Partitioned scheduling of implicit-deadline sporadic task systems under multiple resource constraints

Bipasa Chattopadhyay

Sanjoy Baruah

Abstract—On many multiprocessor platforms each individual processor may have limited amounts of several different kinds of resources such as computing capacity, local memory, and network bandwidth. In order to partition tasks effectively upon such platforms the partitioning algorithm should be cognizant of all the resource constraints. We present and evaluate an algorithm for partitioning a collection of implicit-deadline sporadic tasks upon a multiprocessor platform in a manner that is cognizant of multiple such resource constraints.

I. INTRODUCTION

The advantages of multicore technologies, and the increase in functionality of embedded real-time systems has caused an increasing trend towards the use of multicore CPUs and multiprocessor platforms for implementing embedded real-time systems. One approach for implementing embedded real-time systems on multiprocessor platforms is to statically assign tasks to processors such that every instance of a task executes on the processor to which it has been assigned. This is the well known partitioned scheduling approach. While partitioning tasks it is necessary to ensure that sufficient amounts of each of the resources the tasks need are available on the processors. Some key resources include computing capacity, local (per-core) memory, and network bandwidth.

We study the partitioned Earliest Deadline First (EDF) scheduling of implicit-deadline sporadic task systems [1] on multiprocessor platforms. Earlier work on this problem (see, e.g., [2], [3]) focused exclusively on processors in which only one resource – computing capacity – is available in limited amounts. Some more recent work (e.g., [4], [5]) has considered processors with limited memory in addition to limited computing capacity. Each of the tasks were characterized by a memory requirement, in addition to characterizations of its computational requirements (which was modelled according to the implicit-deadline sporadic task model [1]). In this paper we study the partitioning problem on platforms in which an arbitrary (but fixed) number of different types of resources may be available in limited quantities upon each processor, and the partitioning algorithm must therefore be cognizant of all these limitations when determining the assignment of tasks to processors. This problem is known to be highly intractable. To get an idea of its computational complexity it is worth noting that the simpler problem of partitioning tasks onto multiprocessor platforms, where computing capacity is the only limiting resource on each processor, is NP-complete. Therefore, it is unlikely that there exists an efficient algorithm that can solve the problem exactly; the algorithm that we present here gives an approximate solution. The partitioning problem is similar to the bin packing problem. In [6], approximation algorithms for the multidimensional bin packing problem have been described and analyzed. Although the algorithm that we present is similar to one of the algorithms presented in [6], bin packing algorithms can not be directly applied to solve the task partitioning problem since the goal in task partitioning is to partition the task system onto m processors whereas in bin packing the goal is to pack the items into the least number of bins. Also, in our analysis we derive the resource augmentation [7] bound which is different from the approximation bound with respect to the number of bins, that is derived in the analysis presented in [6].

Further, unlike some earlier algorithms that have been proposed for this problem (e.g., [5]), our algorithm is not based on results concerning integer linear programming; in fact, it will become evident that it is a generalization to multiple resources of the widely-used algorithm in [3] for task partitioning when processing capacity is the sole scarce resource.

The remainder of this paper is organized as follows. In Section II, we describe the task and platform model we use. In Section III, we briefly survey some related work and place this problem within the larger context of multiprocessor real-time scheduling theory. In Section IV, we present Algorithm PARTITION for solving the partitioning problem, and prove that it is correct. We derive some properties of Algorithm PARTITION in Section V, that allow us to make quantitative assertions bounding the degree of deviation of its performance from optimality. In Section VI, we suggest a heuristic for reordering tasks in a task system before partitioning the task system using Algorithm PARTITION. In Section VII, we report on some simulation experiments that compare our algorithm with prior algorithms, and that evaluate the effectiveness of the heuristic of Section VI. For ease of presentation, Sections II-VII deal with platforms with just one kind of resource in addition to computing capacity; we describe generalizations of these sections for dealing with more resources in Section VIII.
The ideas presented in this paper are able to deal with any fixed number of different types of resources. However, for the sake of simplicity we will restrict much of the discussion to systems in which there are two constraining resources - computation capacity, and local memory - on each processor. In Section VIII, we describe how our results generalize to handle more than two types of resources.

We consider a task system model in which all tasks are implicit-deadline sporadic tasks [1] also known as Liu and Layland tasks. A task system, \( \tau \), consists of \( n \) tasks: \( \tau = \{ \tau_1, \tau_2, \ldots, \tau_n \} \), and a task is denoted by \( \tau_i \) (\( i = 1 \) to \( n \)). Each task, \( \tau_i \), is characterized by its

- **Computation requirement**, \( u_i \). Every implicit-deadline task is characterized by a worst-case execution requirement, \( C_i \), and a minimum inter-arrival separation, \( T_i \). Since after partitioning the task system we will be scheduling each processor using EDF, we may use the utilization \( u_i \) of a task, which by definition is \( C_i/T_i \), as its computation requirement.

- **Memory requirement**, \( v_i \). The memory requirement of a task is the fraction of local memory a tasks requires for its exclusive use. For example, local memory may be used to store the executable code of a task on a processor.

We make no assumptions about the relationship between the computation requirement, \( u_i \), and the memory requirement, \( v_i \), for a task, \( \tau_i \); in particular, we do not require that tasks with modest computation requirements have small \( v_i \), and those with large computation requirements have large \( v_i \). (Such restrictions would not allow us to model, e.g., a relatively simple task that is extremely computation-intensive because it repeatedly samples external input at a rapid rate, or a task with large code-size, comprised of much conditional code, that is invoked very infrequently and hence does not place a large computation demand on the processor.)

