# Fast Performance Prediction for Periodic Task Systems *

Xiaobo (Sharon) Hu
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
shu@cse.nd.edu

Gang Quan
Dept. of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
gquan@cse.nd.edu

## ABSTRACT

During design exploration, many implementations of the same system specification may need to be evaluated. In this paper, we present an approach to construct sufficient and necessary conditions for a given system specifications. These conditions can be employed in the design exploration process to rapidly determine if an implementation of the system satisfies the timing constraints. We prove that our conditions always outperform the existing respective conditions. Experimental results are also provided to compare our approach with known scheduling results.

## 1. INTRODUCTION

A challenging issue in hardware-software codesign is rapid design exploration at an early design phase [21, 5]. Since timing estimation must be performed for a large number of design alternatives, fast prediction of the system timing behavior is essential to the success of a design exploration tool. In this paper, we present an efficient method to predict the timing behavior of real-time embedded system. We focus on systems containing periodic tasks (since aperiodic tasks can be handled based on the analysis result for periodic tasks [14]), which are executed on a single processor according to a fix-priority preemptive scheduling scheme. This assumption, although not universal, is true for many embedded systems in practice, particularly for low-cost high-volume consumer products [7].

Many papers have been published that study the problem of predicting if a given real-time system architecture is feasible, i.e., satisfies all the timing constraints (e.g., [1, 2, 8, 9, 13, 16, 22]). Almost all of these results fall into one of the two categories: (i) deriving a closed-form sufficient condition, or (ii) providing an algorithm for checking the feasibility. Employing a sufficient condition from the first category [2, 9], one can rapidly test feasibility of a design alternative. However, since these sufficient conditions are formed so as to be applicable to any periodic task sets, they generally produce rather pessimistic results. That is, many feasible systems do not satisfy these conditions [10, 13]. On the other hand, feasibility-checking algorithms in the second category generally produce more accurate predictions. Unfortunately, these algorithms are much more time consuming.

A somewhat different approach is proposed by Park, $et.al.$, in [17, 18]. Rather than finding sufficient conditions that are applicable to any task set, they present a method to compute a sufficient condition based on the given task periods. However, no comparison is made between the condition derived in [17] and the existing ones. Furthermore, constructing the conditions can be quite costly when the number of tasks is big and the differences between task periods are large. In [18], a simplified approach is proposed, which, however, often produces sufficient conditions that are even more pessimistic than the existing ones in [2, 16].

In this paper, we present our approach to constructing two sufficient conditions for a given system specification (i.e., task timing parameters except the execution time are fixed). Our conditions are obtained by eliminating certain constraints in the problem formulation proposed in [17]. Our first condition is less costly to derive but give exactly the same results as that in [17], and our second condition is much less costly than the first one with little loss in prediction results. We formally prove that both of our conditions lead to better predictions than the existing ones in [2, 16, 18]. We also extend our method to obtain necessary conditions which can help to eliminate those definitely infeasible systems. Experimental results demonstrate that our conditions can even outperform some feasibility-checking algorithms (e.g., [8]). Moreover, the sufficient and necessary conditions for a real-time system to be feasible are derived once the system specification (i.e., task periods and deadlines) is given. Then, for each different implementations (due to the choice of processors and task assignments), the conditions can be readily evaluated with a time complexity being linearly dependent on the number of tasks. Hence, the efficiency of the design exploration process can be greatly improved.

## 2. PRELIMINARY

The system we consider consists of $n$ periodic tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, arranged in the decreasing order of their priorities. Each task is associated with three timing parameters, $T_i, D_i, C_i$. $T_i$ is the shortest time interval between two requests of task $\tau_i$ and is referred to as the *period* of $\tau_i$. $D_i$ denotes the maximum time allowed from initiation to termination of task $\tau_i$ and is referred to as the *deadline* of $\tau_i$. $C_i$ is the time needed for a processor to complete $\tau_i$ without any interruption and is referred to as the execution time of $\tau_i$. We allow that $T_i \geq D_i$. Note that $T_i$ and $D_i$ are fixed when the functional requirements of a system is given but $C_i$ is de-

*This research is supported in part by an External Research Program Grant from Hewlett-Packard Laboratories, and by NSF under grant numbers MIP-9796162 and MIP-9701416.

pendent on the choice of the processor and implementation of $\tau_i$ in an actual design. For a given processor, the processor utilization by $i$ tasks is computed by $U_i = \sum_{j=1}^{i} \frac{C_j}{T_j}$. The tasks are to be scheduled on a single processor by a preemptive fixed-priority scheduler. For ease of presentation, we first assume that the tasks are independent of one another and ignore the communication requirements. We will remove these assumptions in Section 4.

