Task Mapping and Priority Assignment for Soft Real-Time Applications under Deadline Miss Ratio Constraints

SORIN MANOLACHE, PETRU ELES, and ZEBO PENG
Linköping University

Both analysis and design optimisation of real-time systems has predominantly concentrated on considering hard real-time constraints. For a large class of applications, however, this is both unrealistic and leads to unnecessarily expensive implementations. This paper addresses the problem of task priority assignment and task mapping in the context of multiprocessor applications with stochastic execution times and in the presence of constraints on the percentage of missed deadlines. We propose a design space exploration strategy together with a fast method for system performance analysis. Experiments emphasize the efficiency of the proposed analysis method and optimisation heuristic in generating high-quality implementations of soft real-time systems with stochastic task execution times and constraints on deadline miss ratios.

Categories and Subject Descriptors: B.8.0 [**Hardware**]: Performance and Reliability—*General*; C.4 [**Computer Systems Organization**]: Performance of Systems—*Performance attributes*; D.4.7 [**Software**]: Operating Systems—*Organization and Design*

General Terms: Performance, Theory

Additional Key Words and Phrases: Schedulability analysis, soft real-time systems, stochastic task execution times, mapping, priority assignment

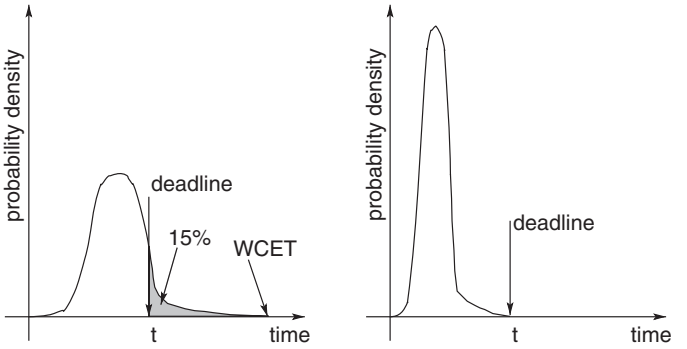
ACM Reference Format:

Manolache, S., Eles, P., and Peng, Z. 2008. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. ACM Trans. Embedd. Comput. Syst. 7, 2, Article 19 (February 2008), 35 pages. DOI = 10.1145/1331331.1331343 http://doi.acm.org/10.1145/1331331.1331343

1. INTRODUCTION

For the large majority of applications, if not all, the task execution times are not fixed, but vary from one activation of the task to the other. This variability may be caused by application-dependent factors (data-dependent loops and branches) [Hughes et al. 2001], architectural factors (unpredictable cache

Authors' addresses: Sorin Manolache, Petru Eles, and Zebo Peng, Linköping University, Sweden.



(a) Slow processor (b) Two times faster processor

Fig. 1. Execution time probability density.

and pipeline behavior, dynamic branch predictors), or environment-dependent factors (network load, for example). In the case of safety-critical applications (avionics, automotive, medicine, and nuclear plant control systems), the designers have to deal with the worst-case scenario, in particular, with worst-case task execution times (WCET). An impressive amount of research results, both for analyzing and designing these systems has been published [Buttazzo 1997].

Designing the system based on the worst-case execution times (WCET) guarantees that no timing requirement is broken. However, for large classes of applications, the soft real-time systems, breaking a timing requirement, though not desirable, is tolerated provided that this happens with a sufficiently low probability. Typical applications are, for example, telecommunications, multimedia applications like MPEG and JPEG, and audio encoding. Thus, for example, Ng et al. report that 1–4 B-frames out of a group-of-pictures (a group of 12 frames ordered as IBBPBBPBBPBB, where I, B, and P are frame types defined by the MPEG standard) may be lost in an MPEG decoding without compromising the required QoS [Ng et al. 2002]. While in the case of safety critical systems, the designers stress safety at the expense of product cost, in the case of soft real-time systems, cost reduction can be a strong incentive for using cheap architectures.

While in hard real-time analysis the tasks are assumed to execute for the amount of time that leads to the worst-case scenario, in soft real-time analysis task execution time probability distributions are preferred in order to be able to determine execution time combinations and their likelihoods. These distributions can be extracted from performance models [van Gemund 1996] by means of analytic methods or simulation, and profiling [van Gemund 2003b; Gautama and van Gemund 2000; Gautama 1998]. Obviously, the worst-case task execution time model is a particular case of such a stochastic one.

Let us consider a cheap processor and a task that runs on it. The probability density function of the task execution time (ETPDF) on the cheap processor is depicted in Figure 1a. If the imposed deadline of the task is t, as shown in the figure, then the cheap processor cannot guarantee that the task will always meet its deadline, as the WCET of the task exceeds the deadline. If no deadline misses were tolerated, a faster and more expensive processor would be needed.

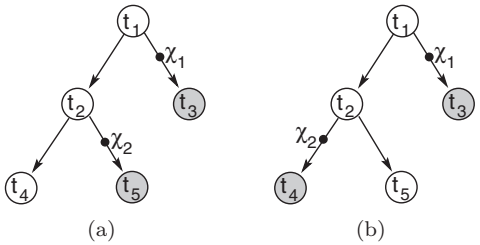


Fig. 2. Motivational example.

The ETPDF of the task on the faster processor is depicted in Figure 1b. In this case, the more expensive processor guarantees that no deadlines are missed. However, if a miss deadline ratio of, at most, 15% is tolerated, then even the cheaper processor would suffice.

The problem of finding the deadline miss ratios, given a hardware platform and an application, is not trivial and has attracted relatively recent research work both for mono [Tia et al. 1995; Lehoczky 1996; Sun et al. 1997; Zhou et al. 1999; Gardner 1999; Gardner and Liu 1999; Hu et al. 2001; Manolache et al. 2001, 2004b; Díaz et al. 2002; Burns et al. 2003] and for multiprocessor systems [Lehoczky 1997; Kalavade and Moghé 1998; Nissanke et al. 2002; Manolache et al. 2002, 2004a; van Gemund 2003a].

This work addresses the complementary problem: given a multiprocessor hardware architecture and a functionality as a set of task graphs, find a task mapping and priority assignment such that the deadline miss ratios satisfy imposed constraints.

A naïve approach to this problem would be to optimize the system based on fixed execution time models (average, median, worst case execution time, etc.) and to hope that the resulting designs would be optimal or close to optimal from the point of view of the percentage of missed deadlines. The following example illustrates the pitfalls of such an approach and emphasizes the need for an optimisation technique, which considers the stochastic execution times. Let us consider the application in Figure 2a. The circles denote the tasks and their shades denote the processors they are mapped onto. The marked edges show the interprocessor communication. The arrows between the tasks indicate their data dependencies. All the tasks have period 20 and the deadline of the task graph is 18. Tasks t_1, t_2, t_3, and t_4 have constant execution times of 1, 6, 7, and 8, respectively. Task t_5 has a variable execution time whose probability is uniformly distributed between 0 and 12. Hence, the average (expected) execution time of task t_5 is 6. The interprocessor communication takes 1 time unit per message. Let us consider the two mapping alternatives depicted in Figure 2a and 2b, respectively. The two Gantt diagrams in Figure 3a and 3b depict the execution scenarios corresponding to the two considered mappings if the execution of task t_5 took the expected amount of time, that is 6. The shaded rectangles depict the probabilistic execution of t_5. A mapping strategy based on the average execution times would select the mapping in Figure 2a as it leads to a shorter response time (15 compared to 17). However, in this case, the worst-case execution time of the task graph is 21. The deadline miss ratio

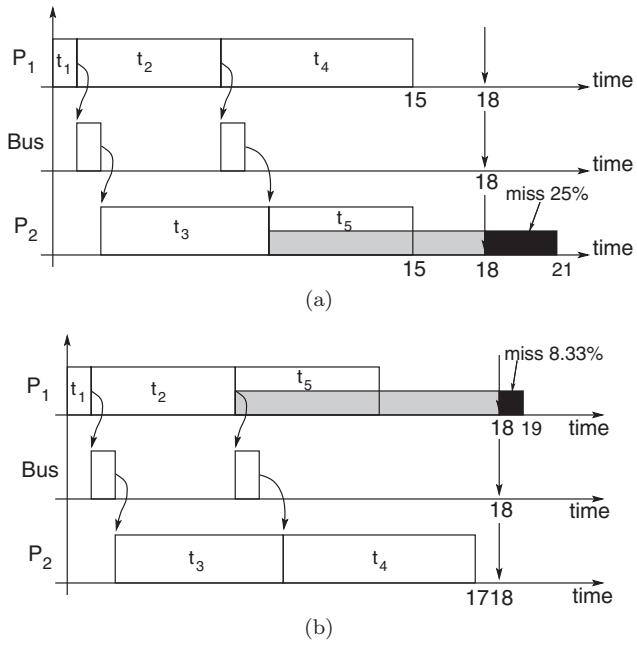


Fig. 3. Gantt diagrams of the two mapping alternatives in Figure 2.

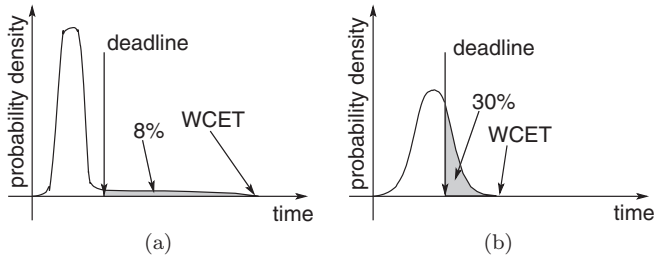


Fig. 4. Motivational example.

of the task graph is $3/12 = 25\%$. If we took into consideration the stochastic nature of the execution time of task t_5, we would prefer the second mapping alternative, because of the better deadline miss ratio of $1/12 = 8.33\%$. If we considered worst-case response times, instead of average ones, then we would chose the second mapping alternative, the same as the stochastic approach. However, approaches based on worst-case execution times can be dismissed by means of very simple counterexamples.

Let us consider a task τ, which can be mapped on processor P_1, or on processor P_2. P_1 is a fast processor with a very deep pipeline. Because of its pipeline depth, mispredictions of target addresses of conditional jumps, though rare, are severely penalized. If τ is mapped on P_1, its ETPDF is shown in Figure 4a. The long and flat density tail corresponds to the rare, but expensive, jump target address misprediction. If τ is mapped on processor P_2, its ETPDF is shown in

Figure 4b. Processor P_2 is slower with a shorter pipeline. The WCET of task τ on processor P_2 is smaller than the WCET if τ ran on processor P_1. Therefore, a design space exploration tool based on the WCET would map task τ on P_2. However, as Figure 4 shows, the deadline miss ratio in this case is larger than if task τ was mapped on processor P_1.

The remainder of the paper is structured as follows. The next section presents some of the related work in the area and underlines our contribution. Section 3 introduces our system model and gives the problem formulation. Section 4 presents the design space exploration strategy detailing the neighborhood restriction heuristic. Section 5 describes the fast approximate method for system analysis. Section 6 presents a set of experiments we conducted in order to evaluate and demonstrate the efficiency of the proposed heuristic. Finally, Section 7 draws conclusions.

2. RELATED WORK AND CONTRIBUTIONS

An impressive amount of work has been carried out in the area of schedulability analysis of applications with worst-case task execution times both for monoprocessor platforms [Liu and Layland 1973; Bini et al. 2001; Leung and Whitehead 1982; Lehoczky et al. 1989; Audsley et al. 1991; Blazewicz 1976; Audsley et al. 1993b; Sun et al. 1997; Spuri and Stankovic 1994; González Harbour et al. 1991] and multiprocessor platforms [Sun and Liu 1995; Sun 1997; Audsley 1991; Audsley et al. 1993a; Tindell and Clark 1994; Palencia Gutiérrez and González Harbour 1998] under fairly general assumptions.

Fewer publications address the analysis of applications with stochastic task execution times. Moreover, most of them consider relatively restricted application classes, limiting their focus on monoprocessor systems, or on exponential task execution time probability distribution functions. Some approaches address specific scheduling policies or assume high-load systems.

Burns et al. [Burns et al. 1999] address the problem of a system breaking its timeliness requirements because of transient faults. In their case, the execution time variability stems from task reexecutions. The shortest interval between two fault occurrences such that no task exceeds its deadline is determined by sensitivity analysis. The probability that the system exceeds its deadline is given by the probability that faults occur at a faster rate than the tolerated one. Broster et al. [Broster et al. 2002] propose a different approach to the same problem. They determine the response time of a task given that it reexecutes $k \in \mathbb{N}$ times resulting for faults. Then, in order to obtain the probability distribution of the response time, they compute the probability of the event that k faults occur. The fault occurrence process is assumed to be a Poisson process in both of the cited works. Burns et al. [2003] extend Broster's approach in order to take into account statistical dependencies among execution times. While their approaches are applicable to systems with sporadic tasks, they are unsuited for the determination of task deadline miss probabilities of tasks with generalised execution time probability distributions. Their approaches are also confined to sets of independent tasks implemented on monoprocessor systems.