The multiprocessor platform is comprised of \( m \) identical preemptive processors denoted \( \pi_1, \pi_2, \ldots, \pi_m \) on which we are to partition a task system \( \tau \) of \( n \) sporadic tasks. We know that EDF is optimal on preemptive uniprocessors [1], [8], and that a necessary and sufficient condition for a collection of tasks to be EDF-schedulable on a preemptive uniprocessor is that the sum of the computation requirements of all the tasks on the processor not exceed the computing capacity of the processor. A correct partitioning of \( \tau \) on the \( m \) processors is therefore one that ensures that

1) the sum of the computation requirements (the \( u_i \) parameters) of all the tasks assigned to each processor does not exceed \( 1 \); and

2) the sum of the memory requirements (the \( v_i \) parameters) of all the tasks assigned to each processor does not exceed \( 1 \).

Various heuristics for task partitioning on processors with limited computing capacity have been studied and evaluated. In [3], heuristics such as First-Fit, Best-Fit, Worst-Fit, First-Fit-Decreasing etc., that have very efficient implementations, have been compared on the basis of their **sufficient schedulability conditions** (for a description of these heuristics please see [3]). Paraphrasing and simplifying slightly, the main result from [3] can be stated as follows: any implicit-deadline sporadic task system satisfying the condition

\[
\sum_{i=1}^{n} u_i \leq m - (m - 1) \times \min \left( \frac{1}{2}, \max_{i=1}^{n} v_i \right)
\]

(1)

is successfully partitioned by the First-Fit-Decreasing (FFD) heuristic upon a platform consisting of \( m \) unit-capacity processors.

A partitioning problem is inherently a decision problem: either a task system is schedulable or not. Partitioning problems fall in the NP-complete class of decision problems, therefore an optimal partitioning algorithm is intractable. To deal with the intractability, we consider designing approximate partitioning algorithms that have efficient run-time complexities. In order to be able to quantitatively discuss the effectiveness of different approximation algorithms it is useful to define the corresponding **optimization** problem. For partitioned scheduling, one such optimization problem, the one we use in this paper, asks: given a task system and a platform, **what is the minimum multiplicative factor by which the resources (i.e., the computing capacity and the local memory) available on each processor in the platform must be augmented, in order for the task system to be determinable to be schedulable or not in polynomial-time?** The minimum multiplicative factor by which the resources need to be augmented above is called the resource augmentation bound of the algorithm.

There is a classification of NP-hard optimization problems according to the difficulty of obtaining approximate solutions to these problems in polynomial-time. In particular, an NP-hard minimization problem is said to be in the class **APX** (for APproXimable) if there is a constant \( c \) such that some polynomial-time algorithm can obtain a solution to any problem instance that is no more than \( c \) times the cost of the (optimal) minimum-cost solution. APX problems for which there exist polynomial-time algorithms that can obtain a solution to any problem instance that is no more than \( c \) times the cost of the (optimal) minimum-cost solution for all \( c > 1 \) are said to be in the class **PTAS** (for Polynomial-Time Approximation Schemes). Here, \( c \) is the approximation bound of the algorithm.

In multiprocessor platforms with just one limited resource (computing capacity) First-Fit has a resource augmentation bound of \((2 - \frac{1}{m})\) [9, P176]. This implies that the parti-
tioning problem for implicit-deadline sporadic task systems is in the class APX. A related problem in [10] that deals with the scheduling of non-preemptive jobs with the objective of minimizing makespan is shown to have a resource augmentation bound of $1 + \epsilon$, for $\epsilon > 0$. Therefore, the minimizing makespan problem is in the class PTAS. It can be shown that both the partitioning problem and minimizing makespan problem are in fact equivalent. Therefore, for a single limited resource the partitioning problem for implicit-deadline sporadic task systems is in fact in the class PTAS.

It can also be shown by reduction to the vector scheduling problem [11], that for two limited resources (computing capacity and local memory) the partitioning problem is also in the class PTAS. The algorithm proposed in [11] is however not implementable in practice because the degree of the polynomial characterizing the run-time of the algorithm is unacceptably large. Two practical algorithms, based on constructing and approximately solving integer linear programs (ILPs), are presented in [5]. One algorithm constructs an ILP from the specification of the task system, and then solves a non-integer “relaxation” to this ILP. This algorithm has a resource augmentation bound of 3, hence is not a PTAS. However, it can be implemented far more efficiently than the algorithm based on vector scheduling in [11]. This algorithm is only applicable for partitioning task systems in which all the computation requirements (the $u_i$'s) and the memory requirements (the $v_i$'s) are < 0.5. The second algorithm presented in [5] has no such restriction. If a task system consists only of tasks with computation and memory requirements < 0.5, the second algorithm simply calls the first algorithm to partition the task system. Therefore, the resource augmentation bound of the second algorithm is at least 3, which is the resource augmentation bound of the first algorithm (The exact resource augmentation bound of the second algorithm is not known). However, unlike the first algorithm, the second algorithm does not always have an efficient run time complexity. For assigning certain tasks the second algorithm needs to solve a pure ILP. Solving a pure ILP for the partitioning problem is NP-hard and the solution can not be obtained efficiently.

