ACKNOWLEDGMENTS

This work is supported in part by a US National Science Foundation (NSF) CAREER award to Fred Chong, by NSF grant CCR-9812415, by an NPSC fellowship to Diana Keen, by grants from Mitsubishi and Altera, and by grants from the University of California Davis Academic Senate. More information is available at http://arch.cs.ucdavis.edu/AP.

REFERENCES

[1] D. Burger and T. Austin,, "The SimpleScalar Tool Set, v2.0," *Computer Architecture News* Vol. 25, No. 3, June 1997.
[2] M. Gokhale, B. Holmes, and K. Lobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer*, vol. 28, no. 4, pp. 23-31, Apr. 1995.
[3] K. Itoh, Y. Nakagome, S. Kimura, and T. Watanabe, "Limitations and Challenges of Multigigabit DRAM Chip Design," *IEEE J. Solid-State Circuits*, vol. 32, no. 5, pp. 624-634, 1997.
[4] Intel, *Intel Architecture Software Developer's Manual*, 1997.
[5] D. Kim, M. Chaudhuri, and M. Heinrich, "Leveraging Cache Coherence in Active Memory Systems," *Proc. 16th Int'l Conf. Supercomputing*, June 2002.
[6] D.M. Keen, "Novel Designs and Uses of Communication in Auxiliary Processing Systems," PhD thesis, Univ. of California Davis, 2002.
[7] K. Murakami, S. Shirakawa, and H. Miyajima, "Parallel Processing RAM Chip with 256Mb DRAM and Quad Processors," *Int'l Solid-State Circuits Conf. Digest of Technical Papers*, 1997.
[8] M. Oskin, F.T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," *Proc. 25th Ann. Int'l Symp. Computer Architecture (ISCA '98)*, 1998.
[9] M. Oskin, J. Hensley, D. Keen, F.T. Chong, M. Farrens, and A. Chopra, "Exploiting ILP in Page-Based Intelligent Memory," *Proc. 32nd Ann. Int'l Symp. Microarchitecture*, Nov. 1999.
[10] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "The Case for Intelligent RAM: IRAM," *IEEE Micro*, Apr. 1997.

Rate-Monotonic Scheduling on Uniform Multiprocessors

Sanjoy K. Baruah, *Member*, *IEEE*, and Joël Goossens

Abstract—The rate-monotonic algorithm is arguably one of the most popular algorithms for scheduling systems of periodic real-time tasks. The rate-monotonic scheduling of systems of periodic tasks on uniform multiprocessor platforms is considered here. A simple, sufficient test is presented for determining whether a given periodic task system will be successfully scheduled by this algorithm upon a particular uniform multiprocessor platform—this test generalizes earlier results concerning rate-monotonic scheduling upon identical multiprocessor platforms.

Index Terms—Uniform multiprocessors; periodic tasks; global scheduling; static priorities; rate-monotonic algorithm.

1 INTRODUCTION

IN the periodic model of hard real-time tasks, a **task** $\tau_i = (C_i, T_i)$ is characterized by two parameters—an execution requirement C_i and a period T_i—with the interpretation that the task generates a *job* at each integer multiple of T_i and each such job has an execution requirement of C_i execution units and must complete by a deadline equal to the next integer multiple of T_i. We assume that preemption is permitted—an executing job may be interrupted and its exeuction resumed later, with no loss or penalty. A periodic task system consists of several independent such periodic tasks that are to execute on a specified preemptive processor architecture. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For each task τ_i, define its *utilization* U_i to be the ratio of τ_i's execution requirement to its period: $U_i \overset{\text{def}}{=} C_i/T_i$. We define the *cumulative utilization* $U(\tau)$ of periodic task system τ (often, the word "cumulative" is dropped and this is called simply the *utilization* of τ) to be the sum of the utilizations of all tasks in τ: $U(\tau) \overset{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. Furthermore, we define the *maximum utilization* $U_{\max}(\tau)$ of periodic task system τ to be the largest utilization of any task in τ: $U_{\max}(\tau) \overset{\text{def}}{=} \max_{\tau_i \in \tau} U_i$.

Without loss of generality, we assume that $T_i \leq T_{i+1}$ for all i, $1 \leq i < n$, i.e., the tasks are indexed according to period.

Run-time scheduling is the process of determining, during the execution of a real-time application system, which job(s) should be executed at each instant in time. Runtime scheduling algorithms are typically implemented as follows: At each time instant, assign a **priority** to each active (informally, a job becomes *active* at its ready time and remains so until it has executed for an amount of time equal to its execution requirement or until its deadline has elapsed) job and allocate the available processors to the highest-priority jobs.

