Technical Report

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

Björn Andersson

Konstantinos Bletsas

HURRAY-TR-091104

Version: 1

Date: 11-03-2009

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

Björn Andersson and Konstantinos Bletsas

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Consider the problem of scheduling a set of implicit deadline sporadic tasks on a heterogeneous multiprocessor so as to meet all deadlines. Tasks cannot migrate and the platform is restricted in that each processor is either of type-1 or type-2 (with each task characterized by a different speed of execution upon each type of processor). We present an algorithm for this problem with a time complexity of O(n*m), where n is the number of tasks and m is the number of processors. It offers the guarantee that if a task set can be scheduled by any non-migrative algorithm to meet deadlines then our algorithm meets deadlines as well if given processors twice as fast. Although this result is proven for only a restricted heterogeneous multiprocessor, we consider it significant for being the first realtime scheduling algorithm to use a low-complexity binpacking approach to schedule tasks on a heterogeneous multiprocessor with provably good performance.

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

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

Abstract

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

We present an algorithm for this problem with a time-complexity of $O(n \cdot m)$, where n is the number of tasks and m is the number of processors. It offers the guarantee that if a task set can be scheduled by any non-migrative algorithm to meet deadlines then our algorithm meets deadlines as well if given processors twice as fast. Although this result is proven for only a restricted heterogeneous multiprocessor, we consider it significant for being the first real-time scheduling algorithm to use a low-complexity bin-packing approach to schedule tasks on a heterogeneous multiprocessor with provably good performance.

1. Introduction

Parallel processing platforms are spreading at an unprecedented rate [8]. Traditionally, parallel processing was used to speed up large computational jobs such as predicting the weather. Today however, parallel processing platforms are also used in low-end and embedded real-time systems thanks to the availability of multicore processors. Such systems often consist of numerous independent tasks.

Designers are well-aware that processing units specialized for a specific function can offer significant performance boost. For example, computer graphics are rendered much faster with a graphics processor than with a general purpose processor. Similar advantages can be obtained using network processors, digital signal processors, SIMD arrays, etc. Consequently, heterogeneous multiprocessors (especially on a single chip) now enjoy widespread use. Virtually all major semiconductor companies are offering or have declared plans to offer heterogeneous multiprocessors implemented on a single chip [11][15][10][14][20].

Despite the widespread availability of heterogeneous multiprocessor platforms and the eagerness to use them, their deployment in embedded systems is a non-trivial task for designers. The complicating factor is that many embedded systems have real-time requirements, whose satisfaction at run-time has to be proven/guaranteed *a priori*. The way tasks are scheduled significantly influences whether their timing requirements are met. For this reason, a comprehensive toolbox of real-time scheduling algorithms and analysis techniques [22][23] have been developed in order to help designers. Unfortunately, few results apply to heterogeneous multiprocessors.

An algorithm for deciding if and only if a task set can be scheduled on a heterogeneous platform exists [5] but it assumes that tasks can migrate. This algorithm is useful for researchers but the assumption that tasks can migrate is often unrealistic in practice, since processing units with different functionalities typically have different instruction sets and data layouts (big-endian/little-endian for example). 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 EDF [19]); 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 schemes and (ii) Integer-Linear-Programming (ILP) modeling offer provably good performance.

Bin-packing schemes are popular for task assignment but unfortunately, the proof techniques used on identical multiprocessors do not easily translate to heterogeneous

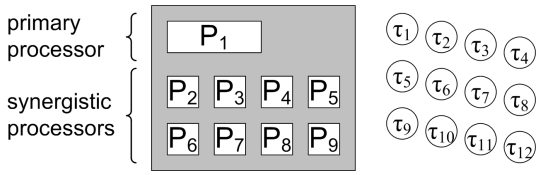

Figure 1. An example of the task-to-processor assignment problem over a multiprocessor with two types of processors. The Cell processor is such an architecture: P_1 is a general purpose processor and P_2-P_9 are 8 vector processors (termed "synergistic"). Each task must be assigned to exactly one processor; a processor may be assigned zero, one or many tasks. All task deadlines must be met with uniprocessor scheduling.

multiprocessors. Consequently, the current literature offers no bin-packing scheme for assigning real-time tasks on heterogeneous multiprocessors. Instead, the task-to-processor assignment problem is modelled [6][7] as Zero-One Integer-Linear-Programming (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 (see [12], p. 245). Via relaxation of the ILP formulation to LP and certain tricks [21], it is possible to design a polynomial-time approximation scheme [6][7]. The derived linear program is solvable in polynomial time [16][17] but unfortunately the degree of the polynomial is high.