Other practical algorithms that have polynomial run-time complexities such as First-Fit, Best-Fit, Worst-Fit, First-Fit-Decreasing etc. for the multidimensional bin packing problem, which is similar to the partitioning problem, have been described and analyzed in [6]. The analysis in [6] shows that the First-Fit algorithm has an approximation bound no greater than $d + 1$ where $d$ is an arbitrary, but fixed, number of dimensions that an item can occupy. This approximation bound is with respect to the number of bins needed to pack the items and it is shown to be tight in [6].

The above result for the bin packing problem can be extended to the partitioning problem. Thus, the First-Fit algorithm for the task partitioning problem can be shown to have an approximation bound $d + 1$ with respect to the number of processors. Here $d$ is an arbitrary, but fixed, number of limited resources available on each processor. For the partitioning algorithm the approximation bound with respect to the number of processors indicates how many additional processors may be needed to ensure the successful partitioning by the algorithm of a task system that can be partitioned by an optimal algorithm. The resource augmentation bound, on the other hand, indicates by how much the resources on the existing processors may need to be inflated to ensure the successful partitioning of a task system that can be partitioned by an optimal algorithm.

In this paper, we describe the First-Fit algorithm for partitioning implicit-deadline task systems onto platforms comprised of $m$ processors on which both computing capacity and local memory are limited, and derive the resource augmentation bound for the algorithm. In Section VIII, we show that the algorithm can be used for partitioning tasks onto processors that have an arbitrary, but fixed, number of limited resources $l$, in addition to computing capacity and we derive the resource augmentation bound, $2 + \ell - (1 + \ell)/m$. If the total number of limited resources is expressed as $d = 1 + \ell$, then the resource augmentation bound becomes $d + 1 - d/m$. When compared to the approximation bound derived in [6] the resource augmentation bound is smaller for smaller values of $m$.

IV. ALGORITHM PARTITION

Given a task system $\tau$ of $n$ implicit-deadline sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$ we want to partition the tasks onto $m$ identical processors $\pi_1, \pi_2, \ldots, \pi_m$ that have unit-capacity and unit-memory. We present an approximate, but efficient algorithm to solve the problem. Figure 1 gives a pseudo-code representation of our algorithm. The algorithm assumes that the collection of tasks $\tau_1, \tau_2, \ldots, \tau_n$ is in any given order and then attempts to assign the tasks onto one of the $m$ processors. We now explain how a task $\tau_i$ is assigned to a processor.

First, let us suppose that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have been successfully assigned. For any processor $\pi_k$, let $\tau(\pi_k)$ denote the tasks among $\tau_1, \tau_2, \ldots, \tau_{i-1}$ that have already been assigned to it. Task $\tau_i$ is assigned to a processor $\pi_k$ only if the following two conditions are satisfied:

$$1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i \quad \text{(Condition 2)}$$

and

$$1 - \sum_{\tau_j \in \tau(\pi_k)} v_j \geq v_i \quad \text{(Condition 3)}$$

If no such $\pi_k$ exists, then the algorithm declares failure: it is unable to partition $\tau$ upon the $m$-processor platform.
PARTITION($\tau, m$)

\[ \text{PARTITIONING SUCCEEDED} \]

\[ \text{PARTITIONING FAILED} \]

The collection of sporadic tasks $\tau = \{\tau_1, \ldots, \tau_n\}$ is to be partitioned on $m$ identical, unit-capacity and unit-memory processors denoted $\pi_1, \ldots, \pi_m$. $\tau(\pi_k)$ denotes the tasks assigned to processor $\pi_k$; initially, $\tau(\pi_k) \leftarrow \emptyset$ for all $k$.

1. for $i \leftarrow 1$ to $n$
   \[ \triangleright i \text{ ranges over the tasks} \]
   2. for $k \leftarrow 1$ to $m$
      \[ \triangleright k \text{ ranges over the processors} \]
      3. if $\tau_i$ satisfies Conditions 2-3 on processor $\pi_k$
         \[ \triangleright \text{assign } \tau_i \text{ to } \pi_k; \]
         \[ \tau(\pi_k) \leftarrow \tau(\pi_k) \cup \{\tau_i\} \]
         4. break;
      5. end (of inner for loop)
      6. if $(k > m)$ return PARTITIONING FAILED
   7. end (of outer for loop)
   8. return PARTITIONING SUCCEEDED

Figure 1. Pseudo-code for partitioning algorithm.

The following lemma asserts that, in assigning a task $\tau_i$ to a processor $\pi_k$, our partitioning algorithm does not adversely affect the schedulability of the tasks previously assigned to the processors.

Lemma 1: If the tasks previously assigned to each processor were EDF-schedulable on that processor and our algorithm assigns task $\tau_i$ to processor $\pi_k$, then the tasks assigned to each processor (including processor $\pi_k$) remain EDF-schedulable on that processor.

Proof:
Observe that the EDF-schedulability of the processors other than processor $\pi_k$ is not affected by the assignment of task $\tau_i$ to processor $\pi_k$. It remains to demonstrate that, if the tasks assigned to $\pi_k$ were EDF-schedulable on $\pi_k$ prior to the assignment of $\tau_i$ and Conditions 2-3 are satisfied, then the tasks on $\pi_k$ remain EDF-schedulable after adding $\tau_i$. To see that this is true, observe that

- Condition 2 ensures that there is sufficient computing capacity to accommodate task $\tau_i$ on processor $\pi_k$; and
- Condition 3 ensures that there is sufficient local memory to accommodate task $\tau_i$ on processor $\pi_k$.