For systems comprised of periodic tasks that are to execute upon a *single shared processor*, a very popular runtime scheduling algorithm is the *rate-monotonic scheduling algorithm* (Algorithm RM) [12]. Algorithm RM assigns each task a priority inversely proportional to its period—the smaller the period, the higher the priority, with ties broken arbitrarily but in a consistent manner: If τ_i and τ_j have equal periods and τ_i's job is given priority over τ_j's job once, then all of τ_i's jobs are given priority over all of τ_j's jobs.

- *S.K. Baruah is with the Department of Computer Science, University of North Carolina, Campus Box 3175, Sitterson Hall, Chapel Hill, NC 27599-3175. E-mail: baruah@cs.unc.edu.*
- *J. Goossens is with the Département d'Informatique, Université Libre de Bruxelles, Boulevard du Triomphe-C.P. 212, 1050 Bruxelles, Belgium. E-mail: Joel.Goossens@ulb.ac.be.*

Manuscript received 21 Feb. 2002; revised 6 Nov. 2002; accepted 6 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115931.

ACKNOWLEDGMENTS

This work is supported in part by a US National Science Foundation (NSF) CAREER award to Fred Chong, by NSF grant CCR-9812415, by an NPSC fellowship to Diana Keen, by grants from Mitsubishi and Altera, and by grants from the University of California Davis Academic Senate. More information is available at http://arch.cs.ucdavis.edu/AP.

REFERENCES

[1] D. Burger and T. Austin,, "The SimpleScalar Tool Set, v2.0," *Computer Architecture News* Vol. 25, No. 3, June 1997.
[2] M. Gokhale, B. Holmes, and K. Lobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer,* vol. 28, no. 4, pp. 23-31, Apr. 1995.
[3] K. Itoh, Y. Nakagome, S. Kimura, and T. Watanabe, "Limitations and Challenges of Multigigabit DRAM Chip Design," *IEEE J. Solid-State Circuits,* vol. 32, no. 5, pp. 624-634, 1997.
[4] Intel, *Intel Architecture Software Developer's Manual,* 1997.
[5] D. Kim, M. Chaudhuri, and M. Heinrich, "Leveraging Cache Coherence in Active Memory Systems," *Proc. 16th Int'l Conf. Supercomputing,* June 2002.
[6] D.M. Keen, "Novel Designs and Uses of Communication in Auxiliary Processing Systems," PhD thesis, Univ. of California Davis, 2002.
[7] K. Murakami, S. Shirakawa, and H. Miyajima, "Parallel Processing RAM Chip with 256Mb DRAM and Quad Processors," *Int'l Solid-State Circuits Conf. Digest of Technical Papers,* 1997.
[8] M. Oskin, F.T. Chong, and T. Sherwood, "Active Pages: A Computation Model for Intelligent Memory," *Proc. 25th Ann. Int'l Symp. Computer Architecture (ISCA '98),* 1998.
[9] M. Oskin, J. Hensley, D. Keen, F.T. Chong, M. Farrens, and A. Chopra, "Exploiting ILP in Page-Based Intelligent Memory," *Proc. 32nd Ann. Int'l Symp. Microarchitecture,* Nov. 1999.
[10] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "The Case for Intelligent RAM: IRAM," *IEEE Micro,* Apr. 1997.

Rate-Monotonic Scheduling on Uniform Multiprocessors

Sanjoy K. Baruah, *Member*, *IEEE*, and Joël Goossens

Abstract—The rate-monotonic algorithm is arguably one of the most popular algorithms for scheduling systems of periodic real-time tasks. The rate-monotonic scheduling of systems of periodic tasks on uniform multiprocessor platforms is considered here. A simple, sufficient test is presented for determining whether a given periodic task system will be successfully scheduled by this algorithm upon a particular uniform multiprocessor platform—this test generalizes earlier results concerning rate-monotonic scheduling upon identical multiprocessor platforms.

Index Terms—Uniform multiprocessors; periodic tasks; global scheduling; static priorities; rate-monotonic algorithm.