Yet, in practice many heterogeneous multiprocessors only use two types of processors; for example one type is a graphics processor and the other one is general-purpose. Intel [15] and AMD [1] plan to ship such chips; FreeScale already does [11]. Graphics processors were traditionally meant just for graphics tasks, hence task assignment was straightforward. Designers today [13] use graphics processors in a wide range of calculations though and this makes task assignment non-trivial. This is accentuated in the Cell processor [14][20] (Figure 1) where the "graphics processor" (called synergistic processor) is Turing-complete, thus able to compute anything that the main processor can. Such chips are now deployed in embedded systems for their excellent performance/cost ratio. CAD tool support for task assignment algorithms exploiting their special structure would tap more of the potential performance.

This motivates our new task assignment algorithm for heterogeneous multiprocessors, which exploits the fact that only two processor types exist. It is a bin-packing algorithm whose time complexity is $O(n \cdot m)$, where n and m are the number of tasks and processors respectively. This algorithm offers the guarantee that if a task set can be scheduled by any non-migrative algorithm to meet deadlines then our algorithm meets deadlines as well provided that it is given processors that are twice as fast.

In the remainder of this paper, Section 2 offers necessary preliminaries. Section 3 presents some useful lemmata, used in Section 4, where we formulate the new algorithm and prove its performance. Section 5 discusses the context of the work and previous work and concludes.

2. Preliminaries

In a computer platform with two 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 is comprised of τ, a set of tasks each of which releases a (potentially infinite) sequence of jobs. The sporadic model is used, i.e. the exact time of a job release is unknown but the time between any two successive job releases of a task τ_i is at least T_i.

A task is assigned to a processor and all jobs released by this task must execute on this processor. The execution requirement (in time units) of some task τ_i depends on the type of processor to which it is assigned. It is $\frac{C_i}{r_{i,1}}$ upon a type-1 host processor but $\frac{C_i}{r_{i,2}}$ upon a type-2 processor. Note that we allow $r_{i,1} = 0$ (or $r_{i,2} = 0$) if task τ_1 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 [19]. A slight adaptation of a previously known result [19] 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 $\sum_{\tau \in \tau[p]} \frac{C_i}{r_{i,z} \cdot T_i} \leq 1$, then all deadlines are met.*

Proof: Follows from Theorem 7 in [19]. □

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

$$\sum_{\tau \in \tau[p]} \frac{C_i}{r_{i,1} \cdot T_i} \leq 1 \ \forall p \in P^1 \quad (1)$$

$$\sum_{\tau \in \tau[p]} \frac{C_i}{r_{i,2} \cdot T_i} \leq 1 \ \forall p \in P^2 \quad (2)$$

Thus our problem of scheduling tasks on a heterogeneous multiprocessor with two types of processors is reduced to finding an assignment of tasks to processors such that the above constraints are satisfied. In the general case, however, even this problem is intractable; see Theorem 1.

Theorem 1. *Deciding if a feasible mapping exists for a given task set and computing platform is NP-complete.*

Proof: If $|P^1|=|P^2|=1$, $r_{i,1}=r_{i,2}=1\ \forall i$ and $\sum_{i\in\tau}\frac{C_i}{T_i}=2$ the problem reduces to PARTITION – see [12], p. 223. □

Faced with this fact, we opt for the design of a non-optimal algorithm which would still offer good performance but which would be of polynomial time complexity.

Commonly, the performance of an algorithm is characterized using the notion of the *utilization bound* [19]: 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 on uniprocessor scheduling [19] 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 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 computing platform Π whose every processor is CPT_A times faster than the corresponding processor in Π'.[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 to meet deadlines (under another algorithm) but which would miss a deadline with the actually used algorithm even if processor speeds were multiplied by an "infinite" factor. Such behavior is undesired in design tools, consequently we aim to design an algorithm with a finite (ideally small) speed competitive ratio.

3. Useful results

Before describing the new algorithm, we will derive some useful statements which facilitate various proofs.

Bin-packing task assignment algorithms are popular in the context of identical [18] and also uniform [3] multiprocessors, as they run fast and offer a finite speed competitive ratio. Yet, the straightforward application of a bin-packing algorithm 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 $1..k$ are characterized by $C_i{=}T_i{=}1$, $r_{i,1}=1$, $r_{i,2}=k$ and tasks indexed $k{+}1..2k$ are characterized by $C_i{=}T_i{=}1$, $r_{i,1}=k$, $r_{i,2}=1$.*

1. Our notion of speed competitive ratio in this paper is equivalent to that in previous work by Baruah [5]. It differs from that used in [3][4].

It is possible to assign tasks such that the condition of Lemma 1 is met for both processors; assigning tasks indexed $1..k$ to P_2 and the rest to P_1 does that. Yet, the application of a normal bin-packing algorithm for identical multiprocessors (such as Next-Fit or 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. Whether under Next-Fit or First-Fit, τ_1 will end up on P_1 (as processors are considered by order of ascending index). Yet, at most one task from among those indexed $1..k$ can be assigned there. Thus, $k-1\geq 2$ tasks (those indexed $2..k$) will then have to be assigned to P_2. The bin-packing scheme would continue trying to assign tasks indexed $k{+}1..2k$ to P_2; none would fit and the algorithm would fail.