The correctness of the partitioning algorithm can now be established by repeated applications of Lemma 1.

Theorem 1: If our partitioning algorithm returns PARTITIONING SUCCEEDED on task system $\tau$, then the resulting partitioning is EDF-schedulable.

Proof:
Observe that the algorithm returns PARTITIONING SUCCEEDED if and only if it has successfully assigned each task in $\tau$ to some processor.

Prior to the assignment of task $\tau_1$, each processor is trivially EDF-schedulable. It follows from Lemma 1 that all processors remain EDF-schedulable after each task assignment as well. Hence, all processors are EDF-schedulable once all tasks in $\tau$ have been assigned.

A. Run-time complexity

Algorithm PARTITION can maintain, for each processor, the cumulative computation and memory requirements of all the tasks that have been assigned to each processor thus far. For each task $\tau_i$ and each processor $\pi_k$, Conditions 2 and 3 can then be evaluated in constant time. Therefore the $i$'th task can be assigned in $O(m)$ time, for each $i$; this yields an overall run-time of $O(n \times m)$, which is linear in the product of the number of tasks and the number of processors.

V. EVALUATION

Algorithm PARTITION is an approximation algorithm that seeks to solve the partitioning problem in polynomial time. In order to quantitatively discuss the effectiveness of Algorithm PARTITION we derive a sufficient schedulability condition for Algorithm PARTITION in Theorem 2 below, and use this schedulability condition to derive the approximation bound, also known as resource augmentation bound, of Algorithm PARTITION in Theorem 3. In the context of the two resource partitioning problems the resource augmentation bound can be conceptualized as follows: if an optimal algorithm can schedule a task system onto $m$ processors then an approximation algorithm is guaranteed to schedule the task system onto $m$ processors if the resources are inflated by a factor equal to the resource augmentation bound. Different approximation algorithms can be compared on the basis of their resource augmentation bounds. The smaller the resource augmentation bound the better -closer to optimal- the algorithm.

We would like to stress that the properties described in Theorems 2–3 are not intended to be used as schedulability tests to determine whether Algorithm PARTITION would successfully schedule a given sporadic task system – since the algorithm itself runs efficiently in polynomial time, the “best” (i.e., most accurate) polynomial-time schedulability test for determining whether a particular task system is successfully scheduled by it is to actually run Algorithm PARTITION and check whether it returns PARTITION SUCCEEDED or not. Rather, these properties are intended to provide a quantitative measure of how effective Algorithm PARTITION is vis a vis the performance of an optimal scheduler.

First some definitions: for a given task system $\tau = \{\tau_1, \ldots, \tau_n\}$, let us define the following notation:

\[ u_{\text{max}}(\tau) \triangleq \max_{i=1}^n u_i \]  \hspace{1cm} (4)

\[ u_{\text{sum}}(\tau) \triangleq \sum_{j=1}^n u_j \]  \hspace{1cm} (5)

\[ u_{\text{max}}(\tau) \triangleq \max_{i=1}^n v_i \]  \hspace{1cm} (6)
Intuitively, \( u_{\text{max}}(\tau) \) represents the maximum computation requirement of any individual task, and \( v_{\text{sum}}(\tau) \) represents the total computation requirement of all the tasks in the task system. Similarly, \( u_{\text{max}}(\tau) \) represents the maximum memory requirement of any individual task, and \( v_{\text{sum}}(\tau) \) represents the total memory requirement of all the tasks in the task system.

Lemma 2 follows immediately.

**Lemma 2:** If task system \( \tau \) is feasible (under either the partitioned or the global paradigm) on an identical multiprocessor platform consisting of \( m \) processors each of computing capacity \( \xi \) and available memory \( \xi \), it must be the case that

\[
\xi \geq \max(u_{\text{max}}(\tau), v_{\text{max}}(\tau)) ,
\]

and

\[
m \cdot \xi \geq \max(u_{\text{sum}}(\tau), v_{\text{sum}}(\tau)) .
\]

**Proof:** Observe that

1) No individual task’s computation requirement may exceed the computing capacity of a processor; i.e., it must be the case that \( u_i \leq \xi \).

2) No individual task’s memory requirement may exceed the amount of memory available on each processor; i.e., it must be the case that \( v_i \leq \xi \).

Taken over all tasks in \( \tau \), these observations together yield the first condition.

In the second condition, the requirement that \( m \cdot \xi \geq v_{\text{sum}}(\tau) \) simply reflects the requirement that the cumulative computation requirement of all the tasks in \( \tau \) not exceed the computing capacity of the platform. Similarly, the requirement that \( m \cdot \xi \geq v_{\text{sum}}(\tau) \) reflects the requirement that the total memory required by all the tasks in \( \tau \) not exceed the memory available on the platform.

Lemma 2 above specifies necessary conditions for our partitioning algorithm to successfully partition a sporadic task system; Theorem 2 below specifies a sufficient condition. But first, a technical lemma that will be used in the proof of Theorem 2.

**Lemma 3:** Suppose that Algorithm PARTITION is attempting to schedule task system \( \tau \) on a platform consisting of unit-capacity and unit-memory processors.

A: If \( v_{\text{sum}}(\tau) \leq 1 \), then Condition 2 is always satisfied.

B: If \( v_{\text{sum}}(\tau) \leq 1 \), then Condition 3 is always satisfied.

