Applying Sensitivity Analysis in Real-Time Distributed Systems

Razvan Racu, Marek Jersak, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
D-38106 Braunschweig / Germany
{racu | jersak | ernst}@ida.ing.tu-bs.de

Abstract

During real-world design of embedded real-time systems, it cannot be expected that all performance data required for scheduling analysis is fully available up front. In such situations, sensitivity analysis is a promising approach to deal with uncertainties that result from incomplete specifications, early performance estimates, late feature requests, and so on. Sensitivity analysis allows the system designer to keep track of the flexibility of the system, and thus to quickly assess the impact of changes of individual hardware and software components on system performance. In this paper we integrate sensitivity analysis into our system-level performance analysis framework SymTA/S and show its benefits during the design of complex, networked multi-processor embedded real-time systems.

1. Introduction

Scheduling analysis for real-time embedded systems has grown beyond single-processor problems and can now cover heterogeneous, networked multi-processor systems. This extension of scope is a major step towards applicability of scheduling analysis for increasingly complex real-world problems. However, several issues remain to be solved before scheduling analysis can achieve widespread acceptance in industries such as automotive, telecommunication or consumer electronics.

A key issue is sensitivity analysis. In a real-world design flow with tight time-to-market pressure, ever changing requirements, and complex supply-chains including platform-based design, subsystem integration and IP-reuse, it cannot be expected that all performance data required for scheduling analysis is fully available up front. Instead, designers must work with incomplete specifications, early performance estimates, numbers asserted by suppliers in contracts but not yet proven, and so on. Additionally, designer must keep future modifications in mind, in particular late feature requests, product variants and the next product generation.

Sensitivity analysis is a promising approach to deal with those design uncertainties. It allows the system designer to keep track of the flexibility of the system, and thus to quickly assess the system-level impact of changes in performance properties of individual hardware and software components. For example, variations in the implementation of different application parts, functional extensions, or changes of timing at subsystem or system interfaces are issues that can turn a previously conforming system into one that violates performance constraints. These and many other variations are only too common in a realistic design-flow.

In this paper we propose a system-level sensitivity analysis technique based on a binary search algorithm. It is important to emphasize that, contrary to previous work, we concern ourselves with the effects of local variations on global system properties which are constrained. In particular we consider end-to-end deadlines, workload on different resources, maximum permissible output jitter and maximum available memory for communication buffers. We focus on variations of task core execution times and resource speeds. However, the proposed algorithm can be easily adapted to almost any system parameter. Our approach has been implemented in SymTA/S, a tool for performance analysis of heterogeneous multi-processor real-time systems.

The paper is structured as follows: in Section 2 we give an overview of existing sensitivity analysis approaches and their limitations. Section 3 presents the SymTA/S tool and gives details about our compositional system-level analysis approach. Our sensitivity analysis algorithm is described in Section 4. In Section 5 and Section 6 we apply the proposed sensitivity analysis algorithms to a hypothetical multi-processor system-on-chip example. Furthermore, we perform a set of experiments to evaluate the complexity of the proposed algorithms. Finally, we conclude and derive some future research ideas.

Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05)
1080-1812/05 $ 20.00 IEEE

2. Related Work

Most analysis techniques known from literature give a pure *Yes/No* answer regarding the timing behavior of a specific system with respect to a set of timing constraints defined for that system. Usually the analyses consider a predefined set of input parameters and determine the response times, and thus, the schedulability of the system.

However, in a realistic system design process it is important to get more information with respect to the effects of parameter variations on system performance, as such variations are inevitable during implementation and integration. Capturing the bounds within which a parameter can be varied without violating the timing constraints offers more flexibility for the system designer and supports future changes. These bounds show how *sensitive* the system or system parts are to system configuration changes.

In 1973, Liu and Layland [10] defined the utilization bound on a resource that guarantees the schedulability of a task set under a rate monotonic scheduling policy. The proposed algorithm is limited by specific system configurations: periodically activated tasks, tasks with deadlines at the end of the period, and tasks that do not share common resources (like semaphores) or that do not inter-communicate.

Later on, Lehoczky et al [9] extended this approach to systems with arbitrary task priority assignment. He defined the *critical scaling factor* Δ^* as the largest possible scaling factor for task execution times that still guarantees the schedulability of the entire task set. However, his approach does not go beyond the limitations mentioned above.

His concept is used later by Katcher et al [8] to define a breakdown utilization for the schedulability analysis of fixed-priority task sets considering the influences of the scheduling kernel. He evaluated and compared different scheduling implementations, but the models he used are still limited to periodically activated tasks with $D = T$.

Steve Vestal [24] proposed the first approach regarding the sensitivity analysis of fixed-priority task sets. He introduced slack variables into the inequalities proposed by Lehoczky [9] in order to transform them into equalities and solve them to obtain the maximum limit of each task core execution time. His approach considers blocking times as well as task sets with linear execution time models. Again, only periodic tasks with deadlines at the end of the period are subject for the analysis.

Yerraballi [25] proposed a similar algorithm in order to identify the sensitivity of schedulability analysis with respect to task execution times. Later on, Cottet and Babau [2] provided a graphical approach to determine the system sensitivity to variations of activation periods, deadlines and release times. The analyses are still constrained by the model used in [10].

