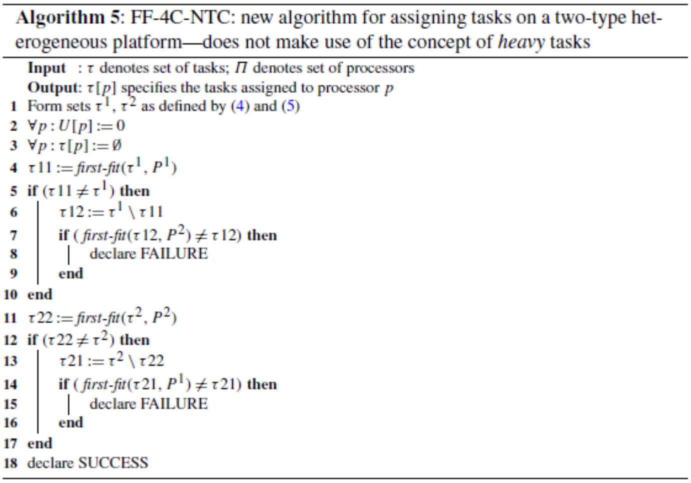
Proof The proof follows from applying $\alpha' = 1$ in Theorem 4. D

6.1.2 Time-complexity of FF-4C

We can use the same reasoning provided for the time-complexity of FF-3C in Sect. 5 for FF-4C as well. FF-4C uses the first-fit sub-routine at most seven times (see Algorithm 4) and each time (i) sorting is performed over at most n tasks whose complexity is $O(n \cdot \log n)$ (ii) sorting is performed over m processors whose complexity is $O(m \cdot \log m)$ and (iii) first-fit bin-packing takes $O(n \cdot m)$ time. Hence, the time-complexity of FF-4C is: $O(n \cdot \max(m, \log n) + m \cdot \log m)$.

6.2 The FF-4C-NTC algorithm

In FF-3C (and also in FF-4C), tasks are categorized as $H1$, $F1$, $H2$ and $F2$ and this makes it possible to prove the speed competitive ratio the way we do it. Unfortunately, this categorization can misguide the algorithm to assign a task in a way which causes



Algorithm 5: FF-4C-NTC: new algorithm for assigning tasks on a two-type heterogeneous platform—does not make use of the concept of *heavy* tasks

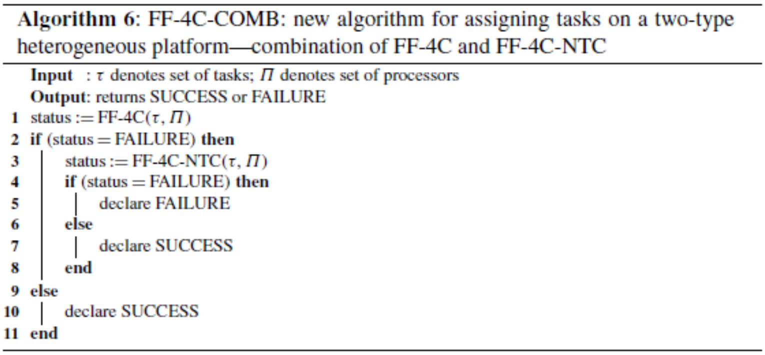
```
Input : $\tau$ denotes set of tasks; $\Pi$ denotes set of processors
Output: $\tau[p]$ specifies the tasks assigned to processor $p$
Form sets $\tau^1$, $\tau^2$ as defined by (4) and (5)
$\forall p : U[p] := 0$
$\forall p : \tau[p] := \emptyset$
$\tau 11 :=$ first-fit($\tau^1$, $P^1$)
if ($\tau 11 \neq \tau^1$) then
    $\tau 12 := \tau^1 \setminus \tau 11$
    if ( first-fit($\tau 12$, $P^2$) $\neq \tau 12$) then
        declare FAILURE
    end
end
$\tau 22 :=$ first-fit($\tau^2$, $P^2$)
if ($\tau 22 \neq \tau^2$) then
    $\tau 21 := \tau^2 \setminus \tau 22$
    if ( first-fit($\tau 21$, $P^1$) $\neq \tau 21$) then
        declare FAILURE
    end
end
declare SUCCESS
```

a failure later on. For example, consider a task set with two tasks τ_1 with $u_1^1 = 0.5$, $u_1^2 = 1.0$ and τ_2 with $u_2^1 = 1.0$, $u_2^2 = 1.0 + E$ and a platform comprising a processor P_1 of type-1 and P_2 of type-2. Clearly, there exists a schedulable assignment of the given task set on the given platform: assign τ_1 to P_2 and τ_2 to P_1. Now let us see what FF-3C does for this problem instance. FF-3C classifies τ_1 and τ_2 as $H1$ and assigns τ_1 to $P1$ and then tries to assign τ_2 to $P1$ but fails. FF-4C also exhibits similar behavior: it assigns τ_1 to $P1$ and then it attempts to assign τ_2 to $P2$ after an unsuccessful attempt to assign it to $P1$ and fails. Hence, both FF-3C and FF-4C fails on this task set. Therefore, we present a new algorithm namely, FF-4C-NTC to handle such cases.