Bernat et al. [Bernat et al. 2002] address a different problem. They determine the frequency with which a single task executes for a particular amount of time, called execution time profile. This is performed based on the execution time profiles of the basic blocks of the task. The strength of this approach is that they consider statistical dependencies among the execution time profiles of the basic blocks. However, their approach would be difficult to extend to the deadline miss ratio analysis of multitask systems because of the complex interleaving that characterizes the task executions in such environments. This would be even more difficult in the case of multiprocessor systems.

Abeni and Buttazzo's work [Abeni and Butazzo 1999] addresses both scheduling and performance analysis of tasks with stochastic parameters. Their focus is on how to schedule both hard and soft real-time tasks on the same processor, in such a way that the hard ones are not disturbed by ill-behaved soft tasks. The performance analysis method is used to assess their proposed scheduling policy (constant bandwidth server) and is restricted to the scope of their assumptions.

Tia et al. [Tia et al. 1995] assume a task model composed of independent tasks. Two methods for performance analysis are given. One of them is just an estimate and is demonstrated to be overly optimistic. In the second method, a soft task is transformed into a deterministic task and a sporadic one. The latter is executed only when the former exceeds the promised execution time. The sporadic tasks are handled by a server policy. The analysis is carried out on this particular model.

Gardner et al. [Gardner 1999; Gardner and Liu 1999], in their stochastic time demand analysis, introduce worst-case scenarios with respect to task release times in order to compute a lower bound for the probability that a job meets its deadline. Their approach, however, does not consider data dependencies among tasks and applications implemented on multiprocessors.

Zhou et al. [Zhou et al. 1999] and Hu et al. [Hu et al. 2001] root their work in Tia's. However, they do not intend to give per-task guarantees, but characterize the fitness of the entire task set. Because they consider all possible combinations of execution times of all requests up to a time moment, the analysis can be applied only to small task sets because of complexity reasons.

Díaz et al. [Díaz et al. 2002] derive the expected deadline miss ratio from the probability distribution function of the response time of a task. The response time is computed based on the system-level backlog at the beginning of each hyperperiod, i.e., the residual execution times of the jobs at those time moments. The stochastic process of the system-level backlog is Markovian and its stationary solution can be computed. Díaz et al. consider only sets of independent tasks and the task execution times may assume values only over discrete sets. In their approach, complexity is mastered by trimming the transition probability matrix of the underlying Markov chain or by deploying iterative methods, both at the expense of result accuracy. According to the published results, the method is exercised only on extremely small task sets.

Kalavade and Moghé [Kalavade and Moghé 1998] consider task graphs where the task execution times are arbitrarily distributed over discrete sets. Their analysis is also based on Markovian stochastic processes. Each state in the process is characterized by the executed time and lead time. The analysis is

performed by solving a system of linear equations. Because the execution time is allowed to take only a finite (most likely small) number of values, such a set of equations is small.

Kim and Shin [Kim and Shin 1996] consider applications that are implemented on multiprocessors and modeled them as queueing networks. They restricted the task execution times to exponentially distributed ones, which reduces the complexity of the analysis. The tasks were considered to be scheduled according to a particular policy, namely first-come-first-served (FCFS). The underlying mathematical model is then the appealing continuous time Markov chain.

In previous work [Manolache et al. 2004b], we presented a memory and time-efficient approach for the deadline miss ratio analysis. While the approach is significantly less restrictive than previous approaches from the assumptions point of view, it is restricted to monoprocessor systems. Therefore, we introduced a different approach [Manolache et al. 2002] in which we overcome this restriction. Nevertheless, the analysis time is too large in order to use the algorithm inside an optimization loop, which is the goal of this paper.

Most of the mentioned approaches are inefficient if used inside optimization loops. In some cases, they work only on severely restricted systems (e.g., small number of tasks, or particular execution time probability distribution functions, or monoprocessor systems), while in other cases they are too slow. Therefore, in this work we propose a fast deadline miss ratio analysis approach that balances speed with analysis accuracy. The approach is discussed in detail in Section 5.

While there still exists a sizable amount of work on analysis of real-time applications with stochastic task execution times, almost no publication addresses the complementary problem: Given a system with stochastic task execution times, optimize the design such that performance metrics are maximized.

Hua et al. [Hua et al. 2003] propose an approach that reduces power consumption of a system while keeping task graph deadline miss ratios below imposed thresholds. The deadline miss ratio analysis deployed by Hua et al. is based on a heuristic that computes upper bounds of task execution times such that the task graph deadline is not violated. Once such a task execution time upper limit assignment is determined, the probability that every task runs less than or equal to its upper execution time limit is determined. If an assignment of upper limits on task execution times is found that leads to successful completion probabilities above the imposed threshold, a task execution order together with task-specific execution voltages are determined. Their approach only considers monoprocessor systems and, therefore, task mapping is not of interest. In this case, the execution time of a task graph is simply the sum of the execution times of its tasks, which greatly simplifies the problem. Another simplifying assumption is that Hua et al. consider discrete execution time probability distributions.

The contribution of our work consists of the following aspects.

1. We show that the stochastic rather than average character of execution times has to be considered in order to minimize deadline miss ratios.

2. We propose a fast heuristic for task mapping and priority assignment with the goal to optimize the deadline miss ratios of tasks and task graphs executed on multiprocessor systems.
3. We introduce a fast approximate analysis algorithm for the deadline miss ratios of tasks and task graphs. This algorithm is used inside the optimization loop of the proposed heuristic.

3. SYSTEM MODEL AND PROBLEM FORMULATION

3.1 Architecture Model

The hardware model consists of a set of *processing elements*. These can be programmable processors of any kind (general purpose, controllers, DSPs, ASIPs, etc.). Let $PE = \{PE_1, PE_2, \ldots, PE_p\}$ denote the set of processing elements. A *bus* may connect two or more processing elements in the set PE. Let $B = \{B_1, B_2, \ldots, B_l\}$ denote the set of buses. Data sent along a bus by a processing element connected to that bus may be read by all processing elements connected to that bus.

Unless explicitly stated, the two types of hardware resources, processing elements, and buses, will not be treated differently, and, therefore, will be denoted with the general term of *processors*. Let $M = p + l$ denote the number of processors and let $P = PE \cup B = \{P_1, P_2, \ldots, P_M\}$ be the set of processors.

3.2 Application Model

The functionality of an application is modeled as a set of *processing tasks*, denoted with $t_1, t_2, \ldots, t_n$. Let PT denote the set of processing tasks. Processing tasks are graphically represented as large circles, as shown in Figure 2a.

Processing tasks may pass messages to each other. The passing of a message between tasks mapped to different processors is modeled as a *communication task*, denoted with χ. Let CT denote the set of communication tasks. They are graphically depicted as small disks, as shown in Figure 2a.

Unless explicitly stated, the processing and the communication tasks will not be differently treated and, therefore, will be denoted with the general term of *tasks*. Let N be the number of tasks and $T = PT \cup CT = \{\tau_1, \tau_2, \ldots, \tau_N\}$ denote the set of tasks.

Data dependencies are graphically depicted as arrows from the sender task to the receiver task, as shown in Figure 2a.

The task that sends the message is the *predecessor* of the receiving task, while the receiving task is the *successor* of the sender. The set of predecessors of task τ is denoted with $^\circ\tau$, while the set of successors of task τ with τ°. A communication task has exactly one predecessor and one successor and both are processing tasks. For illustration (Figure 2a), task t_2 has one predecessor, namely, the processing task t_1, and it has two successors, namely, tasks t_4 and χ_2.

Tasks with no predecessors are called *root* tasks, while tasks with no successors are called *leaf* tasks. In Figure 2a, task t_1 is a root task, while tasks t_3, t_4, and t_5 are leaf tasks.

Let us consider a sequence of tasks $(\tau_1, \tau_2, \ldots, \tau_k)$, $k > 1$. If there exists a data dependency between tasks τ_i and τ_{i+1}, $\forall 1 \leq i < k$, then the sequence $(\tau_1, \tau_2, \ldots, \tau_k)$ forms a *computation path* of length k. We say that the computation path leads from task τ_1 to task τ_k. Task τ_i is an *ancestor task* of task τ_j if there exists a computation path from task τ_i to task τ_j. Complementarily, we say that task τ_i is a *descendant task* of task τ_j if there exists a computation path from task τ_j to task τ_i. We do not allow circular dependencies, i.e., no task can be both the ancestor and the descendant of another task. In Figure 2a, (t_1, t_2, χ_2, t_5) is an example of a computation path of length 4 and task t_2 is an ancestor of tasks t_4, t_5, and χ_2.

The set of tasks of an application is partitioned into g partitions. Any two tasks within the same partition have a common ancestor or a common descendant or are in a predecessor–successor relationship. Two tasks belonging to different partitions have no common ancestor, nor any common descendant, neither are they in a predecessor–successor relationship.

The task partitions are denoted V_1, V_2, ..., V_g. An application consists of a set $\Gamma = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_g\}$ of g *task graphs*, $\Gamma_i = (V_i, E_i \subset V_i \times V_i)$, $1 \leq i \leq g$. A directed edge $(\tau_a, \tau_b) \in E_i$, $\tau_a, \tau_b \in V_i$, represents the data dependency between tasks τ_a and τ_b, denoted $\tau_a \rightarrow \tau_b$.

The application in Figure 2a consists of a single task graph that includes all the processing and communication tasks.

Task *instantiations* (also known as *jobs*) *arrive* periodically. The ith job of task τ is denoted (τ, i), $i \in \mathbb{N}$.

Let $\Pi_T = \{\pi_i \in \mathbb{N} : \tau_i \in T\}$ denote the set of *task periods*, or job interarrival times, where π_i is the period of task τ_i. Instantiation $u \in \mathbb{N}$ of task τ_i demands execution (the job is *released* or the job *arrives*) at time moment $u \cdot \pi_i$. The period π_i of any task τ_i is assumed to be a common multiple of all periods of its predecessor tasks (π_j divides π_i, where $\tau_j \in {}^\circ\tau_i$). Let k_{ij} denote $\frac{\pi_i}{\pi_j}$, $\tau_j \in {}^\circ\tau_i$. Instantiation $u \in \mathbb{N}$ of task τ_i may start executing if, and only if, instantiations $u \cdot k_{ij}, u \cdot k_{ij} + 1, \ldots, u \cdot k_{ij} + k_{ij} - 1$ of tasks τ_j, $\forall \tau_j \in {}^\circ\tau_i$, have completed their execution.

Let $\Pi_\Gamma = \{\pi_{\Gamma_1}, \pi_{\Gamma_2}, \ldots, \pi_{\Gamma_g}\}$ denote the set of *task graph periods*, where π_{Γ_j} denotes the period of the task graph Γ_j. π_{Γ_j} is equal to the least common multiple of all π_i, where π_i is the period of τ_i and $\tau_i \in V_j$. Task $\tau_i \in V_j$ is instantiated $J_i = \frac{\pi_{\Gamma_j}}{\pi_i}$ times during one instantiation of task graph Γ_j. The kth instantiation of task graph Γ_j, $k \geq 0$, denoted (Γ_j, k), is composed of the jobs (τ_i, u), where $\tau_i \in V_j$ and $u \in \{k \cdot J_i, k \cdot J_i + 1, \ldots, k \cdot J_i + J_i - 1\}$. In this case, we say that task instantiation (τ_i, u) belongs to task graph instantiation (Γ_j, k) and we denote it with $(\tau_i, u) \in (\Gamma_j, k)$.

The model, where task periods are integer multiples of the periods of predecessor tasks, is more general than the model, assuming equal task periods all for tasks in the same task graph. This is appropriate, for instance, when modeling protocol stacks. For example, let us consider a part of baseband processing on the GSM radio interface [Mouly and Pautet 1992]. A data frame is assembled out of four radio bursts. One task implements the decoding of radio bursts. Each time a burst is decoded, the result is sent to the frame-assembling task.

Once the frame-assembling task gets all the needed data, that is, every four invocations of the burst decoding task, the frame-assembling task is invoked. This way of modeling is more modular and natural than a model assuming equal task periods, which would have crammed the four invocations of the radio burst decoding task in one task. We think that more relaxed models than ours, with regard to relations between task periods, are not necessary, as such applications would be more costly to implement and are unlikely to appear in common engineering practice.

3.3 Mapping and Execution Times

Processing tasks are *mapped* on processing elements and communication tasks are mapped on buses. All instances of a processing task are executed by the same processing element on which the processing task is mapped. Analogously, all instances of a message are conveyed by the bus on which the corresponding communication task is mapped.

Our heuristic for finding a task mapping is transformational, i.e., it starts from an initial mapping that it transforms in every iteration until a mapping is found that fulfills the problem constraints. Hence, the heuristic operates in every iteration on a system with fully mapped functionality and communication. Therefore, we need to take the mapping into consideration when describing the system model.

