Assigning Real-Time Tasks on Heterogeneous Multiprocessors with Two Unrelated Types of Processors

Björn Andersson, Gurulingesh Raravi and Konstantinos Bletsas
CISTER-ISEP Research Centre, Polytechnic Institute of Porto
Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto, Portugal
bandersson@dei.isep.ipp.pt, ghri@isep.ipp.pt, ksbs@isep.ipp.pt

Abstract

Consider the problem of scheduling a set of implicit-deadline sporadic tasks on a heterogeneous multiprocessor platform to meet all deadlines. Tasks cannot migrate and each processor is either of type-1 or type-2 (with each task having different execution speed on each processor type).

We present a new algorithm, FF-3C, for this problem. FF-3C offers low time-complexity and provably good performance. Specifically, (i) its time-complexity is $O(n \cdot \max(m, \log n))$, where n is the number of tasks and m is the number of processors and (ii) it offers the guarantee that if a task set can be scheduled by an optimal task assignment scheme to meet deadlines then FF-3C meets deadlines as well if given processors twice as fast. We also present several extensions to FF-3C; these offer the same time-complexity and performance guarantee as that of FF-3C but in addition, they offer improved average-case performance. Via experiments with randomly generated task sets, we compare the performance of our new algorithms and two established state-of-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 the algorithm needs, for a given task set, so as to succeed, compared to an optimal task assignment scheme. Overall our new algorithms compare favorably to the state-of-art. One in particular (FF-4C-COMB), in our experimental evaluations, runs 12000 to 160000 times faster and has significantly smaller necessary multiplication factor than state-of-art algorithms.

1. Introduction

Designers have been achieving significant speedup of particular tasks by using specialized processing units (e.g., Graphics Processors for computer graphics or Digital Signal Processors 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 [10, 13, 17]. 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 in real-time embedded systems. 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 [6] but it assumes that tasks can migrate. Also, this assumption is often unrealistic in practice, since processors with different functionalities typically have different instruction sets. 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 Earliest Deadline First scheduling [15]) — the difficult part is the task assignment.

Among known task assignment schemes for multiprocessors in general (i.e., not necessarily heterogeneous), only (i) bin-packing heuristics (e.g., first-fit), (ii) Integer-Linear-Programming (ILP) modeling and (iii) Linear Programming (LP) relaxation approaches for ILP perform provably well. Bin-packing heuristics are popular for task assignment but unfortunately, the proof techniques used on identical multiprocessors (i.e., a task has same execution speed on all processors) do not easily translate to heterogeneous multiprocessors. Consequently, the current literature offers no bin-packing heuristic for assigning real-time tasks on heterogeneous multiprocessors. Instead, task assignment is modeled [7][8] 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 ([11], p. 245). Via relaxation of ILP formulation to LP and certain tricks [18], better time-complexity can be attained [7][8] (polynomial time-complexity for the algorithm in [7] and for the special case of fixed number of processors, the algorithm in [7] has polynomial time-complexity as well). Neither of these algorithms, however, attains low-degree (linear or quadratic) polynomial time-complexity.

In practice, many heterogeneous multiprocessors only use two types of processors (e.g., one type for graphics and one general-purpose). AMD [1] plans to ship such chips; FreeScale already does [10]. Traditionally, graphics processors were meant just for graphics tasks, hence task assignment was trivial. Today though, designers [12] use them for a wide range of calculations and this makes task assignment non-trivial. Notably, the Cell processor [13][17] has Turing-complete "graphics processors" (called synergistic processors), able to compute anything that the main processor can. Unfortunately, the research literature provides no scheduling algorithm that takes advantage of this special structure.

Therefore, in this paper we consider the problem of scheduling without migration, to meet all deadlines, a set of implicit-deadline independent sporadic tasks on a heterogeneous multiprocessor where each processor is either of type-1 or type-2 (with each task having different execution speed on each processor type). We present a new algorithm, FF-3C, for this problem. FF-3C offers low time-complexity and provably good performance. Specifically, (i) its time-complexity is $O(n \cdot \max(m, \log n))$, where n is the number of tasks and m is the number of processors and (ii) it offers the guarantee that if a task set can be scheduled by an optimal task assignment scheme to meet deadlines then FF-3C meets deadlines as well if given processors twice as fast. We also present several extensions to FF-3C; these offer the same time-complexity and performance guarantee as that of FF-3C but in addition, they offer improved average-case performance. Via experiments with randomly generated task sets, we compare the performance of our new algorithms and two established state-of-art algorithms (and variations of the latter). We evaluate 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 particular task set, in order to succeed as compared to an optimal task assignment scheme. Overall our new algorithms compare favorably to the state-of-art. In particular, 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-art algorithms [8, 7].