Let us now provide the bin-packing algorithm with processors $k-1$ times faster. Then, tasks indexed $1..k-1$ will be assigned to P_1 and the k^{th} task to P_2 before considering tasks indexed $k{+}1..2k$. 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.

The above reasoning holds for any $k\geq 3$. By considering $k{\to}\infty$ we obtain that the speed competitive ratio of such bin-packing schemes is infinite.

It can be seen that the cause of the 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 a task may be assigned to a processor where it executes slowly. It seems like a good idea to try to assign each task to the processor where it executes faster. We will use this idea, therefore let us 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:

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

$$\forall\tau_i\in\tau^1 : r_{i,1}\geq r_{i,2} \tag{4}$$

$$\forall\tau_i\in\tau^2 : r_{i,1}< r_{i,2} \tag{5}$$

We proceed with two useful observations (their correctness is evident; for formal proof, see the Appendix).

Lemma 2. *If there is a task τ_i in τ^1 such that $1<\frac{C_i}{r_{i,1}\cdot T_i}$, it is then impossible to meet deadlines. Likewise if there is a task τ_i in τ^2 such that $1<\frac{C_i}{r_{i,2}\cdot T_i}$.*

Proof: See the Appendix. □

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

$$\sum_{i\in\tau^1}\frac{C_i}{r_{i,1}\cdot T_i}+\sum_{i\in\tau^2}\frac{C_i}{r_{i,2}\cdot T_i}>|P^1|+|P^2| \tag{6}$$

Proof: See the Appendix. □

3.1. Inequalities which we will use

We 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 $\frac{v_i}{w_i} \geq \frac{v_i}{w_i}$. (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 $\sum_{i=1}^{n} x_i \cdot v_i \leq \text{CAP}$
and $0 \leq x_i \leq 1$
and x_i is a real number.

(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 Fractional Knapsack Problem can be solved by the following algorithm:*

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

This is known from undergraduate textbooks (for example, see Chapter 16.2 in [9]). We now consider a multiprocessor 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*

$$\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} > \frac{|P^1|}{2} - x \tag{7}$$

and for every pair of tasks $\tau_i \in A1$ and $\tau_j \in \tau \setminus A1$ it holds that $\frac{r_{i,1}}{r_{i,2}} - 1 \geq \frac{r_{j,1}}{r_{j,2}} - 1$. Let $A2$ denote $\tau^1 \setminus A1$.

Let B1 denote a subset of τ^1 such that

$$\sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} \leq \frac{|P^1|}{2} - x \tag{8}$$

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

$$\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in A2} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i} \tag{9}$$

Proof: Let us arbitrarily choose $A1$ and B1 as defined above. Using Inequalities 7 and 8 we clearly get:

$$\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} > \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} \tag{10}$$

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

Instance1.
CAP = left-hand side of Inequality 10,
$v_i = \left(\frac{1}{r_{i,2}} - \frac{1}{r_{i,1}}\right)\frac{C_i}{T_i}$,
$w_i = \frac{C_i}{r_{i,1} \cdot T_i}$

Instance2.
CAP = right-hand side of Inequality 10,
$v_i = \left(\frac{1}{r_{i,2}} - \frac{1}{r_{i,1}}\right)\frac{C_i}{T_i}$,
$w_i = \frac{C_i}{r_{i,1} \cdot T_i}$

Using, Inequality 10, we get:

$$CAP_{Instance1} > CAP_{Instance2} \tag{11}$$

where $CAP_{Instance1}$ is defined as CAP in Instance1 and $CAP_{Instance2}$ is defined analogously.

Observe that Instance1 and Instance 2 differ only in their value of CAP. Instance1 can be perceived as a relaxed version of Instance2. Therefore we have:

$$SUMVALUE_{Instance1} \geq SUMVALUE_{Instance2} \tag{12}$$

where $SUMVALUE_{Instance1}$ is defined as the value of the variable $SUMVALUE$ in Instance1 when the algorithm in Lemma 4 terminates and $SUMVALUE_{Instance2}$ is defined analogously. Observe that this choice of $CAP_{Instance1}$ ensures that, on line 15 of the pseudocode, x_i is assigned the value 1 when the algorithm of Lemma 4 takes Instance1 as input. Note that

$$SUMVALUE_{Instance1} = \sum_{i \in A1} \left(\frac{r_{i,1}}{r_{i,2}} - 1\right)\frac{C_i}{r_{i,1} \cdot T_i} \tag{13}$$

because of the definition of A1. Also, note that

$$SUMVALUE_{Instance2} \geq \sum_{i \in B1} \Big(\frac{r_{i,1}}{r_{i,2}} - 1\Big) \frac{C_i}{r_{i,1} \cdot T_i} \quad (14)$$

because, in Instance2, the set B1 cannot produce a solution for the fractional knapsack problem that is higher than the optimal one. Applying Equation 13 and Inequality 14 on Inequality 12 yields:

$$\sum_{i \in A1} \Big(\frac{r_{i,1}}{r_{i,2}} - 1\Big) \frac{C_i}{r_{i,1} \cdot T_i} \geq \sum_{i \in B1} \Big(\frac{r_{i,1}}{r_{i,2}} - 1\Big) \frac{C_i}{r_{i,1} \cdot T_i} \quad (15)$$

