On-line Scheduling on Uniform Multiprocessors*

Shelby Funk† Joël Goossens† Sanjoy Baruah†

†University of North Carolina
Chapel Hill, North Carolina
{funk,baruah}@cs.unc.edu

‡Université Libre de Bruxelles
Brussels, Belgium
Joel.Goossens@ulb.ac.be

Abstract

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. The on-line scheduling of hard-real-time systems, in which all jobs must complete by specified deadlines, on uniform multiprocessor machines is considered. It is known that on-line algorithms tend to perform very poorly in scheduling such hard-real-time systems on multiprocessors; resource-augmentation techniques are presented here that permit on-line algorithms to perform better than may be expected given the inherent limitations. Results derived here are applied to the scheduling of periodic task systems on uniform multiprocessor machines.

1. Introduction

In hard-real-time systems, there are certain basic units of work, known as jobs, which must be executed in a timely manner. In one popular model of hard-real-time systems, each job is assumed to be characterized by three parameters – an arrival time, an execution requirement, and a deadline, with the interpretation that the job must be executed for an amount equal to its execution requirement between its arrival time and its deadline.

Multiprocessor Machines. The scheduling of hard-real-time systems has been much studied, particularly upon uniprocessor platforms — upon machines in which there is exactly one shared processor available, and all the jobs in the system are required to execute on this single shared processor.

In multiprocessor platforms there are several processors available upon which these jobs may execute. In this paper, we will be studying the scheduling of hard-real-time systems on multiprocessor platforms, under the following assumptions:

Job preemption is permitted. That is, a job executing on a processor may be preempted prior to completing execution, and its execution may be resumed later. We assume that there is no penalty associated with such preemption.

Job migration is permitted. That is, a job that has been preempted on a particular processor may resume execution on the same or a different processor. Once again, we assume that there is no penalty associated with such migration.

Job parallelism is forbidden. That is, each job may execute on at most one processor at any given instant in time.

A taxonomy of multiprocessor platforms. In much previous work concerning hard-real-time scheduling on multiprocessors, it has been assumed that all processors are identical. However, scheduling theorists distinguish between at least three different kinds of multiprocessor machines:

Identical parallel machines: These are multiprocessors in which all the processors are identical, in the sense that they have the same computing power.

Uniform parallel machines: By contrast, each processor in a uniform parallel machine is characterized by its own computing capacity, with the interpretation that a job that executes on a processor of computing capacity $s$ for $t$ time units completes $s \times t$ units of execution. (Observe that identical parallel machines are a special case of uniform parallel machines, in which the computing capacities of all processors are equal.)

*Supported in part by the National Science Foundation (Grant Nos. CCR-9704206, CCR-9972105, CCR-9988327, and ITR-0082866).
**Unrelated parallel machines:** In unrelated parallel machines, there is an execution rate \( r_{i,j} \) associated with each job-processor ordered pair \((J_i, P_j)\), with the interpretation that job \( J_i \) completes \((r_{i,j} \times t)\) units of execution by executing on processor \( P_j \) for \( t \) time units.

Real-time scheduling theorists have extensively studied uniprocessor hard-real-time scheduling; recently, steps have been taken towards obtaining a better understanding of hard-real-time scheduling on identical multiprocessors (see, e.g., [4, 1, 12, 3, 2]). However, not much is known about hard-real-time scheduling on uniform or unrelated multiprocessors. While it can be (and often is) argued that the unrelated parallel machines model is a theoretical abstraction of little significance to the designers of real-time systems, we believe that the uniform parallel machines model is a very relevant one for modelling many actual application systems. There are several reasons for this:

- The existence of this model gives application system designers the freedom to use processors of different speeds, rather than constraining them to always use identical processors. In fact, uniform multiprocessor platforms are already commercially available — for instance the Compaq AlphaServer GS series (specifically the series GS 160 & GS 320 — see, e.g., [6]) supports mixed processor speeds with up to 32 mixed processors.

- Even when all the processors available are identical, they may not all be exclusively available for the execution of the real-time periodic tasks, but may be required to devote a certain fraction of their computing capacity to some other (non real-time) tasks. Each such processor can be modelled by another of lower computing capacity, with this computing capacity indicative of the fraction of its actual computing capacity that can be devoted to periodic tasks.

- As new and faster processors become available, one may choose to improve the performance of a system by upgrading some of its processors. If the only model we have available is the identical multiprocessors model, we must necessarily replace all the processors simultaneously. With the uniform parallel machines model, we can however choose to replace just a few of the processors, or indeed simply add some faster processors while retaining all the previous processors.

**On-line scheduling on multiprocessors.** On-line scheduling algorithms make scheduling decisions at each time-instant based upon the characteristics of the jobs that have arrived thus far, with no knowledge of jobs that may arrive in the future. Several uniprocessor on-line scheduling algorithms, such as the earliest deadline first scheduling algorithm (EDF) [11, 5] and the Least Laxity algorithm [14] are known to be optimal in the sense that if a set of jobs can be scheduled such that all jobs complete by their deadlines, then these algorithms will also schedule these sets of jobs to meet all deadlines. For multiprocessor systems, however, no on-line scheduling algorithm can be optimal: this was shown for the simplest (identical) multiprocessor model by Hong and Leung [8], and the techniques from [8] can be directly extended to the more general (uniform and unrelated) machine models.

An important advance in the field of on-line scheduling on multiprocessors was obtained by Phillips, Stein, Torng, and Wein [1], who explored the use of resource-augmentation techniques for the on-line scheduling of real-time jobs\(^1\). Phillips et al. investigated whether an on-line algorithm, if provided with faster processors than necessary for feasibility, could perform better than is implied by the above non-optimality results. In [1], they showed that the obvious extension to the Earliest Deadline First algorithm for identical multiprocessors could make the following performance guarantee

If a set of jobs is feasible on a \( m \) identical processors, then the same set of jobs will be scheduled to meet all deadlines by EDF on a \( m \) identical processors in which the individual processors are \((2 - \frac{1}{m})\) times as fast as in the original system.