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 (all task deadlines are equal to their minimum inter-arrival times) 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 utilisation of task τ_i depends on the type of processor to which it is assigned. The utilisation of task τ_i is U_i^1 if τ_i is assigned to a type-1 processor. Analogously, the utilisation of task τ_i is U_i^2 if τ_i is assigned to a type-2 processor. Note that we allow $U_i^1 = \infty$ (or $U_i^2 = \infty$) if task τ_i cannot be assigned at all to a type-1 (or type-2) processor.

Let $\tau[p]$ denote the set of tasks assigned to processor p. Earliest-Deadline-First (EDF) is a very popular algorithm in uniprocessor scheduling [15]. A slight adaptation of a previously known result [15] gives us:

Lemma 1. *If all tasks in $\tau[p]$ are scheduled under EDF on processor p (which is of type-z, where z stands for 1 or 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:

$$\sum_{\tau_i \in \tau[p]} U_i^1 \leq 1 \; \forall p \in P^1 \tag{1}$$

$$\sum_{\tau_i \in \tau[p]} U_i^2 \leq 1 \; \forall p \in P^2 \tag{2}$$

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 [7]. We therefore aim for a non-optimal algorithm of polynomial time-complexity which would still offer good performance.

Commonly, the performance of an algorithm is characterized using the notion of the *utilisation bound* [15]: an algorithm with a utilisation bound of UB is always capable of scheduling any task set with a utilisation up to UB so as to meet deadlines. This definition has been used in uniprocessor scheduling [15] and multiprocessors with identical processors [2]. 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 an algorithm A is defined as the lowest number such that for every task set τ and computing platform $\Pi^{'}$ it holds that if it is possible for a non-migrative algorithm to meet all deadlines of τ on $\Pi^{'}$ then algorithm A meets all deadlines of τ on a computing platform Π whose every processor is CPT_A times faster than the corresponding processor in $\Pi^{'}$.[1]

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.

3. Useful results

Bin-packing heuristics are popular for assigning tasks on identical [14] or uniform [4] 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 – see Appendix in [3]. It can be seen (in Appendix in [3]) 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:

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:

$$\tau = \tau^1 \cup \tau^2 \tag{3}$$

$$\forall \tau_i \in \tau^1 : U_i^1 \leq U_i^2 \tag{4}$$

$$\forall \tau_i \in \tau^2 : U_i^1 > U_i^2 \tag{5}$$

We proceed with two useful observations (their correctness is evident; for formal proofs, see the Appendix in [3]).

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

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

$$\sum_{i \in \tau^1} U_i^1 + \sum_{i \in \tau^2} U_i^2 > |P^1| + |P^2| \tag{6}$$

[1]Our notion of speed competitive ratio in this paper is equivalent to that in previous work by Baruah [6]. It differs from the one used in [5].

We next highlight how the problem in consideration is related to other known computational problems, to help with proofs later. If you read this paper for the first time, you may want to skip this section now and revisit 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 item, indexed i, while x_i as the fraction of it that is employed). 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 $\sum_{i=1}^{n} x_i \cdot w_i \leq \mathrm{CAP}$
and x_i is a real number and $0 \leq x_i \leq 1$
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. *The following algorithm optimally solves the Fractional knapsack Problem:*

```
1. reindex tuples $\{v_i, w_i\}$ by order of descending $v_i/w_i$
2. for i:= 1 to n do $x_i$:=0; end for
3. i:=1; SUMWEIGHT:=0; SUMVALUE:=0;
4. while ((SUMWEIGHT+$w_i \leq CAP$) and ($i \leq n$)) do
5.     $x_i$:=1;
6.     SUMWEIGHT:=SUMWEIGHT+$w_i$;
7.     SUMVALUE:=SUMVALUE+$v_i$;
8.     i:=i+1;
9. end while
10. if $i \leq n$ then
11.     $x_i$:=(CAP-SUMWEIGHT)/$w_i$;
12.     SUMWEIGHT:=SUMWEIGHT+$w_i \cdot x_i$;
13.     SUMVALUE:=SUMVALUE+$v_i \cdot x_i$;
14. end if
```

This is found in undergraduate textbooks (e.g., Chapter 16.2 in [9]). Now consider a scheduling problem:

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

$$\sum_{i \in A1} U_i^1 > \frac{|P^1|}{2} - x \tag{7}$$

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

$$\sum_{i \in B1} U_i^1 \leq \frac{|P^1|}{2} - x \tag{8}$$

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

$$\sum_{i \in A1} U_i^1 + \sum_{i \in A2} U_i^2 + \sum_{i \in \tau^2} U_i^2 \leq \sum_{i \in B1} U_i^1 + \sum_{i \in B2} U_i^2 \quad (9)$$

Proof. Let us arbitrarily choose $A1$, $B1$ as defined. We will prove that this implies Inequality 9. Using Inequalities 7 and 8 we clearly get:

$$\sum_{i \in A1} U_i^1 > \sum_{i \in B1} U_i^1 \quad (10)$$

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

Instance1:
CAP = left-hand side of Inequality 10.
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 the algorithm in Lemma 4 terminates with Instance1 as input.