1 INTRODUCTION

IN the periodic model of hard real-time tasks, a **task** $\tau_i = (C_i, T_i)$ is characterized by two parameters—an execution requirement C_i and a period T_i—with the interpretation that the task generates a *job* at each integer multiple of T_i and each such job has an execution requirement of C_i execution units and must complete by a deadline equal to the next integer multiple of T_i. We assume that preemption is permitted—an executing job may be interrupted and its exeuction resumed later, with no loss or penalty. A periodic task system consists of several independent such periodic tasks that are to execute on a specified preemptive processor architecture. Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ denote a periodic task system. For each task τ_i, define its *utilization* U_i to be the ratio of τ_i's execution requirement to its period: $U_i \stackrel{\text{def}}{=} C_i/T_i$. We define the *cumulative utilization* $U(\tau)$ of periodic task system τ (often, the word "cumulative" is dropped and this is called simply the *utilization* of τ) to be the sum of the utilizations of all tasks in τ: $U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$. Furthermore, we define the *maximum utilization* $U_{\max}(\tau)$ of periodic task system τ to be the largest utilization of any task in τ: $U_{\max}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} U_i$.

Without loss of generality, we assume that $T_i \leq T_{i+1}$ for all i, $1 \leq i < n$, i.e., the tasks are indexed according to period.

Run-time scheduling is the process of determining, during the execution of a real-time application system, which job(s) should be executed at each instant in time. Runtime scheduling algorithms are typically implemented as follows: At each time instant, assign a **priority** to each active (informally, a job becomes *active* at its ready time and remains so until it has executed for an amount of time equal to its execution requirement or until its deadline has elapsed) job and allocate the available processors to the highest-priority jobs.

For systems comprised of periodic tasks that are to execute upon a *single shared processor*, a very popular runtime scheduling algorithm is the *rate-monotonic scheduling algorithm* (Algorithm RM) [12]. Algorithm RM assigns each task a priority inversely proportional to its period—the smaller the period, the higher the priority, with ties broken arbitrarily but in a consistent manner: If τ_i and τ_j have equal periods and τ_i's job is given priority over τ_j's job once, then all of τ_i's jobs are given priority over all of τ_j's jobs.

• *S.K. Baruah is with the Department of Computer Science, University of North Carolina, Campus Box 3175, Sitterson Hall, Chapel Hill, NC 27599-3175. E-mail: baruah@cs.unc.edu.*
• *J. Goossens is with the Département d'Informatique, Université Libre de Bruxelles, Boulevard du Triomphe-C.P. 212, 1050 Bruxelles, Belgium. E-mail: Joel.Goossens@ulb.ac.be.*

Manuscript received 21 Feb. 2002; revised 6 Nov. 2002; accepted 6 Jan. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 115931.

In [1], the rate-monotonic scheduling of periodic real-time task systems upon a more general machine model than the uniprocessor model was studied. In *identical multiprocessors*, there are several identical processors upon which the jobs generated by the periodic tasks execute. As in the uniprocessor case, Algorithm RM upon identical multiprocessors assigns tasks priorities in inverse proportion to their periods and selects for execution at each instant in time the (jobs of) the highest-priority tasks that have jobs awaiting execution. In this paper, we study the scheduling of periodic task systems upon an even more general machine model: the **uniform multiprocessor** model. In contrast to identical multiprocessors (in which all processors are assumed to be equally powerful), each processor in a uniform multiprocessor machine is characterized by a *speed* or computing capacity, with the interpretation that a job executing on a processor with speed s for t time units completes $(s \times t)$ units of execution. We consider the preemptive *global* scheduling of hard-real-time systems on uniform multiprocessor platforms. By **global** scheduling, we mean that individual tasks are not associated to specific processors; rather the *interprocessor migration* of individual jobs is permitted (i.e., a preempted job may resume execution or a different processor with no cost or penalty).[1] In many systems, this may not be valid in that interprocessor migration incurs a cost since the runtime state of the executing task must also be migrated to the destination processor. Nevertheless, we can easily *bound* from above (as in [2], [3]) the maximum number of such interprocessor migrations that any individual job will have to undergo; the total cost of all such migrations can be amortized among the individual jobs by "charging" each job for a certain number of such migrations (i.e., by inflating each job's execution requirement by an appropriate amount). In this manner, we abstract away the effects of interprocessor migrations from our formal model.

2 Model and Background

In Section 2.1 below, we 1) introduce some definitions and formal notation concerning scheduling upon uniform multiprocessor platforms and 2) present, without proof, previously published results from elsewhere that will be used in this paper. In Section 2.2, we place the research presented here in its proper context with respect to prior research in real-time and multiprocessor scheduling theory.

2.1 Formal Model

We start out by providing a formal definition of *uniform multiprocessors*.

Definition 1. *Let π denote a* **uniform multiprocessor platform**.