The algorithm FF-4C-NTC classifies tasks as τ^1 and τ^2 as defined by Inequalities (4) and (5), and for each class, assigns tasks in order of decreasing u_i^2/u_i^1 for type-1 processors and decreasing u_i^1/u_i^2 for type-2 processors, respectively with ties broken favoring the task with lower identifier. FF-4C-NTC does not classify τ^1 into $H1$ and $F1$ nor τ^2 into $H2$ and $F2$ (as was the case with FF-3C and FF-4C): It only considers favorite/non-favorite processor types and disregards the information (used by both FF-3C and FF-4C) whether a task is heavy or not. The pseudo-code of FF-4C-NTC is shown in Algorithm 5. The algorithm first tries to assign tasks from τ^1 on their favorite processors of type P^1 using first-fit and if any of these tasks could not be assigned then it tries to assign them on their non-favorite processor type P^2—and analogously for τ^2. For the above example, FF-4C-NTC assigns τ_1 to P_2 and τ_2 to P_1.

FF-4C-NTC also has the same time-complexity of $O(n \cdot \max(m, \log n) + m \cdot \log m)$ as the previously discussed algorithms.

This algorithm will be used as a sub-routine in our next algorithm, namely FF-4C-COMB, discussed in Sect. 6.3. We will not use FF-4C-NTC as a stand-alone algorithm and hence we will not discuss its speed competitive ratio.


Algorithm 6: FF-4C-COMB: new algorithm for assigning tasks on a two-type heterogeneous platform—combination of FF-4C and FF-4C-NTC

```
Input  : $\tau$ denotes set of tasks; $\Pi$ denotes set of processors
Output: returns SUCCESS or FAILURE
status := FF-4C($\tau$, $\Pi$)
if (status = FAILURE) then
    status := FF-4C-NTC($\tau$, $\Pi$)
    if (status = FAILURE) then
        declare FAILURE
    else
        declare SUCCESS
    end
else
    declare SUCCESS
end
```

6.3 The FF-4C-COMB algorithm

As discussed in earlier sections, for some task sets FF-4C succeeds whereas FF-4C-NTC fails and for other task sets FF-4C-NTC succeeds whereas FF-4C fails. FF-4C-COMB exploits this fact by making use of both the algorithms to get the best out of the two—pseudo-code is listed in Algorithm 6. It first attempts to assign the task set with FF-4C and, upon failing, it tries with FF-4C-NTC.

6.3.1 The speed competitive ratio of FF-4C-COMB

In this section, we establish the speed competitive ratio of FF-4C-COMB.

Lemma 11 *Let α denote a real number*: $0 < \alpha \leq \frac{1}{2}$.
Let us derive a new task set τ ' from the task set τ as follows:

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

***If** for τ , it holds that*:

$$\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}$$
$$\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)$$

then

$$\textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \Rightarrow \textit{sched}\left(\text{FF-4C-COMB}, \tau, \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right)$$

Proof An equivalent claim is that if a task set τ is not schedulable under FF-4C-COMB over a computing platform Π then the task set τ' would likewise be unschedulable, using any algorithm, over computing platform Π. We will prove this by contradiction.

Assume that FF-4C-COMB has failed to assign τ on Π but it is possible (using an algorithm OPT) to assign τ' on Π. Since FF-4C-COMB failed to assign τ on Π, it follows that FF-4C-COMB declared FAILURE. We explore the only possibility for this to occur:

Failure on Line 5 in FF-4C-COMB: For FF-4C-COMB to declare FAILURE on this line, FF-4C must have failed on Line 1 (in Algorithm 6). But, from Lemma 9 we know that

$$sched\left(\textit{nmo-feasible}, \tau', \Pi\left(P^1, P^2\right)\right) \Rightarrow sched\left(\text{FF-4C}, \tau, \Pi\left(P^1, P^2\right)\right)$$

Since FF-4C declared FAILURE, it must hold that τ' is (nmo-) infeasible on Π. Hence, OPT produces an infeasible assignment—this is a contradiction. D

As done previously for FF-3C and FF-4C, the following lemma expresses this relationship with α', an expression of a bound on the utilization of a task set (τ') on which we apply the optimal algorithm.

Lemma 12 *Let α' denote a real number*: $0 < \alpha' \leq 1$.

Let us derive a new task set τ from the task set τ' as follows:

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

If *for τ', it holds that*:

$$\begin{aligned} \forall \tau_i \in \tau' : &\quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and} \\ \forall \tau_i \in \tau' : &\quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right) \end{aligned}$$

then

$$sched\left(\textit{nmo-feasible}, \tau', \Pi\left(P^1, P^2\right)\right) \Rightarrow sched\left(\text{FF-4C-COMB}, \tau, \Pi\left(P^1, P^2\right)\right)$$

Proof The reasoning is analogous to the proof of Lemma 8. D

The following theorem expresses the above result in terms of the additional processor speed required by FF-4C-COMB as compared to that of an optimal algorithm for scheduling a given task set.

Theorem 6 *Let α' denote a real number* $0 < \alpha' \leq 1$.

***If** for a task set τ', it holds that*:

$$\forall \tau_i \in \tau' : \quad \left(u_i^{1'} \leq \alpha'\right) \vee \left(1 < u_i^{1'}\right) \quad \textit{and}$$
$$\forall \tau_i \in \tau' : \quad \left(u_i^{2'} \leq \alpha'\right) \vee \left(1 < u_i^{2'}\right)$$

then

$$\begin{aligned} & \textit{sched}\left(\textit{nmo-feasible}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right)\right) \\ & \quad \Rightarrow \textit{sched}\left(\text{FF-4C-COMB}, \tau', \Pi\left(\left|P^1\right|, \left|P^2\right|\right) \times \left\langle 1+\alpha', 1+\alpha'\right\rangle\right) \end{aligned}$$

Proof The theorem directly follows from Lemma 12. D

Theorem 7 *The speed competitive ratio of FF-*4*C-COMB is at most* 2.