That is, EDF is an on-line algorithm that can guarantee to successfully schedule on \( m \) identical processors any set of jobs that are feasible on \( m \) processors \( m/(2m-1) \) times as fast as those available to EDF.

**This research.** Our contributions, described in this paper, are twofold. First, we explore the issue of on-line scheduling of hard-real-time systems on uniform multiprocessors. Given the specifications of a uniform multiprocessor platform \( \pi \), we apply resource-augmentation techniques to obtain a condition (Theorem 1) upon the specifications of any another uniform multiprocessor platform \( \pi' \) such that if \( \pi' \) satisfies this condition, then any hard-real-time task system feasible on \( \pi \) will meet all deadlines when scheduled on \( \pi' \) using EDF.

One of the benefits of having provably optimal on-line scheduling algorithms (such as EDF, in the uniprocessor case) available is that they permit a separation of the concerns of feasibility-analysis and run-time scheduling during the design of real-time application systems — provided that the system designer can guarantee that a proposed design is feasible, the optimal on-line scheduling algorithm

---

\(^1\)Resource augmentation as a technique for improving the performance on on-line scheduling algorithms was formally proposed by Kalyanasundaram and Pruhs [10].
can be used as a run-time scheduling algorithm which will guarantee to meet all deadlines. In such frameworks, run-time scheduling can be looked upon as a kind of “service” that is provided by a real-time platform to an application system, subject to the following contract – if the application system guarantees to only generate feasible collections of jobs, the service-provider (i.e., the real-time platform) guarantees to meet all deadlines. We believe that our results should be viewed in the same context, and therein lies their significance: Provided that an application system only generates collections of jobs that are feasible on a particular platform $\pi$, our results permit us to derive the specifications of another system $\pi'$ such that the run-time platform $\pi'$ will meet all deadlines if the EDF on-line scheduling algorithm is used. Hence once again the system-designer can focus upon designing feasible systems (although the platform upon which feasibility must be guaranteed is a “virtual” one $\pi$, and not the actual run-time platform $\pi'$ upon which the system will actually execute). Given the result of Hong and Leung [8], this is the best kind of result we can hope to obtain, since no uniform multiprocessor on-line algorithm for scheduling hard-real-time systems can possibly be optimal. Our second major contribution in this paper concerns the problem of scheduling periodic task systems [11] upon uniform multiprocessors:

- We extend some previous results of Horvath et al. [9] to obtain an exact (i.e., necessary and sufficient) test for determining whether a given periodic task system is feasible on a specified uniform multiprocessor machine.

- We apply this exact feasibility test, and our result concerning on-line scheduling (Theorem 1), to derive a sufficient condition for a periodic task system to successfully meet all deadlines when scheduled using EDF.

That is, given the specifications of a periodic task system and a platform upon which it is to execute, we can determine whether it is (i) feasible (by any run-time algorithm) upon this platform, and (ii) EDF-feasible – i.e., whether it can be successfully scheduled to meet all deadlines by EDF – upon this platform.

Organization of this document. The remainder of this paper is organized as follows. In Section 2 we describe our system, our task/job model, and we define our EDF implementation for uniform multiprocessor platforms. In Section 3, we present (and prove the correctness of) our resource-augmentation technique for on-line scheduling of real-time jobs using EDF. In Section 4, we apply this technique to the scheduling of periodic task systems on uniform multiprocessor platforms. Concluding remarks are provided in Section 5.

2. Definitions and assumptions

Throughout this paper, we will consider the scheduling of hard-real-time systems upon a uniform multiprocessor platform comprised of $m$ processors. Each processor is characterized by a single parameter denoting the speed or computing capacity of the processor. We use the notation $\pi = [s_1, s_2, \ldots, s_m]$ to represent the $m$-processor uniform multiprocessor platform in which the processors have computing capacities $s_1, s_2, \ldots, s_m$ respectively; without loss of generality, we assume that these speeds are indexed in a non-increasing manner: $s_j \geq s_{j+1}$ for all $j, 1 \leq j < m$. We assume that all speeds are positive and that there is at least one processor — i.e., $s_i > 0$ for all $i$ and $m \geq 1$.

We will assume that a hard-real-time system may be modeled as an arbitrary collection of individual 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 $[r_j, d_j]$.

Periodic tasks. The periodic task model [11] has proven very useful for the modelling and analysis of real-time computer application systems. In this model, each recurring real-time process is modelled as a periodic task, and is characterized by two parameters – an execution requirement and a period. (While the execution time may be any non-negative number, deadlines are assumed here to be non-negative rational numbers.) In this section, we will study the scheduling of real-time systems that can be completely modelled as finite collections of such periodic tasks. Accordingly, we will model a real-time system $\tau = \{T_1, T_2, \ldots, T_n\}$ as being comprised of a collection of $n$ periodic tasks. We will assume that all the system parameters – the number of tasks in the system, and the execution requirement and period parameters of each task – are a priori known. Each periodic task generates an infinite sequence of jobs which need to be executed by the system. A periodic task $T_i = (e_i, p_i)$ with execution requirement parameter $e_i$ and period parameter $p_i$ generates a job at each instant $k \cdot p_i$, which needs to execute for $e_i$ units by a deadline of $(k + 1) \cdot p_i$, for all non-negative integers $k$. (In the remainder of this section, we will often use the symbol $\tau$ itself to denote the infinite collection of jobs generated by the tasks in periodic task system $\tau$.)