- *The number of processors comprising π is denoted by $m(\pi)$.*
- *For all i, $1 \le i \le m(\pi)$, the speed (the computing capacity) of the ith-fastest processor in π is denoted by $s_i(\pi)$.*
- *The total computing capacity of all the processors in π is denoted by $S(\pi)$: $S(\pi) \stackrel{\text{def}}{=} \sum_{i=1}^{m(\pi)} s_i(\pi)$.*

Definition 2 (Greedy scheduling algorithm). *A uniform multiprocessor scheduling algorithm is said to be greedy if it satisfies all three of the following conditions:*

1. *It never idles a processor when there are jobs awaiting execution.*
2. *If it must idle some processor (because fewer active jobs are available than processors), then it idles the slowest processors. That is, if it is the case that, at some instant t the jth-slowest processor is idled, then the kth-slowest processor is also idled at that instant t, for all $k > j$.*
3. *It always executes higher-priority jobs upon faster processors. That is, if the jth-slowest processor is executing job J at time t, it must be the case that the priority of J is no smaller than the priorities of the jobs (if any) executing on the kth slowest processor, for all $k > j$.*

In the remainder of this paper, we will assume that Algorithm RM is implemented in a greedy manner upon uniform multiprocessors.

We now define two important additional parameters of uniform multiprocessor platforms; as we will see later in this paper, these parameters succinctly capture the characteristics of uniform multiprocessors that are particularly relevant in determining whether a task system is successfully scheduled upon the platform by Algorithm RM.

Definition 3 (λ and μ). *For any uniform multiprocessor platform π, we define a parameter $\underline{\lambda(\pi)}$ as follows:*

$$\lambda(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i+1}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\}. \tag{1}$$

For any uniform multiprocessor platform π, we define a parameter $\underline{\mu(\pi)}$ as follows:

$$\mu(\pi) \stackrel{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\}. \tag{2}$$

These parameters $\lambda(\pi)$ and $\mu(\pi)$ of a uniform multiprocessor system π intuitively measure the "degree" by which π differs from an identical multiprocessor platform (in the sense that $\lambda(\pi) = (m(\pi) - 1)$ and $\mu(\pi) = m(\pi)$ if π is comprised of identical processors and both parameters become progressively smaller as the speeds of the processors differ from each other by greater amounts; in the extreme, we would have $\lambda(\pi) = 0$ and $\mu(\pi) = 1$). Although $\lambda(\pi)$ and $\mu(\pi)$ are very similar to each other (and can, in fact, be represented in terms of each other), we find it more convenient to define them both separately.

Real-time job instances. At times, we find it convenient to represent a real-time system using a model that is somewhat more general than the periodic task model. We will then assume that a hard-real-time **instance** is modeled as a collection of independent jobs. Each **job** $J_j = (r_j, c_j, d_j)$ is characterized by an arrival time r_j, an execution requirement c_j, and a deadline d_j, with the interpretation that this job needs to execute for c_j units over the interval $[r_j, d_j)$. (Thus, the periodic task $\tau_i = (C_i, T_i)$ generates an infinite sequence of jobs with parameters $(k \cdot T_i, C_i, (k+1) \cdot T_i)$, $k = 0, 1, 2, \ldots$; in the remainder of this paper, we will sometimes use the symbol τ itself to denote the infinite set of jobs generated by the tasks in periodic task system τ.)

Definition 4 $\mathbf{W}(\mathbf{A}, \pi, \mathbf{I}, \mathbf{t})$). *Let I denote any collection of jobs and π any uniform multiprocessor platform. For any algorithm A and time instant $t \ge 0$, let $W(A, \pi, I, t)$ denote the amount of work done by algorithm A on jobs of I over the interval $[0, t)$, while executing on π.*

The following theorem was proven in [10]; we will be using it later in this paper.

Theorem 1 (From [10]). *Let π_o and π denote uniform multiprocessor platforms. Let A_o denote any uniform multiprocessor scheduling algorithm and A any greedy uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms π_o and π:*

1. By contrast, **partitioned scheduling** partitions the set of tasks among the various processors and requires that all the jobs generated by a task execute upon its assigned processor. There are advantages and disadvantages to both partitioned and global scheduling—for a discussion of the trade off, see Section 2.2 and consult [2], [3].

$$S(\pi) \geq S(\pi_o) + \lambda(\pi) \cdot s_1(\pi_o), \tag{3}$$

then, for any collection of jobs I and any time-instant $t \geq 0$,

$$W(A, \pi, I, t) \geq W(A_o, \pi_o, I, t). \tag{4}$$

2.2 Context for This Research

We now place this work in the proper context with respect to related research. In order to do so, we need to elaborate upon several key concepts.