Proof The proof follows from applying $\alpha' = 1$ in Theorem 6. D

6.3.2 Time-complexity of FF-4C-COMB

We know that both FF-4C and FF-4C-NTC have the same time-complexity of $O(n \cdot \max(m, \log n) + m \cdot \log m)$. FF-4C-COMB (pseudo-code in Algorithm 5) calls FF-4C first and upon failing it calls FF-4C-NTC. Hence, time-complexity of FF-4C-COMB is also $O(n \cdot \max(m, \log n) + m \cdot \log m)$.

7 Experimental setup and results

After seeing the theoretical bounds of our algorithms, we wanted to evaluate their performance and compare it with state-of-the-art. For this purpose, we looked at the following issues: (i) how well our algorithms perform compared to state-of-the-art in successfully assigning the tasks to processors, i.e., how much faster processors our algorithms need in order to assign a task set compared to state-of-the-art algorithms?, (ii) how fast our algorithms run compared to state-of-the-art algorithms? and (iii) how much pessimism is there in our theoretically derived performance bounds?

In order to answer these questions, we performed two sets of experiments. First, we compared the performance of our algorithms with two state-of-the-art algorithms (Baruah 2004b, 2004c). Both Baruah (2004b, 2004c) proposed solutions with speed competitive ratio of 2. Hence, we evaluated the performance of our algorithms with Baruah (2004b, 2004c) by setting $\alpha' = 1$ when their speed competitive ratio becomes 2 as well. We observed that, in our experiments with randomly generated task sets, our algorithms perform better in practice than state-of-the-art. We also observed that our algorithms run significantly faster compared to state-of-the-art. Then, we simulated our algorithms for different values of α'. We observed that even for this improved analysis case (where the speed competitive ratio is quantified with task set parameters as opposed to a constant number (Andersson et al. 2010)), they still perform better than indicated by the speed competitive ratio. We now discuss both the cases in detail.

7.1 Comparison with state-of-the-art

We implemented two versions of Baruah (2004c) (SKB-RTAS and SKB-RTAS-IMP) and two versions of Baruah (2004b) (SKB-ICPP and SKB-ICPP-IMP). SKB-RTAS and SKB-ICPP follow from the corresponding papers; the -IMP variants are our improved versions of the respective algorithms (see description below). We implemented all algorithms using C on Windows XP on an Intel Core2 (2.80 GHz) machine. For SKB-algorithms we also used a state-of-art LP/ILP solver, IBM ILOG CPLEX (IBM Inc. 2011).

In Baruah (2004c), a two step algorithm to assign tasks on a heterogeneous platform is proposed. The algorithm is as follows:

1. The assignment problem is formulated as ILP and then relaxed to LP. The LP formulation is solved using an LP solver. Tasks are then assigned to the processors according to the values of the respective indicator variables in the solution. Using certain tricks (Potts 1985), it is shown that there exists a solution (for example, the solution that lies on the vertex of the feasible region) to the LP formulation in which all but at most $m - 1$ tasks are integrally assigned to processors where m is the number of processors.
2. The remaining at most $m - 1$ tasks are integrally assigned on the remaining capacity of the processors using “exhaustive enumeration”.

While assigning the remaining tasks in Step 2, the author illustrates with an example that the utilization of the task under consideration is compared against the value $1 - z$ for assignment decisions *on any processor*, where z (returned by the LP solver) is the maximum utilized fraction of any processor—SKB-RTAS implements this (pessimistic) rule. Since the actual remaining capacity of each processor[6] can easily be computed from the LP solver solution, SKB-RTAS-IMP uses that, instead of $1 - z$, to test assignments, for improved average-case performance.

In Baruah (2004b), author proposes a two step algorithm, namely *taskPartition*, to assign tasks on a heterogeneous platform. The algorithm is as follows:

1. This step is similar to Step 1 of (Baruah 2004c) as described above.
2. The remaining at most $m - 1$ tasks are assigned using the bipartite matching technique such that at most one task from the $m - 1$ remaining tasks is assigned to each processor.

Let $r_1, r_2, \ldots, r_k$ denote the distinct utilization values in the given task set sorted in the increasing order, where $1 \leq k \leq m \times n$. The two step algorithm is called repeatedly by a procedure, namely *optSrch*, with different values of r_i, $1 \leq i \leq k$. When *taskPartition* is called by *optSrch* with a r_i, all the utilizations that are greater than r_i are set to infinity. The procedure *optSrch* checks for the condition $U^{r_i}_{OPT} \leq 1 - r_i$

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

in order to determine whether a feasible mapping has been obtained by *taskPartition* where $U^{r_i}_{OPT}$ denotes the value of objective function of the vertex solution returned by LP solver—SKB-ICPP implements this feasibility test. This pessimistic condition severely impacts performance. Hence, SKB-ICPP-IMP implements a better feasibility condition which checks that the sum of utilizations of all the tasks assigned to each processor does not exceed its computing capacity thereby improving its performance significantly in practice.

The necessary multiplication factor is defined as the amount of extra speed of processors the algorithm needs, for a given task set, so as to succeed, as compared to an optimal task assignment algorithm. We assess (i) the average-case performance of algorithms by creating a histogram of necessary multiplication factor and also (ii) the average run-time of each algorithm. Since all the SKB-algorithms use CPLEX, an external program, for assigning tasks to processors (for solving LP), they are penalized by the startup time and reading of the problem instance from an input file—we refer to this overhead as *CPLEX overhead*. We deal with this issue by measuring the average time for CPLEX overhead and subtract it from the measured running time of those algorithms that rely on CPLEX. In particular, SKB-ICPP and SKB-ICPP-IMP invoke CPLEX multiple times for a single task set. So, we record, for such algorithms for each task set how many times CPLEX was invoked and subtract as many times the average CPLEX overhead.