We define the utilization $u_i$ of task $T_i$ to be the ratio of its execution requirement to its period: $u_i \triangleq e_i / p_i$. Without loss of generality, we assume that the tasks in $\tau$ are indexed according to non-increasing utilization: $u_i \geq u_{i+1}$ for all $i, 1 \leq i < n$. 


Work-conserving scheduling algorithms. In the context of uniprocessor scheduling, a work-conserving scheduling algorithm is defined to be one that never idles the (sole, shared) processor while there is any active job awaiting execution. This definition extends in a rather straightforward manner to the identical multiprocessor case: an algorithm for scheduling on identical multiprocessors is defined to be work-conserving if it never leaves any processor idle while there remain active jobs awaiting execution.

We define a uniform multiprocessor scheduling algorithm to be work-conserving if and only if it satisfies the following conditions:

- No processor is idled while there are active jobs awaiting execution.
- If at some instant there are fewer than $m$ active jobs awaiting execution (recall that $m$ denotes the number of processors in the uniform multiprocessor platform), then the active jobs are executed upon the fastest processors. That is, it is the case that at any instant $t$ if the $j$-th-slowest processor is idled by the work-conserving scheduling algorithm, then the $k$-th-slowest processor is also idled at instant $t$, for all $k > j$.

EDF on uniform processors. Recall that the earliest deadline first scheduling algorithm (EDF) chooses for execution at each instant in time the currently active job[s] that have the smallest deadlines. In this research, we assume that EDF is implemented upon uniform multiprocessor systems according to the following rules:

1. No processor is idled while there is an active job awaiting execution.
2. When fewer than $m$ jobs are active, they are required to execute upon the fastest processors while the slowest are idled.
3. Higher priority jobs are executed upon faster processors. More formally, if the $j$-th-slowest processor is executing job $J_g$ at time $t$ under our EDF implementation, it must be the case that the deadline of $J_g$ is not greater than the deadlines of jobs (if any) executing on the $(j+1)$-th-, $(j+2)$-th-, ..., $m$-th-slowest processors.

The first two conditions above imply that EDF is a work-conserving scheduling algorithm.

3. EDF scheduling on uniform multiprocessors

Recall that it follows from the result of Hong and Leung [8] that no uniform multiprocessor on-line scheduling algorithm can be optimal. Suppose that a given set of jobs is known to be feasible on a given $m$-processor uniform multiprocessor platform $\pi$. In this section, we obtain conditions upon some other $m$-processor uniform multiprocessor platform $\pi'$ under which the EDF scheduling algorithm guarantees to meet all deadlines for this set of jobs on $\pi'$.

In Lemma 1 below, we first prove a general result that relates the amount of work done at each instant in time by any work-conserving scheduling algorithm (such as EDF) executing on $\pi'$ with the amount of work done by an optimal scheduling algorithm executing on $\pi$, when both algorithms are executing the same set of jobs. We use this lemma in Theorem 1 to determine conditions under which EDF executing on $\pi'$ will meet all deadlines of a set of jobs known to be feasible on $\pi$.

First, some additional notation.

Definition 1 ($W(A, \pi, I; t)$). Let $I$ denote any set of jobs, and $\pi$ any uniform multiprocessor platform. For any algorithm $A$ and time instant $t \geq 0$, 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 $\pi$.

Definition 2 ($S_j$). Let $\pi$ denote an $m$-processor uniform multiprocessor platform with processor capacities $s_1, s_2, \ldots, s_m$. $s_j \geq s_{j+1}$ for all $1 \leq j < m$. Define $S_j$ as follows:

$$S_j \equiv \sum_{\ell=1}^{j-1} s_{\ell}$$

for all $1 \leq j \leq m$.

Similarly, if the speeds of $\pi'$ are given by $s'_1, s'_2, \ldots, s'_m$, with $s'_j \geq s'_{j+1}$ for all $1 \leq j < m$, we define $S'_j$ by

$$\sum_{\ell=1}^{j-1} s'_{\ell}$$

for all $1 \leq j \leq m$.

Definition 3 ($\lambda_\pi$). Let $\pi$ denote an $m$-processor uniform multiprocessor platform with processor capacities $s_1, s_2, \ldots, s_m$. $s_j \geq s_{j+1}$ for all $1 \leq j < m$. Define $\lambda_\pi$ as follows:

$$\lambda_\pi \equiv \max_{j=1}^{m} \left\{ \frac{\sum_{k=j+1}^{m} s_k}{s_j} \right\}.$$  (1)

This parameter $\lambda_\pi$ of a uniform multiprocessor system $\pi$ intuitively measures the “degree” by which $\pi$ differs from an identical multiprocessor platform (in the sense that $\lambda_\pi = (m-1)$ if $\pi$ is comprised of $m$ identical processors, and becomes progressively smaller as the speeds of the processors differ from each other by greater amounts; in the extreme, if $s_1 > 0$ and $s_2 = s_3 = \cdots = s_m = 0$ would have $\lambda_\pi = 0$).
Lemma 1 below specifies a condition (Condition 2 below) upon the uniform multiprocessor platforms $\pi$ and $\pi'$ under which any work-conserving algorithm $A'$ (such as EDF) executing on $\pi'$ is guaranteed to complete at least as much work by each instant in time $t$ as any other algorithm $A$ (including an optimal algorithm) executing on $\pi$, when both algorithms are executing on any set of jobs $I$. This condition is expressed as a constraint on the parameter $\lambda_{m'}$ of the uniform multiprocessor platform $\pi'$. Condition 2 expresses the additional computing capacity needed by $\pi'$ (i.e., the amount by which the total computing capacity of $\pi'$ must exceed that of $\pi$) in terms of this $\lambda_{m'}$ parameter, and the speed of the fastest processor in $\pi$ — the smaller the value of $\lambda_{m'}$ (the more $\pi'$ deviates from being an identical multiprocessor), the smaller the amount of this excess processing capability needed.