Let $MapP : PT \rightarrow PE$ be a surjective function that maps processing tasks on the processing elements. $MapP(t_i) = P_j$ indicates that processing task t_i is executed on the processing element P_j. Let $MapC : CT \rightarrow B$ be a surjective function that maps communication tasks on buses. $MapC(\chi_i) = B_j$ indicates that the communication task χ_i is performed on the bus B_j. For notation simplicity, $Map : T \rightarrow P$ is defined, where $Map(\tau_i) = MapP(\tau_i)$ if $\tau_i \in PT$ and $Map(\tau_i) = MapC(\tau_i)$ if $\tau_i \in CT$. Conversely, let $\mathcal{T}_p = \{\tau \in T : Map(\tau) = p \in P\}$ denote the set of tasks that are mapped on processor p. Let $\mathcal{T}_\tau$ be a shorthand notation for $\mathcal{T}_{Map(\tau)}$. Let PE_τ denote the set on processors on which task τ may be mapped.

The mapping is graphically indicated by the shading of the task. In Figure 2a, tasks t_1, t_2, and t_4 are mapped on processing element PE_1 and tasks t_3 and t_5 on processing element PE_2. Both communication task χ_1 and χ_2 are mapped on bus B_1.

An edge connects either two processing tasks that are mapped on the same processing element or a processing task to a communication task or a communication task to a processing task.

For a processing task t_i, $\forall 1 \leq i \leq n$, let Ex_{t_i} denote its execution time on processing element $MapP(t_i)$. Let ϵ_{t_i} be the probability density of Ex_{t_i}.

Let t_i and t_j be any two processing tasks such that task t_i is a predecessor of task t_j ($t_i \in {}^\circ t_j$) and tasks t_i and t_j are mapped on the same processing element ($MapP(t_i) = MapP(t_j)$). In this case, the time of the communication between task t_i and t_j is considered to be part of the execution time of task t_i. Thus, the execution time probability density ϵ_{t_i} accounts for this intraprocessor communication time.

Let t_i and t_j be two processing tasks, let χ be a communication task, let PE_a and PE_b be two distinct processing elements, and let B be a bus such that all of the following statements are true:

—Processing tasks t_i and t_j are mapped on processing elements PE_a and PE_b, respectively ($MapP(t_i) = PE_a$ and $MapP(t_j) = PE_b$).
—Communication task χ is mapped on bus B ($MapC(\chi) = B$).
—Bus B connects processing elements PE_a and PE_b.
—Task χ is a successor of task t_i and a predecessor of task t_j ($\chi \in t_i^\circ \wedge \chi \in {}^\circ t_j$).

The transmission time of the message that is passed between tasks t_i and t_j on the bus B is modeled by the execution time Ex_χ of the communication task χ. Let ϵ_χ denote the probability density of Ex_χ.

Without making any distinction between processing and communication tasks, we let Ex_i denote an execution (communication) time of an instantiation of task $\tau_i \in T$ and we let $ET = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_N\}$ denote the set of N *execution time probability density functions (ETPDFs)*. No restriction is imposed on the execution time probability distribution functions, i.e. they are generalised probability distributions.

3.4 Real-Time Requirements

The real-time requirements are expressed in terms of deadlines. Let $\Delta_T = \{\delta_i \in \mathbb{N} : \tau_i \in T\}$ denote the set of *task deadlines*, where δ_i is the deadline of task τ_i. If job (τ_i, u) has not completed its execution at time $u \cdot \pi_i + \delta_i$, then the job is said to have missed its deadline.

Let $\Delta_\Gamma = \{\delta_{\Gamma_j} \in \mathbb{N} : \Gamma_j \in \Gamma\}$ denote the set of task graph deadlines, where δ_{Γ_j} is the deadline of task graph Γ_j. If there exists at least one task instantiation $(\tau_i, u) \in (\Gamma_j, k)$, such that (τ_i, u) has not completed its execution at time moment $k \cdot \pi_{\Gamma_j} + \delta_{\Gamma_j}$, we say that task graph instantiation (Γ_j, k) has missed its deadline.

If $D_i(t)$ denotes the number of jobs of task τ_i that have missed their deadline over a time span t and $A_i(t) = \lfloor \frac{t}{\pi_i} \rfloor$ denotes the total number of jobs of task τ_i over the same time span, then $\lim_{t\to\infty} \frac{D_i(t)}{A_i(t)}$ denotes the *expected deadline miss ratio* of task τ_i. Let $D_{\Gamma_j}(t)$ denote the number of task graph deadline misses in the interval $[0, t)$ and let $A_{\Gamma_j}(t) = \lfloor \frac{t}{\pi_{\Gamma_j}} \rfloor$ denote -3 the number of instantiations of task graph Γ_j in the same interval. Then, $\lim_{t\to\infty} \frac{D_{\Gamma_j}(t)}{A_{\Gamma_j}(t)}$ denotes the *expected deadline miss ratio* of task graph Γ_j.

Let $Missed_T = \{m_{\tau_1}, m_{\tau_2}, \ldots, m_{\tau_N}\}$ be the set of expected deadline miss ratios per task. Similarly, the set $Missed_\Gamma = \{m_{\Gamma_1}, m_{\Gamma_2}, \ldots, m_{\Gamma_g}\}$ is defined as the set of expected deadline miss ratios per task graph.

The designer may specify upper bounds for tolerated deadline miss ratios, both for tasks and for task graphs. Let $\Theta_T = \{\theta_{\tau_1}, \theta_{\tau_2}, \ldots, \theta_{\tau_N}\}$ be the set of deadline miss thresholds for tasks and let $\Theta_\Gamma = \{\theta_{\Gamma_1}, \theta_{\Gamma_2}, \ldots, \theta_{\Gamma_g}\}$ be the set of deadline miss thresholds for task graphs.

Some tasks or task graphs may be designated as being *critical* by the designer, which means that deadline miss thresholds are not allowed to be

violated. The deadline miss *deviation* of task τ, denoted dev_τ, is defined as

$$dev_\tau = \begin{cases} \infty & m_\tau > \theta_\tau, \tau \text{ critical} \\ m_\tau - \theta_\tau & m_\tau > \theta_\tau, \tau \text{ not critical} \\ 0 & m_\tau \leq \theta_\tau \end{cases} \quad (1)$$

Analogously, we define the deadline miss deviation of a task graph. Designating some tasks as being critical and assigning a deadline miss threshold of 0 to them allows us to specify systems with both hard and soft real-time tasks.

We say that a task graph instantiation (Γ, k), $k \geq 0$, is active in the system at time t if there exists at least one task instantiation $(\tau, u) \in (\Gamma, k)$ such that job (τ, u) has not completed its execution at time t.

Whenever a new instantiation of task graph $\Gamma_i, \forall 1 \leq i \leq g$, arrives and there exists an already active instantiation in the system at the time of the arrival of the new instantiation, the existing active instantiation of task graph Γ_i is *discarded*. Discarding a task graph implies:

—The running jobs belonging to the task graph to be discarded are immediately removed from the processors they run onto. These jobs are eliminated from the system, i.e., their execution is never resumed and all resources that they occupy (locks, memory, process control blocks, file control blocks, etc.) are freed.

—The ready-to-run and blocked-on-I/O jobs belonging to the task graph to be discarded are immediately removed from the ready-to-run and waiting-on-I/O queues of the scheduler. They are also eliminated from the system.

In the common case of more than one task mapped on the same processor, a scheduler selects the next task to run based on the priority associated to the task off line. Because of complexity reasons when dealing with the analysis of systems with stochastic task execution times, we impose that task execution is nonpreemptive.[1] To circumvent such limitation, the designer may define preemption points for a task. The task is decomposed at these points into subtasks. The task execution may then be preempted at the preemption points.

3.5 Problem Formulation

The problem addressed in this paper is formulated as follows.

The problem input consists of

—The set of processing elements PE, the set of buses B, and their connection to processors,

—The set of task graphs Γ,

—The set of task periods Π_T and the set of task graph periods Π_Γ,

—The set of task deadlines Δ_T and of task graph deadlines Δ_Γ,

—The set PE_τ of allowed mappings of τ for all tasks $\tau \in T$,

[1]Based on these assumptions, and considering the communication tasks, we are able to model any priority based bus arbitration protocol as, for instance, CAN [Bosch 1991].

—The set of execution time probability density functions corresponding to each processing element $p \in PE_\tau$ for each task τ,
—The set of communication time probability density functions corresponding to each bus $b \in PE_\chi$ for each communication task χ,
—The set of task deadline miss thresholds Θ_T and the set of task graph deadline miss thresholds Θ_Γ, and
—The set of tasks and task graphs that are designated as being critical.

The problem output consists of a mapping and priority assignment such that the cost function[2]

$$\sum dev = \sum_{i=1}^{N} dev_{\tau_i} + \sum_{i=1}^{g} dev_{\Gamma_i} \tag{2}$$

giving the sum of miss deviations is minimized. If a mapping and priority assignment is found such that $\sum dev$ is finite, it is guaranteed that the deadline miss ratios of all critical tasks and task graphs are below their imposed thresholds.

Because the defined problem is NP-hard (see the complexity of the classical mapping problem [Garey and Johnson 1979]), we have to rely on heuristic techniques for solving the formulated problem.

Two problems have to be solved:

—find an efficient design space exploration strategy, and
—develop a fast and sufficiently accurate analysis, providing the needed indicators.

An accurate estimation of the miss deviation, which is used as a cost function for the optimization process, is in itself a complex and time-consuming task [Manolache et al. 2002]. Therefore, a fast approximation of the cost function value is needed to guide the design space exploration.

Section 4 discusses the first subproblems, while Section 5 focuses on the system analysis we propose.

4. MAPPING AND PRIORITY ASSIGNMENT HEURISTIC

In this section, we propose a design space exploration strategy that maps tasks to processors and assigns priorities to tasks in order to minimize the cost function defined in Eq. (2). The exploration strategy is based on the tabu search (TS) heuristic [Glover 1989].

4.1 The Tabu Search Based Heuristic

Tabu search is a heuristic introduced by Glover [Glover 1989]. It has been successfully used for various system-level design problems, such as yield maximization problems [Alippi et al. 2003], FIR filter design [Traferro et al. 1999], mapping and scheduling of task graphs on SoCs [Wild et al. 2003], and latency-area

[2]The designers may add weights to the terms of the sums if different criticality degrees need to be assigned to the deadline miss events.

tradeoffs for NoCs [Cardoso et al. 2005]. Several researchers have compared tabu search with other optimization heuristics, such as simulated annealing and genetic algorithms and shown the superiority of tabu search with regard to optimization time and quality of results [Hajji et al. 2002; Wild et al. 2003; Pierre and Houeto 2002; Krishnamachari and Wicker 2000; Eles et al. 1997].

We believe that the message of our paper, namely, that considering the probability distributions of the task execution times is crucial for obtaining low deadline miss ratios, would hold even if any other search methods were deployed. Our primary goals are to show that optimization considering stochastic execution times is possible, how it can be done, how the analysis can be made fast enough, what the basic moves for design space exploration are, and that using tabu search is an efficient alternative for design space exploration. After reading this paper, and using the analysis approach, as well as the moves defined by us, it will be a relatively straightforward exercise for the interested reader to implement the design space exploration using simulated annealing (SA), genetic algorithms (GA), or a particular, customized design space exploration approach.

We use an extended variant of Tabu search, which is described in this section. The variant is not specific to a particular problem. After explaining the heuristic in general, we will become more specific at the end of the section where we illustrate the heuristic in the context of task mapping and priority assignment.

Typically, optimization problems are formulated as follows: Find a configuration, i.e. an assignment of values to parameters that characterize a system, such that the configuration satisfies a possibly empty set of imposed constraints and the value of a cost function is minimal for that configuration.

We define the design space $\mathcal{S}$ as a set of points (also called solutions), where each point represents a configuration that satisfies the imposed constraints. A *move* from one solution in the design space to another solution is equivalent to assigning a new value to one or more of the parameters that characterise the system. We say that we obtain solution s_2 by applying the move m on solution s_1, and we write $s_2 = m(s_1)$. Solution s_1 can be obtained back from solution s_2 by applying the *negated move* m, denoted $\overline{m}$ ($s_1 = \overline{m}(s_2)$).

Solution s' is a *neighbor* of solution s if there exists a move m such that solution s' can be obtained from solution s by applying move m. All neighbors of a solution s form the neighborhood $V(s)$ of that solution ($V(s) = \{q : \exists m$ such that $q = m(s)\}$).