Then we can reason as follows.

$$\begin{aligned}(15) \Leftrightarrow \sum_{i \in A1} \left(\frac{1}{r_{i,2}} - \frac{1}{r_{i,1}}\right) \frac{C_i}{T_i} \geq \sum_{i \in B1} \left(\frac{1}{r_{i,2}} - \frac{1}{r_{i,1}}\right) \frac{C_i}{T_i} \Leftrightarrow \\ -\Big(\sum_{i \in A1} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} - \sum_{i \in A1} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i}\Big) \\ \leq -\Big(\sum_{i \in B1} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} - \sum_{i \in B1} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i}\Big)\end{aligned}$$

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

$$\sum_{i \in \tau^1} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} = \sum_{i \in B1} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i} \quad (16)$$

Adding these two together produces the inequality:

$$\begin{aligned}\sum_{i \in \tau_1} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau_2} \frac{C_i}{r_{i,2} \cdot T_i} \\ -\Big(\sum_{i \in A1} \frac{C_i}{r_{i,2} \cdot T_i} - \sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i}\Big) \\ \leq \sum_{i \in B1} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i} \\ -\Big(\sum_{i \in B1} \frac{C_i}{r_{i,2} \cdot T_i} - \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i}\Big) \quad (17)\end{aligned}$$

Rearranging the terms yields:

$$\begin{aligned}\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} + \Big(\sum_{i \in \tau^1} \frac{C_i}{r_{i,2} \cdot T_i} - \sum_{i \in A1} \frac{C_i}{r_{i,2} \cdot T_i}\Big) \\ + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} \\ + \Big(\sum_{i \in B1} \frac{C_i}{r_{i,2} \cdot T_i} - \sum_{i \in B1} \frac{C_i}{r_{i,2} \cdot T_i}\Big) + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i} \\ = \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i} \quad (18)\end{aligned}$$

Exploiting the fact that $A2 = \tau^1 \setminus A1$ gives us:

$$\begin{aligned}\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in A2} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \\ \leq \sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in B2} \frac{C_i}{r_{i,2} \cdot T_i}\end{aligned}$$

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 τ^{H1} and τ^{H2} are two disjoint subsets of τ. We apply Lemma 5 on $\tau \setminus (\tau^{H1} \cup \tau^{H2})$ and then add the same sum to the 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 $(\tau^1 \setminus (\tau^{H1} \cup \tau^{H2}))$ such that*

$$\sum_{i \in A1} \frac{C_i}{r_{i,1} \cdot T_i} \geq \frac{|P^1|}{2} - x \quad (19)$$

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

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

$$\sum_{i \in B1} \frac{C_i}{r_{i,1} \cdot T_i} \leq \frac{|P^1|}{2} - x \quad (20)$$

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

$$\begin{aligned}\sum_{i \in \tau^{H1}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{H2}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} + \\ \sum_{i \in A1} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in A2} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^2 \setminus (\tau^{H1} \cup \tau^{H2})} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} \\ \leq \sum_{i \in \tau^{H1}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{H2}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} + \\ \sum_{i \in B1} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in B2} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} \quad (21)\end{aligned}$$

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 preferably be assigned to the type of processor on which it runs faster.

Idea2. A task which has a utilization less than 50% on one type of processor and utilization greater than

50% on a processor of the other type of processor should be assigned to the former type of processor. This is a special case of Idea1 but we state it separately because this facilitates creating an algorithm with the desired speed competitive ratio. The rationale behind this idea is that we are interested in comparing the performance of our new algorithm versus every other algorithm using processors of half the speed; by following Idea2, we create assignments that mimic what every other algorithm does (assuming that the other algorithm successfully assigns tasks).