**Lemma 1** Let $\pi$ denote an $m$-processor uniform multiprocessor platform with processor capacities $s_1, s_2, \ldots, s_m$, $s_j \geq s_{j+1}$ for all $j$, $1 \leq j < m$. Let $\pi'$ denote an $m'$-processor uniform multiprocessor platform with processor capacities $s'_1, s'_2, \ldots, s'_{m'}$, $s'_{j} \geq s'_{j+1}$ for all $j$, $1 \leq j < m'$. Let $A$ denote any $m$-processor multiprocessor scheduling algorithm, and $A'$ any work-conserving $m'$-processor uniform multiprocessor scheduling algorithm. If the following condition is satisfied by platforms $\pi$ and $\pi'$:

$$s'_m \geq \lambda_{m'} \cdot s_1 + S_m$$  \hspace{1cm} (2)

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

$$W(A', \pi', I, t) \geq W(A, \pi, I, t).$$  \hspace{1cm} (3)

**Proof.** The proof is by contradiction. Suppose that it is not true; i.e., there is some collection of jobs $I$ and some time-instant by which work-conserving algorithm $A'$ executing on $\pi'$ has performed strictly less work than some other algorithm $A$ executing on $\pi$. Let $J_o = (r_o, c_o, d_o)$ denote some job in $I$ with the earliest arrival time such that there is some time-instant $t_o$ satisfying

$$W(A', \pi', I, t_o) < W(A, \pi, I, t_o),$$

and the amount of work done on job $J_o$ by time-instant $t_o$ in $A'$ is strictly less than the amount of work done of $J_o$ by time-instant $t_o$ in $A$.

By our choice of $r_o$, it must be the case that

$$W(A', \pi', I, r_o) \geq W(A, \pi, I, r_o).$$

Therefore, the amount of work done by $A$ over $[r_o, t_o]$ is strictly greater than the amount of work done by $A'$ over the same interval.

Let $x_{\ell}$ denote the cumulative length of time over the interval $[r_{\ell}, t_{\ell}]$ during which $A'$ is executing on $\ell$ processors, $1 \leq \ell \leq m$ (Hence, $t_o - r_o = x_1 + x_2 + \cdots + x_m$). We make the following two observations:

- Since $A'$ is a work-conserving scheduling algorithm, job $J_o$, which has not completed by instant $t_o$ in the schedule generated by $A'$, must be executing at all time-instants during which some processor is idled by $A'$. During the instants at which $\ell$ processors are non-idling, $\ell < m$, all these non-idled processors have computing capacity $\geq s'_{\ell}$ — this follows from the definition of “work-conserving” and the fact that $A'$ is a work-conserving algorithm. Therefore, it follows that job $J_o$ has executed for at least $\sum_{j=1}^{m-1} x_j s_j'$ units by time $t_o$ in the schedule generated by $A'$ on $\pi'$, while it could have executed for at most $s_1 \left( \sum_{j=1}^{m} x_j \right)$ units in the schedule generated by Algorithm $A$ on $\pi$. We therefore have

$$\sum_{j=1}^{m-1} x_j s_j' < s_1 \left( \sum_{j=1}^{m} x_j \right).$$  \hspace{1cm} (4)

Multiplying both sides of Inequality 4 above by $\lambda_{m'}$, and noting that

$$x_j s_j' \lambda_{m'} \geq \left( x_j s_j' \frac{S_m - S'_1}{s_j} \right) = x_j (S'_m - S'_j),$$

we obtain

$$\sum_{j=1}^{m-1} \left( x_j (S'_m - S'_j) \right) < s_1 \lambda_{m'} \left( \sum_{j=1}^{m} x_j \right).$$  \hspace{1cm} (5)

- The total amount of work done by $A'$ executing on $\pi'$ during $[r_o, t_o]$ is given by $\sum_{j=1}^{m} (x_j S'_j)$, while the total amount of work done by $A$ executing on $\pi$ during this same interval is bounded from above by the capacity of $\pi$, and is hence $\leq (\sum_{j=1}^{m} x_j) \cdot S_m$. We thus obtain the inequality

$$\sum_{j=1}^{m} (x_j S'_j) < \left( \sum_{j=1}^{m} x_j \right) \cdot S_m.$$  \hspace{1cm} (6)
Adding Inequalities 5 and 6, we obtain
\[
\sum_{j=1}^{m} (x_j s'_{j}) + \sum_{j=1}^{m} (x_j (S_{m} - s'_{j})) < \\
\sum_{j=1}^{m} x_j (s_1 \lambda_{\pi'} + S_{m})
\]
\[
\equiv x_m s'_{m} + \sum_{j=1}^{m-1} [x_j (s'_{j} + s'_{m} - S_{j})] < \\
\sum_{j=1}^{m} x_j (s_1 \lambda_{\pi'} + S_{m})
\]
\[
\equiv x_m s'_{m} + \sum_{j=1}^{m-1} (x_j s'_{m}) < \sum_{j=1}^{m} x_j (s_1 \lambda_{\pi'} + S_{m})
\]
\[
\equiv S'_{m} \cdot \sum_{j=1}^{m} x_j < \sum_{j=1}^{m} x_j (s_1 \lambda_{\pi'} + S_{m})
\]
\[
\equiv S'_{m} < s_1 \lambda_{\pi'} + S_{m}
\]
which contradicts the assumption made in the statement of the lemma (Inequality 2).

Lemma 1 allows us to reason about the total execution of work-conserving algorithms. The next theorem shows that we can use this knowledge to deduce whether a work-conserving algorithm can feasibly schedule a task set: it states that any collection of jobs \( I \) that is feasible on a uniform multiprocessor platform \( \pi \) will be scheduled to meet all deadlines by Algorithm EDF on any platform \( \pi' \) satisfying Condition 2 of Lemma 1.