Our goal is to find conditions that can be used to check if a processor definitely can or cannot complete the tasks on or before their deadlines (also referred to as the system is feasible or not). We introduce the following definitions.

DEFINITION 1. *Feasibility upper bound (or simply feasibility bound), $B_i^F$, is a bound on the processor utilization such that the $i$ tasks are definitely feasible if $U_i < B_i^F$.*

DEFINITION 2. *Infeasibility lower bound (or simply infeasibility bound), $B_i^{\overline{F}}$, is a bound on the processor utilization such that the $i$ tasks are definitely not feasible if $U_i > B_i^{\overline{F}}$.*

It follows that the feasibility bound and infeasibility bound can be used to form sufficient and necessary conditions, respectively, for testing feasibility. If an implementation is such that $B_i^F \leq U_i \leq B_i^{\overline{F}}$, whether the implementation is feasible or not cannot be determined by these conditions and a time-consuming feasibility-checking algorithm would be needed. Therefore, it is desirable to have larger feasibility bounds and smaller infeasibility bounds.

The well-known results by Liu and Layland in [16] is a feasibility bound, i.e.,

$$B_n^F = n(2^{\frac{1}{n}} - 1), \tag{1}$$

A better feasibility bound is presented in [2] as follows:

$$B_n^F = \begin{cases} (n-1)(2^{\frac{\delta}{n-1}} - 1) + 2^{1-\delta} - 1 & \delta < 1 - 1/n \\ n(2^{\frac{1}{n}} - 1) & \delta \geq 1 - 1/n \end{cases} \tag{2}$$

where $\delta = \max_{1 \leq i \leq n} S_i - \min_{1 \leq i \leq n} S_i$, and $S_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$. Note that both bounds only apply to systems in which $D_i = T_i$ for $i = 1, \cdots, n$. An often used infeasibility bound is simply $B_i^{\overline{F}} = 1$. As pointed out in the introduction, these bounds can be rather ineffective for many task systems.

Park, $et.al.$, presented an LP-based approach to compute feasibility bounds [17, 18] formulated as follows.

$$\begin{aligned} \textbf{Minimize:} \quad & B_i^F = \sum_{j=1}^{i} \frac{C_j}{T_j} \\ \textbf{Subject To:} \quad & \\ & \sum_{j=1}^{i} C_j \lceil \frac{pT_k}{T_j} \rceil \geq pT_k \\ & \qquad p = 1, 2, \cdots, \lfloor T_i/T_k \rfloor \\ & \qquad k = 1, 2, \cdots, i-1. \\ & \sum_{j=1}^{i} C_j \lceil \frac{D_i}{T_j} \rceil \geq D_i \end{aligned} \tag{3}$$

The idea is to find the minimum processor utilization by $i$ tasks subject to that there is no idle time before the first instance of $\tau_i$ is finished. For each task $\tau_i$, $i = 1, 2, \cdots, n$, an corresponding LP instance can be constructed. By solving the LP instances, one obtains feasibility bounds $B_i^F$ for $i = 1, 2, \cdots, n$. The minimum of all $B_i^F$ is taken as the system-wise feasibility bound. To see why the solution is a valid feasibility bound, let us assume that the tasks are assigned to a particular processor and the resulting processor utilization is $U_i$. If $U_i < B_i^F$, at least one of the constraints in (3) must be violated. That is, the first request of $\tau_i$ as well as higher priority tasks *can be completed* before either one of the $pT_k$'s or $D_i$, which is equivalent to the system being feasible [13].

One drawback of the approach is that the number of constraints in an LP-instance can become very large if the differences between task periods are large. Such task systems may occur in real applications. For example, in the engine control example described in [12], some periods are more than 100 times of some others. For this reason, the same authors modified their LP formulation by keeping only the last constraint and eliminating the rest from (3) [18]. However, it is not difficult to find cases where the bounds obtained from [18] are worse than those in (1) and (2).