Based on these ideas, we will use the concepts of τ^1 and τ^2 (already defined in Section 2). We also define:

$$\tau^{H1} = \{\tau_i \in \tau^1 : \frac{C_i}{T_i \cdot r_{i,2}} > 1/2\} \quad (22)$$

$$\tau^{H2} = \{\tau_i \in \tau^2 : \frac{C_i}{T_i \cdot r_{i,1}} > 1/2\} \quad (23)$$

$$\tau^{F1} = \tau^1 \setminus \tau^{H1} \quad (24)$$

$$\tau^{F2} = \tau^2 \setminus \tau^{H2} \quad (25)$$

Intuitively, τ^{H1} and τ^{H2} identify those tasks which should be assigned based on Idea2. τ^{H1} stands for "Set of tasks that are heavy if they are not assigned to their favorite processor, of type-1." Analogous for τ^{H2}. Also, intuitively, τ^{F1} and τ^{F2} identify those tasks which should be assigned based on Idea1. τ^{F1} stands for "Set of tasks that have a processor of type-1 as its favorite and for which heaviness should not be considered." Analogous for τ^{F2}.

Figure 2 shows the new algorithm FF-3C. The intuition behind the design of our algorithm is that first we assign tasks to their favorite processors so that the tasks are not heavy (lines 4-5). Then we assign the remaining tasks to their favorite processors (lines 6-7). Then if there are remaining tasks these tasks have to be assigned to processors that are not their favorite (line 12 and line 20). The name FF-3C is derived from the fact that first-fit is used to assign a task to a processor and a task has three chances to be used by first-fit. A task has the chance to be assigned by first-fit if it follows Idea2 (to avoid making a task heavy). Then a task has the chance to be assigned to its favorite processor. And then a task has a chance to be assigned to a processor which is not its favorite processor.

The algorithm FF-3C keeps track of processor utilizations in a global vector U, initialized to zero (line 2).

As already mentioned, the algorithm FF-3C performs several passes with first-fit bin-packing. It uses a subroutine `first-fit` 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 tasks that were successfully assigned. Figure 3 shows pseudo-code for `first-fit`.

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

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

Figure 2. The new algorithm, FF-3C

1. **function** first-fit(ts : set of tasks; ps : set of processors) return set of tasks
2. assigned_tasks := $\emptyset$
3. Order the tasks ts and order the processors ps. This order should be maintained during the execution of the function first-fit
4. τ_i := first task in ts
5. P_p := first processor in ps
6. Let k denote the type of processor P_p (either 1 or 2)
7. **if** U[p]+$\frac{C_i}{T_i \cdot r_{i,k}} \leq 1$ **then**
8. U[p] := U[p]+$\frac{C_i}{T_i \cdot r_{i,k}}$
9. τ[p] := τ[p] $\cup \{\tau_i\}$
10. assigned_tasks := assigned_tasks $\cup \{\tau_i\}$
11. **if** remaining tasks exist in ts **then**
12. τ_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_p := next processor in ps
20. go to line 6.
21. **else**
22. **return** assigned_tasks
23. **end if**
24. **end if**

Figure 3. First-fit bin-packing

Theorem 2. *The speed competitiveness 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 al-

gorithm, over computing platform $\Pi^{'}$ with processors each half as fast as the corresponding one in Π. This, we will prove (by contradiction). From the definition of $\Pi^{'}$:

$$\frac{r^{'}_{i,1}}{r_{i,1}} = \frac{r^{'}_{i,2}}{r_{i,2}} = \frac{1}{2} \; \forall i \tag{26}$$

Assume that FF-3C has failed to assign τ on Π but it is possible (using an algorithm OPT) to assign τ on $\Pi^{'}$. 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 $|\tau^{H1}| \leq |P^1|$ then there would be no failure on line 4 in FF-3C. Therefore, we know that $|\tau^{H1}| \geq |P^1|$+1. Hence OPT must assign at least one task in τ^{H1} on a processor in P^2. Let τ_i denote this task. Using the definition of τ^{H1} gives us that $\frac{C_i}{T_i \cdot r_{i,2}} > 1/2$ and applying Equation 26 yields $\frac{C_i}{T_i \cdot r^{'}_{i,2}} > 1$. But since OPT assigned τ_i on a processor in P^2 it must be that $\frac{C_i}{T_i \cdot r^{'}_{i,2}} \leq 1$. This is a contradiction.

Failure on line 5 in FF-3C.

This results in a contradiction. It can be shown because this case is symmetric to the case above.

Failure on line 9 in FF-3C.

From the case, we obtain that $\tau^{F11} \subset \tau^{F1}$ and $\tau^{F22} \subset \tau^{F2}$. Therefore, there was a task $\tau_{failed1} \in \tau^{F1}$ which could not be assigned on any processor in P^1 and there was a task $\tau_{failed2} \in \tau^{F2}$ which could not be assigned on any processor in P^2. Consequently, we obtain:

$$\forall p \in P^1 : U[p] + \frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}} > 1 \tag{27}$$

$$\text{and} \;\; \forall p \in P^2 : U[p] + \frac{C_{failed2}}{T_{failed2} \cdot r_{failed2,2}} > 1 \tag{28}$$