**Theorem 1** Let \( I \) denote an instance of jobs that is feasible on an \( m \)-processor uniform multiprocessor platform \( \pi \). Let \( \pi' \) denote another \( m \)-processor uniform multiprocessor platform. Let the parameter \( \lambda_{\pi'} \) of \( \pi' \) be as defined in Lemma 1 (Equation 1):
\[
\lambda_{\pi'} \overset{\text{def}}{=} \max_{j=1}^{m} \left\{ \sum_{k=j+1}^{m} \frac{s'_k}{s'_{j}} \right\}.
\]
If Condition 2 of Lemma 1 is satisfied by platforms \( \pi \) and \( \pi' \):
\[
S'_{m} \geq \lambda_{\pi'} \cdot s_1 + S_{m}
\]
then \( I \) will meet all deadlines when scheduled using the EDF algorithm executing on \( \pi' \).

**Proof.** As a consequence of \( \pi \) and \( \pi' \) satisfying Condition 2, it follows from Lemma 1 that the work done at any time-instant \( t \) by EDF scheduling \( I \) on \( \pi' \) is at least as much as the work done by that time-instant \( t \) by an optimal scheduling algorithm executing \( I \) on \( \pi \):
\[
W(EDF, \pi', I, t) \geq W(\text{opt}, \pi, I, t) \quad \text{for all } t \geq 0,
\]
where \( \text{opt} \) denotes an algorithm that generates a schedule for \( I \) which meets all deadlines on \( \pi \) — since \( I \) is assumed feasible on \( \pi \), such a schedule exists.

We now prove by induction that \( I \) is scheduled by EDF to meet all deadlines on \( \pi' \). The induction is on the number of jobs in \( I \). Specifically, let \( I_k := \{J_1, \ldots, J_k\} \) denote the \( k \) jobs of \( I \) with the highest EDF-priority.

**Base case.** Since \( I_0 \) denotes the empty set, \( I_0 \) can clearly be scheduled by EDF to meet all deadlines on \( \pi' \).

**Induction step.** Assume that EDF can schedule \( I_k \) on \( \pi' \) for some \( k \) and consider the EDF-generated schedule of \( I_{k+1} \) on \( \pi' \). Note that \( I_k \subset I_{k+1} \) and that the \( J_{k+1} \) does not effect the scheduling decisions made by EDF on the jobs \( \{J_1, J_2, \ldots, J_k\} \) while it is scheduling \( I_{k+1} \). That is, the schedule generated by EDF for \( \{J_1, J_2, \ldots, J_k\} \) while scheduling \( I_{k+1} \), is identical to the schedule generated by EDF while scheduling \( I_k \); hence by the induction hypothesis, these \( k \) highest priority jobs \( \{J_1, J_2, \ldots, J_k\} \) of \( I_{k+1} \) all meet their deadlines. It remains to prove that \( J_{k+1} \) also meets its deadline.

Let us now turn our attention to the schedules generated by \( \text{opt} \) executing on \( \pi \). Since \( I \) is assumed to be feasible on \( \pi \), it follows that \( I_{k+1} \) is also feasible on \( \pi \) and hence \( \text{opt} \) will schedule \( I_{k+1} \) on \( \pi \) to meet all deadlines. That is,
\[
W(\text{opt}, \pi, I_{k+1}, d_{k+1}) = \sum_{l=1}^{k+1} c_l,
\]
where \( d_{k+1} \) denotes the latest deadline of a job in \( I_{k+1} \). By Lemma 1
\[
W(EDF, \pi', I_{k+1}, d_{k+1}) \geq W(\text{opt}, \pi, I_{k+1}, d_{k+1}) = \sum_{l=1}^{k+1} c_l.
\]
Since the total execution requirement of all the jobs in \( I_{k+1} \) is \( \sum_{l=1}^{k+1} c_l \) it follows that job \( J_{k+1} \) meets its deadline.

We have thus shown that EDF successfully schedules all the jobs of \( I_{k+1} \) to meet their deadlines on \( \pi' \). The theorem follows.

As an immediate corollary to Theorem 1 above, we obtain the result of Philips, Stein, Torng, and Wein [1] concerning EDF-scheduling on identical multiprocessors:

**Corollary 1.1** If a set of jobs is feasible on an identical \( m \)-processor platform, then the same set of jobs will be scheduled to meet all deadlines by EDF on an identical \( m \)-processor platform in which the individual processors are \((2 - \frac{1}{m})\) times as fast as in the original system.

**Proof.** If we require the platforms \( \pi \) and \( \pi' \) of the statement of Theorem 1 to be comprised of identical multiprocessors
of speeds \( s \) and \( s' \) respectively, the parameter \( \lambda_{s'} \) evaluates to \((m-1)\):

\[
\lambda_{s'} \overset{\text{def}}{=} \max_{j=1}^{m} \left\{ \frac{\sum_{k=j+1}^{m} s'_{k}}{s'_{j}} \right\} = \max_{j=1}^{m} \left\{ \frac{(j-1)s'}{s'} \right\} = m - 1
\]

The condition \( S'_{m} \geq \lambda_{s'} \cdot s_{1} + S_{m} \) from the statement of Theorem 1 is hence equivalent to

\[
S'_{m} \geq \lambda_{s'} \cdot s_{1} + S_{m} \\
\equiv m \cdot s' \geq (m-1) \cdot s + m \cdot s \\
\equiv m \cdot s' \geq (m-1 + m) \cdot s \\
\equiv s' \geq \left( 2 - \frac{1}{m} \right) \cdot s
\]

from which the corollary follows.