# 3. LP-BASED BOUND COMPUTATION

In this section, we present our LP-based approach for computing feasibility and infeasibility bounds and prove that these bounds are better than the existing ones.

## 3.1 Feasibility bound

For computing feasibility bounds, we formulate the problem as an LP problem similar to [17]. However, by making use of some interesting observations, we can reduce the number of constraints used in [17]. The first feasibility bound is obtained by using at most half of the constraints as in [17] while the bound value is the same as that in [17]. The number of constraints used in calculating the second bound equals to the number of tasks (instead of depends on task periods). We will further show that both our bounds are never smaller than those in [2, 16] and are better for most systems.

Observe that the constraints in (3) is constructed for $D_i$ and each $pT_k$, $p = 1, \cdots, \lfloor T_k/T_j \rfloor$ and $k = 1, \cdots, i-1$. We refer to these points as *scheduling points*. One way to reduce the number of constraints is to reduce the unnecessary constraints. Given two constraints

$$\mathbf{H}(p,k): \quad \sum_{j=1}^{i} C_j \lceil \frac{pT_k}{T_j} \rceil \geq pT_k, \tag{4}$$

$$\mathbf{H}(q,l): \quad \sum_{j=1}^{i} C_j \lceil \frac{qT_l}{T_j} \rceil \geq qT_l, \tag{5}$$

We say that $\mathbf{H}(p,k)$ is *redundant* or $\mathbf{H}(q,l)$ *dominates* $\mathbf{H}(p,k)$ if $\frac{\lceil \frac{pT_k}{T_j} \rceil}{pT_k} \geq \frac{\lceil \frac{qT_l}{T_j} \rceil}{qT_l}$ for all $j = 1, 2, \cdots, i$, and denote by $\mathbf{H}(p,k) \preceq \mathbf{H}(q,l)$. Then, $\mathbf{H}(p,k)$ can be removed from the constraint set in (3) since it is automatically satisfied when $\mathbf{H}(q,l)$ is satisfied. According to the definition, we can check if any constraint is redundant. To check all pairs of constraints, the worst-case time complexity is $O(mn \log m)$, where $m$ is the total number of scheduling points and $n$ is the number of tasks. Several observations allowing us to reduce the time needed for checking redundancy are presented as follows (We omit the proofs due to page limit).

LEMMA 1. *Given a set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, consider two scheduling points, $pT_k$ and $qT_l$. If $pT_k = \frac{1}{2}qT_l$, then $\mathbf{H}(p,k) \preceq \mathbf{H}(q,l)$.*

LEMMA 2. *Given a set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, let $\frac{T_k}{T_j} = A(k,j) + \epsilon(k,j)$ for $k, j = 1, \cdots, n$, where $A(k,j)$ is an integer and $0 < \epsilon(k,j) < 1$. If $p \cdot \epsilon(k,j)$ is an integer, then $\frac{\lceil \frac{pT_k}{T_j} \rceil}{pT_k} \leq \frac{\lceil \frac{qT_k}{T_j} \rceil}{qT_k}$ for any $q = 1, 2, \cdots$.*

By applying the above two lemmas, removing redundant constraints can be greatly simplified. In fact, according to Lemma 1, one can immediately remove those constraints corresponding to scheduling points $pT_k$ for $p = 1, \cdots, \lfloor T_i/2T_k \rfloor$ without any comparison. That is, at least half of the constraints can be eliminated from (3) by Lemma 1 alone.

If both the number of tasks and the number of scheduling points are large, the number of constraints left after removing all the redundant constraints may still be quite large. We now construct another LP instance in which the number of constraints equals to the number of tasks and show that the feasibility bound obtained is still very effective. The modified LP is given in the following:

$$\textbf{Minimize:} \qquad B_i^F = \sum_{j=1}^{i} \frac{C_j}{T_j}$$

**Subject to:**

$$\sum_{j=1}^{i} C_j \lceil \frac{\lfloor \frac{D_i}{T_k} \rfloor \cdot T_k}{T_j} \rceil \geq \lfloor \frac{D_i}{T_k} \rfloor \cdot T_k \qquad (6)$$
$$k = 1, 2, \cdots, i-1.$$
$$\sum_{j=1}^{i} C_j \lceil \frac{D_i}{T_j} \rceil \geq D_i$$

To see why the solution of the above LP is a valid feasibility bound, similar argument as in Section 2 can be applied. Assume that the tasks are assigned to a particular processor and the resulting processor utilization is $U_i$. If $U_i < B_i^F$, the given task execution times must violate at least one of the constraints in (6). That is, the first request of $\tau_i$ as well as higher priority tasks *can be completed* before either one of the $(\lfloor \frac{D_i}{T_k} \rfloor \cdot T_k)$ or $D_i$. Thus, the system is feasible as far as $\tau_i$ is concerned.