We have considered the following as CPLEX overhead: (i) starting CPLEX from our program through a system call and (ii) reading of an input file (i.e., problem instance) by CPLEX. We measured the total time that CPLEX takes to start and read the largest input file possible for our simulation (i.e., problem involving 12 tasks and 6 processors). We measured this time for 200 iterations (same as the number of task sets for which we have computed the average execution times) and took the average of these measurements. This average value was subtracted (i) once for every measurement of SKB-RTAS and SKB-RTAS-IMP and (ii) r times for every measurement of SKB-ICPP and SKB-ICPP-IMP where r is the number of different $u_{i,j}$ values the algorithm tries for each task set.

The problem instances (number of tasks, their utilizations and number of processors of each type) were generated randomly. Each problem instance had at most 12 tasks and at most 3 processors of each type. We term a task set *critically feasible* if it is feasible on a given heterogeneous multiprocessor platform but rendered infeasible if u^1_i and u^2_i of all the tasks in the system are increased by an arbitrarily small factor. To obtain critically feasible task sets from randomly generated task sets, we perform the assignment with ILP as discussed in Baruah (2004b) and obtain z—the utilization of the most utilized processor, and then multiply all the task utilizations by a factor of $\frac{1}{z}$ and repeatedly feed back to CPLEX till $0.98 < z \leq 1$.

We ran each algorithm on 15000 critically feasible task sets to obtain the necessary multiplication factor. The pseudo-code for determining the necessary multiplication factor for task sets is shown in Algorithm 7. We input a task set to algorithm A (where A can be: FF-3C, FF-4C, FF-4C-NTC, FF-4C-COMB, SKB-RTAS, SKB-RTAS-IMP, SKB-ICPP or SKB-ICPP-IMP) and if the algorithm cannot find a feasible mapping, we increment the multiplication factor by a small step, i.e., STEP = 0.01 and divide the original u^1_i and u^2_i of each task by the new multiplication factor (whose value is

Algorithm 7: Pseudo-code to determine necessary multiplication factor for an algorithm

```
STEP := 0.01
for i = 1 to 15000 do //for each critically feasible task set
    found_mult_fact := false
    mult_fact := 1
    Let curr_$\tau$ denote the critically feasible task set under consideration
    while (found_mult_factor /= true) do
        Multiply the utilizations of all the tasks in curr_$\tau$ by a factor of $\frac{1.0}{mult\_fact}$; let temp_$\tau$
        denote the resulting task set
        result := mappingAlgo(temp_$\tau$ , assign_info) // assign_info is an output
           variable which contains the task assignment information
        if (result = SUCCESS) then
            found_mult_fact := true
            print mult_fact, assign_info
        else
            mult_fact := mult_fact + STEP
        end
    end
end
```

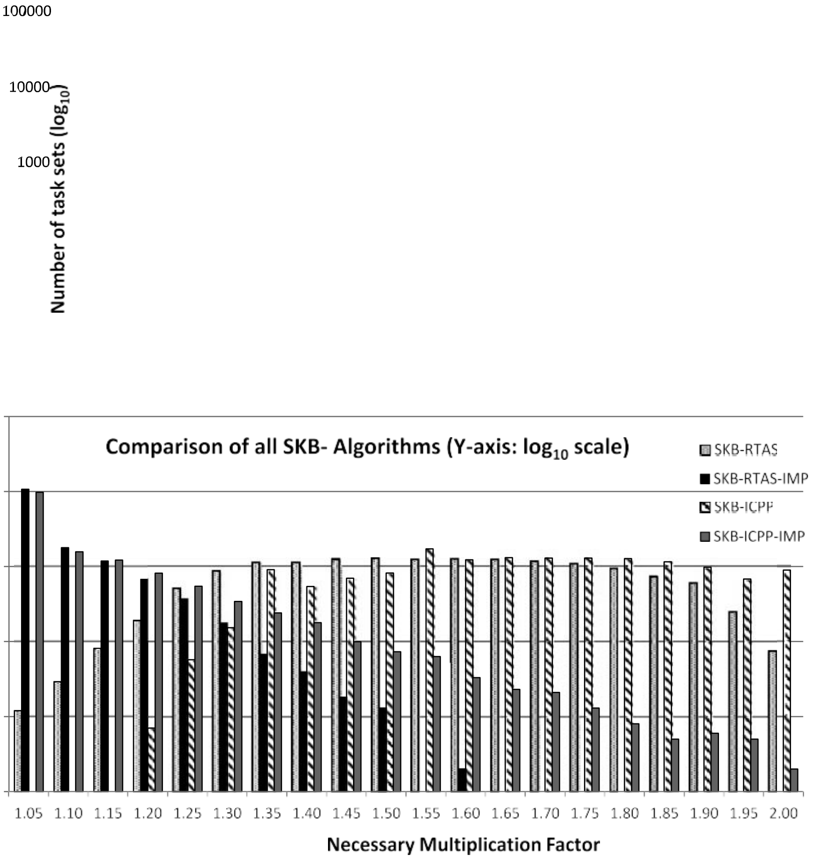


Fig. 2 Comparison of necessary multiplication factor for all the SKB-algorithms (if an algorithm has low necessary multiplication factor for many task sets then the algorithm performs well)

now 1.01) and feed this task set to algorithm A. These steps (multiplication factor adjustment and feeding back of the derived task set) are repeated till the algorithm succeeds, which gives us the necessary multiplication factor. This entire procedure is repeated for each of the 15000 task sets. With this procedure, we obtain a histogram of necessary multiplication for different algorithms.