The exploration algorithm is shown in Figure 5. The exploration starts from an initial solution, labeled also as the current solution (line 1) considered as the globally best solution so far (line 2). The initial solution may be generated with a fast greedy algorithm or may be even randomly generated. The initial solution has typically a reduced impact (or no impact at all) on the quality of the final solution and on the convergence rate. This insensitivity to the initial solution is because of the diversification phase (explanation later in this Section) of tabu search and the fact that tabu search does not get stuck in local minima of the search space. These two properties ensure that unexplored parts of the search space are eventually explored by the tabu search heuristic.

```
(1)   crt_sol = init_sol
(2)   global_best_sol = crt_sol
(3)   global_best = cost(crt_sol)
(4)   TM = ∅
(5)   since_last_improvement = 0
(6)   iteration_count = 1
(7)   CM = set_of_candidate_moves(crt_sol)
(8)   (chosen_move, next_sol_cost) = choose_move(CM)
(9)   while iteration_count < max_iterations do
(10)    while since_last_improvement < W do
(11)      next_sol = move(crt_sol, chosen_move)
(12)      TM = TM ∪ {chosen_move}
(13)      since_last_improvement + +
(14)      iteration_count + +
(15)      crt_sol = next_sol
(16)      if next_sol_cost < global_best_cost then
(17)        global_best_cost = next_sol_cost
(18)        global_best_sol = crt_sol
(19)        since_last_improvement = 0
(20)      end if
(21)      CM = set_of_candidate_moves(TM, crt_sol)
(22)      (chosen_move, next_sol_cost) = choose_move(CM)
(23)    end while
(24)    since_last_improvement = 0
(25)    (chosen_move, next_sol_cost) = diversify(TM, crt_sol)
(26)    iteration_count + +
(27)  end while
(28)  return global_best_sol
```

Fig. 5. Design space exploration algorithm.

The cost function is evaluated for the current solution (line 3). We keep track of a list of moves *TM* that are marked as *tabu*. A solution will leave the tabu list after a certain number of iterations (the tabu tenure). Initially the list is empty (line 4).

We construct *CM*, a subset of the set of all moves that are possible from the current solution point (line 7). Let $N(CM)$ be the set of solutions that can be reached from the current solution by means of a move in *CM*.[3] The cost function is evaluated for each solution in $N(CM)$. Let $m \leq m'$ if $cost(m(crt\_sol)) \leq cost(m'(crt\_sol))$. A move in *CM* is selected (line 8) as follows.

If $\exists m \in CM$ such that $m \leq m' \forall m' \in CM \wedge cost(m(crt\_sol)) \leq global\_best$, then move m is selected. Else, if $\exists m \in CM \backslash TM$ such that $m \leq m' \forall m' \in CM \backslash TM$, then move m is selected. Else, $m \in TM$ such that $m \leq m' \forall m' \in TM$ is selected.

The new solution is obtained by applying the chosen move m on the current solution (line 11). The reverse of move $\overline{m}$ is marked as tabu such that m will not be reversed in the next few iterations (line 12). The new solution becomes the current solution (line 15). If it is the case (line 16), the new solution becomes also the globally best solution reached so far (lines 17–18). However, it should be noted that the new solution could have a larger cost than the current solution. This could happen if there are no moves that would improve on the current

[3]If *CM* is the set of *all* possible moves from *crt_sol* then $N(CM) = V(crt\_sol)$ W.

solution or all such moves would be tabu. By this, the heuristic has the potential to escape from local minima. Placing the reverse of the most recent moves into the list *TM* of tabu moves also avoids cycling, which could occur if the move returns to a recently visited solution. The procedure of building the set of candidate moves and then choosing one according to the criteria listed above is repeated. If no global improvement has been noted for the past W iterations, the loop (lines 10–23) is interrupted (line 10). In this case, a diversification phase follows (line 25) in which a rarely used move is performed in order to force the heuristic to explore different regions in the design space. The whole procedure is repeated until the heuristic iterated for a specified maximum number of iterations (line 9). The procedure returns the solution characterized by the lowest cost function value that it found during the design space exploration (line 28).

Two issues are of utmost importance when tailoring the general tabu search-based heuristic, described above for particular problems.

First, there is the definition of what is a legal move. On one hand, the transformation of a solution must result in another solution, i.e., the resulting parameter assignment must satisfy the set of constraints. On the other hand, because of complexity reasons, certain restrictions must be imposed on what constitutes a legal move. For example, if any transformation were a legal move, the neighborhood of a solution would comprise the entire solution space. In this case, it is sufficient to run the heuristic for just one iteration ($max\_iterations = 1$), but that iteration would require an unreasonably long time, as the whole solution space would be probed. Nevertheless, if moves were too restricted, a solution could be reached from another solution only after applying a long sequence of moves. This makes the reaching of the far-away solution unlikely. In this case, the heuristic would be inefficient as it would circle in the same region of the solution space until a diversification step would force it out.

The second issue is the construction of the subset of candidate moves. One solution would be to include all possible moves from the current solution in the set of candidate moves. In this case, the cost function, which sometimes can be computationally expensive, has to be calculated for all neighbors. Thus, we would run the risk to render the exploration slow. If we had the possibility to quickly assess which are promising moves, we could include only those in the subset of candidate moves.

For our particular problem, namely, the task mapping and priority assignment, each task is characterized by two attributes: its mapping and its priority. In this context, a move in the design space is equivalent to changing one or both attributes of one single task.

In the following section, we discuss the issue of constructing the subset of candidate moves.

4.2 Candidate Move Selection

The cost function is evaluated $|CM|$ times at each iteration, where $|CM|$ is the cardinality of the set of candidate moves. Let us consider that task τ, mapped on processor P_j, is moved to processor P_i and there are q_i tasks on processor

P_i. Task τ can take one of $q_i + 1$ priorities on processor P_i. If task τ is not moved to a different processor, but only its priority is changed on processor P_j, then there are $q_j - 1$ possible new priorities. If we consider all processors, there are $M - 2 + N$ possible moves for each task τ, as shown in the equation

$$q_j - 1 + \sum_{\substack{i=1 \\ i \neq j}}^{M} (q_i + 1) = M - 2 + \sum_{i=1}^{M} q_i = M - 2 + N \tag{3}$$

where N is the number of tasks and M is the number of processors. Hence, if all possible moves are candidate moves,

$$N \cdot (M - 2 + N) \tag{4}$$

moves are possible at each iteration. Therefore, a key to the efficiency of the algorithm is the intelligent selection of the set CM of candidate moves. If CM contained only those moves that had a high chance to drive the search toward good solutions, then fewer points would be probed, leading to a speed up of the algorithm.

In our approach, the set CM of candidate moves is composed of all moves that operate on a subset of tasks. Tasks are assigned scores and the chosen subset of tasks is composed of the first K tasks with respect to their score. Thus, if we included all possible moves that modify the mapping and/or priority assignment of only the K highest-ranked tasks, we would reduce the number of cost function evaluations N/K times.

We illustrate the way the scores are assigned to tasks based on the example in Figure 2b. As a first step, we identify the critical paths and the noncritical computation paths of the application. The average execution time of a computation path is given by the sum of the average execution times of the tasks belonging to the path. A path is critical if its average execution time is the largest among the paths belonging to the same task graph. For the example in Figure 2b, the critical path is $t_1 \rightarrow t_2 \rightarrow t_4$, with an average execution time of $1+6+8 = 15$. In general, noncritical paths are those paths starting with a root node or a task on a critical path, ending with a leaf node or a task on a critical path and containing only tasks that do not belong to any critical path. For the example in Figure 2b, noncritical paths are $t_1 \rightarrow t_3$ and $t_2 \rightarrow t_5$.

For each critical or noncritical path, a path-mapping vector is computed. The mapping vector is a P-dimensional integer vector, where P is the number of processors. The modulus of its projection along dimension p_i is equal to the number of tasks that are mapped on processor p_i and that belong to the considered path. For the example in Figure 2b, the vectors corresponding to the paths $t_1 \rightarrow t_2 \rightarrow t_4$, $t_1 \rightarrow t_3$ and $t_2 \rightarrow t_5$ are $2\mathbf{i} + 1\mathbf{j}$, $1\mathbf{i} + 1\mathbf{j}$, and $2\mathbf{i} + 0\mathbf{j}$ respectively, where $\mathbf{i}$ and $\mathbf{j}$ are the versors along the two dimensions. Each task is characterized by its task-mapping vector, which has a modulus of 1 and is directed along the dimension corresponding to the processor on which the task is mapped. For example, the task-mapping vectors of t_1, t_2, t_3, t_4, and t_5 are $1\mathbf{i}$, $1\mathbf{i}$, $1\mathbf{j}$, $1\mathbf{j}$, and $1\mathbf{i}$, respectively.

Next, for each path and for each task belonging to that path, the angle between the path- and the task-mapping vectors is computed. For example, the

task-mapping vectors of tasks t_1 and t_2 form an angle of arccos $\frac{2}{\sqrt{5}}$ with the path-mapping vector of critical path $t_1 \rightarrow t_2 \rightarrow t_4$ and the task-mapping vectors of task t_1 and t_3 form an angle of 45° with the path-mapping vector of the non-critical path $t_1 \rightarrow t_3$. The mapping vector of task t_4 forms an angle of arccos $\frac{1}{\sqrt{5}}$ with the mapping vector of the critical path $t_1 \rightarrow t_2 \rightarrow t_4$.

The score assigned to each task is a weighted sum of angles between the task's mapping vector and the mapping vectors of the paths to whom the task belongs. The weights are proportional to the relative criticality of the path. In our example, we weight angles between task-mapping vectors and mapping vectors of critical paths with 4, while angles between task-mapping vectors and mapping vectors of noncritical paths have weight 1. Hence, the score of task t_1 is $4.0 \cdot \arccos \frac{2}{\sqrt{5}} + 1.0 \cdot \frac{\pi}{4} \approx 2.64$, that of task t_2 is $4.0 \cdot \arccos \frac{2}{\sqrt{5}} \approx 1.85$, that of task t_3 is $\frac{\pi}{4} \approx 0.78$, that of task t_4 is $4.0 \cdot \arccos \frac{1}{\sqrt{5}} \approx 4.43$, and the score of task t_5 is 0. Task t_4 has the highest score and moves that change its mapping or priority are promising candidate moves. Intuitively, this approach attempts to map the tasks that belong to critical paths on the same processor (in our example, to move task t_4 from the shaded to the white processor such that the whole critical path $t_1 \rightarrow t_2 \rightarrow t_4$ is mapped on the same processor). In order to avoid processor overload, the scores are penalized if the task is intended to be moved on highly loaded processors.

Once scores have been assigned to tasks, the first $K = N/c$ tasks are selected according to their scores. In our experiments, we use $c = 2$. In order to further reduce the search neighbourhood, not all possible moves that change the task mapping and/or priority assignment of one task are chosen. Only two processors are considered as target processors for each task. The selection of those two processors is made based on scores assigned to processors. These scores are a weighted sum of the potential reduction of interprocessor communication and processor load.

We will illustrate how the scores are computed using the application example in Figure 2a. On the white processor, task t_2 communicates with one task (t_5) over the bus, while it does not use the bus to communicate with tasks t_1 and t_4. If we moved task t_2 on the shaded processor, we would obtain two interprocessor communications (from task t_1 and to task t_4) and one intraprocessor communication (to task t_5). Thus, moving task t_2 would imply increasing the IPC from 1 to 2, i.e., an increase of 100%.

If we moved task t_3 from the shaded processor to the white processor, all the interprocessor communication caused by task t_3 disappears (task t_3 communicates only with task t_1, which is on the white processor). We would reduce the interprocessor communication with 100%. If the score contained only the IPC reductions, the algorithm would try to cluster all tasks on a single processor. In order to counter that, we added the processor load as a penalty in the score calculation. Moving task t_3 from the shaded to the white processor would reduce the IPC of task t_3 with 100%, but would also increase the load of the white processor with the load caused by task t_3. As the white processor has to cope with an average work load of 15 units (the average execution times of tasks t_1, t_2, and t_4), the 100% reduction would be penalized with an amount proportional to 15.

Because we add two different types of quantities (IPC modification and processor load modification), we need to weight them differently in order to get meaningful scores. The weights are obtained empirically. We observe which are the typical IPC modifications and which are the typical processor load modifications as a result of moving a task from one processor to another. In our case, we weight the processor load modification with -5.5 and the IPC modification with 1. The processor load is weighted with a negative weight, in order to penalize overload.

On average, there will be N/M tasks on each processor. Hence, if a task is moved to a different processor, it may take $N/M + 1$ possible priorities on its new processor. By considering only $N/2$ tasks and only two target processors for each task, we restrict the neighborhood to

$$N/2 \cdot 2 \cdot (1 + N/M) = N \cdot (1 + N/M) \tag{5}$$

candidate moves on average. This restricted neighborhood is approximately

$$N \cdot (M - 2 + N)/(N \cdot (1 + N/M)) \approx M \tag{6}$$

times smaller than the neighborhood that resulted from the application of all possible moves. We will denote this method as the restricted neighborhood search. In Section 6, we will compare the restricted neighborhood search with an exploration of the complete neighborhood.

5. ANALYSIS

This section presents our analysis algorithm that is used for evaluation of the cost function that is driving the design space exploration. The first part of the section discusses the deduction of the algorithm itself, while the second part presents some considerations on the approximations that were made and their impact regarding the accuracy.