Dynamic and static priorities. As stated in the introduction, runtime scheduling algorithms typically assign a priority to each active job at each time instant and allocate the available processors to the highest-priority jobs. With respect to certain runtime scheduling algorithms, it is possible that some tasks τ_i and τ_j both have active jobs at times t_1 and t_2 such that, at time t_1, τ_i's job has higher priority than τ_j's, while, at time t_2, τ_j's job has higher priority than τ_i's. Runtime scheduling algorithms that permit such "switching" of the order of priorities between tasks are known as **dynamic** priority algorithms. The earliest deadline first scheduling algorithm (Algorithm EDF) [12], [8] is an example of a dynamic priority algorithm. By contrast, **static** priority algorithms (such as Algorithm RM, studied in this paper) must satisfy the constraint that, for every pair of tasks τ_i and τ_j, whenever τ_i and τ_j both have active jobs, it is always the case that the same task's jobs have higher priority.

Partitioned and global scheduling. In designing scheduling algorithms for multiprocessor environments, one can distinguish between at least two distinct approaches. In **partitioned** scheduling, all jobs generated by a task are required to execute on the *same* processor. **Global scheduling**, by contrast, permits *task migration* (i.e., different jobs of an individual task may execute upon different processors) as well as job-migration: An individual job that is preempted may resume execution upon a processor different from the one upon which it had been executing prior to preemption.

It has been proven by Leung and Whitehead [11] that the partitioned and global approaches to static-priority scheduling on identical multiprocessors are *incomparable* in the sense that 1) there are task systems that are feasible on m identical processors under the partitioned approach but for which no priority assignment exists which would cause all jobs of all tasks to meet their deadlines under global scheduling on the same m processors and 2) there are task systems that are feasible on m identical processors under the global approach, but which cannot be partitioned into m distinct subsets such that each individual partition is uniprocessor static-priority feasible. This result of Leung and Whitehead [11] provides a very strong motivation to study both the partitioned and the nonpartitioned approaches to static-priority multiprocessor (identical as well as uniform) scheduling since it is provably true that neither approach is strictly better than the other.

Related work. Since the seminal paper by Liu and Layland in 1973 [12], a tremendous amount of excellent research has been done on static-priority scheduling upon uniprocessors—for a comprehensive survey, see [4].

Real-time scheduling upon uniform multiprocessors is a relatively new area of research. In [5], the dynamic-priority scheduling of periodic task systems upon uniform multiprocessors subject to certain preemption constraints (formalized as the *integer boundary constraint*) was studied: The general problem was proven intractable. In [10], online scheduling of collections of independent jobs upon uniform multiprocessors was considered and sufficient conditions obtained for determining whether an instance of jobs known to be feasible upon a particular uniform multiprocessor platform would be EDF-feasible upon a different uniform multiprocessor platform (where "EDF" denotes the Earliest Deadline First scheduling algorithm). To our knowledge, there has been no prior work done on static-priority scheduling upon uniform multiprocessors The partioned approach to static-priority scheduling upon has been extensively studied [9], [6], [7], [13] for identical multiprocessors. Much of this research has adopted the approach of using bin-packing-like algorithms for partitioning a given set of periodic tasks among a set of processors such that each partition is uniprocessor feasible under the rate-monotonic algorithm, e.g., Davari and Dhall [6] presented a polynomial-time algorithm which they proved would partition a set of periodic tasks into no more than twice as many partitions as an optimal algorithm would (equivalently, they devised a polynomial-time algorithm for partitioned static-priority multiprocessor scheduling that uses at most twice as many processors as an optimal algorithm would). More recently, Oh and Baker [13] presented a partitioned static-priority multiprocessor scheduling algorithm that schedules any task system with utilization $\leq m(\sqrt{2}-1)$ on m processors—this represents a utilization of approximately 42 percent of the capacity of the multiprocessor platform. We expect that these results can be generalized to uniform multiprocessors by applying techniques that have been developed for the bin-packing problem when bin-sizes are not the same.

The global approach to the static-priority scheduling of periodic task systems upon identical multiprocessors was studied in [1] and conditions were derived for determining whether a given periodic task system will be scheduled to meet all deadlines by Algorithm RM upon m unit-capacity processors. It was shown that any periodic task system in which each task's utilization is no more than one-third and the sum of the utilizations of all the tasks is no more than $m/3$ is successfully scheduled by Algorithm RM upon m unit-capacity processors—to our knowledge, this is the first (and so far, only) nontrivial utilization-based static-priority feasibility condition for the global scheduling of periodic real-time tasks upon multiprocessors.