Instance2:
CAP = left-hand side of Inequality 10.
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 the algorithm in Lemma 4 terminates with Instance2 as input.

Instance3:
CAP = right-hand side of Inequality 10.
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 the algorithm in Lemma 4 terminates with Instance3 as input.

Instance4:
CAP = right-hand side of Inequality 10.
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 the algorithm in Lemma 4 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 the 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$_1$.
O13: From O12 and O5: SUMVALUE$_3$≤SUMVALUE$_2$.

Using O13 and the definitions of the instances and 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:

$$\sum_{i \in B1} (U_i^2 - U_i^1) \leq \sum_{i \in A1} (U_i^2 - U_i^1) \quad (11)$$

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

$$\sum_{i \in \tau^1} U_i^2 + \sum_{i \in \tau^2} U_i^2 = \sum_{i \in B1} U_i^2 + \sum_{i \in B2} U_i^2 \quad (12)$$

Combining Inequality 11 and Equation 12 gives us:

$$\sum_{i \in \tau^1} U_i^2 + \sum_{i \in \tau^2} U_i^2 - \Big(\sum_{i \in A1} U_i^2 - \sum_{i \in A1} U_i^1 \Big)$$
$$\leq \sum_{i \in B1} U_i^2 + \sum_{i \in B2} U_i^2 - \Big(\sum_{i \in B1} U_i^2 - \sum_{i \in B1} U_i^1 \Big) \quad (13)$$

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

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

This is the statement of the lemma. □

Lemma 5 considers the task set 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. We apply Lemma 5 on $\tau \setminus (H1 \cup H2)$ and then add the same sum to both sides of Inequality 9. This gives us:

Lemma 6. *Consider n tasks and a heterogeneous multiprocessor conforming to the system model (and notation) of Section 2. Let x denote a number such that $0 \leq x \leq \frac{|P^1|}{2}$. Let $A1$ denote a subset of $(\tau^1 \backslash (H1 \cup H2))$ such that*

$$\sum_{i \in A1} U_i^1 > \frac{|P^1|}{2} - x \quad (14)$$

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

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

$$\sum_{i \in B1} U_i^1 \leq \frac{|P^1|}{2} - x \quad (15)$$

Let B2 denote $(\tau \setminus (H1 \cup H2)) \backslash B1$. It then holds that:

$$\sum_{i \in H1} U_i^1 + \sum_{i \in H2} U_i^2 + \sum_{i \in A1} U_i^1 + \sum_{i \in A2} U_i^2 + \sum_{i \in \tau^2 \setminus (H1 \cup H2)} U_i^2$$
$$\leq \sum_{i \in H1} U_i^1 + \sum_{i \in H2} U_i^2 + \sum_{i \in B1} U_i^1 + \sum_{i \in B2} U_i^2$$

Lemma 6 will be useful for proving the performance of our new algorithm, formulated in Section 4.

4. The new algorithm

Our goal is to design an algorithm with a speed competitive ratio 2. The new algorithm is based on two ideas.

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

Idea2: A task with utilisation above 50% on its non-favourite type of processor should be assigned to its favourite type of processor. This special case of Idea1 is stated separately because this facilitates creating an algorithm with the desired speed competitive ratio: Since we will compare the performance of our new algorithm versus every other algorithm that uses processors of at most half 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 Section 2), we also define the disjoint sets:

$$H1 = \{\tau_i \in \tau^1 : U_i^2 > 1/2\} \quad (16)$$
$$H2 = \{\tau_i \in \tau^2 : U_i^1 > 1/2\} \quad (17)$$
$$F1 = \tau^1 \setminus H1 \quad (18)$$
$$F2 = \tau^2 \setminus H2 \quad (19)$$

A task is termed to be *heavy on type-1 processors* (or, respectively, type-2 processors) if its utilisation 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 favourite which are heavy if they are assigned to their non-favourite processor type (type-2)". Analogous for $H2$. (Obviously, a task in $H1$ or $H2$ might also be heavy on its favourite 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 favourite and which are not heavy on either processor type". Analogous for $F2$. From the definitions of $H1$, $H2$, $F1$, and $F2$ (and Inequalities 4 and 5), we have:

$$\tau_i \in H1 \Rightarrow \quad U_i^2 > \frac{1}{2} \quad (20)$$
$$\tau_i \in H2 \Rightarrow \quad U_i^1 > \frac{1}{2} \quad (21)$$
$$\tau_i \in F1 \Rightarrow \quad U_i^1 \leq \frac{1}{2} \text{ and } U_i^2 \leq \frac{1}{2} \quad (22)$$
$$\tau_i \in F2 \Rightarrow \quad U_i^1 \leq \frac{1}{2} \text{ and } U_i^2 < \frac{1}{2} \quad (23)$$

```
Output: $\tau$[p] specifies the tasks assigned to processor $p$.
1.  Form sets $H1$, $H2$, $F1$, $F2$ as defined by Eq. 16-19
2.  $\forall p$: U[p] := 0
3.  $\forall p$: $\tau$[p] := $\emptyset$
4.  if first-fit( $H1$, $P^1$) $\neq H1$ then declare FAILURE
5.  if first-fit( $H2$, $P^2$) $\neq H2$ then declare FAILURE
6.  $F11$ := first-fit( $F1$, $P^1$)
7.  $F22$ := first-fit( $F2$, $P^2$)
8.  if $(F11 = F1) \wedge (F22 = F2)$ then declare SUCCESS
9.  if $(F11 \neq F1) \wedge (F22 \neq F2)$ then declare FAILURE
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
```

Figure 1. The new algorithm, FF-3C

Figure 1 shows the new algorithm, FF-3C. The intuition behind the design of FF-3C is that first we assign tasks to their favourite processors which would be heavy on other processor type (lines 4-5). Then we assign the non-heavy tasks to their favourite processors (lines 6-7). Then, if there are remaining non-heavy tasks, these have to be assigned to processors that are not their favourite (lines 12 and 20).

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 favourite and (iii) to its non-favourite processor type.

As already mentioned, the algorithm FF-3C performs several passes with first-fit bin-packing. It uses a subroutine `first-fit` (see Figure 2 for pseudocode) 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.

We next establish the competitive ratio of FF-3C.

```
1.  function first-fit( ts : set of tasks; ps : set of processors)
        return set of tasks
2.   assigned_tasks := $\emptyset$
3.   If ps consists of type-1 (type-2) processors, then order
     ts by decreasing $U_i^2/U_i^1$ (resp., increasing $U_i^2/U_i^1$).
     Use any order for processors ps, but maintain it
     during the execution of the function first-fit.
4.   $\tau_i$ := first task in ts
5.   $p$ := first processor in ps
6.   Let $k$ denote the type of processor $p$ (either 1 or 2)
7.   if U[p]+$U_i^k \leq 1$ then
8.     U[p] := U[p]+$U_i^k$
9.     $\tau$[p] := $\tau$[p] $\cup \{\tau_i\}$
10.    assigned_tasks := assigned_tasks $\cup \{\tau_i\}$
11.    if remaining tasks exist in ts then
12.      $\tau_i$ := next task in ts
13.      go to line 5.
14.    else
15.      return assigned_tasks
16.    end if
17.  else
18.    if remaining processors exist in ps then
19.      $p$ := next processor in ps
20.      go to line 6.
21.    else
22.      return assigned_tasks
23.    end if
24.  end if
```

Figure 2. First-fit bin-packing

Theorem 1. *The speed competitive ratio of FF-3C is at most 2.*

Proof. An equivalent claim is that any task set τ which is not schedulable under FF-3C over a computing platform Π would likewise be unschedulable, using any algorithm, over computing platform Π' each of whose processors has at most half the speed of the corresponding processor in Π. This, we will prove (by contradiction). From the definition of Π':

$$\forall i : \frac{U_i^1}{U_i^{1'}} = \frac{U_i^2}{U_i^{2'}} = \frac{1}{2} \tag{24}$$

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 follows that FF-3C declared FAILURE. We explore all possibilities for this to occur:

Failure on line 4 in FF-3C.
If $\sum_{i\in H1} U_i^1 \leq \frac{|P^1|}{2}$ then (from a well-known result [16]), first-fit succeeds. Therefore, we know that

$$\sum_{i\in H1} U_i^1 > \frac{|P^1|}{2} \overset{(24)}{\Rightarrow} \sum_{i\in H1} U_i^{1'} > |P^1|$$

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 Inequality 20 and Equation 24 we get $U_i^{2'} > 1$, hence 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. Consequently, we obtain:

$$\forall p \in P^1 : U[p] + U_{failed1}^1 > 1 \tag{25}$$

$$\text{and } \forall p \in P^2 : U[p] + U_{failed2}^2 > 1 \tag{26}$$

Since $\tau_{failed1} \in F1$, Inequality 22 gives us $U_{failed1}^1 \leq \frac{1}{2}$. Analogously, $U_{failed2}^2 \leq \frac{1}{2}$. Using these on Inequalities 25 and 26 gives:

$$\forall p \in P^1 : U[p] > 1/2 \tag{27}$$

$$\text{and } \forall p \in P^2 : U[p] > 1/2 \tag{28}$$

Observing that tasks assigned on processors in P^1 are a subset of τ^1 and using Inequality 27 gives us:

$$\sum_{\tau_i \in \tau^1} U_i^1 > \frac{|P^1|}{2} \tag{29}$$

With analogous reasoning, we obtain:

$$\sum_{\tau_i \in \tau^2} U_i^2 > \frac{|P^2|}{2} \tag{30}$$

Observing these two inequalities and Equation 24 and Lemma 3 gives us that OPT fails to assign tasks 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 in FF-3C there was a task $\tau_{failed} \in (F1 \setminus F11)$ which was attempted to each of the processors in P^2 but all of them failed. Hence, we have:

$$\forall p \in P^2 : U[p] + U_{failed}^2 > 1 \tag{31}$$

We can add these inequalities together and get:

$$\sum_{p \in P^2} U[p] > |P^2| \cdot (1 - U_{failed}^2) \tag{32}$$

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 in FF-3C. We also know that $\tau^{F12assigned} \subset F12$. Hence:

$$\sum_{i \in (H2 \cup F22 \cup F12)} U_i^2 > |P^2| \cdot (1 - U_{failed}^2)$$

Since $\tau_{failed} \in F1 \setminus F11 \subseteq F1$, using $\tau_{failed} \in F1$ and Inequality 22 on the above inequality yields:

$$\sum_{i \in (H2 \cup F22 \cup F12)} U_i^2 > \frac{|P^2|}{2} \tag{33}$$

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} mentioned above or it may be different. Because it was not possible to assign $\tau_{failed1}$ on any of the processors in P^1, we have:

$$\forall p \in P^1 : U[p] + U^1_{failed1} > 1 \tag{34}$$

Adding these inequalities together gives us:

$$\sum_{p \in P^1} U[p] > |P^1| \cdot (1 - U^1_{failed1}) \tag{35}$$

We know that the tasks assigned to processors in P^1 just after executing line 6 in FF-3C are $H1 \cup F11$. Therefore, we have:

$$\sum_{i \in (H1 \cup F11)} U^1_i > |P^1| \cdot (1 - U^1_{failed1}) \tag{36}$$

Since $\tau_{failed} \in (F1 \setminus F11) \subseteq F1$, using $\tau_{failed} \in F1$ and Inequality 22 on Inequality 36 yields:

$$\sum_{i \in (H1 \cup F11)} U^1_i > \frac{|P^1|}{2} \tag{37}$$

Let us now discuss OPT, the algorithm which succeeds in assigning the task set τ on the computer platform $\Pi^{'}$. Let us discuss tasks in $H1$. From the definition, we know that:

$$\forall \tau_i \in H1 : U^2_i > 1/2 \tag{38}$$

Using Equation 24 gives us:

$$\forall \tau_i \in H1 : U^{2'}_i > 1 \tag{39}$$

If $\exists \tau_i \in H1 \ : U^1_i > \frac{1}{2}$, then $\exists \tau_i \in H1 \ : U^{1'}_i > 1$ and using $\tau_i \in H1$ and Inequality 4 gives $\exists \tau_i \in H1 \subseteq \tau^1 : \ U^{2'}_i > 1$. Hence such a task cannot be assigned by OPT on any processor of $\Pi^{'}$ (of any type) and this is a contradiction. Hence we can assume that $\forall \tau_i \in H1 : \ U^1_i \leq \frac{1}{2}$, from which we get:

$$\forall \tau_i \in H1 : \ U^{1'}_i \leq 1 \tag{40}$$

Using Inequalities 39 and 40 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:

$$\sum_{\tau_i \in (H1 \cup \tau^{OPT1})} U^{1'}_i \leq |P^1| \tag{41}$$

$$\text{and} \quad \sum_{\tau_i \in (H2 \cup \tau^{OPT2})} U^{2'}_i \leq |P^2| \tag{42}$$

Using Equation 24 gives us:

$$\sum_{\tau_i \in (H1 \cup \tau^{OPT1})} U^1_i \leq \frac{|P^1|}{2} \tag{43}$$

$$\text{and} \quad \sum_{\tau_i \in (H2 \cup \tau^{OPT2})} U^2_i \leq \frac{|P^2|}{2} \tag{44}$$

We can now reason about the inequalities we obtained about the assignments of FF-3C and OPT. Rewriting Inequalities 37 and 43 respectively yields:

$$\sum_{i \in F11} U^1_i > \frac{|P^1|}{2} - \sum_{i \in H1} U^1_i \tag{45}$$

$$\sum_{\tau_i \in \tau^{OPT1}} U^1_i \leq \frac{|P^1|}{2} - \sum_{\tau_i \in H1} U^1_i \tag{46}$$

We can see that Inequalities 45 and 46 with $x = \sum_{i \in H1} U^1_i$ ensure that the assumptions of Lemma 6 are true, given the ordering of $F1$ during assignment over P^1 (line 3 in Fig 2), which ensures that $\forall \tau_i \in F11, \forall \tau_j \in F12 : \ \frac{U^2_i}{U^1_i} \geq \frac{U^2_j}{U^1_j}$. Using Lemma 6 gives us:

$$\sum_{i \in H1} U^1_i + \sum_{i \in H2} U^2_i + \sum_{i \in F11} U^1_i + \sum_{i \in F12} U^2_i + \sum_{i \in F22} U^2_i$$
$$\leq \sum_{i \in H1} U^1_i + \sum_{i \in H2} U^2_i + \sum_{i \in \tau^{OPT1}} U^1_i + \sum_{i \in \tau^{OPT2}} U^2_i$$