5.1 Analysis Algorithm

The cost function that is driving the design space exploration is $\sum dev$, where dev is the miss deviation, as defined in Eq. (2). The miss deviation for each task is obtained as the result of a performance analysis of the system.

In previous work [Manolache et al. 2002], we presented a performance analysis method for multiprocessor applications with stochastic task executions times. The method is based on the Markovian analysis of the underlying stochastic process. As the latter captures all possible behaviors of the system, the method gives great insight regarding the system's internals and bottlenecks. However, its relatively large analysis time makes its use inappropriate inside an optimization loop. Therefore, we propose an approximate analysis method of polynomial complexity. The main challenge is in finding those dependencies among random variables that are weak and can be neglected, such that the analysis becomes of polynomial complexity and the introduced inaccuracy is within reasonable bounds.

Before proceeding with the exposition of the approximate analysis approach, we introduce the notation that we use in the sequel.

The finishing time of the jth job of task τ is the time moment when (τ, j) finishes its execution. We denote it with $F_{\tau,j}$. The deadline miss ratio of a job is the probability that its finishing time exceeds its deadline:

$$m_{\tau,j} = 1 - \mathsf{P}(F_{\tau,j} \leq \delta_{\tau,j}) \tag{7}$$

The ready time of (τ, j) is the time moment when (τ, j) is ready to execute, i.e., the maximum of the finishing times of jobs in its predecessor set. We denote the ready time with $A_{\tau,j}$ and we write

$$A_{\tau,j} = \max_{\sigma \in {}^\circ\tau} F_{\sigma,j} \tag{8}$$

The starting time of (τ, j) is the time moment when (τ, j) starts executing. We denote it with $S_{\tau,j}$. Obviously, the relation

$$F_{\tau,j} = S_{\tau,j} + Ex_{\tau,j} \tag{9}$$

holds between the starting and finishing times of (τ, j), where $Ex_{\tau,j}$ denotes the execution time of (τ, j). The ready and starting times of a job may differ because the processor might be busy at the time the job becomes ready for execution. The ready, starting, and finishing times are all random variables.

Let $L_{\tau,j}(t)$ be a function that takes value 1 if (τ, j) is running at time moment t and 0 otherwise. In other words, if $L_{\tau,j}(t) = 1$, processing element $Map(\tau)$ is busy executing job j of task τ at time t. If (τ, j) starts executing at time t, $L_{\tau,j}(t)$ is considered to be 1. If (τ, j) finishes its execution at time t', $L_{\tau,j}(t')$ is considered to be 0. For simplicity, in the sequel, we will write $L_{\tau,j}(t)$ when we mean $L_{\tau,j}(t) = 1$. Also, $L_\tau(t)$ is a shorthand notation for $\sum_{j\in\mathbb{N}} L_{\tau,j}(t)$. $L_{\tau,j}(t) = 1$ if an instantiation of task τ is running at time t.

Let $I_{\tau,j}(t)$ be a function that takes value 1 if

—All tasks in the ready-to-run queue of the scheduler on processor $Map(\tau)$ at time t have a lower priority than task τ, and

—$\sum_{\sigma\in\mathcal{T}_\tau\setminus\{\tau\}} L_\sigma(t) = 0$,

and it takes value 0 otherwise, where $\mathcal{T}_\tau = \mathcal{T}_{Map(\tau)}$ is the set of tasks mapped on the same processor as task τ. Intuitively, $I_{\tau,j}(t) = 1$ implies that (τ, j) could start running on processing element $Map(\tau)$ at time t if (τ, j) becomes ready at or prior to time t. Let $I_{\tau,j}(t, t')$ be a shorthand notation for $\exists\xi \in (t, t'] : I_{\tau,j}(\xi) = 1$, i.e., there exists a time moment ξ in the right semiclosed interval $(t, t']$, such that (τ, j) could start executing at ξ if it become ready at or prior to ξ.

In order to compute the deadline miss ratio of (τ, j) (Eq. 7), we need to compute the probability distribution of the finishing time $F_{\tau,j}$. This, in turn, can be precisely determined (Eq. 9) from the probability distribution of the execution time $Ex_{\tau,j}$, which is an input data, and the probability distribution of the starting time of (τ, j), $S_{\tau,j}$. Therefore, in the sequel, we focus on determining $\mathsf{P}(S_{\tau,j} \leq t)$.

We start by observing that $I_{\tau,j}(t, t+h)$ is a necessary condition for $t < S_{\tau,j} \leq t + h$. Thus,

$$\mathsf{P}(t < S_{\tau,j} \leq t + h) = \mathsf{P}(t < S_{\tau,j} \leq t + h \cap I_{\tau,j}(t, t + h)) \tag{10}$$

We can write

$$\begin{aligned}&\mathsf{P}(t < S_{\tau,j} \leq t+h \cap I_{\tau,j}(t,t+h))\\ &\quad = \mathsf{P}(t < S_{\tau,j} \cap I_{\tau,j}(t,t+h))\\ &\qquad - \mathsf{P}(t+h < S_{\tau,j} \cap I_{\tau,j}(t,t+h))\end{aligned} \tag{11}$$

Furthermore, we observe that the event

$$t+h < S_{\tau,j} \cap I_{\tau,j}(t,t+h)$$

is equivalent to

$$\begin{aligned}&(t+h < A_{\tau,j} \cap I_{\tau,j}(t,t+h))\\ &\cup (sup\{\xi \in (t,t+h] : I_{\tau,j}(\xi) = 1\} < A_{\tau,j} \leq t+h \cap I_{\tau,j}(t,t+h))\end{aligned}$$

In other words, (τ, j) starts executing after $t+h$ when the processor was available sometimes in the interval $(t, t+h]$ if, and only if, (τ, j) became ready to execute *after* the latest time in $(t, t+h]$ at which the processor was available. Thus, we can rewrite Eq. (11) as follows:

$$\begin{aligned}&\mathsf{P}(t < S_{\tau,j} \leq t+h \cap I_{\tau,j}(t,t+h))\\ &= \mathsf{P}(t < S_{\tau,j} \cap I_{\tau,j}(t,t+h))\\ &\quad - \mathsf{P}(t+h < A_{\tau,j} \cap I_{\tau,j}(t,t+h))\\ &\quad - \mathsf{P}(sup\{\xi \in (t,t+h] : I_{\tau,j}(\xi) = 1\} < A \leq t+h\\ &\qquad \cap I_{\tau,j}(t,t+h))\end{aligned} \tag{12}$$

After some manipulations involving negations of the events in the above equation, and by using Eq. (10), we obtain

$$\begin{aligned}&\mathsf{P}(t < S_{\tau,j} \leq t+h) = \mathsf{P}(A_{\tau,j} \leq t+h \cap I_{\tau,j}(t,t+h))\\ &- \mathsf{P}(S_{\tau,j} \leq t \cap I_{\tau,j}(t,t+h))\\ &- \mathsf{P}(sup\{\xi \in (t,t+h] : I_{\tau,j}(\xi) = 1\} < A_{\tau,j} \leq t+h\\ &\qquad \cap I_{\tau,j}(t,t+h))\end{aligned} \tag{13}$$

When h becomes very small, the last term of the right-hand side of the above equation becomes negligible relative to the other two terms. Hence, we write the final expression of the distribution of $S_{\tau,j}$ as follows:

$$\begin{aligned}&\mathsf{P}(t < S_{\tau,j} \leq t+h)\\ &\approx \mathsf{P}(A_{\tau,j} \leq t+h \cap I_{\tau,j}(t,t+h))\\ &\quad - \mathsf{P}(S_{\tau,j} \leq t \cap I_{\tau,j}(t,t+h))\end{aligned} \tag{14}$$

We observe from Eq. (14) that the part between t and $t+h$ of the probability distribution of $S_{\tau,j}$ can be calculated from the probability distribution of $S_{\tau,j}$ for time values less than t. Thus, we have a method for an iterative calculation of $\mathsf{P}(S_{\tau,j} \leq t)$, in which we compute

$$\mathsf{P}(kh < S_{\tau,j} \leq (k+1)h), \quad k \in \mathbb{N}$$

at iteration $k+1$ from values obtained during previous iterations.

A difficulty arises in the computation of the two joint distributions in the right-hand side of Eq. (14). The event that a job starts or becomes ready prior to time t *and* that the processor may start executing it in a vicinity of time t is a very complex event. It depends on many aspects, such as the particular order of execution of tasks on different (often all) processors, and on the execution time of different tasks, quite far from task τ in terms of distance in the computation tree. Particularly, the dependence on the execution order of tasks on different processors makes the exact computation of

$$\mathsf{P}(A_{\tau,j} \leq t + h \cap I_{\tau,j}(t, t + h))$$

and

$$\mathsf{P}(S_{\tau,j} \leq t \cap I_{\tau,j}(t, t + h))$$

of exponential complexity. Nevertheless, exactly this multitude of dependencies of events $I_{\tau,j}(t, t+h)$, $A_{\tau,j} \leq t+h$, and $S_{\tau,j} \leq t$ on various events makes the dependency weak among the aforementioned three events. Thus, we approximate the right-hand side of Eq. (14) by considering the joint events as if they were conjunctions of independent events. Hence, we approximate $\mathsf{P}(t < S_{\tau,j} \leq t + h)$ as follows:

$$\begin{aligned} &\mathsf{P}(t < S_{\tau,j} \leq t + h) \\ &\quad \approx (\mathsf{P}(A_{\tau,j} \leq t + h) - \mathsf{P}(S_{\tau,j} \leq t)) \cdot \mathsf{P}(I_{\tau,j}(t, t + h)) \end{aligned} \tag{15}$$

The impact of the introduced approximation on the accuracy of the analysis is discussed in Section 5.2, based on a nontrivial example.

In order to fully determine the probability distribution of $S_{\tau,j}$ (and implicitly of $F_{\tau,j}$ and the deadline miss ratio), we need the probability distribution of $A_{\tau,j}$ and the probability $\mathsf{P}(I_{\tau,j}(t, t+h))$. Based on Eq. (8), if the finishing times of all tasks in the predecessor set of task τ were statistically independent, we could write

$$\mathsf{P}(A_\tau \leq t) = \prod_{\sigma \in {}^\circ\tau} \mathsf{P}(F_\sigma \leq t). \tag{16}$$

In the majority of cases, the finishing times of all tasks in the predecessor set of task τ are not statistically independent. For example, if there exists a task α and two computation paths $\alpha \rightarrow \sigma_1$ and $\alpha \rightarrow \sigma_2$, where tasks σ_1 and σ_2 are predecessors of task τ, then the finishing times F_{σ_1} and F_{σ_2} are not statistically independent. The dependency becomes weaker the longer these computation paths are. Also, the dependency is weakened by the other factors that influence the finishing times of tasks σ, for example, the execution times and execution order of the tasks on processors $Map(\sigma_1)$ and $Map(\sigma_2)$. Even if no common ancestor task exists among any of the predecessor tasks σ, the finishing times of tasks σ may be dependent, because they or some of their predecessors are mapped on the same processor. However, these kind of dependencies are extremely weak as shown by Kleinrock [Kleinrock 1964] for computer networks and by Li [Li and Antonio 1997] for multiprocessor applications. Therefore, in practice, Eq. (16) is a good approximation.

Last, we determine the probability $\mathsf{P}(I_{\tau,j}(t, t + h))$, i.e., the probability that processor $Map(\tau)$ may start executing (τ, j) sometimes in the interval $(t, t+h]$.

```
(1)  Sort all tasks in topological order of the task
     graph and put the sorted tasks in sequence T
(2)  For all (τ, j), such that τ has no predecessors
     determine P(A_{τ,j} ≤ t)
(3)  For all (τ, j), let P(I_{τ,j}(0, h)) = 1
(4)  for t := 0 to LCM step h do
(5)    for each τ ∈ T do
(6)      compute P(A_τ ≤ t)                          Eq.(16)
(7)      compute P(t < S_{τ,j} ≤ t + h)              Eq.(15)
(8)      compute P(F_{τ,j} ≤ t + h)                  Eq.(9)
(9)      compute P(L_{τ,j}(t + h))                   Eq.(18)
(10)     compute P(I_{τ,j}(t, t + h))                Eq.(17)
(11)   end for
(12) end for
(13) compute the deadline miss ratios                Eq.(7)
```

Fig. 6. Approximate analysis algorithm.

This probability is given by the probability that no task is executing at time t, i.e.,

$$\mathsf{P}(I_{\tau,j}(t, t+h)) = 1 - \sum_{\sigma \in \mathcal{T}_\tau \setminus \{\tau\}} \mathsf{P}(L_\sigma(t) = 1) \tag{17}$$

The probability that (τ, j) is running at time t is given by

$$\mathsf{P}(L_{\tau,j}(t)) = \mathsf{P}(S_{\tau,j} \leq t) - \mathsf{P}(F_{\tau,j} \leq t) \tag{18}$$

The analysis algorithm is shown in Figure 6. The analysis is performed over the interval $[0, LCM)$, where LCM is the least common multiple of the task periods. The algorithm computes the probability distributions of the random variables of interest parsing the set of tasks in their topological order. Thus, we make sure that ready times propagate correctly from predecessor tasks to successors.