3 RM-Feasibility upon Uniform Processors

For any scheduling algorithm A and any processor platform π, we say that τ is **A-feasible** upon π if τ meets all deadlines when scheduled upon π by algorithm A. We say τ is **feasible** upon π if it can be scheduled to meet all deadlines upon π by an optimal algorithm. In this section, we obtain sufficient conditions for determining whether any periodic task system is RM-feasible upon any given uniform multiprocessor platform.

Consider a periodic task system τ and a uniform multiprocessor platform π and suppose that τ and π satisfy the following relationship:

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau). \tag{5}$$

Let us consider the RM-scheduling of τ upon π.

Without loss of generality, assume that ties are broken by Algorithm RM such that τ_i has greater priority than τ_{i+1} for all i, $1 \leq i < n$. Notice that whether jobs of τ_k meet their deadlines under Algorithm RM depends only upon the jobs generated by the tasks $\{\tau_1, \tau_2, \ldots, \tau_k\}$ and are completely unaffected by the presence of the tasks $\tau_{k+1}, \ldots, \tau_n$. For $k = 1, 2, \ldots, n$, let us define the task-set $\tau^{(k)}$ as follows:

$$\tau^{(k)} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_k\}.$$

Lemma 1. *Task system $\tau^{(k)}$ is feasible on a uniform multiprocessor platform π_o satisfying the conditions:*

1. $S(\pi_o) = U(\tau^{(k)})$ *and*
2. $s_1(\pi_o) = U_{\max}(\tau^{(k)})$.

Proof. Set π_o equal to a uniform multiprocessor consisting of k processors, with computing capacities equal to $C_1/T_1, C_2/T_2, \ldots, C_k/T_k$, respectively. It is easy to see that $\tau^{(k)}$

is feasible upon this π_o: an optimal scheduling algorithm OPT would simply schedule each task exclusively upon the processor that has computing capacity equal to that task's utilization. □

From Condition 3 and Lemma 1 above and the fact that Algorithm RM is greedy, it follows that, if parameters $S(\pi)$ and $\lambda(\pi)$ of uniform multiprocessor platform π satisfy

$$S(\pi) \geq U(\tau^{(k)}) + \lambda(\pi) \cdot U_{\max}(\tau^{(k)}), \tag{6}$$

then it will be true that

$$\begin{aligned} & W(\mathrm{RM}, \pi, \tau^{(k)}, t) \geq W(\mathrm{OPT}, \pi_o, \tau^{(k)}, t) \\ \equiv & \left(W(\mathrm{RM}, \pi, \tau^{(k)}, t) \geq t \cdot \left(\sum_{j=1}^{k} U_j \right) \right), \end{aligned} \tag{7}$$

where π_o and OPT are as described in the proof of Lemma 1.

Recall that we are assuming that π and τ satisfy Condition 5. Since

$$\left(2 \cdot U(\tau) \geq 2 \cdot U(\tau^{(k)}) \geq U(\tau^{(k)})\right) \text{ and } (\mu(\pi) \geq \lambda(\pi)),$$

we may conclude that

$$\begin{aligned} & S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \quad \text{(By Condition 5)} \\ & \Rightarrow S(\pi) \geq U(\tau^{(k)}) + \lambda(\pi) \cdot U_{\max}(\tau^{(k)}). \end{aligned}$$

Hence, (7) is seen to be true for all $t \geq 0$, i.e., *at any time-instant t, the amount of work done on $\tau^{(k)}$ by Algorithm RM executing on π is at least as much as the amount of work done on $\tau^{(k)}$ by OPT on π_o.*

Lemma 2. *All jobs of τ_k meet their deadlines when $\tau^{(k)}$ is scheduled upon π using Algorithm RM.*

Proof. Let us assume that the first $(\ell - 1)$ jobs of τ_k have met their deadlines under Algorithm RM; we will prove below that the ℓth job of τ_k also meets its deadline. The correctness of Lemma 2 will then follow by induction on ℓ, starting with $\ell = 1$.