Applying Inequalities 43 and 44 to the inequality above gives us:

$$\sum_{i \in H1} U^1_i + \sum_{i \in H2} U^2_i + \sum_{i \in F11} U^1_i + \sum_{i \in F12} U^2_i + \sum_{i \in F22} U^2_i \leq \frac{|P^1|}{2} + \frac{|P^2|}{2} \tag{47}$$

Applying Inequalities 33 and 37 to Inequality 47 gives us:

$$\frac{|P^1|}{2} + \frac{|P^2|}{2} < \frac{|P^1|}{2} + \frac{|P^2|}{2} \tag{48}$$

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 theorem holds. □

5. Time complexity of the new algorithm

We will show that the time-complexity of FF-3C is a polynomial function of the number of tasks (n) and processors (m). By inspection of the pseudocode for FF-3C (Figure 1), 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 $O(n \log n)$ e.g., using Heapsort.

- Sorting is performed over either P^1 or P^2 (i.e., at most m processors). Since the order does not matter – only that an order exists – the complexity is $O(m)$.

- First-fit bin-packing is performed ($O(n \cdot m)$).

Thus the time-complexity of the algorithm is at most

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

6. Extensions to FF-3C

In this section, we will discuss how to enhance FF-3C to attain better average-case performance.

FF-4C: 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 favourite processor type then FF-3C declares failure even without trying to assign them on their non-favourite processor type (line 4 and 5 in Figure 1). In an extreme case, FF-3C would fail with a system comprised of (i) a heavy task of type H1 (or H2) which can fit on processor of type P^2 (or P^1) and (ii) zero processors of type P^1 (or P^2) and infinite number of processors of type P^2 (or P^1). FF-4C, an enhanced version of FF-3C, overcomes this drawback and gives better average-case performance than FF-3C. FF-4C, upon failing to assign tasks in H1 (or H2) on processors of type P^1 (or P^2), tries to assign those unassigned tasks onto their non-favourite processors of type P^2 (or P^1).

FF-4C-NTC: One observation about the classification of tasks into H1, F1 and H2, F2 by FF-3C (and also FF-4C) is that it can misguide the algorithm to assign a task in a way which causes a failure later on. For example, consider the following system with two tasks τ_1 with U_1^1=0.99, U_1^2=1.0 and τ_2 with U_2^1=0.495, U_2^2=2.0 and processors P_1 of type-1 and P_2 of type-2. FF-3C (also FF-4C) classifies τ_1 as H1 and assigns it to P_1 and τ_2 as F1 and fails to assign it to either P_1 or P_2. The intuitive way would be to just classify tasks as τ^1 or τ^2 (as defined by Inequalities 4 and 5) and for each class, assign 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. Hence, this version of FF-4C does not classify τ^1 into H1 and F1 nor τ^2 into H2 and F2: It only considers favourite/non-favourite processor types and disregards whether a task is heavy or not. The algorithm first tries to assign (using first-fit as shown in Figure 2) tasks from τ^1 on their favourite processors of type P^1 and if any of these tasks could not be assigned then it tries to assign them on their non-favourite processor type P^2 – and analogously for τ^2. For the above example, FF-4C-NTC would assign τ_1 to P_2 and τ_2 to P_1.

FF-4C-COMB: For some task sets FF-4C succeeds and FF-4C-NTC fails but for others the inverse occurs. FF-4C-COMB exploits this by first attempting task assignment with FF-4C and, upon failing, also trying FF-4C-NTC.

The speed competitive ratio and time-complexity of these algorithms are same as that of FF-3C. The proofs (and their pseudocode) can be found in Appendix in [3].

7. Experimental setup and results

We experimentally compare the performance of our algorithms and two prior state-of-art algorithms. We implemented two versions of [8] (SKB-RTAS and SKB-RTAS-IMP) and two versions of [7] (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 in C on Windows XP on an Intel Core2 (2.80 GHz). For SKB- algorithms we also used a state-of-art LP/ILP solver, IBM ILOG CPLEX.

In [8], it is stated that LPRelax-Feas(Γ, Π) assigns at least ($n-m+1$) tasks to processors (n is number of tasks, m is number of processors); any remaining tasks (at most $m-1$) are assigned by exhaustive enumeration. While assigning these tasks, the author illustrates with an example that their utilisation can be compared against ($1-z$) for assignment decisions *on any processor*, where z (returned by the LP solver) is the maximum utilised fraction of any processor – SKB-RTAS implements this (pessimistic) rule. Since the actual remaining capacity of each processor[2] can be easily computed from the LP solver solution, SKB-RTAS-IMP uses that, instead of ($1-z$), to test assignments, for improved average-case performance.