Figure 2 shows the comparison of all the versions of SKB-algorithms. The SKB-RTAS-IMP and SKB-ICPP-IMP with their improved tests (to check the feasibility of task assignment to processors) give better average-case performance compared

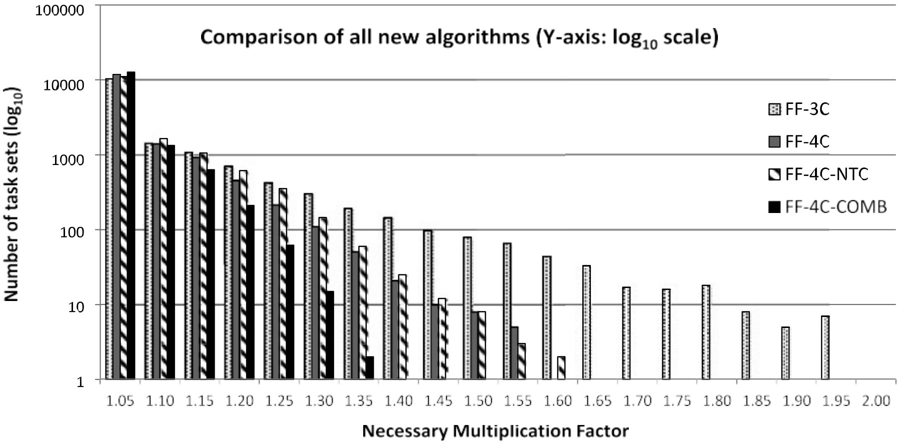


Fig. 3 Comparison of necessary multiplication factor for all of our FF-algorithms (if an algorithm has low necessary multiplication factor for many task sets then the algorithm performs well)

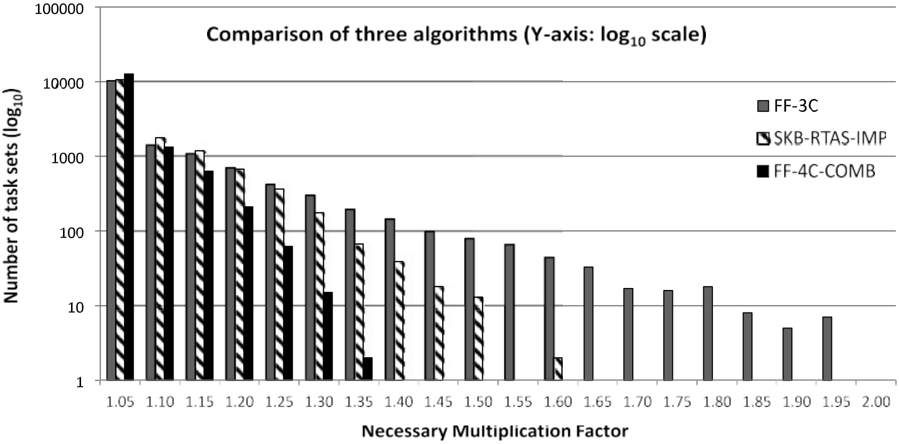


Fig. 4 Comparison of necessary multiplication factor for three algorithms (if an algorithm has low necessary multiplication factor for many task sets then the algorithm performs well)

to their counterparts. As we can see, SKB-RTAS-IMP gives the best performance among all the SKB-algorithms.

Figure 3 shows the performance of all our FF-algorithms. As we can see, FF-3C performs poorly compared to the other three, and FF-4C-COMB gives the best performance among all the FF-algorithms as it makes use of both FF-4C and FF-4C-NTC algorithms (whose performance lies between FF-3C and FF-4C-COMB).

Since SKB-RTAS-IMP offered the best necessary multiplication factor among all the SKB-algorithms and FF-4C-COMB offered the best necessary multiplication factor among all the FF-algorithms, we only depict these along with FF-3C since it is the baseline of all our algorithms in Fig. 4. As seen in our experiments, the necessary multiplication factor of FF-4C-COMB never exceeded 1.35 whereas for FF-3C

Table 4 Average execution time of our algorithms (in microseconds)

Multiplication factor	New Algorithms Measured average execution time			
	FF-3C	FF-4C	FF-4C-NTC	FF-4C-COMB
1.00	0.84	0.73	0.97	1.06
1.25	0.53	0.55	0.54	0.56
1.50	0.49	0.48	0.46	0.48
1.75	0.49	0.46	0.40	0.42
2.00	0.51	0.47	0.43	0.50

Table 5 Average execution time of SKB-algorithms (in microseconds) with the CPLEX overhead

Multiplication factor	Old Algorithms Measured avg. execution time including CPLEX overhead			
	SKB-RTAS	SKB-RTAS-IMP	SKB-ICPP	SKB-ICPP-IMP
1.00	32477.35	32562.27	394753.66	369170.79
1.25	31665.74	31525.82	393745.52	325010.43
1.50	31747.28	31740.34	381912.81	297383.55
1.75	31749.19	31598.63	337205.23	290102.20
2.00	31752.65	31781.70	291689.45	287692.93

Table 6 Average execution time of SKB-algorithms (in microseconds) after subtracting the CPLEX overhead

Multiplication factor	Old Algorithms Measured avg. execution time excluding CPLEX overhead			
	SKB-RTAS	SKB-RTAS-IMP	SKB-ICPP	SKB-ICPP-IMP
1.00	14263.68	14348.60	164551.87	161689.21
1.25	13452.07	13312.15	163565.96	149459.82
1.50	13533.61	13526.67	161373.08	140211.38
1.75	13535.52	13384.96	151003.87	137302.53
2.00	13538.98	13568.03	137989.63	136490.37

and SKB-RTAS-IMP this factor is close to 2.00 and 1.60 respectively. Therefore, FF-4C-COMB offers significantly better average-case performance compared to state-of-the-art.