Line 7 of the algorithm computes the probability that job (τ_i, j) starts its execution sometime in the interval $(t, t+h]$ according to Eq. (15). The finishing time of the job may lie within one of the intervals $(t + BCET_i, t + h + BCET_i]$, $(t + BCET_i + h, t + 2h + BCET_i], \ldots, (t + WCET_i, t + h + WCET_i]$, where $BCET_i$ and $WCET_i$ are the best- and worst-case execution times of task τ_i, respectively. There are $\lceil (WCET_i - BCET_i)/h \rceil$ such intervals. Thus, the computation of the probability distribution of the finishing time of the task (line 8) takes $\lceil |ETPDF_i|/h \rceil$ steps, where $|ETPDF_i| = WCET_i - BCET_i$.

Let $|ETPDF| = \max_{1 \leq i \leq N} \lceil |ETPDF_i|/h \rceil$. Then, the complexity of the algorithm is $O(N \cdot LCM/h \cdot \lceil |ETPDF|/h \rceil)$, where N is the number of processing and communication tasks.

The choice of the discretization resolution h is done empirically such that we obtain a fast analysis with reasonable accuracy for the purpose of task mapping. The designers could use an arbitrary task mapping and priority assignment and analyze the system with a more accurate, but slower, method (e.g., Manolache et al. [2002]). They could then experiment with various values for h and use the results of the slower method as a reference. Once a convenient value for h is found, it is used for the fast analysis that drives the optimization.

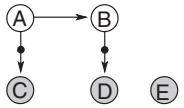


Fig. 7. Application example.

5.2 Approximations

We have made several approximations in the algorithm described in the previous section. These are:

1. The discretization approximation used throughout the approach, i.e., the fact that the probability distributions of interest are all computed at discrete times $\{0, h, 2h, \ldots, \lfloor LCM/h \rfloor\}$,
2. $\mathsf{P}(A_\tau \leq t) \approx \prod_{\sigma \in \circ\tau} \mathsf{P}(F_\sigma \leq t)$,
3. $\mathsf{P}(A_{\tau,j} \leq t + h \cap I_{\tau,j}(t, t+h)) \approx \mathsf{P}(A_{\tau,j} \leq t + h) \cdot \mathsf{P}(I_{\tau,j}(t, t+h))$ and $\mathsf{P}(S_{\tau,j} \leq t \cap I_{\tau,j}(t, t+h)) \approx \mathsf{P}(S_{\tau,j} \leq t) \cdot \mathsf{P}(I_{\tau,j}(t, t+h))$.

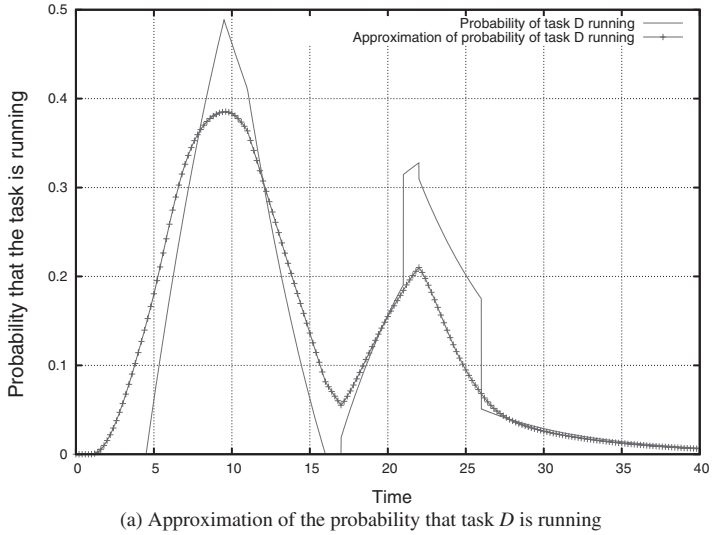
The first approximation is inevitable when dealing with continuous functions. Moreover, its accuracy may be controlled by choosing different discretization resolutions h.

The second approximation is typically accurate as the dependencies between the finishing times F_σ are very weak [Kleinrock 1964] and we will not focus on its effects in this discussion.

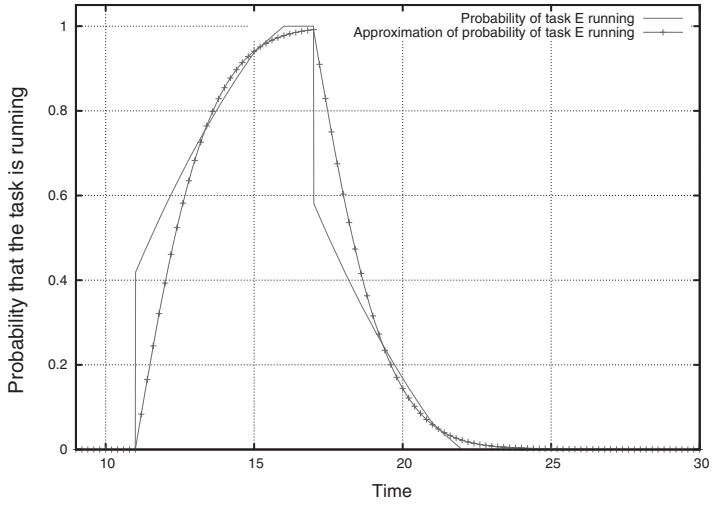
In order to discuss the last approximation, we will introduce the following example. Let us consider the application depicted in Figure 7. It consists of five tasks, grouped into two task graphs $\Gamma_1 = \{A, B, C, D\}$ and $\Gamma_2 = \{E\}$. Tasks A and B are mapped on the first processor, while tasks C, D, and E are mapped on the second processor. The two black dots on the arrows between tasks A and C, and tasks B and D represent the interprocessor communication tasks. Tasks C, D, and E have fixed execution times of 4, 5, and 6 time units, respectively. Tasks A and B have execution times with exponential probability distributions, with average rates of $1/7$ and $1/2$, respectively.[4] Each of the two interprocessor communications takes 0.5 time units. Task A arrives at time moment 0, while task E arrives at time moment 11. Task E is the highest priority task. The deadline of both task graphs is 35.

Because of the data dependencies between the tasks, task D is the last to run among the task graph Γ_1. The probability that processor two is executing task D at time t is analytically determined and plotted in Figure 8a as a function of t. On the same figure, we plotted the approximation of the same probability as obtained by our approximate analysis method. The probability that task E is running at time t and its approximation are shown in Figure 8b.

[4]We chose exponential execution time probability distributions only for the scope of this illustrative example. Thus, we are able to easily deduce the exact distributions in order to compare them to the approximated ones. Note that our approach is not restricted to exponential distributions and we use generalized distributions throughout the experimental results.



(a) Approximation of the probability that task D is running



(b) Approximation of the probability that task E is running

Fig. 8. Approximation accuracy.

In Figure 8a, we observe that large approximation errors occur at times around the earliest possible start time of task D, i.e., around time 4.5.[5] We can write

$$\begin{aligned} &\mathsf{P}(A_D \leq t + h \cap I_D(t, t + h)) \\ &\quad = \mathsf{P}(I_D(t, t + h) | A_D \leq t + h) \cdot \mathsf{P}(A_D \leq t + h) \end{aligned}$$

[5]The time when task D becomes ready is always *after* the time when task C becomes ready. Task C is ready the earliest at time 0.5, because the communication $A \rightarrow C$ takes 0.5 time units. The execution of task C takes 4 time units. Therefore, the processor is available to task D the earliest at time 4.5.

$\mathsf{P}(I_D(t, t+h)|A_D \leq t+h)$ is interpreted as the probability that task D may start to run in the interval $(t, t+h]$ *knowing* that it became ready to execute prior to time $t+h$. If $t+h < 4.5$, and we took into consideration the fact that $A_D \leq t+h$, then we know for sure that task C could not have yet finished its execution of four time units (see footnote). Therefore,

$$\mathsf{P}(I_D(t, t+h)|A_D \leq t+h) = 0, t+h < 4.5$$

However, in our analysis, we approximate $\mathsf{P}(I_D(t, t+h)|A_D \leq t+h)$ with $\mathsf{P}(I_D(t, t+h))$, i.e., we do *not* take into account that $A_D \leq t$. Not taking into account that task D became ready prior to time t, opens the possibility that task A has not yet finished its execution at time t. In this case, task C has not yet become ready, and the processor on which tasks C and D are mapped could be idle. Thus,

$$\mathsf{P}(I_D(t, t+h)) \neq 0$$

because the processor might be free if task C has not yet started. This illustrates the kind of approximation errors introduced by

$$\mathsf{P}(A_D \leq t+h \cap I_D(t, t+h)) \approx \mathsf{P}(I_D(t, t+h)) \cdot \mathsf{P}(A_D \leq t+h)$$

However, what we are interested in is a high-quality approximation closer to the *tail* of the distribution, because typically there is the deadline. As we can see from the plots, the two curves almost overlap for $t > 27$. Thus, the approximation of the deadline miss ratio of task D is very good. The same conclusion is drawn from Figure 8b. In this case, too, we see a perfect match between the curves for time values close to the deadline.

Finally, we assessed the quality of our approximate analysis on larger examples. We compare the processor load curves obtained by our approximate analysis (AA) with processor load curves obtained by our high-complexity performance analysis (PA) developed in previous work [Manolache et al. 2002]. The benchmark application consists of 20 processing tasks mapped on two processors and three communication tasks mapped on a bus connecting the two processors. Figure 9 gives a qualitative measure of the approximation. It depicts the two processor load curves for a task in the benchmark application. One of the curves was obtained with PA and the other with AA. A quantitative measure of the approximation is given in Table I. We present only the extreme values for the average errors and standard deviations. Thus, row 1 in the table, corresponding to task 19, shows the largest obtained average error, while row 2, corresponding to task 13, shows the smallest obtained average error. Row 3, corresponding to task 5, shows the worst obtained standard deviation, while row 4, corresponding to task 9, shows the smallest obtained standard deviation. The average of standard deviations of errors over all tasks is around 0.065. Thus, we can say with 95% confidence that AA approximates the processor load curves with an error of ± 0.13.

Although the accuracy of this analysis is good, its main purpose is to be used inside the design space exploration loop. Once the final optimized design has been produced, our approach from previous work [Manolache et al. 2002] will be used in order to evaluate the accurate deadline miss ratio.

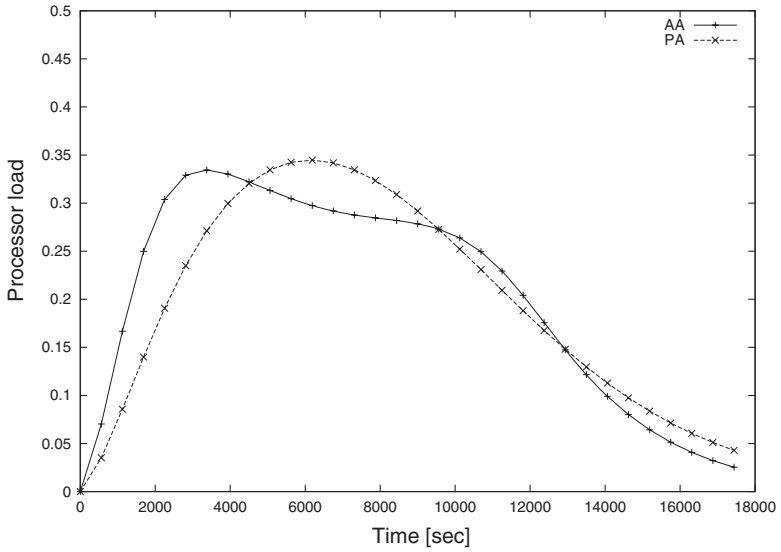


Fig. 9. Approximation accuracy.

Table I. Approximation Accuracy

Task	Average error	Standard deviation of errors
19	0.056351194	0.040168796
13	0.001688039	0.102346107
5	0.029250265	0.178292338
9	0.016695770	0.008793487

6. EXPERIMENTAL RESULTS

The proposed heuristic for task mapping and priority assignment has been experimentally evaluated on randomly generated benchmarks and on a real-life example. This section presents the experimental setup and comments on the obtained results. The experiments were run on a desktop PC with an AMD Athlon processor clocked at 1533 MHz.