In [7], the author compares whether $U_{OPT}^{r_i} \leq (1 - r_i)$ (in procedure *optSrch*) to identify whether a feasible mapping has been obtained by the algorithm (namely, $taskPartition$) where $U_{OPT}^{r_i}$ is the vertex solution's objective function value returned by LP solver by setting all the utilisations of those tasks which are greater than r_i to ∞ – SKB-ICPP implements this feasibility test. This pessimistic test severely impacts performance. Hence, SKB-ICPP-IMP implements a better feasibility condition which checks that the sum of utilisations of all the tasks assigned to each processor does not exceed its computing capacity thereby improving its performance significantly in practice.

We assess (i) the average-case performance of algorithms by creating a histogram of necessary multiplication

[2]The actual remaining capacity on processor p is $1 - \sum_{i:x_{i,p}=1} u_{i,p}$ where $x_{i,p}$ is the indicator variable used in [8].

factor (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 scheme) 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 file (referred to 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. Hence, we record, for such algorithms for each task set how many times CPLEX was invoked and subtract as many times the average CPLEX overhead.

The problem instances (number of tasks, their utilisations 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 all U_i^1 and U_i^2 are increased by an arbitrarily small factor. To obtain critically feasible task sets from the randomly generated task sets, we perform assignment with the ILP approach as in [7] and obtain z, the utilisation of the most utilised processor, and then multiply all task utilisations by $1.0/z$ and repeatedly feed back to CPLEX till $0.98 < z \leq 1$.

We ran each algorithm on 15000 generated critically feasible task sets to obtain the necessary multiplication factor. 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_i^1 and U_i^2 of each task by the new multiplication factor (whose value is 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 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. (Detailed results are provided in Appendix in [3].) Among the previously known algorithms, we found that SKB-ICPP-IMP offered the best necessary multiplication factor. Among the new algorithms, we found that FF-4C-COMB performed best. Therefore, we only depict these (and FF-3C since it is a baseline of all our algorithms) in Figure 3. As seen in our experiments, the necessary multiplication factor of FF-4C-COMB never exceeded 1.325 whereas for FF-3C and SKB-ICPP-IMP this factor is close to two. Therefore, FF-4C-COMB offers significantly better average-case performance compared to prior state-of-art.

We also measured the running times of each algorithm (Table 1). As mentioned, we deal with the CPLEX overhead for the SKB-algorithms for fair evaluation. We can see that, in our experiment, our proposed algorithms all run in less than 1.1 μs but the SKB algorithms had running times in the range of 13500 to 160000 μs. Hence all of our algorithms run at least 12000 times faster.

8. 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 [13][17][1][10]. FF-3C is low-degree polynomial in time-complexity, i.e., faster than algorithms based on ILP (or relaxation to LP).

We designed variations on FF-3C with better average-case performance and with the same time-complexity and the same speed competitive ratio. In particular we note 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 the prior state-of-art algorithms [8, 7].

Acknowledgements

This work was partially funded by the Portuguese Science and Technology Foundation (Fundação para a Ciência ea Tecnologia - FCT), Luso-American Development Foundation (FLAD) and the RECOMP project[3].

References

[1] AMD Inc. AMD fusion family of APUs: Enabling a superior, immersive PC experience. http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf, 2010.

[2] B. Andersson, S. Baruah, and J. Jonsson. Static-Priority Scheduling on Multiprocessors. In *Proceedings of the* 22^{nd} *IEEE Real-Time Systems Symposium*, pages 193–202, 2001.

[3] B. Andersson, G. Raravi, and K. Bletsas. Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. Technical report, CISTER/ISEP, Polytechnic Institute of Porto, HURRAY-TR-100103, http://www.hurray.isep.ipp.pt/docs/, 2010.

[3]The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user uses the information at its sole risk and liability.