**Proof:**

The proof of A is straightforward, since violating Condition 2 requires that \( (u_i + \sum_{j \in T(\tau_k)} u_j) \) exceed 1. Similarly, the proof of B follows from the observation that violating Condition 3 requires that \( (v_i + \sum_{j \in T(\tau_k)} v_j) \) exceed 1.

Thus, any implicit-deadline sporadic task system satisfying all of \( u_{\text{sum}}(\tau) \leq 1 \) and \( v_{\text{sum}}(\tau) \leq 1 \) is successfully scheduled by our algorithm. We will describe, in Theorem 2, what happens when one or more of these conditions are not satisfied; Lemmas 4-5 below derive technical results that are used in proving Theorem 2.

**Lemma 4:** Suppose \( u_{\text{sum}}(\tau) > 1 \), let the tasks \( \tau_1, \tau_2, \ldots, \tau_{i-1} \), in \( \tau \), be successfully mapped by the partitioning algorithm onto the available processors. When the partitioning algorithm is attempting to assign \( \tau_i \), if Condition 2 fails on \( m_1 \) processors then the following inequality must hold:

\[
m_1 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i}
\]

**Proof:**

Since none of the \( m_1 \) processors satisfy Condition 2 for task \( \tau_i \), it must be the case that there is not enough remaining computing capacity on each such processor to accommodate the computation requirement of task \( \tau_i \). Therefore, strictly more than \( (1 - u_i) \) of the computing capacity of each such processor has been consumed by the tasks already assigned to these processors; summing over all \( m_1 \) processors and noting that the tasks already assigned \( (\tau_1, \tau_2, \ldots, \tau_{i-1}) \) to these processors is a subset of the tasks in \( \tau \), we obtain the following:

\[
(1 - u_i)m_1 < \sum_{j=1}^{i-1} u_j
\]

\[
\Rightarrow (1 - u_i)m_1 + u_i < \sum_{j=1}^{i} u_j
\]

\[
\equiv (1 - u_i)m_1 + u_i < \sum_{j=1}^{i} u_j \leq \sum_{j=1}^{n} u_j
\]

\[
\equiv m_1 < \frac{u_{\text{sum}}(\tau) - u_i}{1 - u_i} \left( \sum_{j=1}^{n} u_j = u_{\text{sum}}(\tau) \right)
\]

which is as asserted by the lemma.

Note that, according to Lemma 3 (Part A), if \( u_{\text{sum}}(\tau) \leq 1 \) then Condition 2 is always satisfied. In this lemma we consider that Condition 2 fails on \( m_1 \) processors when attempting to map task \( \tau_i \). Therefore, by Lemma 3 (Part A), the value of \( u_{\text{sum}}(\tau) \) should be greater than 1, which is as stated in the Lemma. For \( u_{\text{sum}}(\tau) > 1 \), if \( u_i = u_{\text{max}}(\tau) \), i.e. if task \( \tau_i \) has the maximum computation requirement in \( \tau \), then we obtain the following upper bound on the value of \( m_1 \):

\[
m_1 < \frac{u_{\text{sum}}(\tau) - u_{\text{max}}(\tau)}{1 - u_{\text{max}}(\tau)}
\]

**Lemma 5:** Suppose \( v_{\text{sum}}(\tau) > 1 \), let the tasks \( \tau_1, \tau_2, \ldots, \tau_{i-1} \), in \( \tau \), be successfully mapped by the partitioning algorithm onto the available processors. When the partitioning algorithm is attempting to assign \( \tau_i \), if Condition 3 fails on \( m_2 \) processors then the following inequality must hold:

\[
m_2 < \frac{v_{\text{sum}}(\tau) - v_i}{1 - v_i}
\]
Proof:
The proof is similar to the proof for Lemma 4. Also, just like in Lemma 4 we can show that for \( v_{\text{sum}}(\tau) > 1 \), if \( v_i = v_{\text{max}}(\tau) \), i.e. if task \( \tau_i \) has the maximum memory requirement in \( \tau \), then we obtain the following upper bound on the value of \( m_2 \):

\[
m_2 < \frac{v_{\text{sum}}(\tau) - v_{\text{max}}(\tau)}{1 - v_{\text{max}}(\tau)}
\]  

We now present a sufficient schedulability condition for Algorithm \textsc{Partition} which is applicable when \( u_{\text{sum}}(\tau) > 1 \) and \( v_{\text{sum}}(\tau) > 1 \):

Theorem 2: A sporadic task system \( \tau \) such that \( u_{\text{sum}}(\tau) > 1 \) and \( v_{\text{sum}}(\tau) > 1 \) is successfully scheduled by our algorithm on \( m \) unit-capacity and unit-memory processors, for any \( m \geq \frac{(u_{\text{sum}}(\tau) - u_{\text{max}}(\tau)) + (v_{\text{sum}}(\tau) - v_{\text{max}}(\tau))}{1 - v_{\text{max}}(\tau)} \)

Proof:
Our proof is by contradiction – we will assume that our algorithm fails to partition task system \( \tau \) on \( m \) processors, and prove that, in order for this to be possible, \( m \) must violate Inequality 12 above.