Suppose that $\frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}} > 1/2$. We know that $\tau_{failed1} \in \tau^{F1}$ and this gives us $r_{failed1,1} \geq r_{failed1,2}$ which gives us $\frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,2}} > 1/2$. This implies that $\tau_{failed1} \in \tau^{H1}$ but this is impossible because τ^{H1} and τ^{F1} are disjoint. Therefore, we have that: $\frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}} \leq 1/2$. With analogous reasoning, we obtain: $\frac{C_{failed2}}{T_{failed2} \cdot r_{failed2,2}} \leq 1/2$. Using these inequalities on Inequalities 27 and 28 gives:

$$\forall p \in P^1 : U[p] > 1/2 \tag{29}$$

$$\text{and} \;\; \forall p \in P^2 : U[p] > 1/2 \tag{30}$$

Observing that tasks assigned on processors in P^1 are a subset of τ^1 and using Inequality 29 gives us:

$$\sum_{\tau_i \in \tau^1} \frac{C_i}{T_i \cdot r_{i,1}} > \frac{|P^1|}{2} \tag{31}$$

With analogous reasoning, we obtain:

$$\sum_{\tau_i \in \tau^2} \frac{C_i}{T_i \cdot r_{i,2}} > \frac{|P^2|}{2} \tag{32}$$

Observing these two inequalities and Equation 26 and Lemma 3 gives us that OPT fails to assign tasks on $\Pi^{'}$. This is a contradiction.

Failure on line 15 in FF-3C.

From the case, we obtain that $\tau^{F11} \subset \tau^{F1}$ and $\tau^{F22} = \tau^{F2}$. Therefore, there was a task $\tau_{failed} \in (\tau^{F1} \setminus \tau^{F11})$ which was attempted to each of the processors in P^2. But all of them failed. Therefore, we have:

$$\forall p \in P^2 : U[p] + \frac{C_{failed}}{T_{failed} \cdot r_{failed,2}} > 1 \tag{33}$$

We can add these inequalities together and get:

$$\sum_{p \in P^2} U[p] > |P^2| \cdot (1 - \frac{C_{failed}}{T_{failed} \cdot r_{failed,2}}) \tag{34}$$

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

$$\sum_{i \in (\tau^{H2} \cup \tau^{F22} \cup \tau^{F12})} \frac{C_i}{T_i \cdot r_{i,2}} > |P^2| \cdot (1 - \frac{C_{failed}}{T_{failed} \cdot r_{failed,2}}) \tag{35}$$

We also know that since $\tau_{failed} \in \tau^{F1}$ it follows that τ_{failed} is not in τ^{H1} and hence:

$$\frac{C_{failed}}{T_{failed} \cdot r_{failed,2}} \leq 1/2 \tag{36}$$

But since $\tau_{failed1} \in \tau^{F1} \subseteq \tau^1$, using Inequality 4 gives us:

$$\frac{C_{failed}}{T_{failed} \cdot r_{failed,1}} \leq 1/2 \tag{37}$$

Combining this with Inequality 34 yields:

$$\sum_{i \in (\tau^{H2} \cup \tau^{F22} \cup \tau^{F12})} \frac{C_i}{T_i \cdot r_{i,2}} > \frac{|P^2|}{2} \tag{38}$$

We also know that FF-3C has executed line 6 and when it performed first-fit-bin-packing, there must have been a task $\tau_{failed1} \in (\tau^{F1} \setminus \tau^{F11})$ 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:

$$\forall p \in P^1 : U[p] + \frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}} > 1 \quad (39)$$

Adding these inequalities together gives us:

$$\sum_{p \in P^1} U[p] > |P^1| \cdot (1 - \frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}}) \quad (40)$$

We know that the tasks assigned to processors in P^1 are $\tau^{H1} \cup \tau^{F11}$. Therefore, we have:

$$\sum_{i \in (\tau^{H1} \cup \tau^{F11})} \frac{C_i}{T_i \cdot r_{i,1}} > |P^1| \cdot (1 - \frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}}) \quad (41)$$

We also know that since $\tau_{failed1} \in \tau^{F1}$ it follows (from the definition τ^{H1} in the beginning of this section) that $\tau_{failed1}$ is not in τ^{H1} and hence:

$$\frac{C_{failed1}}{T_{failed1} \cdot r_{failed1,1}} \leq 1/2 \quad (42)$$

Combining them yields:

$$\sum_{i \in (\tau^{H1} \cup \tau^{F11})} \frac{C_i}{T_i \cdot r_{i,1}} > \frac{|P^1|}{2} \quad (43)$$

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

$$\forall \tau_i \in \tau^{H1} : \frac{C_i}{T_i \cdot r_{i,2}} > 1/2 \quad (44)$$