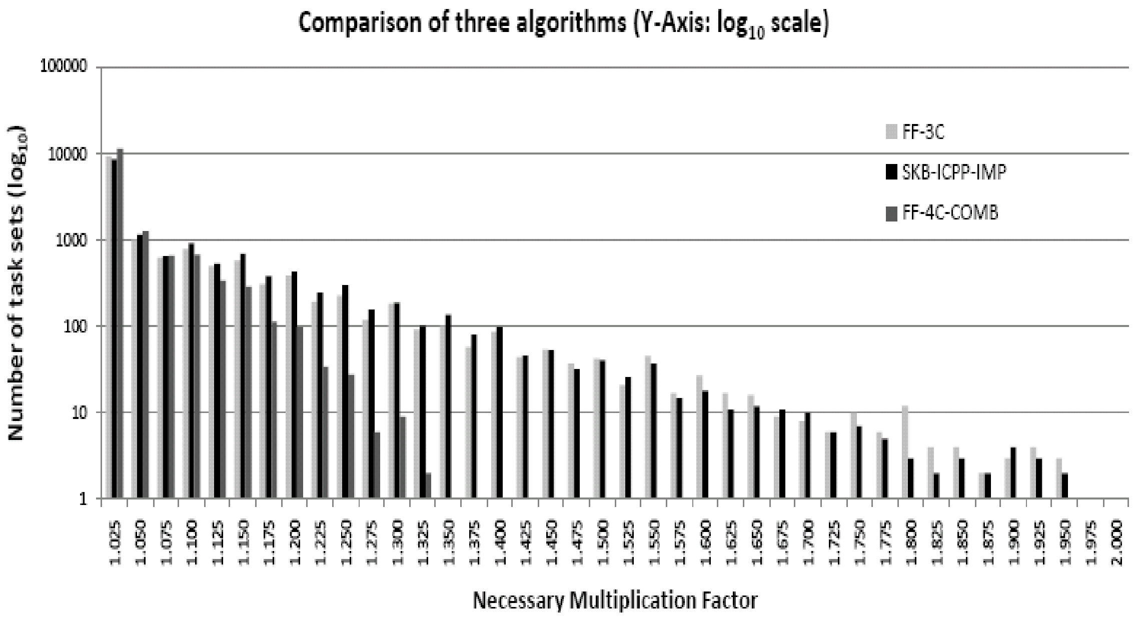

Figure 3. Comparison of necessary multiplication factor for three algorithms (smaller is better)

	New Algorithms				Old Algorithms							
	Measured avg exec time				Measured avg exec time incl CPLEX overhead				Measured avg exec time incl CPLEX overhead — avg CPLEX overhead			
Multiplication factor	FF-3C	FF-4C	FF-4C -NTC	FF-4C -COMB	SKB-RTAS	SKB-RTAS -IMP	SKB-ICPP	SKB-ICPP -IMP	SKB-RTAS	SKB-RTAS -IMP	SKB-ICPP	SKB-ICPP -IMP
1.00	0.85	0.76	0.93	1.08	32481.61	32545.39	394715.80	369120.15	14324.45	14388.23	164603.39	161727.00
1.25	0.52	0.52	0.51	0.53	31657.49	31572.03	393758.65	325045.97	13500.33	13414.87	163646.24	149405.05
1.50	0.49	0.49	0.45	0.48	31751.65	31729.69	381899.86	297359.20	13594.49	13572.52	161185.38	140149.17
1.75	0.47	0.46	0.42	0.46	31744.69	31582.66	337182.98	290084.67	13587.52	13425.49	151049.23	137254.26
2.00	0.49	0.48	0.40	0.48	31736.95	31768.30	291714.93	287719.46	13579.79	13611.13	137972.10	136531.41

Table 1. Comparison of average execution time of algorithms (in microseconds)

[4] B. Andersson and E. Tovar. Competitive Analysis of Partitioned Scheduling on Uniform Multiprocessors. In *Proceedings of the* 15^{th} *International Workshop on Parallel and Distributed Real-Time Systems*, pages 1–8, 2007.

[5] B. Andersson and E. Tovar. Competitive Analysis of Static-Priority of Partitioned Scheduling on Uniform Multiprocessors. In *Proceedings of the* 13^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 111–119, 2007.

[6] S. Baruah. Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms. In *Proceedings of the* 25^{th} *IEEE International Real-Time Systems Symposium*, pages 37–46, 2004.

[7] S. Baruah. Partitioning real-time tasks among heterogeneous multiprocessors. In *Proc. of the* 33^{rd} *International Conference on Parallel Processing*, pages 467–474, 2004.

[8] S. Baruah. Task partitioning upon heterogeneous multiprocessor platforms. In *Proceedings of the* 10^{th} *IEEE International Real-Time and Embedded Technology and Applications Symposium*, pages 536–543, 2004.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd Ed.* McGraw-Hill, 2001.

[10] Freescale Semiconductor. Freescale unveils versatile multicore processor for telematics, consumer and industrial applications. http://media.freescale.com/phoenix.zhtml?c=196520&p=irol-newsArticle&ID=1004385, 2007.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, 1979.

[12] D. Geer. Taking the Graphics processor Beyond Graphics. *IEEE Computer*, 38(9):14–16, 2005.

[13] M. Gschwind, H. P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. Synergistic Processing in Cell's Multicore Architecture. *IEEE Micro*, 26(2):10–24, 2006.

[14] T. G. C. Lavarenne and Y. Sorel. Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors. In *Proceedings of the* 7^{th} *International Workshop on Hardware/Software Codesign*, pages 74–48, 1999.

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

[16] M. López, J. L. Díaz, and D. F. García. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems Journal*, 28:39–68, 2004.

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

[18] C. N. Potts. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10:155–164, 1985.