Let us suppose that our partitioning algorithm fails to obtain a partition for \( \tau \) on \( m \) unit-capacity processors. In particular, let us suppose that task \( \tau_i \) cannot be mapped on to any processor. Let \( m_1 \) and \( m_2 \) denote (as in Lemmas 4-5 above) the number of processors on which Conditions 2 and 3 have failed respectively when we attempted to assign \( \tau_i \) to some processor. It is necessary that:

\[
m_1 + m_2 \geq m
\]

(By Lemmas 4 and 5 respectively)

\[
\frac{(u_{\text{sum}}(\tau) - u_{\text{max}}(\tau)) + (v_{\text{sum}}(\tau) - v_{\text{max}}(\tau))}{1 - v_{\text{max}}(\tau)} \geq m_1 + m_2
\]

\[
\Rightarrow \frac{(u_{\text{sum}}(\tau) - u_{\text{max}}(\tau)) + (v_{\text{sum}}(\tau) - v_{\text{max}}(\tau))}{1 - v_{\text{max}}(\tau)} > m
\]

Taking the contrapositive, it follows that the negation of Equation 13 is sufficient to ensure that our partitioning algorithm will successfully partition \( \tau \) on \( m \) unit-capacity and unit-memory processors, as is claimed by the theorem.

A. Resource augmentation bound

The technique of resource augmentation [7] compares the performance of an algorithm with that of a hypothetical optimal one, under the assumption that the algorithm under discussion has access to more resources than the optimal algorithm. Resource augmentation has come to be widely used as a metric for comparing the effectiveness of different approximation algorithms for solving a given scheduling problem: according to this metric, the lower the resource augmentation bound of an algorithm the better -closer to optimal- it is. An algorithm with resource augmentation bound equal to one is, by definition, an optimal algorithm.

Using Theorem 2 above, we now present resource-augmentation characterizations of our partitioning algorithm when it is used for partitioning arbitrary and constrained sporadic task systems.

Theorem 3: Algorithm \textsc{Partition} makes the following performance guarantee: if an implicit-deadline sporadic task system is feasible on \( m \) identical processors each of a particular computing capacity and memory, then Algorithm \textsc{Partition} will successfully partition this task system upon a platform comprised of \( m \) processors that each have \( (3 - \frac{2}{m}) \) times the computing capacity and memory as the original.

Proof:
Let us assume that \( \tau = \{\tau_1, \tau_2, \ldots, \tau_n\} \) is feasible on \( m \) processors each of computing capacity and memory equal to \( \xi \). Since \( \tau \) is feasible on \( m \xi \) speed processors, it follows from Lemma 2 that the tasks in \( \tau \) satisfy the following properties:

\[
u_{\text{max}}(\tau) \leq \xi, v_{\text{max}}(\tau) \leq \xi,
\]

and

\[
u_{\text{sum}}(\tau) \leq m \cdot \xi, v_{\text{sum}}(\tau) \leq m \cdot \xi
\]

Suppose once again that \( \tau \) is successfully scheduled by Algorithm \textsc{Partition} on \( m \) unit-capacity and unit-memory processors.

Now we have four possibilities:

Case 1 \( u_{\text{sum}}(\tau) \leq 1, v_{\text{sum}}(\tau) \leq 1 \): According to Lemma 3 (Part A) and (Part B), Conditions 2 and 3 are always satisfied. For this case Algorithm \textsc{Partition} is an optimal algorithm and the resource augmentation bound is 1.

Case 2 \( u_{\text{sum}}(\tau) \leq 1, v_{\text{sum}}(\tau) > 1 \): According to Lemma 3 (Part A), Condition 2 is always satisfied. Therefore the problem reduces to a problem in which there is only one limited resource. In this case we know from [9, P.176] that the resource augmentation bound is \( 2 - \frac{1}{m} \).

Case 3 \( u_{\text{sum}}(\tau) > 1, v_{\text{sum}}(\tau) \leq 1 \): Same as above.

Case 4 \( u_{\text{sum}}(\tau) > 1, v_{\text{sum}}(\tau) > 1 \):

For this case we know from Theorem 2 that the task system is schedulable if:

\[
m \geq \frac{(u_{\text{sum}}(\tau) - u_{\text{max}}(\tau)) + (v_{\text{sum}}(\tau) - v_{\text{max}}(\tau))}{1 - v_{\text{max}}(\tau)}
\]

We obtain an upper bound on the RHS of the above equation when:

- \( u_{\text{sum}}(\tau) = m \cdot \xi \) and \( v_{\text{sum}}(\tau) = m \cdot \xi \).
- \( u_{\text{max}}(\tau) = \xi \) and \( v_{\text{max}}(\tau) = \xi \).

Therefore, the task system is schedulable if:
\[ m \geq \frac{m \xi - \xi}{1 - \xi} + \frac{m \xi - \xi}{1 - \xi} \]
\[ \equiv m \geq \frac{\xi}{1 - \xi} \left( 2m - 2 \right) \]
\[ \equiv m - m \xi \geq \left( 2m - 2 \right) \xi \]
\[ \equiv \frac{1}{\xi} \geq \left( 3 - \frac{2}{m} \right) \]

which is as claimed in the statement of the theorem.

VI. HEURISTIC IMPROVEMENTS

The description of Algorithm \textit{PARTITION}, and the derivation of its resource augmentation bound, make no assumptions about the order in which the tasks are considered for placement on the processors. In implementing Algorithm \textit{PARTITION}, however, we may want to consider the tasks according to some ordering that enhances the likelihood that a given task system will be successfully partitioned. (For instance, since it is, intuitively speaking, more difficult to place a task that has larger computation and memory requirements, it may be better to consider such tasks for placement earlier, when more computing capacity and memory are available on the processors.)