The benchmark set consisted of 396 applications. The applications contained t tasks, clustered in g task graphs and mapped on p processors, where $t \in \{20, 22, \ldots, 40\}$, $g \in \{3, 4, 5\}$, and $p \in \{3, 4, \ldots, 8\}$. For each combination of t, g, and p, two applications were randomly generated. Three mapping and priority assignment methods were run on each application. All three implement a tabu search algorithm with the same tabu tenure, termination criterion, and number of iterations, after which a diversification phase occurs. In each iteration, the first method selects the next point in the design space while considering the *entire* neighborhood of design space points. Therefore, we denote it ENS, exhaustive neighborhood search. The second method considers only a restricted neighborhood of design space points when selecting the next design transformation. The restricted neighborhood is defined as explained in Section 4.2. We call the second method RNS, restricted neighborhood search. Both ENS and RNS use the same cost function, defined in Eq. (2) and calculated according to

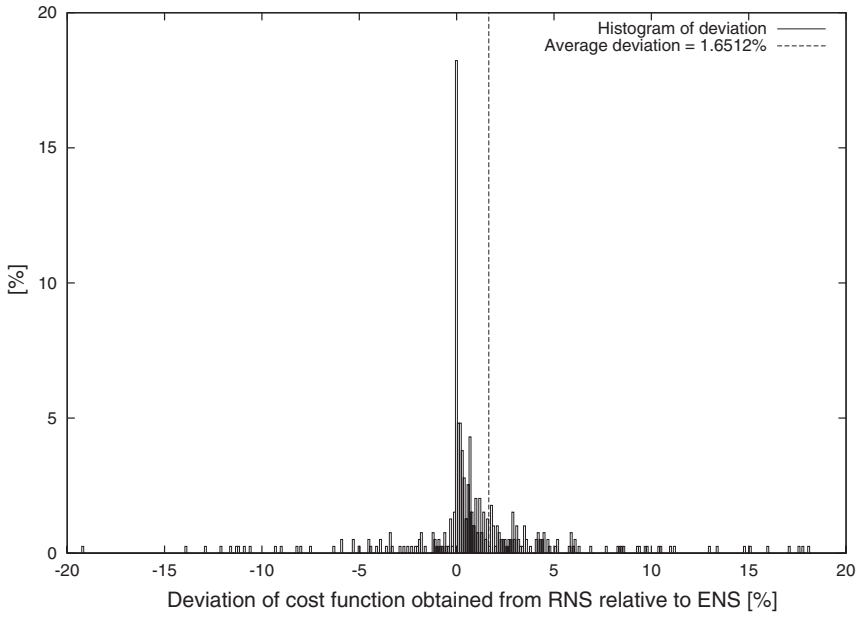


Fig. 10. Cost obtained by RNS versus ENS.

the approximate analysis described in Section 5. The third method considers the tasks as having fixed execution times, equal to the average task execution times. It is very similar to ENS, as it uses a tabu search with an exhaustive neighborhood search. The difference to ENS is that the third method is driven by a different cost function, namely $\sum lax_\tau$, where lax_τ is defined as follows

$$lax_\tau = \begin{cases} \infty & F_\tau > \delta_\tau \wedge \tau \text{ is critical} \\ F_\tau - \delta_\tau & \text{otherwise} \end{cases} \tag{19}$$

The third method is abbreviated LO-AET, laxity optimization based on average execution times. Once LO-AET has produced a solution, the cost function defined in Eq. (2) is calculated and reported for the produced mapping and priority assignment.

6.1 RNS and ENS: Quality of Results

The first issue we look at is the quality of results obtained with RNS compared to those produced by ENS. The deviation of the cost function obtained with RNS relative to the cost function obtained by ENS is defined as

$$\frac{cost_{RNS} - cost_{ENS}}{cost_{ENS}} \tag{20}$$

Figure 10 depicts the histogram of the deviation over the 396 benchmark applications. The relative deviation of the cost function appears on the x axis. The value on the y axis corresponding to a value x on the x axis indicates the percentage of the 396 benchmarks that have a cost function deviation equal to x. On average, RNS is only 1.65% worse than ENS. In 19% of the cases, the

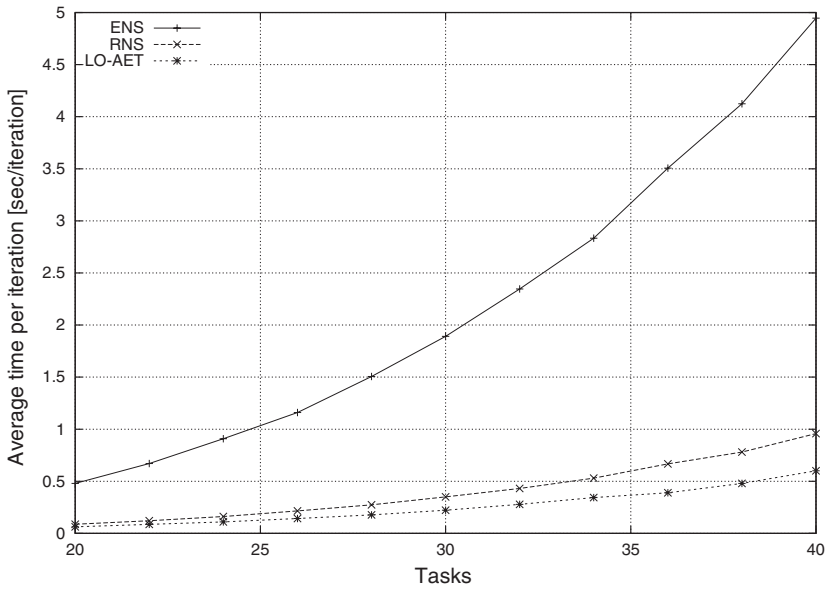


Fig. 11. Run times of RNS versus ENS.

obtained deviation was between 0 and 0.1%. Note that RNS can obtain better results than ENS (negative deviation). This is because of the intrinsically heuristic nature of tabu search.

6.2 RNS and ENS: Exploration Time

As a second issue, we compared the runtimes of RNS, ENS, and LO-AET. Figure 11 shows the average times needed to perform one iteration in RNS, ENS, and LO-AET, respectively. It can be seen that RNS runs on average 5.16–5.6 times faster than ENS. This corresponds to the theoretical prediction, made at in Section 4.2, stating that the neighborhood size of RNS is M times smaller than the one of ENS when $c = 2$. In our benchmark suite, M, the number of processors, is between 3 and 8, averaging to 5.5. We also observe that the analysis time is close to quadratic in the number of tasks, which again corresponds to the theoretical result that the size of the search neighborhood is quadratic in N, the number of tasks.

We finish the tabu search when $40 \cdot N$ iterations have executed, where N is the number of tasks. In order to obtain the execution times of the three algorithms, one needs to multiply the numbers on the ordinate in Figure 11 with $40 \cdot N$. For example, for 40 tasks, RNS takes circa 26 min while ENS takes roughly 2 hr 12 min.

6.3 RNS and LO-AET: Quality of Results and Exploration Time

The LO-AET method is marginally faster than RNS. However, as shown in Figure 12, the value of the cost function obtained by LO-AET is, on average, almost an order of magnitude worse (9.09 times) than the one obtained by RNS.

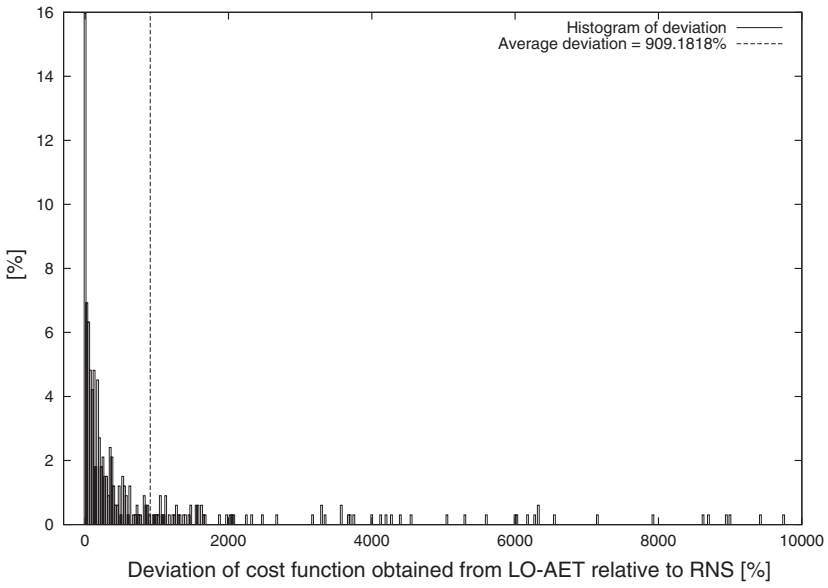


Fig. 12. Cost obtained by LO-AET versus RNS.

This supports one of the main messages of this article, namely, that considering a fixed execution time model for optimization of systems is completely unsuitable if deadline miss ratios are to be improved. Although LO-AET is able to find a good implementation in terms of average execution times, it turns out that this implementation is very poor from the point of view of deadline miss ratios. What is needed is a heuristic like RNS, which is explicitly driven by deadline miss ratios during design space exploration.

6.4 Real-Life Example: GSM Voice Decoding

Last, we considered an industrial-scale real-life example from the telecommunication area, namely a smart GSM cellular phone [Schmitz 2003], containing voice encoder and decoder, an MP3 decoder, as well as a JPEG encoder and decoder.

In GSM, a second of human voice is sampled at 8 kHz, and each sample is encoded on 13 bits. The resulting stream of 13,000 bytes per second is then encoded using so-called regular pulse excitation long-term predictive transcoder (GSM 06.10 specification [ETS]). The encoded stream has a rate of 13,000 bits per second, i.e., a frame of 260 bits arrives every 20 ms. Such a frame is decoded by the application shown in Figure 13. It consists of one task graph of 34 tasks mapped on two processors. The task partitioning and profiling was done by M. Schmitz [Schmitz 2003]. The period of every task is equal to the frame period, namely 20 ms. The tasks process an input block of 260 bits. The layout of a 260-bit frame is shown on the top of Figure 13, where the correspondence between the various fields in the frame and the tasks processing them is also depicted.

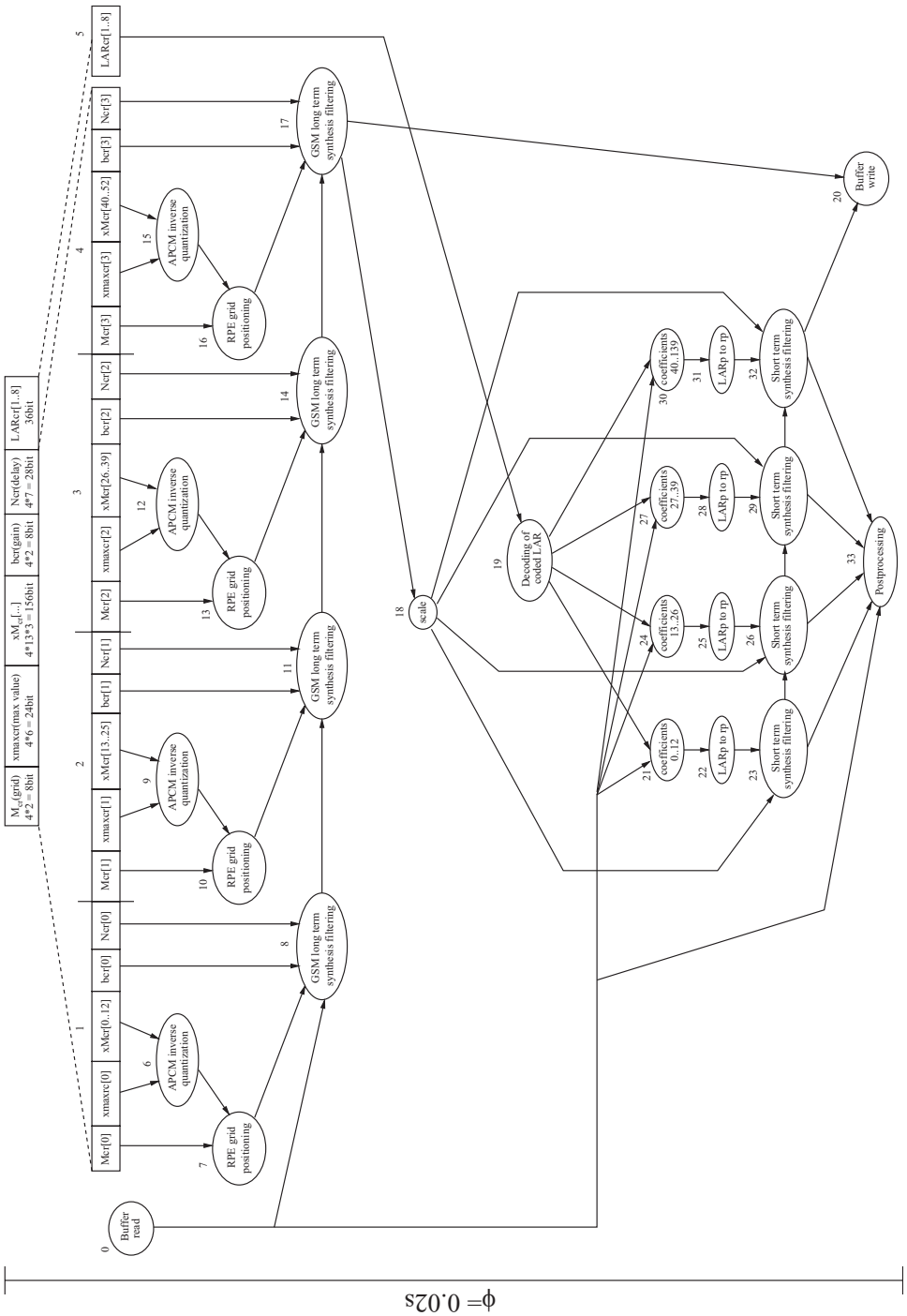


Fig. 13. Task graph modeling GSM voice decoding. (From M. Schmitz's PhD thesis.)