Using Equation 26 gives us:

$$\forall \tau_i \in \tau^{H1} : \frac{C_i}{T_i \cdot r'_{i,2}} > 1 \quad (45)$$

Therefore, OPT would fail if a task in τ^{H1} was assigned to a processor in P^2. Since we know that OPT succeeded, it follows that every task in τ^{H1} is assigned to a processor in P^1. With analogous reasoning, we have that every task in τ^{H2} is assigned to a processor in P^2. Let τ^{OPT1} denote the tasks (except those from τ^{H1}) that were assigned to processors in P^1 by OPT. Analogously, let τ^{OPT2} denote the tasks (except those from τ^{H2}) that were assigned to processors in P^2 by OPT. Therefore (using Inequalities 1 and 2), we know that:

$$\sum_{\tau_i \in (\tau^{H1} \cup \tau^{OPT1})} \frac{C_i}{T_i \cdot r'_{i,1}} \leq |P^1| \quad (46)$$

$$\text{and} \quad \sum_{\tau_i \in (\tau^{H2} \cup \tau^{OPT2})} \frac{C_i}{T_i \cdot r'_{i,2}} \leq |P^2| \quad (47)$$

Using Equation 26 gives us:

$$\sum_{\tau_i \in (\tau^{H1} \cup \tau^{OPT1})} \frac{C_i}{T_i \cdot r_{i,1}} \leq \frac{|P^1|}{2} \quad (48)$$

$$\text{and} \quad \sum_{\tau_i \in (\tau^{H2} \cup \tau^{OPT2})} \frac{C_i}{T_i \cdot r_{i,2}} \leq \frac{|P^2|}{2} \quad (49)$$

Having obtained inequalities about the assignments of FF-3C and OPT, we can now reason about them. Rewriting Inequality 43 yields:

$$\sum_{i \in \tau^{F11}} \frac{C_i}{T_i \cdot r_{i,1}} > \frac{|P^1|}{2} - \sum_{i \in \tau^{H1}} \frac{C_i}{T_i \cdot r_{i,1}} \quad (50)$$

Also, simple rewriting of Inequality 48 yields:

$$\sum_{\tau_i \in \tau^{OPT1}} \frac{C_i}{T_i \cdot r_{i,1}} \leq \frac{|P^1|}{2} - \sum_{\tau_i \in \tau^{H1}} \frac{C_i}{T_i \cdot r_{i,1}} \quad (51)$$

We can see that Inequalities 50 and 51 with $x = \sum_{i \in \tau^{H1}} \frac{C_i}{T_i \cdot r_{i,1}}$ ensure that the assumptions of Lemma 6 are true. Using Lemma 6 gives us:

$$\sum_{i \in \tau^{H1}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{H2}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{F11}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{F12}} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^{F22}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} \leq \sum_{i \in \tau^{H1}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{H2}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} + \sum_{i \in OPT1} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in OPT2} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} \quad (52)$$

Applying Inequality 48 and Inequality 49 on Inequality 52 gives us:

$$\sum_{i \in \tau^{H1}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{H2}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{F11}} \frac{1}{r_{i,1}} \cdot \frac{C_i}{T_i} + \sum_{i \in \tau^{F12}} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau^{F22}} \frac{1}{r_{i,2}} \cdot \frac{C_i}{T_i} \leq \frac{|P^1|}{2} + \frac{|P^2|}{2}$$

Applying Inequality 38 and Inequality 43 on the above inequality gives us:

$$\frac{|P^1|}{2} + \frac{|P^2|}{2} < \frac{|P^1|}{2} + \frac{|P^2|}{2} \quad (53)$$

This is a contradiction.

Failure on line 23 in FF-3C.

A contradiction results – the proof is analogous that for the case "Failure on line 15 in FF-3C".

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

5. Discussion and Conclusions

The model considered is restricted but captures many current and future single-chip heterogeneous multiprocessors. The Cell processor [14][20] is comprised of a Power processor and 8 synergistic processor elements. The planned AMD Fusion [1] is similarly arranged, with a processor and graphics processors integrated onto a single chip. A difference from the Cell processor is that the graphics processors are not Turing-complete, hence there may be some programs which they cannot execute but the main processor can. This can be easily modelled by treating the execution rate of those tasks on the graphics processor as zero. Intel has similar plans [15]. Similar solutions are already in the marketplace, such as the MPC5121e processor [11] and the network processor [10] from Freescale. In fact, most network processors are heterogeneous multiprocessors with two types of processors.

We specify preemptive EDF scheduling on each processor but do not specify which processor performs the scheduling. In the Cell processor, the synergistic processors can only execute tasks assigned to them by the Power processor; they are not autonomous. Our approach can be applied though by having the Power processor keep track of all tasks (ready, runnable or blocked) of all processors (i.e. both the Power processor and the synergistic ones) and then notify each processor which task to execute and when. Similarly with AMD Fusion and MPC5121e.