The ℓth job of τ_k arrives at time-instant $(\ell - 1)T_k$, has a deadline at time-instant ℓT_k, and needs C_k units of execution. From (7), we have

$$W(\mathrm{RM}, \pi, \tau^{(k)}, (\ell - 1)T_k) \geq (\ell - 1)T_k \left(\sum_{j=1}^{k} U_j \right) \tag{8}$$

Also, at least $(\ell - 1) \cdot T_k \cdot (\sum_{j=1}^{k-1} U_j)$ units of this execution by Algorithm RM were of tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$—this follows from the fact that exactly $(\ell - 1)T_k U_k$ units of τ_k's work have been generated prior to instant $(\ell - 1)T_k$; the remainder of the work executed by Algorithm RM must therefore be generated by $\tau_1, \tau_2, \ldots, \tau_{k-1}$.

The cumulative execution requirement of all the jobs generated by the tasks $\tau_1, \tau_2, \ldots, \tau_{k-1}$ that arrive prior to the deadline of τ_k's ℓth job is bounded from above by

$$\begin{aligned} & \sum_{j=1}^{k-1} \left\lceil \frac{\ell T_k}{T_j} \right\rceil C_j \\ & < \sum_{j=1}^{k-1} \left(\frac{\ell T_k}{T_j} + 1 \right) C_j \\ & = \ell T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j. \end{aligned}$$

As we have seen above (the discussion following (8)), at least $(\ell - 1) \cdot T_k \cdot \sum_{j=1}^{k-1} U_j$ of this work gets done prior to time-instant $(\ell - 1)T_k$; hence, at most

$$\left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \tag{9}$$

remains to be executed *after* time-instant $(\ell - 1)T_k$.

The amount of processor capacity left unused by $\tau_1, \ldots, \tau_{k-1}$ during the interval $[(\ell - 1)T_k, \ell T_k)$ is therefore no smaller than

$$S(\pi) \cdot T_k - \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right). \tag{10}$$

Observe that *not all of this capacity is available* to τ_k's ℓth job; in particular, at any instant at which several processors are simultaneously available, τ_k can execute upon only one of these processors and the excess capacity upon the remaining processors is wasted. So, how much of this processor capacity identified in (10) above is guaranteed to be available to τ_k? To determine this, observe that, since Algorithm RM is a greedy scheduling algorithm if several processors are simultaneously available, then τ_k will execute upon the *fastest* available processor. Suppose that τ_k is executing upon the *j*th-fastest processor at some instant in time (for some j, $1 \leq j \leq m(\pi)$), then the $(j+1)$th, $(j+2)$th, ..., *m*th-fastest processors may all be idled at this instant, but all processors that are faster than the *j*th faster processor are busy and do not contribute to the excess capacity computed above. That is, the fraction of excess capacity that is used at this instant is at least $(s_j(\pi)/[s_j(\pi) + s_{j+1}(\pi) + \cdots + s_{m(\pi)}(\pi)])$. By definition of the parameter $\mu(\pi)$ (Definition 3), this is $\geq \frac{1}{\mu(\pi)}$. Therefore, a lower bound on the amount of execution available to τ_k's ℓth job over $[(\ell - 1)T_k, \ell T_k)$ is given by

$$\frac{1}{\mu(\pi)} \cdot \left[S(\pi) \cdot T_k - \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \right].$$

For the ℓth job of τ_k to meet its deadline, it suffices that this amount be at least as large at τ_k's execution requirement, i.e.,

$$\frac{1}{\mu(\pi)} \cdot \left[S(\pi) \cdot T_k - \left(T_k \sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j \right) \right] \geq C_k,$$

which is true by definition of $U_{\max}(\tau)$:

$$\begin{aligned} & U_{\max}(\tau) \geq U_k \\ \equiv & \frac{1}{\mu(\pi)} \cdot [U_{\max}(\tau) \cdot \mu(\pi)] \geq U_k \\ & (\textit{By Condition } 5 \textit{ and the fact that } \sum_{j=1}^{k-1} U_j \leq U(\tau)) \Rightarrow \\ & \frac{1}{\mu(\pi)} \cdot \left[S(\pi) - 2 \cdot \sum_{j=1}^{k-1} U_j \right] \geq U_k \\ & (\textit{Since } T_k \geq T_j \textit{ for } j < k) \Rightarrow \\ & \frac{1}{\mu(\pi)} \cdot \left[S(\pi) - \left(\frac{\sum_{j=1}^{k-1} U_j + \sum_{j=1}^{k-1} C_j}{T_k} \right) \right] \geq U_k, \end{aligned}$$

and, hence, proves the property. □

Theorem 2. *Given a periodic task system τ and a uniform multiprocessor platform π,*

$$S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau)$$

is a sufficient condition for ensuring that τ is RM-feasible upon π.