For any two tasks \( \tau_i \) and \( \tau_j \), let us say that \( \tau_i \succeq \tau_j \) if \( u_i \geq u_j \) and \( v_i \geq v_j \). A straightforward extension of the “decreasing” concept in First-Fit-Decreasing yields the following rule for ordering the tasks:

- If \( \tau_i \succeq \tau_j \) then consider \( \tau_i \) before considering \( \tau_j \) (ties broken arbitrarily).

When we have tasks \( \tau_i \) and \( \tau_j \) such that neither \( \tau_i \succeq \tau_j \) nor \( \tau_j \succeq \tau_i \) hold (i.e., if \( \left( u_i > u_j \right) \) and \( v_i < v_j \)) or \( \left( u_i < u_j \right) \) and \( v_i > v_j \)), there are several possible generalizations to the FFD rule that we can come up with. Inspired by the proof of Theorem 2: consider tasks in descending order of \( f_i \), where \( f_i \) is defined as follows:

\[ f_i \equiv \frac{v_{\text{sum}}(\tau) - v_i}{1 - v_i} + \frac{v_{\text{sum}}(\tau) - u_i}{1 - u_i} \] (14)

Note that, if there are no memory constraints (all the \( v_i \)'s are zero), then ordering as per decreasing \( f_i \) reduces to FFD for partitioning tasks onto processors with limited computing capacity.

VII. EXPERIMENTAL EVALUATION

In this section we experimentally evaluate whether our heuristic for re-ordering tasks improves the schedulability of Algorithm \textit{PARTITION}. But first we compare Algorithm \textit{PARTITION} and the partitioning algorithm described in [5], that efficiently solves an ILP using a “non-integer” relaxation, on the basis of schedulability. We call the latter algorithm, \textit{LPR\textsc{Partiton}}. In order to assess the schedulability of the algorithms, we generated task systems and measured the number of task systems that were successfully scheduled by each algorithm.

Each task system consists of tasks that are characterized by the values \( \left( u_i, v_i \right) \) which correspond to computation and memory requirements respectively. The values \( u_i \) and \( v_i \) were generated using three uniform, three bimodal, and three exponential distributions as described in [12]. The ranges for the uniform distributions were \([0.001, 0.1]\) (light), \([0.1, 0.4]\) (medium), and \([0.5, 0.9]\) (heavy). For the bimodal distributions, utilizations uniformly ranged over \([0.001, 0.5]\) or \([0.5, 0.9]\) with respective probabilities of \(8/9\) and \(1/9\) (light), \(6/9\) and \(3/9\) (medium), and \(4/9\) and \(5/9\) (heavy). For the exponential distributions, utilizations were generated with a mean of 0.10 (light), 0.25 (medium), and 0.50 (heavy). With exponential distributions, points that fell outside the allowed range of \([0, 1]\) were discarded. Please note that for our experiments the values of \( u_i \) and \( v_i \) for each task in a task system were chosen from the same distribution. For example, the values of \( u_i \) and \( v_i \) for all tasks in a task system could be chosen from the uniform light distribution. The maximum total computation and memory requirements of the generated task systems were set to some value \( M \leq m \). We added tasks to a task system, \( \tau \), until either \( u_{\text{sum}}(\tau) \) or \( v_{\text{sum}}(\tau) \) exceeded \( M \) and then discarded the last-added task so that \( u_{\text{sum}}(\tau) \leq M \leq m \) and \( v_{\text{sum}}(\tau) \leq M \leq m \).

We have shown, in Theorem 3, that Algorithm \textit{PARTITION} has a resource augmentation bound of \((3 - \frac{2}{m})\); for large values of \( m \) this bound approaches 3. Algorithm \textit{LPR\textsc{Partiton}} is also known to have a resource augmentation bound of 3. But does this imply that they have comparable schedulability? The resource augmentation bound of an algorithm is one worst-case metric of the algorithm’s performance, in that it indicates that there exists at least one task system that can be scheduled on the given processors by an optimal algorithm that can be scheduled by the algorithm only if the resources on the processors are augmented by a factor equal to the resource augmentation bound. However, two algorithms with equal resource augmentation bounds need not successfully schedule the same task systems.

We conducted a series of simulation experiments to compare the relative performance of Algorithm \textit{PARTITION} and Algorithm \textit{LPR\textsc{Partiton}}, when scheduling randomly-generated task systems. We simulated the partitioning of task systems onto processors with both algorithms and measured the number of task systems that were successfully partitioned by each algorithm. Algorithm \textit{LPR\textsc{Partiton}} can only be used to schedule tasks for which the values \( u_i \) and \( v_i \) are \(< 0.5\). Therefore, for this experiment the values of \( u_i \) and \( v_i \) for all tasks were chosen only from the uniform light and the uniform medium distributions. For \( m = 2, 4, 6, 8 \) processors and for each value of \( M \) starting from a chosen value, \( \leq m/2 \), and incremented
in steps of 0.1 until $M = m$, we generated 100 task systems with values of $u_i$ and $v_i$ for all tasks chosen from the uniform light distribution. We partitioned the task systems onto processors using Algorithm PARTITION and Algorithm LPRPARTITION. We repeated the experiment for tasks chosen from the uniform medium distribution.