We also measured the running times of each algorithm for the same task set. Table 4 shows the running time of FF-algorithms, Table 5 shows the running time of SKB-algorithms with CPLEX overhead and finally Table 6 shows the running time of SKB-algorithms after subtracting the measured CPLEX overhead (from the values shown in Table 5). We deal with the CPLEX overhead in the SKB-algorithms for fair evaluation. We can see that, in the experiments, our proposed algorithms all run in less than 1.1 μs (Table 4) but SKB-algorithms have running times in the range of

13500 to 160000 μs (Table 6). Hence all of our algorithms run at least 12000 times faster.

7.2 Evaluation of our algorithms for different values of α'

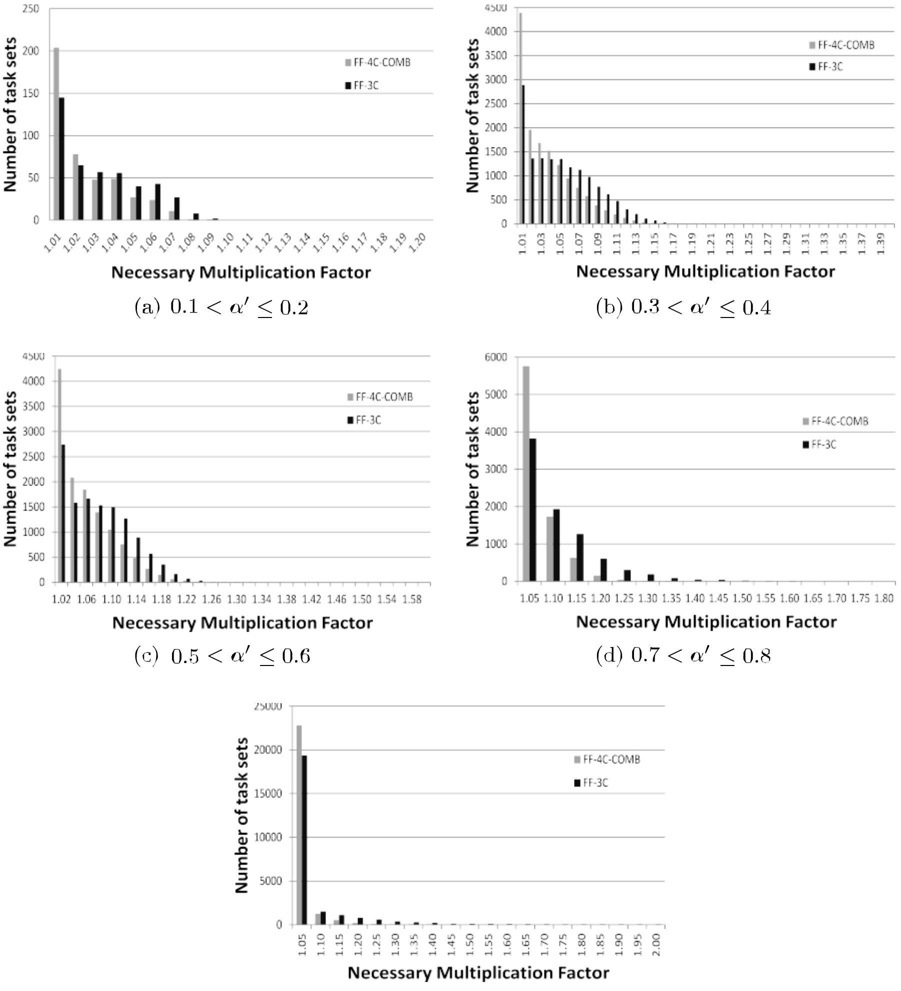
We evaluated the performance of our algorithms for different values of α'. We randomly generated 100000 critically feasible task sets. Each critically feasible task set had at most 25 tasks and at most 2 processors of each type.[7] We then classified the critically feasible task sets based on the value of α' of each task set into ten groups—for a given critically feasible task set, if $\alpha' \leq 0.1$ then the task set belongs to the first group, if $0.1 < \alpha' \leq 0.2$ then the task set belongs to the second group, … , and finally if $0.9 < \alpha' \leq 1.0$ then the task set belongs to the tenth group. Then, we ran each algorithm, i.e., FF-3C, FF-4C, FF-4C-NTC and FF-4C-COMB for the above generated critically feasible task sets and observed their necessary multiplication factors. We plotted the histogram of necessary multiplication factors for each of these algorithms for task sets in each of the groups. Since the experiments in previous subsection have confirmed that FF-4C-COMB performs better compared to all other algorithms and since FF-3C is the baseline of all our algorithms, we only depict these.

Figure 5 shows the performance of FF-3C and FF-4C-COMB algorithms. We only show the results obtained for five cases, i.e., $0.1 < \alpha' \leq 0.2$, $0.3 < \alpha' \leq 0.4$, … , $0.9 < \alpha' \leq 1.0$. The observations for other cases follow the same trend. As we can see from the graphs, for the vast majority of task sets, the algorithms perform much better than indicated by their speed competitive ratio, even when we consider the speed competitive ratio as a function of task set parameters.