Since $\lfloor \frac{D_i}{T_k} \rfloor \cdot T_k$ is one of the scheduling points before $D_i$, the constraints in (6) is a subset of those in (3). Hence, it is possible for the bound obtained from (6) to be smaller than that from (3). However, we shall show that the feasibility bound obtained from the modified LP instance is still mostly larger than the bounds given in [2, 16, 18]. Recall that the bound in [18] is obtained by solving an LP which has the same objective function as in (6) but only uses the last constraint in (6). The smaller constraint set results in a large feasible solution region which in turn gives a smaller objective function value. Therefore, the bound value obtained in [18] will always be less than or equal to the bound obtained from (6). Our experimental results show that the two bound values can be quite different for same tasks.

To show that the bound obtained from (6) is larger than or equals to the ones in [2, 16], we only need to compare our bound with that in [2], since the bound in [2] has been proved to be better than that in [16]. We introduce two lemmas for the proof.

LEMMA 3. *Given a set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, assume that $T_1 \leq \cdots \leq T_n \leq 2^\delta T_1$ ($\delta \leq 1$), and $D_i = T_i$. The feasibility bound obtained from (6) is greater than or equal to the bound computed from (2).*

**Proof:** For the given task set, the constraints in our LP instance for computing $B_i^F$ become

$$\sum_{j=1}^{k-1} 2C_j + \sum_{j=k}^{i} C_j \geq T_k \qquad k = 1, \cdots, i \qquad (7)$$

The bound in (2) is obtained by solving the following non-linear optimization problem with $C_j$ and $T_j$ (for $j = 1, \cdots, n$) as variables.

$$\textbf{Minimize:} \qquad U_i(\underline{C}, \underline{T}) = \sum_{j=1}^{i} \frac{C_j}{T_j}$$

**subject to:**

$$\sum_{j=1}^{k-1} 2C_j + \sum_{j=k}^{i} C_j \geq T_k \qquad (8)$$
$$k = 1, \cdots, i$$
$$0 \leq C_j \leq T_j \qquad j = 1, \cdots, i$$
$$T_1 \leq \cdots \leq T_n \leq 2^\delta T_1, \quad \delta \leq 1$$

For the given task set, the solution obtained from solving the LP instance defined by the objective function in (6) and the constraints in (7) definitely satisfies the constraints in (8). That is, it is a feasible solution of (8). Since the non-linear programming problem allows both $C_j$ and $T_j$ to change, $\min U_i(\underline{C}, \underline{T})$ must be less than or equal to $B_i^F$ obtained from our LP instance in (6). □

LEMMA 4. *Given a set of tasks, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, assume that $T_1 \leq \cdots \leq T_n$ and $D_i = T_i$. The feasibility bound obtained from (6) is greater than or equal to the bound computed from (2).*

The proof for Lemma 4 can be obtained by comparing the constraints in (7) with the general constraint set in (6), and shown that any feasible solution satisfying the latter constraint set is also a feasible solution for the former constraint set. Hence, the bound for the task set in Lemma 4 is always greater than or equals to the bound for the system in Lemma 3. (We omit the detailed proof due to the page limit.)

Note that the bound in (2) is valid for the task systems in Lemma 4. However, for systems in which task priorities do not follow the rate monotonic rule (rate monotonic implies that $i < j$ if and only if $T_i < T_j$), the bound in (2) can no longer be used. Our LP formulation in (6), on the other hand, can still be readily applied. Furthurmore, the LP-based approach can be generalized to produce even better bounds for given system specifications. For example, if it is known that the complexity of $\tau_i$ is higher than that of $\tau_j$, we can add an additional constraint, $C_i > C_j$, to the LP in (3) or (6). Such constraints for certain task can be readily obtained from the system specification regardless of the processors to be used. Other constraints such as the minimum and maximum of a task execution time [22] can also be used.