For all tasks, the deadline is equal to the period. No tasks are critical in this application, but the deadline miss threshold of every task is 0. Hence, the value of the cost function defined in Eq. (2) is equal to the sum of the deadline miss ratios of all 34 tasks and the deadline miss ratio of the entire application.

The restricted neighborhood search found a task mapping and priority assignment of cost 0.0255 after probing 729, 662 potential solutions in 1 hr 31 min on an AMD Athlon clocked at 1533 MHz. This means that the deadline miss ratio of the voice decoding application, if the tasks are mapped and their priority is assigned as found by the RNS, is less than 2.55%.

7. CONCLUSIONS

In this paper, we addressed the problem of design optimization of soft real-time systems with stochastic task execution times under deadline miss ratio constraints. The contribution is threefold. First, we have shown that methods considering fixed execution time models are unsuited for this problem. Second, we presented a design space exploration strategy based on tabu search for task mapping and priority assignment. Third, we introduced a fast and approximate analysis for guiding the design space exploration. Experiments demonstrated the efficiency of the proposed approach.

REFERENCES

ABENI, L. AND BUTAZZO, G. 1999. QoS guarantee using probabilistic deadlines. In *Proceedings of the* 11*th Euromicro Conference on Real-Time Systems*. 242–249.

ALIPPI, C., GALBUSERA, A., AND STELLINI, M. 2003. An application-level synthesis methodology for multidimensional embedded processing systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22,* 11, 1457–1470.

AUDSLEY, N. C. 1991. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Tech. Rep. YCS 164, Department of Computer Science, University of York. (Dec.).

AUDSLEY, N. C., BURNS, A., DAVIS, R. I., TINDELL, K. W., AND WELLINGS, A. J. 1991. Hard real-time scheduling: The deadline monotonic approach. In *Proceedings of the* 8*th IEEE Workshop on Real-Time Operating Systems and Software*. 133–137.

AUDSLEY, N., BURNS, A., RICHARDSON, M., TINDELL, K., AND WELLINGS, A. 1993a. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal 8,* 5, 284–292.

AUDSLEY, N. C., BURNS, A., RICHARDSON, M. F., AND WELLINGS, A. J. 1993b. Incorporating unbounded algorithms into predictable real-time systems. *Computer Systems Science and Engineering 8,* 3, 80–89.

BERNAT, G., COLIN, A., AND PETTERS, S. 2002. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the* 23*rd Real-Time Systems Symposium*. 279–288.

BINI, E., BUTAZZO, G., AND BUTAZZO, G. 2001. A hyperbolic bound for the rate monotonic algorithm. In *Proceedings of the* 13*th Euromicro Conference on Real-Time Systems*. 59–66.

BLAZEWICZ, J. 1976. Scheduling dependent tasks with different arrival times to meet deadlines. In *Modeling and Performance Evaluation of Computer Systems*, E. Gelenbe and H. Bellner, Eds. North-Holland, Amsterdam.

Bosch. 1991. *CAN Specification*. Bosch, Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Germany.

BROSTER, I., BURNS, A., AND RODRIGUEZ-NAVAS, G. 2002. Probabilistic analysis of CAN with faults. In *Proceedings of the* 23*rd Real-Time Systems Symposium*.

BURNS, A., PUNNEKKAT, S., STRIGINI, L., AND WRIGHT, D. 1999. Probabilistic scheduling guarantees for fault-tolerant real-time systems. In *Proceedings of the* 7*th International Working Conference on Dependable Computing for Critical Applications*. 339–356.

BURNS, A., BERNAT, G., AND BROSTER, I. 2003. A probabilistic framework for schedulability analysis. In *Proceedings of the Third International Embedded Software Conference, EMSOFT*, R. Alur and I. Lee, Eds. Number LNCS 2855 in Lecture Notes in Computer Science. 1–15.

BUTTAZZO, G. C. 1997. *Hard Real-Time Computing Systems*. Kluwer Academic Publ., Norwell, MA.

CARDOSO, R., KREUTZ, M., CARRO, L., AND SUSIN, A. 2005. Design space exploration on heterogeneous network-on-chip. In *Proceedings of the IEEE International Symposium on Circuits and Systems*. Vol. 1. 428–431.

DÍAZ, J. L., GARCÍA, D. F., KIM, K., LEE, C.-G., LO BELLO, L., LÓPEZ, J. M., MIN, S. L., AND MIRABELLA, O. 2002. Stochastic analysis of periodic real-time systems. In *Proceedings of the* 23*rd Real-Time Systems Symposium*.

ELES, P., PENG, Z., KUCHCINSKY, K., AND DOBOLI, A. 1997. System-level hardware/software partitioning based on simulated annealing and tabu search. *Journal on Design Automation for Embedded Systems 2*, 5–32.

European telecommunications standards institute. http://www.etsi.org/.

GARDNER, M. 1999. Probabilistic analysis and scheduling of critical soft real-time systems. Ph.D. thesis, University of Illinois at Urbana-Champaign.

GARDNER, M. AND LIU, J. W. 1999. *Analyzing Stochastic Fixed-Priority Real-Time Systems*. Springer, New York. 44–58.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability*. Freeman, San Francisco, CA.

GAUTAMA, H. 1998. A probabilistic approach to the analysis of program execution time. Tech. Rep. 1-68340-44(1998)06, Faculty of Information Technology and Systems, Delft University of Technology.

GAUTAMA, H. AND VAN GEMUND, A. J. C. 2000. Static performance prediction of data-dependent programs. In *Proceedings of the* 2*nd International Workshop on Software and Performance*. 216–226.

GLOVER, F. 1989. Tabu search—Part I. *ORSA Journal on Computing 1,* 3, 190–206.

GONZÁLEZ HARBOUR, M., KLEIN, M. H., AND LEHOCZKY, J. P. 1991. Fixed priority scheduling of periodic rasks with varying execution priority. In *Proceedings of the IEEE Real Time Systems Symposium*. 116–128.

HAJJI, O., BRISSET, S., AND BROCHET, P. 2002. Comparing stochastic optimization methods used in electrical engineering. *IEEE Transactions on Systems, Man and Cybernetics 7,* 6–9, 6.

HU, X. S., ZHOU, T., AND SHA, E. H.-M. 2001. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9,* 6 (Dec.), 833–844.

HUA, S., QU, G., AND BHATTACHARYYA, S. 2003. Exploring the probabilistic design space of multimedia systems. In *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping*.

HUGHES, C., KAUL, P., ADVE, S., JAIN, R., PARK, C., AND SRINIVASAN, J. 2001. Variability in the execution of multimedia applications and implications for architecture. In *Proceedings of the* 28*th International Symposium on Computer Architecture*. 254–265.

KALAVADE, A. AND MOGHÉ, P. 1998. A tool for performance estimation of networked embedded end-systems. In *Proceedings of the* 35*th Design Automation Conference*. 257–262.

KIM, J. AND SHIN, K. G. 1996. Execution time analysis of communicating tasks in distributed systems. *IEEE Transactions on Computers 45,* 5 (May), 572–579.

KLEINROCK, L. 1964. *Communication Nets: Stochastic Message Flow and Delay*. McGraw-Hill, New York.

KRISHNAMACHARI, B. AND WICKER, S. 2000. Optimization of fixed network design in cellular systems using local search algorithms. In *Proceedings of the 52nd Vehicular Technology Conference*. Vol. 4. 1632–1638.

LEHOCZKY, J. P. 1996. Real-time queueing theory. In *Proceedings of the* 18*th Real-Time Systems Symposium*. 186–195.

LEHOCZKY, J. P. 1997. Real-time queueing network theory. In *Proceedings of the* 19*th Real-Time Systems Symposium*. 58–67.

LEHOCZKY, J., SHA, L., AND DING, Y. 1989. The rate monotonic scheduling algorithm: Exact characterization and average case behaviour. In *Proceedings of the* 11*th Real-Time Systems Symposium*. 166–171.

LEUNG, J. Y. T. AND WHITEHEAD, J. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation 2,* 4, 237–250.

LI, Y. A. AND ANTONIO, J. K. 1997. Estimating the execution time distribution for a task graph in a heterogeneos computing system. In *Proceedings of the Heterogeneous Computing Workshop*.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM 20,* 1 (Jan.), 47–61.

MANOLACHE, S., ELES, P., AND PENG, Z. 2001. Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *Proceedings of the* 13*th Euromicro Conference on Real Time Systems*. 19–26.

MANOLACHE, S., ELES, P., AND PENG, Z. 2002. Schedulability analysis of multiprocessor real-time applications with stochastic task execution times. In *Proceedings of the* 20*th International Conference on Computer Aided Design*. 699–706.

MANOLACHE, S., ELES, P., AND PENG, Z. 2004a. Optimization of soft real-time systems with deadline miss ratio constraints. In *Proceedings of the* 10*th IEEE Real-Time and Embedded Technology and Applications Symposium*. 562–570.

MANOLACHE, S., ELES, P., AND PENG, Z. 2004b. Schedulability analysis of applications with stochastic task execution times. *ACM Transactions on Embedded Computing Systems 3,* 4, 706–735.

MOULY, M. AND PAUTET, M.-B. 1992. *The GSM System for Mobile Communication*. Palaiseau.

NG, J., LEUNG, K., WONG, W., LEE, V. C., AND HUI, C. K. 2002. Quality of service for MPEG video in human perspective. In *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications*. 233–241.

NISSANKE, N., LEULSEGED, A., AND CHILLARA, S. 2002. Probabilistic performance analysis in multiprocessor scheduling. *Computing and Control Engineering Journal 13,* 4 (Aug.), 171–179.

PALENCIA GUTIÉRREZ, J. C. AND GONZÁLEZ HARBOUR, M. 1998. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the* 19*th IEEE Real Time Systems Symposium*. 26–37.

PIERRE, S. AND HOUETO, F. 2002. Assigning cells to switches in cellular mobile networks using taboo search. *IEEE Transactions on Systems, Man and Cybernetics 32,* 3, 351–356.

SCHMITZ, M. 2003. Energy minimisation techniques for distributed embedded systems. Ph.D. thesis, Dept. of Computer and Electrical Enginieering, Univ. of Southampton, UK.

SPURI, M. AND STANKOVIC, J. A. 1994. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers 43,* 12 (Dec.), 1407–1412.

SUN, J. 1997. Fixed-priority end-to-end scheduling in distributed real-time systems. Ph.D. thesis, University of Illinois at Urbana-Champaign.

SUN, J. AND LIU, J. W. S. 1995. Bounding the end-to-end response time in multiprocessor real-time systems. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*. 91–98.

SUN, J., GARDNER, M. K., AND LIU, J. W. S. 1997. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Transactions on Software Engineering 23,* 10 (Oct.), 604–615.

TIA, T.-S., DENG, Z., SHANKAR, M., STORCH, M., SUN, J., WU, L.-C., AND LIU, J. W. S. 1995. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. 164–173.

TINDELL, K. AND CLARK, J. 1994. Holistic schedulability analysis for distributed real-time systems. *Euromicro Jurnal on Microprocessing and Microprogramming (Special Issue on Parallel Embedded Real-Time Systems) 40*, 117–134.

TRAFERRO, S., CAPPARELLI, F., PIAZZA, F., AND UNCINI, A. 1999. Efficient allocation of power of two terms in FIR digital filter design using tabu search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 3*, 411–414.

VAN GEMUND, A. J. 1996. Performance modelling of parallel systems. Ph.D. thesis, Delft University of Technology.

VAN GEMUND, A. 2003a. Symbolic performance modeling of parallel systems. *IEEE Transactions on Parallel and Distributed Systems 14,* 2 (Feb.), 154–165.

VAN GEMUND, A. J. C. 2003b. Symbolic performance modeling of parallel systems. *IEEE Transactions on Parallel and Distributed Systems 14*, 2, 154–165.

WILD, T., FOAG, J., PAZOS, N., AND BRUNNBAUER, W. 2003. Mapping and scheduling for architecture exploration of networking SoCs. In *Proceedings of the 16th International Conference on VLSI Design*. 376–381.

ZHOU, T., HU, X. S., AND SHA, E. H.-M. 1999. A probabilistic performace metric for real-time system design. In *Proceedings of the 7th International Workshop on Hardware-Software Co-Design*. 90–94.

Received March 2006; revised August 2006; accepted October 2006