Proof. When τ and π satisfy the condition of the theorem (which is exactly Condition 5), we have shown (in Lemma 2 above) that Algorithm RM schedules $\tau^{(k)}$ in such a manner that all jobs of the lowest-priority task τ_k complete by their deadlines. The correctness of Theorem 2 immediately follows, by induction on k. □

We can apply Theorem 2 to identical multiprocessors to obtain a result similar to a result in [1]:

Corollary 1. *Any periodic task system in which each task's utilization is no more than one-third and the sum of the utilizations of all the tasks is no more than $m/3$ is successfully scheduled by Algorithm RM upon m unit-capacity processors.*

Proof. Let τ denote the periodic task system: by the antecedents of the corollary, $U(\tau) \leq m/3$ and $U_{\max}(\tau) \leq 1/3$. Let π denote an m-processor identical multiprocessor platform comprised of unit-capacity processors; by definition,

$$\mu(\pi) \overset{\text{def}}{=} \max_{i=1}^{m(\pi)} \left\{ \frac{\sum_{j=i}^{m(\pi)} s_j(\pi)}{s_i(\pi)} \right\} = \left\{ \frac{m}{1} \right\} = m.$$

By Theorem 2, τ is RM-feasible upon π if

$$\begin{aligned} & S(\pi) \geq 2 \cdot U(\tau) + \mu(\pi) \cdot U_{\max}(\tau) \\ & \equiv \left(m \geq 2 \cdot \frac{m}{3} + m \cdot \frac{1}{3} \right) \\ & \equiv (m \geq m), \end{aligned}$$

which is true. □

4 Conclusions

Our contribution in this paper can be summarized as follows: *We have obtained here the first nontrivial feasibility test for a static-priority scheduling algorithm that adopts a global approach to task allocation upon uniform multiprocessors.* That is, we have studied the behavior of Algorithm RM—one such previously defined [12], [1] static-priority global scheduling algorithm—upon uniform multiprocessor platforms. We have obtained simple sufficient conditions for determining whether any given periodic task system will be succesfully scheduled by Algorithm RM upon a given uniform multiprocessor platform.

Acknowledgments

This work was supported in part by the US National Science Foundation (Grant Nos. CCR-9988327, ITR-0082866, and CCR-0204312).

References

[1] B. Andersson, S. Baruah, and J. Jonsson, "Static-Priority Scheduling on Multiprocessors," *Proc IEEE Real-Time Systems Symp.*, pp. 193-202, Dec. 2001.

[2] B. Andersson and J. Jonsson, "Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or Not to Partition," *Proc. Int'l Conf. Real-Time Computing Systems and Applications*, pp. 337-346, Dec. 2000.

[3] B. Andersson and J. Jonsson, "Some Insights on Fixed-Priority Preemptive Non-Partitioned Multiprocessor Scheduling," *Proc. Real-Time Systems Symp.—Work-In-Progress Session*, Nov. 2000.

[4] N.C. Audsley, A. Burns, R.I. Davis, K.W. Tindell, and A.J. Wellings, "Fixed Priority Preemptive Scheduling: An Historical Perspective," *Proc. Real-Time Systems 8*, pp. 173-198, 1995.

[5] S. Baruah, "Scheduling Periodic Tasks on Uniform Processors," *Proc. Euromicro Workshop Real-time Systems*, pp. 7-14, June 2000.

[6] S. Davari and S.K. Dhall, "On a Real-Time Task Allocation Problem," *Proc. 19th Hawaii Int'l Conf. System Science*, Jan. 1985.

[7] S. Davari and S.K. Dhall, "An On-Line Algorithm for Real-Time Tasks Allocation," *Proc. Real-Time Systems Symp.*, pp. 194-200, 1986.

[8] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processors," *Proc. IFIP Congress*, pp. 807-813, 1974.

[9] S.K. Dhall and C.L. Liu, "On a Real-Time Scheduling Problem," *Operations Research*, vol. 26, pp. 127-140, 1978.

[10] S. Funk, J. Goossens, and S. Baruah, "On-Line Scheduling on Uniform Multiprocessors," *Proc. IEEE Real-Time Systems Symp.*, pp. 183-192, Dec. 2001.

[11] J. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks," *Performance Evaluation*, vol. 2, pp. 237-250, 1982.

[12] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[13] D.I. Oh and T.P. Baker, "Utilization Bounds for N-Processor Rate Monotone Scheduling with Static Processor Assignment," *Real-Time Systems: The Int'l J. Time-Critical Computing*, vol. 15, pp. 183-192, 1998.