## 3.2 Infeasibility Bound

The above LP formulation can be extended to compute an infeasibility bound on the processor utilization of $i$ tasks. In this case, we

are interested in finding the maximum utilization above which the task set will be infeasible. We construct the following LP instance:

**Maximize:** $B_i^{\overline{F}} = \sum_{j=1}^{i} \frac{C_j}{T_j}$

**Subject to:**

$$\sum_{j=1}^{i} C_j \lceil \frac{D_i}{T_j} \rceil \geq D_i \quad (9)$$

If a processor utilization $U_i$ is greater than $B_i^{\overline{F}}$, the constraint is violated and the tasks cannot be completed before the deadline. Therefore, $B_i^{\overline{F}}$ is a valid infeasibility upper bound.

If there is no other information available, the infeasibility bound computed above is generally very close to one. However, if more information about the task execution time, such as the lower and upper bounds on $C_i$ and the relative magnitude of $C_i$'s, is available, it can be used to formulate additional constraints in the above LP instance, and hence tighter infeasibility bounds can be obtained.

# 4. EXTENSIONS TO THE TASK MODEL

This section discusses the way we handle tasks that need to communicate with each other by sending and receiving data or messages. As in many existing approaches, e.g.,[4, 22], the overhead for such communication can be modeled as a fixed amount of execution times if the two communicating tasks reside on two different hardware components, and is assumed to be zero otherwise. The challenge is to maintain consistency in data transfers.

First, consider the communicating tasks with the same period. If the priorities of the tasks are assigned such that receiving tasks always have lower priorities than the corresponding sending ones, the tasks can be scheduled as if they are independent without causing any data inconsistency. Thus, any analysis result derived for independent task systems are still valid for such communicating task systems and our LP-based method is readily applicable. Though may not be optimal, such an assignment is often used in existing work (e.g., [1, 4, 22]). If a priority assignment violates the above condition, data dependencies must be considered during scheduling. We are working on modifying our approach for such systems.

There exist systems in which the frequencies of sending and receiving tasks can be different. A key requirement in such a system is to avoid shared data being accessed by more than one task at the same time. Such a requirement falls into the category of task synchronization. Common synchronization primitives include semaphores, locks and monitors [20]. The use of these or equivalent methods is necessary to protect the consistency of shared data or to guarantee the proper use of non-preemptable resources. However, their use may jeopardize the ability of the system to meet its timing requirements. Two priority inheritance protocols are proposed to deal with the synchronization problem in [20]. Based on the priority ceiling protocol in [20], the execution of task $\tau_i$ can be delayed by at most the time duration of $Z_i$, the maximum time in which $\tau_i$ can be blocked waiting for a lower priority task to complete the use of shared data. It follows that $Z_i$ can be treated as a part of the execution time of $\tau_i$, i.e., the execution time of $\tau_i$ can be considered as $\hat{C}_i = C_i + Z_i$, where $C_i$ is the original task execution time, and the feasibility and infeasibility bounds can be computed correspondly.

# 5. EXPERIMENTAL RESULTS

In this section, we use some experimental results to compare the performance of our approach with those of several existing approaches, including the LP-based bound computation approach by Park, et. al. [17], another bound-based approach in [2], and a new scheduling algorithm in [8].

The task sets in our experiment is constructed as follows. A task set contains the number of tasks between 10 and 70. For task sets with a given number of tasks, we construct 10 groups, each of which contains 100 task sets. The task sets within one group have the same task periods but different execution times (which models the design exploration process examining a number of different implementations). Both task periods and execution times are randomly generated. Only task sets with utilization higher than the bound given by Liu and Layland [16] are used in the experiment, since all these algorithms are able to correctly predict the schedulability of the task set with utilization lower than Liu and Layland bound [16].

Two measures, the prediction ratio and running time, are used in evaluating the performance of different approaches. Assume the number of feasible task sets predicted by one approach is $M$. Using the exhaustive method discussed in [13], the number of feasible task sets can be exactly determined and let this number be $N$. The prediction ratio is thus defined as $M/N$. It captures the quality of an approach. The average prediction ratio over different period groups of task sets is used in the experiment to eliminate the possible random effect. The running time of an approach includes both the bound computation time (if appropriate) and the feasibility checking time.