Theorem 1 characterizes a uniform multiprocessor platform \( \pi' \) according to its parameter “\( \lambda_{s'} \)” (as defined in Equation 1), and relates the EDF-feasibility of a system, known to be feasible on some platform \( \pi \), to the cumulative capacities of \( \pi \) and \( \pi' \), the speed \( s_{1} \) of the fastest processor in \( \pi \), and this parameter \( \lambda_{s'} \) of platform \( \pi' \). Theorem 2 below asserts that, for this particular characterization of a uniform multiprocessor system, the bound represented by Theorem 1 is a tight one and EDF is optimal in the sense that no other on-line scheduling algorithm can make a better guarantee:

**Theorem 2** There exist uniform multiprocessor platforms \( \pi \) and \( \pi' \) and an instance \( I \) of hard-real-time jobs such that

- Instance \( I \) is feasible on platform \( \pi \).
- The relationship between the various parameters of \( \pi \) and \( \pi' \) is as follows
  \[
  S'_{m} = \lambda_{s'} \cdot s_{1} + S_{m} - \epsilon \cdot \quad (7)
  \]

(where \( \epsilon \) is assumed to be an arbitrarily small positive number; i.e., the condition of Theorem 1 is not satisfied by an arbitrarily small amount \( \epsilon \).)

- Instance \( I \) is infeasible on uniform multiprocessor platform \( \pi' \).

**Proof.** Let \( \eta = \frac{s}{s'} \). Consider a system \( \pi = [1, 1] \) comprised of two identical unit-speed processors; an instance \( I = \{(0, 1, 1), (0, 1, 1)\} \) of two jobs each arriving at time zero, and needing to execute for one unit by a deadline equal to one; and a platform \( \pi' = \left[ \frac{2-\eta}{1+\eta}, \frac{\eta(2-\eta)}{1+\eta} \right] \) comprised of one processor of speed \( \frac{2-\eta}{1+\eta} \) and another processor of speed \( \frac{\eta(2-\eta)}{1+\eta} \). Observe that

\[
\lambda_{s'} \cdot s_{1} + S_{m} - \epsilon = \eta + 2 - \epsilon = \eta + 2 - \epsilon = S'_{m}.
\]

Therefore, Equation 7 is satisfied; however, it is easy to see that there is no way to meet both jobs’ deadlines upon \( \pi' \), since instance \( I \) requires two units of execution over the interval \([0, 1]\), while platform \( \pi' \) can accommodate only \((2 - \eta)\) units of execution over this interval.

**4. Scheduling periodic task systems on uniform multiprocessors**

In this section, we will apply the theory developed in Section 3 above to study the deadline-based scheduling of periodic task systems on uniform multiprocessor platforms. Although scheduling a periodic task system (as defined above, in Section 2) is not an “on-line” problem in the sense that all task parameters are assumed known beforehand, the results in Section 3 nevertheless turn out to be useful towards developing a framework for scheduling periodic task systems on uniform multiprocessors.

In Section 4.1, we extend previous results concerning the scheduling of jobs on uniform multiprocessors, to obtain an exact test for determining whether a given periodic task system is feasible on a particular uniform multiprocessor platform. Then in Section 4.2 we use this exact feasibility test, along with the results we had obtained in Section 3, to design a test for determining whether a given periodic task system will be successfully scheduled by EDF on a specified uniform multiprocessor platform.

**4.1. An exact feasibility condition**

Horvath et al. [9] have studied the problem of scheduling a set of individual preemptable jobs, all of which have the same arrival time and deadlines equal to infinity, on a given uniform multiprocessor platform in order to minimize the makespan - the length of the resulting schedule\(^2\). The following result was proved in [9].

**Theorem 3 (Horvath et al. [9])** Consider a set of \( n \) jobs \( \{J_{1}, J_{2}, \ldots, J_{n}\} \), each with arrival time equal to zero, deadline equal to infinity, and indexed according to non-increasing execution requirements (i.e., \( J_{i} = (0, c_{i}, \infty) \), and \( c_{i} \geq c_{i+1} \) for all \( i, 1 \leq i < n \)). Let \( \pi \) denote a uniform multiprocessor platform comprised of \( m \) processors with speeds \( s_{1}, s_{2}, \ldots, s_{m}, s_{i} \geq s_{i+1} \) for all \( i, 1 \leq i < m \). Let \( C_{i}, 1 \leq i \leq n \), be defined as follows

\[
C_{i} = c_{1} + c_{2} + \cdots + c_{i},
\]

and \( S_{i}, 1 \leq i \leq m \), be defined as follows

\[
S_{i} = s_{1} + s_{2} + \cdots + s_{i}.
\]

\(^2\)In the scheduling-theoretic notation of Graham et al. [7], this is the “Q / pmtn / C_{max}” problem.
That is, \( C_i \) denotes the cumulative execution requirement of the \( i \) largest jobs, and \( S_i \) the cumulative computing capacity of the \( i \) fastest processors. The minimum makespan for scheduling these \( n \) jobs \( \{J_1, J_2, \ldots, J_n\} \) on the uniform multiprocessor platform \( \pi \) is given by

\[
\max \left( \max_{i=1}^{m} \left( \frac{C_i}{S_i}, \frac{C_n}{S_m} \right) \right).
\] 

Below, we show how we can transform the problem of scheduling periodic tasks on uniform multiprocessors to the \( Q / pmtn / C_{\max} \) problem. The proof of the following theorem presents the details.

**Theorem 4** Consider a set \( \tau = \{T_1, \ldots, T_n\} \) of periodic tasks indexed according to non-increasing utilization (i.e., \( u_i \geq u_{i+1} \) for all \( i, 1 \leq i < n \), where \( u_i \) def \( \frac{C_i}{S_i} \)). Let 

\[
U_i = \sum_{j=1}^{i} u_j \text{ for all } i, 1 \leq i \leq n.
\]

Let \( \tau' \) be a set of \( i \) non-periodic jobs, with the \( i \)th job \( J_i \) having arrival time zero, a deadline at time infinity, and an execution requirement equal to \( C_i \) def \( u_i \Delta \). Using the result of Horvath et al. [9] described above, we know that the minimum makespan for \( \tau' \) on \( \pi \) is

\[
\max \left( \max_{i=1}^{m} \left( \frac{C_i}{S_i}, \frac{C_n}{S_m} \right) \right).
\]

By definition, \( C_i = \sum_{j=1}^{i} C_j = \sum_{j=1}^{i} u_j \Delta = \Delta \cdot U_i \).