The graphs in Figures 2, and 3 show our results for $m = 2$, and $m = 8$ processors. We observe that greater number of task systems were successfully scheduled by Algorithm PARTITION than by Algorithm LPRPARTITION. We also generated the graphs for $m = 4$, and $m = 6$ processors and the results were consistent with the results we obtained for $m = 2$, and $m = 8$ processors. The additional graphs are not included in this paper because of space limitations; they can be found online.\(^1\)

In order to experimentally evaluate whether the heuristic presented in Section VI improves the schedulability of Algorithm PARTITION, we generated task systems and reordered the tasks in each task system according to the heuristic. We then used Algorithm PARTITION to partition both the generated task system and the reordered task system; the difference being only in the order in which the tasks are considered for assignment to processors. To determine schedulability we measured the number of task systems that were successfully partitioned in both cases. For this experiment all distributions were used to generate tasks for task systems. For $m = 2, 4, 6, 8$ processors and for each $M$ starting from a chosen value, $\leq m/2$, and incremented in steps of 0.1 until $M = m$, we generated 100 task systems. Note that, for a set of 100 task systems, the values of $u_i$ and $v_i$ for all tasks were generated from one distribution and then the experiment was repeated for the other distributions. From our results we made the following observations.

Observation 1. In Figure 4, and in additional graphs that can be found at this website\(^1\) we observe that for $m = 2$ processors our heuristic does not significantly improve upon the number of task systems that were successfully partitioned when compared to partitioning with no-reordering. However, as the number of processors increases the number of task systems that were successfully partitioned with our heuristic was greater than the number of task systems that were successfully partitioned with no-reordering. This is because with lesser number of processors there is not much variability in the assignment of tasks onto processors. For example, in the case of two processors, a task can be assigned to either of only two processors. Thus, the order in which the tasks are considered for assignment to processors does not significantly affect the likelihood of the task system being partitioned.

Observation 2. In Figure 5(a), for $m = 8$ processors and uniform heavy distribution, equal number of task systems were partitioned with our heuristic and with no-reordering.

\(^1\)At http://www.cs.unc.edu/~bipasa/additional_graphs.pdf
For the uniform heavy distribution, the values of $u_i$ and $v_i$ for all tasks are $\geq 0.5$. For such tasks only one of each task can be assigned to a processor irrespective of the order in which the tasks are considered for assignment onto processors. Therefore, reordering or not does not affect the schedulability of the task system. For the bimodal and exponential heavy distributions, the values of $u_i$ and $v_i$ for the tasks are in the range $[0.001, 0.9]$ and $[0, 1]$ respectively. For these distributions, Figures 5(b) and 5(c) show an increase in schedulability with our heuristic.

In general we observe that reordering tasks according to our heuristic before partitioning using Algorithm PARTITION increases the schedulability of Algorithm PARTITION. Additional graphs can be found at this website \footnote{Link to website} and the results of these graphs are consistent with the observations we made.

VIII. EXTENDING TO > 2 DISTINCT RESOURCE TYPES

In Sections II-VII above, we have restricted our attention to the partitioning of sporadic task systems upon platforms in which each processor has limited amounts of two resources: computing capacity and local memory. Our results from these sections are easily generalized to platforms in which there are multiple resources on each processor (in addition to computing capacity), each available in limited quantities. In order for these generalizations to hold, it is required that each such additional resource be allocated, in the specified amount, “permanently” to each task throughout the duration of the run-time of the system.
More formally, suppose that there are $\ell$ kinds of resources (in addition to computing capacity). We characterize each sporadic task $\tau_i$ by

- Its computation requirement, using the traditional implicit-deadline sporadic-tasks model: $\tau_i = (C_i,T_i)$
- Its resource requirements $v_{i}[1], v_{i}[2], \ldots, v_{i}[\ell]$, with $v_{i}[p]$ denoting the fraction of the $p$th resource that is locally available on each processor that must be reserved for the exclusive use of this task.

In determining whether such a task $\tau_i$ “fits” on processor $\pi_k$, Algorithm PARTITION must ensure that $\pi_k$ has enough of each of the $\ell$ resources; Condition 3 is therefore replaced by the more general condition:

$$\forall p : 1 \leq p \leq \ell : 1 - \sum_{\tau_j \in \pi_k} v_{j}[p] \geq v_{i}[p]$$  \hspace{1cm} (15)$$

It can be shown that with this generalization to Condition 3, the resource augmentation bound of Theorem 3 becomes

$$2 + \frac{1+\ell}{m}$$

Observe that for $\ell \leftarrow 1$ we have, once again, the system model discussed in Sections II-VII, and the bound above becomes exactly the bounds of Theorem 3.

IX. Conclusions

As embedded devices, such as smart-phones, increasingly come to be implemented upon multicore and multiprocessor platforms, it becomes increasingly important that platform resources be efficiently managed. However, much prior scheduling-theoretic research on obtaining multiprocessor implementations of real-time systems has focused almost exclusively on processor computing capacity, to the exclusion of other resources, such as memory, that are also available in scarce quantities on individual processors or cores. A few results have been obtained (for example, [11] and [4], [5]) concerning the scheduling of systems in which the usage of multiple resources must be simultaneously optimized. The work in this paper further explores the issue of resource allocation and scheduling on platforms that require the simultaneous management of multiple resources. We have presented, and proved both the correctness and effectiveness of an algorithm for partitioning a collection of implicit-deadline sporadic tasks on memory-constrained multiprocessor platforms. Unlike some other algorithms that have been proposed for this problem, the algorithm is easy to implement and has a good run-time computational complexity - linear in the product of the number of tasks in the task system and the number of processors. We have also indicated how the algorithm may be extended to deal with additional resource constraints.

REFERENCES