The experimental results for quality and running time of these algorithms are presented in Figure 1 and Table 1, respectively. Here, **LP-0** refers to the results by [17], **LP-1** refers to the approach which applies Lemma 1 to eliminate half of the constraints from the linear programming problems in [17], **LP-2** refers to the approach shown in (6), **DCT** refers to the results by the the algorithm in [8], and **Burch** refers to the results by [2].

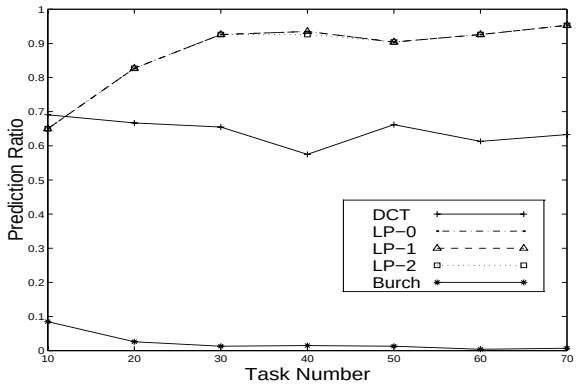


**Figure 1: Quality of different approaches**

Figure 1 shows that the LP-based approaches, i.e. **LP-0**,**LP-1**, and **LP-2**, greatly outperform **Burch**, and lead to much higher prediction ratios than **DCT** in most cases, especially for task sets with a large number of tasks. Furthermore, it is interesting to notice that **LP-2** can save CPU time dramatically with little loss of quality, compared with **LP-0** and **LP-1**. Even by simply eliminating the redundant constraints, **LP-1** can save a large amount of CPU time compared with **LP-0**. For example, when task number is 70, **LP-1** only needs about 35% of the CPU time that **LP-0** takes. Park's modified approach [18] is also tested in our program. Task sets

are randomly generated with utilization near but less than Liu and Layland bound [16]. With that approach, only 3% of the task sets can be correctly predicted to be schedulable when each task set contains 20 tasks.

| CPU Time (s) | | Algorithm | | | | |
|---|---|---|---|---|---|---|
| | | DCT | LP-0 | LP-1 | LP-2 | Burch |
| Number of Tasks | 10 | 0.54 | 7.95 | 7.9 | 7.55 | 0.12 |
| | 20 | 1.83 | 32.86 | 23.66 | 17.17 | 0.18 |
| | 30 | 3.96 | 65.12 | 45.61 | 29.14 | 0.15 |
| | 40 | 6.96 | 224.97 | 116.44 | 44.35 | 0.12 |
| | 50 | 10.53 | 394.08 | 203.26 | 65.09 | 0.16 |
| | 60 | 14.92 | 950.44 | 408.13 | 92.9 | 0.13 |
| | 70 | 20.3 | 2812.96 | 995.73 | 132.54 | 0.2 |

**Table 1: Running time of different approaches**

From Table 1, it seems that the LP-based approaches take much longer CPU time than **DCT** does. However, the bound calculation time, which accounts for almost 100% of the total CPU time for the LP-based approaches, is counted for 10 times in Table 1 , while in design space exploration, after the periods of the tasks are given, only one bound calculation is needed. Moreover, after the utilization bounds are available, the complexity for feasibility checking is only $O(n)$ for the LP-based approaches, where it is $O(n^3)$ for **DCT** [8]. Hence, our LP-based approach is more desirable in design space exploration in terms of both quality and running time.

# 6. CONCLUSION

We have presented an approach to constructing sufficient and necessary conditions for a given system specification. These conditions can be used in the design exploration process to rapidly determine the feasibility of an design alternative. We have formally proved that our conditions always lead to better prediction results, in terms of the percentage of correctly predicted result, than the existing respective conditions. Furthermore, the experimental data have shown that our conditions can even outperform more costly feasibility-checking algorithms.

We emphasize that our approach is especially appealing to design space exploration, where system specification is given and many implementations of the systems need to be compared. The feasibility conditions in our approach are are effective because they are derived for a given system specification, instead of for general real-time systems. Furthermore, these conditions are efficient as they can be applied to very rapidly evaluate the feasibility of a given implementation.