Therefore, the minimum makespan for \( \tau' \) is

\[
\Delta \cdot \max \left( \max_{i=1}^{m} \left( \frac{U_i}{S_i}, \frac{U_n}{S_m} \right) \right)
\]

which is at most \( \Delta \) by Equations 9 and 10. Now let \( \sigma_1 \) denote a schedule that completes execution of \( \tau' \) by time \( \Delta \). We can obtain a schedule for the periodic task system \( \tau \) on \( \pi \), by simply repeating the schedule \( \sigma_1 \) infinitely often every \( \Delta \) time units. Since \( \Delta \) divides each deadline of \( \tau \) evenly, the utilization of each job between its arrival time and its deadline is \( u_i \), therefore all the deadlines are met.

Suppose now that the periodic task system \( \tau \) is feasible on \( \pi \), and let \( \sigma_2 \) be a schedule on \( \pi \) in which all jobs of all tasks in \( \tau \) complete by their deadlines. Let \( P \) denote the least common multiple of the periods of all the tasks in \( \tau \). In \( \sigma_2 \), each task \( T_i \) is scheduled for exactly \( u_i \cdot P \) units of execution over the interval \([0, P]\). It therefore directly follows that the \( n \) non-periodic jobs \( \tau'' \) def \( \{K_1, K_2, \ldots, K_n\} \), with the \( i \)th job \( K_i \) arriving at instant zero, with a deadline at time infinity, and having an execution requirement equal to \( u_i \cdot P \), can be scheduled to complete on the processors \( \pi \) such that all jobs complete execution by time \( P \)- indeed, \( \sigma_2 \) is a schedule that accomplishes this. We can obtain a schedule \( \sigma_3 \) for \( \tau'' \) on \( \pi \) by simply "compressing" the time-line of \( \sigma_2 \) by the factor \( \frac{P}{\Delta} \) and scheduling \( J_i \) in \( \sigma_3 \) whenever \( K_i \) is scheduled in \( \sigma_2 \). (Observe that when \( \sigma_3 \) is compressed, the execution requirements become \( u_i P \Delta = \Delta \cdot u_i \).) Therefore, \( \tau'' \) can be scheduled on the processors \( \pi \) such that all jobs complete execution by time \( \Delta \) if \( \tau \) is feasible on \( \pi \). Clearly \( \Delta \) cannot be less than the minimum makespan given in Theorem 3 — i.e.,

\[
\Delta \geq \max \left( \max_{i=1}^{m} \left( \frac{\Delta \cdot U_i}{S_i}, \frac{\Delta \cdot U_n}{S_m} \right) \right).
\]

Dividing by \( \Delta \) gives

\[
1 \geq \max \left( \max_{i=1}^{m} \left( \frac{U_i}{S_i}, \frac{U_n}{S_m} \right) \right).
\]

This is equivalent to Conditions 9 and 10.

We have thus seen that the periodic tasks \( \tau \) are feasible on the uniform processors \( \pi \) if and only if the non-periodic jobs \( \tau'' \) can be scheduled on \( \pi \) with a makespan of at most \( \Delta \). Furthermore, \( \tau'' \) can be scheduled on \( \pi \) with a makespan of \( \Delta \) if and only if Conditions 9 and 10 are satisfied.

**4.2. EDF-scheduling of periodic tasks**

In Section 3, we had identified a condition under which a collection of jobs known to be feasible upon a uniform multiprocessor platform \( \pi \) could be EDF-scheduled to meet all deadlines upon a uniform multiprocessor platform \( \pi' \) — this condition (Condition 2) relates the parameters of \( \pi' \) to the computing capacity of the faster processor, and the total computing capacity, of \( \pi \). In this section, we will use the results of Section 4.1 above to first identify a uniform multiprocessor platform upon which a given periodic task system \( \tau \) is feasible (Theorem 5 below); next, (Theorem 6 

\[\text{[Evidence: PhD dissertation, 2023]}
\]

\[\text{[Evidence: PhD dissertation, 2023]}
\]

\[\text{[Evidence: PhD dissertation, 2023]}
\]
below), we will apply Theorem 1 to the parameters of this uniform multiprocessor to determine a sufficient condition for \( \tau \) to be successfully scheduled by EDF on any given uniform multiprocessor platform \( \pi' \).

**Definition 4** \((P(\tau, m, k))\) Let \( \tau = \{T_1, \ldots, T_n\} \) denote a collection of periodic tasks indexed according to non-increasing utilization (i.e., \( u_i \geq u_{i+1} \) for all \( i, 1 \leq i < n \), where \( u_i \overset{\text{def}}{=} \frac{c_i}{p_i} \)). Let \( m \) be any positive integer. For \( 1 \leq k \leq m \), define \( P(\tau, m, k) \) recursively as follows:

- If \( u_1 > \frac{U_n}{m} \), let \( P(\tau, m, 1) = u_1 \) and \( P(\tau, m, k) = P(\tau - \{T_1\}, m - 1, k - 1) \) for \( k = 2, \ldots, m \).
- If \( u_1 \leq \frac{U_n}{m} \), let \( P(\tau, m, k) = \frac{U_n}{m} \) for \( k = 1, \ldots, m \).

where \( U_n \overset{\text{def}}{=} \sum_{i=1}^{n} u_i \).