8 Clustered scheduling

Having seen the excellent performance of FF-3C—both its performance bound and its performance in experimental evaluation—we now consider a heterogeneous multiprocessor platform where processors have two types (just like before) but processors of each type are organized into clusters of processors. Such a problem has been studied in the past in the context of identical multiprocessor platform (Qi et al. 2010). However, to the best of our knowledge, no such work exists for heterogeneous platforms with two types of processors. We define a cluster of processors as a subset of all processors such that (i) all processors in the cluster are of the same type, (ii) each cluster has equal number of processors and (iii) a task can migrate between processors in the same cluster. Note that a consequence of (ii) is that there is no “remaining cluster” with fewer processors. Related to (iii), the assumption that migration is allowed, we assume that an optimal migrative scheduling algorithm is used to sched-

[7] Since we only evaluate FF-algorithms in this batch of experiments and do not run SKB-algorithms which make use of linear programming solvers thereby taking much longer to output the solution, we could afford to set a higher bound on the number of tasks in each problem instance compared to previous set of experiments.



(a) $0.1 < \alpha' \leq 0.2$

(b) $0.3 < \alpha' \leq 0.4$

(c) $0.5 < \alpha' \leq 0.6$

(d) $0.7 < \alpha' \leq 0.8$

(e) $0.9 < \alpha' \leq 1.0$

Fig. 5 Performance of FF-3C and FF-4C-COMB algorithm in terms of necessary multiplication factors for different values of *α*' (if an algorithm has low necessary multiplication factor for many task sets then the algorithm performs well)

ule the tasks in each cluster; the research literature offers several such algorithms for implicit-deadline sporadic scheduling, for example, the class of Pfair scheduling algorithms (Anderson and Srinivasan 2000), sporadic EKG with timeslot being the greatest common divisor of minimum inter-arrival times (Andersson and Bletsas 2008) and the DP-WRAP framework (Levin et al. 2010).

For the purpose of discussing scheduling on such a platform, let k denote the number of processors in a cluster. Clearly, we have $\frac{|P^1|}{k}$ clusters containing type-1 processors and we have $\frac{|P^2|}{k}$ clusters containing type-2 processors. Let α denote a real number: $0 < \alpha \leq \frac{1}{2}$. Let τ denote the task set such that:

$$\forall \tau_i \in \tau : \quad \left(u_i^1 \le \alpha \right) \vee \left(1 - \alpha < u_i^1 \right) \text{ and}$$
$$\forall \tau_i \in \tau : \quad \left(u_i^2 \le \alpha \right) \vee \left(1 - \alpha < u_i^2 \right)$$

We want to schedule the task set τ on computing platform Π (with clusters as mentioned above) with migration allowed between processors in the same cluster. We formulate this problem as follows:

Partition the set of tasks into $\frac{|P^1|}{k} + \frac{|P^2|}{k}$ partitions where

- $\frac{|P^1|}{k}$ of the partitions are said to be of type-1
- $\frac{|P^2|}{k}$ of the partitions are said to be of type-2
- for each type-1 partition p, it holds that: $\sum_{\tau_j \in p} u_j^1 \le k$
- for each type-2 partition p, it holds that: $\sum_{\tau_j \in p} u_j^2 \le k$

Once partitions have been formed, it is straightforward to assign tasks to processors (all tasks in a type-1 partition are assigned to a cluster of type-1 processor; analogously for type-2). Let us consider the special case where it holds that $\tau = F1 \cup F2$. The partitioning problem described above can be rewritten as follows:

Partition the set of tasks into $\frac{|P^1|}{k} + \frac{|P^2|}{k}$ partitions where

- $\frac{|P^1|}{k}$ of the partitions are said to be of type-1
- $\frac{|P^2|}{k}$ of the partitions are said to be of type-2
- for each type-1 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^1}{k} \le 1$
- for each type-2 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^2}{k} \le 1$

Because of $\tau = F1 \cup F2$, we have that all tasks are “light”. Hence if the utilization of each task would be divided by k then all tasks would still be light. This, combined with Lemma 7, gives us that:

- **If** it is possible to partition the set of tasks into $\frac{|P^1|}{k} + \frac{|P^2|}{k}$ partitions where
  - $\frac{|P^1|}{k}$ of the partitions are said to be of type-1
  - $\frac{|P^2|}{k}$ of the partitions are said to be of type-2
  - for each type-1 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^1}{k} \le 1$
  - for each type-2 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^2}{k} \le 1$
- **then** FF-3C can be used to assign tasks to partitions (where a partition corresponds to a processor in FF-3C) and this outputs a partitioning of the set of tasks into $\frac{|P^1|}{k} + \frac{|P^2|}{k}$ partitions where
  - $\frac{|P^1|}{k}$ of the partitions are said to be of type-1
  - $\frac{|P^2|}{k}$ of the partitions are said to be of type-2
  - for each type-1 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^1}{k} \le \frac{1}{1-\frac{\alpha}{k}}$
  - for each type-2 partition p, it holds that: $\sum_{\tau_j \in p} \frac{u_j^2}{k} \le \frac{1}{1-\frac{\alpha}{k}}$

Hence, we can see that FF-3C can be used to solve the problem of scheduling implicit-deadline sporadic tasks on a heterogeneous multiprocessor with clustering. Clearly, we do not have to use FF-3C; we can use any of its (average-case performance) improved versions, i.e., FF-4C, FF-4C-NTC, FF-4C-COMB.