Our current results are applicable to systems with communications among tasks. However, we require that the task priorities be assigned without violating the precedence constraints and we are working on extending our approach to general task systems.

# 7. REFERENCES


[1] N. Audsley, A. Burns, M. Richardson, K. Tindell and A.J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284-292, 1993.

[2] A. Burchard, J.Liebeherr, Y. Oh and S.H. Son, "New strategies for assigning real-time tasks to multiprocessor systems," *IEEE Transactions on Computers*, vol. 44, no. 12, pp. 1429-1442, December, 1995.

[3] S.-T. Cheng and A.K. Agrawala, "Allocation and scheduling of real-time periodic tasks with relative timing constraints," *Proceedings of the Second International Workshop on Real-Time Computing Systems and Applications*, pp. 210-217, 1995.

[4] R.P. Dick and J.K. Jha, "MOGAC: A multiobjective genetic algorithm for the co-synthesis of hardware-software embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 920-935, vol. 17, no. 10, Oct 1998.

[5] D.D. Gajski, F. Vahid, S. Narayan and J. Gong, *Specification and Design of Embedded Systems*, Prentice Hall, Englewoo Cliffs, New Jersey, 1994.

[6] R.K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 29-40, September 1993.

[7] W.A. Halang and A.D. Stoyenko, "Next generation of real-time operating systems: industrial perspective," *Proceedings of the NATO Advanced Study Institute on Real Time Computing*, pp. 595-596, 1994.

[8] C.-C. Han and H.-Y. Tyan "A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms," *Proceedings of the Real-Time Systems Symposium, pp. 36-45, 1997.*

*[9] M.G. Harbour, M.H.Klein and J.P. Lehoczky, "Timing analysis for fixed-priority scheduling of hard real-time systems,"* IEEE Transactions on Software Engineering, *vol. 20, no. 1, pp. 13-28, January, 1994.*

*[10] X. Hu and R. Sambandam, "Predicting timing behavior in architectural design exploration of real-time embedded systems,"* Proceedings of the 34th IEEE/ACM Design Automation Conference, *June 1997, pp. 157-160.*

*[11] J. P. Huang, "Modeling of software partition for distributed real-time applications",* IEEE Transactions on Software Engineering, *pp. 1113-1126, October 1985.*

*[12] X. Hu and J.G. D'Ambrosio, "Hardware/software partitioning for real-time embedded systems," Journal of Design Automation for Embedded Systems. vol. 2, no. 3/4, pp. 339-358, 1997.*

*[13] J. Lehoczky, L. Sha and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior,"* Proceedings of the 1989 IEEE Real-time System Symposium, *pp. 166-171, 1989.*

*[14] J. Lehoczky and S. Ramos-Thue, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems,"* Proceedings of the 11992 IEEE Real-time System Symposium, *pp. 110-123, 1992.*

*[15] J. Y-T., Leung, "A new algorithm for scheduling periodic, real-time tasks",* Algorithmica, *vol. 4, pp. 209-219, 1989.*

*[16] C. L. Liu, and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment,"* Journal of the ACM, *vol. 20, no. 1, pp. 46-61, 1973.*

*[17] D. Park, S. Natarajan, A. Kanevsky, and M.J. Kim, "A generalized utilization bound test for fixed-priority real-time scheduling,"* Proceedings of the Second International Workshop on Real-Time Computing Systems and Applications, *pp. 73-76, Oct. 1995.*

*[18] D. Park, S. Natarajan, and A. Kanevsky, "Fixed-priority scheduling of real-time systems using utilization bounds,"* Journal of Systems Software, *vol. 33, pp. 57-63, 1996.*

*[19] K. Ramamritham, and J. A. Stankovic, "Dynamic task scheduling in distributed real-time systems",* IEEE Software, *vol. 1, no. 3, pp. 65-75, July, 1984.*

*[20] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization,"* IEEE Transactions on Computers, *vol. 39, no. 9, pp. 1175-1185, Sept. 1990.*

*[21] W. Wolf, "Hardware-software co-design of embedded systems,"* Proceedings of the IEEE, *vol. 82, no. 7, pp. 967-989, July 1994.*

*[22] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems,"* Proceedings of the International Conference on Computer Design (ICCD'95), *pp. 64-69, October 1995.*