As seen, the model studied in this paper is of significant practical interest and its importance is expected to increase further in the future. But while the hardware/software codesign community has dealt with the problem of scheduling real-time tasks (see for example [18]), algorithms are evaluated by simulation only; not by proofs of their performance. Scheduling theorists however offer proofs of algorithm performance. Multiprocessors are commonly categorized as:

Identical – All processors have the same speed.

Uniform – Each processor has a speed. A processor with a higher speed executes every task proportionately faster.

Heterogeneous (sometimes called unrelated parallel machines) – A matrix, indexed with i and p, gives the speed that task i has when it executes on processor p.

Clearly, heterogeneous multiprocessors are the most general model but are still plagued by requiring scheduling algorithms to have a large computational complexity if provably good performance must be achieved. We claim that our model of heterogeneous multiprocessors with two types of processors (i) is capable of modelling many of those heterogeneous multiprocessors that are of practical interest and (ii) allows the design of algorithms that run much faster but still maintain provably good performance. Indeed, our new algorithm for task assignment over a heterogeneous multiprocessor with only two types of processors is certainly faster than algorithms based on Integer Linear Programming (or relaxation to Linear Programming).

References

[1] AMD Inc. AMD Completes ATI Acquisition and Creates Processing Powerhouse (Press release). http://www.amd.com/us-en/Corporate/VirtualPressRoom/0,,51_104_543~113741,00.html, October 2006.

[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 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] S. Borkar. Thousand Core Chips - A Technology Perspective. In *Proceedings of the 44th ACM/IEEE Design Automation Conference*, pages 746–749, 2007.

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

[10] Freescale Semiconductor. C-5 Network Processor (NP). http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=C-5&nodeId=01DFTQ3126q62S.

[11] Freescale Semiconductor. Freescale Unveils Multicore Processor for Telematics, Consumer, Industrial Applications. http://parts.ihs.com/news/freescale-multicore-processor.htm, May 2007. (Press release).

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

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

[14] 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.

[15] Intel Corporation. Intel Developer Forum Day 1 News Disclosures From Beijing (Press release). http://www.intel.com/pressroom/archive/releases/20070416supp.htm, April 2007.

[16] N. Karmakar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[17] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademia Nauk*, 20:1093–1096, 1979.

[18] 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.

[19] 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.

[20] 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.

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

[22] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: a framework for developing real-time systems. *Proc. of the IEEE*, 82(1):68–82, 1994.

[23] K. W. Tindell. An Extensible Approach for Analysing Fixed Priority Hard Real-Time Tasks. Technical Report YCS 189, Dept. of Computer Science, University of York, UK, 1992.

Appendix

Lemma 2. *If there is a task τ_i in τ^1 such that $1 < \frac{C_i}{r_{i,1} \cdot T_i}$, it is then impossible to meet deadlines. Likewise if there is a task τ_i in τ^2 such that $1 < \frac{C_i}{r_{i,2} \cdot T_i}$.*

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

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

$$\sum_{i \in \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} > |P^1| + |P^2| \quad (54)$$

Proof: The proof is by contradiction. Let τ be a task set for which Inequality 54 holds. Assume then that a feasible partitioning of τ exists.

Given that τ is feasible, the set of constraints expressed by Inequalities 1 and 2 must hold. Then, from Inequalities 1 and 2 respectively we have:

$$\sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,1} \cdot T_i} \leq 1 \ \forall p \in P^1 \quad (55)$$

$$\sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,2} \cdot T_i} + \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq 1 \ \forall p \in P^2 \quad (56)$$

However, from Inequalities 5 and 4 respectively:

$$(5) \Rightarrow \frac{C_i}{r_{i,1} \cdot T_i} > \frac{C_i}{r_{i,2} \cdot T_i} \forall i \in \tau^2 \quad (57)$$

$$\text{and} \quad (4) \Rightarrow \frac{C_i}{r_{i,1} \cdot T_i} \leq \frac{C_i}{r_{i,2} \cdot T_i} \forall i \in \tau^1 \quad (58)$$

Then $(55) \overset{(57)}{\Rightarrow}$

$$\sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} < 1 \ \forall p \in P^1 \quad (59)$$

Likewise, $(56) \overset{(58)}{\Rightarrow}$

$$\sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq 1 \ \forall p \in P^2 \quad (60)$$

We can combine Inequalities 59 and 60 into:

$$\sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq 1 \ \forall p \quad (61)$$

Via summation of Inequality 61 over all p we obtain

$$\sum_{p} \sum_{i \in \tau[p] \cap \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{p} \sum_{i \in \tau[p] \cap \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq \sum_{p} 1$$

$$\Rightarrow \sum_{i \in \tau^1} \frac{C_i}{r_{i,1} \cdot T_i} + \sum_{i \in \tau^2} \frac{C_i}{r_{i,2} \cdot T_i} \leq |P^1| + |P^2| \quad (62)$$

This contradicts Inequality 54. □