**Theorem 5** Consider a set \( \tau = \{T_1, \ldots, T_n\} \) of periodic tasks indexed according to non-increasing utilization. Let \( \pi \) denote the \( m \)-processor uniform multiprocessor platform in which the processors have speeds \( s_k = P(\tau, m, k) \), \( 1 \leq k \leq m \):

\[ \pi = [P(\tau, m, 1), P(\tau, m, 2), \ldots, P(\tau, m, m)]. \]

Periodic task system \( \tau \) is schedulable on this uniform multiprocessor platform \( \pi \).

**Proof.** By construction, \( \pi \) satisfies the Conditions 9 and 10 of Theorem 4. Therefore, \( \tau \) is schedulable on \( \pi \). \n
The following corollary follows directly from Theorem 5, and the definitions of \( P(\tau, m, k) \).

**Corollary 5.1** The \( m \)-processor uniform multiprocessor platform

\[ \pi = [P(\tau, m, 1), P(\tau, m, 2), \ldots, P(\tau, m, m)] \]

of Theorem 5 satisfies the following two properties:

1. The fastest processor has computing capacity \( \max \left( u_1, \frac{U_n}{m} \right) \).
2. The sum of the processor capacities is \( U_n \).

**Theorem 6** Let \( \pi' = [s'_1, s'_2, \ldots, s'_m] \) denote any \( m \)-processor uniform multiprocessor platform, and let \( \lambda_{\pi'} \) be as defined in Lemma 1 (Equation 1):

\[ \lambda_{\pi'} \overset{\text{def}}{=} \max_{j=1}^{m} \left\{ \sum_{i=j+1}^{m} \frac{s'_i}{s'_j} \right\}. \]

Periodic task system \( \tau \) will meet all deadlines when scheduled on \( \pi' \) using EDF, if the following condition holds

\[ S'_m \geq \lambda_{\pi'} \cdot \left\{ \max \left( u_1, \frac{U_n}{m} \right) \right\} + U_n. \tag{11} \]

**Proof.** By Theorem 5 and Corollary 5.1, \( \tau \) is feasible on an \( m \)-processor uniform multiprocessor platform \( \pi = (s_1, s_2, \ldots, s_m) \) satisfying

\[ s_1 = \max(u_1, \frac{U_n}{m}), \quad \text{and} \quad S'_m = U_n. \]

Hence by Theorem 1, \( \tau \) will meet all deadlines when it is scheduled using EDF on \( \pi' \), if

\[ S'_m \geq \lambda_{\pi'} \cdot \left\{ \max \left( u_1, \frac{U_n}{m} \right) \right\} + U_n, \]

and the theorem follows.

We now illustrate the use of Theorem 6 by an example.

**Example 1** Consider a task system \( \tau \) comprised of five tasks:

\[ \tau = \{(15, 10), (4, 5), (12, 20), (6, 15), (2, 10)\}; \]

for this system, \( u_1 = 1.5, u_2 = 0.8, u_3 = 0.6, u_4 = 0.4, \) and \( u_5 = 0.2 \). Suppose that \( \tau \) is to be EDF-scheduled on the uniform multiprocessor platform \( \pi' = [3, 1, 0.5] \) — will all deadlines be met?

By Equation 1, the value of \( \lambda_{\pi'} \) for the uniform multiprocessor platform \( \pi' \) is

\[ \lambda_{\pi'} = \max \left( \frac{1 + 0.5}{3}, \frac{0.5}{1} \right) = \frac{1}{2}, \]

and the total computing capacity is

\[ 3 + 1 + 0.5 = 4.5. \]

By Theorem 5 and Corollary 5.1, \( \tau \) is feasible on some \( 3 \)-processor uniform multiprocessor platform having a total computing capacity of

\[ 1.5 + 0.8 + 0.6 + 0.4 + 0.2 = 3.5 \]

and with the fastest processor having computing capacity

\[ s_1 = \max(1.5, \frac{3.5}{3}) = 1.5. \]

The Condition 11 is therefore

\[
\begin{align*}
4.5 & \geq 0.5 \cdot 1.5 + 3.5 \\
\Xi 4.5 & \geq 0.75 + 3.5 \\
\Xi 4.5 & \geq 4.25
\end{align*}
\]

and \( \tau \) can consequently be scheduled by EDF to meet all deadlines on \( \pi' \).
5. Conclusions

Knowledge of good on-line scheduling algorithms seems necessary in order to gain a deeper understanding of real-time scheduling within a particular model — witness the pervasive role played by EDF in many of the seminal papers on uniprocessor real-time scheduling theory. While EDF is known to be optimal in the uniprocessor context, it has been shown that no multiprocessor on-line scheduling algorithm can be optimal.

In this research, we have explored the design of suitable on-line scheduling algorithms for use in uniform multiprocessor platforms (since identical multiprocessors are a special case of uniform multiprocessors, our results hold for identical multiprocessors as well). Our hypothesis was that despite its non-optimality, EDF is an appropriate algorithm to use for on-line scheduling on uniform multiprocessors. Our findings strongly support this hypothesis:

- For a particular characterization of uniform multiprocessors — by its $\lambda$ parameter (Definition 3) and its cumulative computing capacity — EDF is an optimal algorithm (Theorem 2).
- Many of the advantages of using EDF in the uniprocessor context (efficient implementation [13]; the fact that the exact computation requirements of jobs are not needed for making scheduling decisions; bounds upon the number of (global) context switches; etc.) continue to hold in the multiprocessor case.
- An efficient test (Theorem 1) can determine whether any instance of hard-real-time jobs known to be feasible on a particular platform can be scheduled by EDF to meet all deadlines upon another platform.

As an illustration of the applicability of EDF as an on-line scheduling algorithm for uniform multiprocessor platforms, we have applied the theory we have developed to devise an efficient test for determining whether any system of periodic tasks is EDF-feasible upon a given multiprocessor platform.

References