9 Discussion and conclusions

The heterogeneous multiprocessor computational model (i.e., unrelated parallel machines) is more general than identical or uniform multiprocessors, in terms of the systems that it can accommodate. Generally, this called for algorithms with large computational complexity, for provably good performance. We partially solve the issue via a scheduling algorithm for multiprocessors consisting of two unrelated processor types. This restricted model is of great practical interest, as it captures many current/future single-chip heterogeneous multiprocessors (AMD Inc 2010; Gschwind et al. 2006; Maeda et al. 2005; Freescale Semiconductor 2007). Our proposed algorithm, FF-3C, is low-degree polynomial in time-complexity, i.e., faster than algorithms based on ILP (or its relaxation to LP). We also proved that the speed competitive ratio of FF-3C is $\frac{1}{1-\alpha}$ where $0 < \alpha \leq \frac{1}{2}$.

Further, we designed variations of FF-3C which provide better average-case performance and have the same time-complexity and the same speed competitive ratio as that of FF-3C. We would like to mention that, in our experimental evaluations, one of our new algorithms, FF-4C-COMB, runs 12000 to 160000 times faster and has significantly smaller necessary multiplication factor than state-of-the-art algorithms (Baruah 2004b, 2004c). We also presented a version of FF-3C targeted for two-type heterogeneous multiprocessors where processors are organized into clusters and where migration of tasks is allowed between processors of the same cluster.

Acknowledgements This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project Ref. FCOMP-01-0124-FEDER-022701; by FCT and COMPETE (ERDF), within REHEAT project, ref. FCOMP-01-0124-FEDER-010045; by FCT and ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/66771/2009.

References

AMD Inc (2010) AMD fusion family of APUs: Enabling a superior, immersive PC experience. http://www.amd.com/us/Documents/48423_fusion_whitepaper_WEB.pdf

AMD Inc (2011a) AMD embedded G-series platform. http://www.amd.com/us/products/embedded/processors/Pages/g-series.aspx

AMD Inc (2011b) The AMD fusion family of APUs. http://sites.amd.com/us/fusion/apu/Pages/fusion.aspx

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

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

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

Andersson B, Tovar E (2007b) Competitive analysis of static-priority of partitioned scheduling on uniform multiprocessors. In: Proceedings of the 13th IEEE international conference on embedded and real-time computing systems and applications, pp 111–119

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

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

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

Baruah S (2004b) Partitioning real-time tasks among heterogeneous multiprocessors. In: Proc of the 33rd international conference on parallel processing, pp 467–474

Baruah S (2004c) Task partitioning upon heterogeneous multiprocessor platforms. In: Proceedings of the 10th IEEE international real-time and embedded technology and applications symposium, pp 536–543

Coffman EG, Garey MR, Johnson DS (1997) Approximation algorithms for bin packing: a survey. In: Approximation algorithms for NP-hard problems. PWS, Boston, pp 46–93

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms, 2nd edn. McGraw-Hill, New York

Dertouzos M (1974) Control robotics: The procedural control of physical processes. In: Proceedings of IFIP congress (IFIP’74), pp 807–813

Freescale Semiconductor (2007) i.MX applications processors. http://www.freescale.com/webapp/sps/site/homepage.jsp?code=IMX_HOME

Garey MR, Johnson DS (1979) Computers and intractability: A guide to the theory of NP-completeness. Freeman, New York

Geer D (2005) Taking the graphics processor beyond graphics. IEEE Comput 38(9):14–16

Grandpierre T, Lavarenne C, Sorel Y (1999) Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In: Proceedings of the 7th international workshop on hardware/software codesign, pp 74–78

Gschwind M, Hofstee HP, Flachs B, Hopkins M, Watanabe Y, Yamazaki T (2006) Synergistic processing in cell’s multicore architecture. IEEE MICRO 26(2):10–24

Hochbaum D, Shmoys D (1986) A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. In: Proc of the sixth conference on foundations of software technology and theoretical computer science, pp 382–393

Horowitz E, Sahni S (1976) Exact and approximate algorithms for scheduling nonidentical processors. J ACM 23:317–327

IBM Inc (2005) The cell project at IBM research. http://www.research.ibm.com/cell/

IBM Inc (2011) IBM ILOG CPLEX optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/

IEEE Spectrum (2011) With Denver project NVIDIA and ARM join CPU-GPU integration race. http://spectrum.ieee.org/tech-talk/semiconductors/processors/with-denver-project-nvidia-and-arm-join-cpugpu-integration-race

Intel Corporation (2011) The 2nd generation Intel Core processor family. http://www.intel.com/en_IN/consumer/products/processors/core-family.htm

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

Levin G, Funk S, Sadowskin C, Pye I, Brandt S (2010) DP-FAIR: A simple model for understanding optimal multiprocessor scheduling. In: Proceedings of the 22nd Euromicro conference on real-time systems, pp 3–13

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

López M, Díaz J, García D (2004) Utilization bounds for EDF scheduling on real-time multiprocessor systems. Real-Time Syst 28:39–68

Maeda S, Asano S, Shimada T, Awazu K, Tago H (2005) A real-time software platform for the Cell processor. IEEE MICRO 25(5):20–29

NVIDIA (2011) Tegra 2 and Tegra 3 super processors. http://www.nvidia.com/object/tegra-superchip.html

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

Qi X, Zhu D, Aydin H (2010) A study of utilization bound and run-time overhead for cluster scheduling in multiprocessor real-time systems. In: Proceedings of the 16th IEEE international conference on embedded and real-time computing systems and applications, pp 3–12

Texas Instruments (2011) OMAP application processors: OMAP 5 platform. http://www.ti.com/ww/en/omap/omap5/omap5-platform.html

Wiese A, Bonifaci V, Baruah S (2012) Partitioned EDF scheduling on a few types of unrelated multiprocessors. Tech rep. Available at http://www.cs.unc.edu/~baruah/Submitted/2012-k-unrelated.pdf