The analysis given by Punnekkat [15] uses a combination of a binary search algorithm and a modified version of the response time schedulability tests proposed by Audsley and Tindell [1][23]. However, no details are given regarding the system models that can be investigated and no mathematical support is provided together with the binary search algorithm. Moreover, the presented set of experiments are still restricted to periodic task with deadlines less than periods.

In [16] Regehr introduced a binary search algorithm in order to determine systems that, on one hand, offer a high flexibility with respect to task execution times and, on the other hand, they guarantee a minimum number of threads required to run a given task set.

Recently, ETAS and Live Devices developed the *RTA-OSEK* [11] product that has a built-in sensitivity analysis engine for embedded systems based on OSEK-OS.

3. The SymTA/S Approach

3.1. Overview

SymTA/S [4] is a software tool for formal performance analysis of heterogeneous SoCs and distributed systems. The core of SymTA/S is our recently developed technique to couple scheduling analysis algorithms using event streams [18, 21]. Event streams describe the possible I/O timing of tasks and are characterized by appropriate event models such as periodic events with jitter or bursts and sporadic events. At the system level, event streams are used to connect local analyses according to the systems application and communication structure.

In contrast to previous work, SymTA/S explicitly supports the combination and integration of different kinds of analysis techniques known from real-time research. For this purpose, it is essential to transition between the often incompatible event stream models resulting from the dissimilitude of the local techniques. This kind of incompatibility appears, for instance, between an analysis technique assuming periodic events with jitter and an analysis technique requiring sporadic events. In SymTA/S we use *event model interfaces (EMIFs)* and *event adaptation functions (EAFs)* to realize these essential transitions [18].

However, integration of heterogeneous systems is not the sole domain of application for EMIFs and EAFs. In SymTA/S so-called shapers can be inserted into any event stream. Shapers are basically EMIF-EAF combinations which manipulate an event stream, and thus, the interaction between two components. They provide control about the timing of exchanged events and data and consequently also about performance dependencies. We have shown in [20] that this is especially

important to break up non-functional dependency cycles and to reduce transient load peaks in dynamic systems. In other words, due to the event model transformation provided by EMIFs and EAFs, SymTA/S is able to analyze many real world examples that holistic approaches [23, 14] cannot handle.

In order to perform a system level analysis, SymTA/S locally performs existing scheduling analyses (e.g. RMS, TDMA, Round Robin, etc.) and propagates their results to the neighboring components. This analysis-propagate mechanism is repeated iteratively until all components are analyzed, which means that all output streams remained unchanged.

The above described basic SymTA/S approach has been recently extended to support multi-rate systems, tasks with multiple activating inputs (OR- or AND-concatenated), conditional communication and functional cycles [5, 7]. These major extensions enable SymTA/S to cope with complex real-world applications.

Furthermore, SymTA/S is able to consider system context information to tighten analysis bounds. We define as a system context all kinds of correlations between activating events that go beyond the possible timing of consecutive events in one event stream. *Inter* event stream contexts, initially introduced by Tindell [22] and generalized by Palencia and Harbour [13], consider possible phases between events in different event streams, thus allowing to calculate a tighter number of interrupts of a task by other tasks sharing the same component. *Intra* event stream contexts, initially introduced by Mok and Chen [12], consider correlations between successive computation or communication requests, thus allowing to calculate a tighter load for a number of successive activations of a task. Both types of contexts lead to the calculation of shorter worst-case and longer best-case response times. In [6] we presented the generalization of intra event stream contexts, the combination of both types of contexts during analysis, and explicit distinction between different types of events on one hand, and different task behaviors on the other. The latter is crucial for subsystem integration and compositional performance analysis, since different types of events are a property of the sender, while different behaviors are a property of the receiver.

SymTA/S additionally features a powerful design-space exploration engine [3] which currently employs evolutionary algorithms. The designer defines the search-space, for example valid priority assignments, maximum buffer sizes or valid mappings of tasks to resources. He defines optimization criteria which the algorithm then uses to find pareto-optimal solutions. During the search, the best solutions found so far are displayed. The designer can stop exploration at any time, take a closer look at particular solutions, or modify the search-space or optimization criteria.

3.2. Analysis model

A task is activated due to an activating event. Activating events can be generated in a multitude of ways, including expiration of a timer, external or internal interrupt, and task chaining. In order to execute, a task needs to be mapped on a *computation* or a *communication* resource. A task mapped on a *computation* resource is called *computation task* or sometimes *process*. Similarly, a task mapped on a *communication resource* is refered as *communication task* or sometimes *channel*. The task *core execution time* represents the time a computation/communication task needs to complete assuming the resource was exclusively assigned to it, i.e no interruption occurs during execution. Since the internal execution time of a task is usually not constant, it is generally expressed in form of a $[best-case, worst-case]$ interval. As multiple tasks can share the same resource, a *scheduler* grants the resource to a specific task, out of the set of active tasks, according to a particular scheduling policy. In SymTA/S, *scheduling analysis* calculates both worst-case and best-case task response times, i.e. the time between task activation and task completion, for all tasks sharing a resource under the control of a scheduler. Scheduling analysis guarantees that all observable response times will fall into the calculated $[best-case, worst-case]$ interval. Therefore, we say that scheduling analysis is conservative.

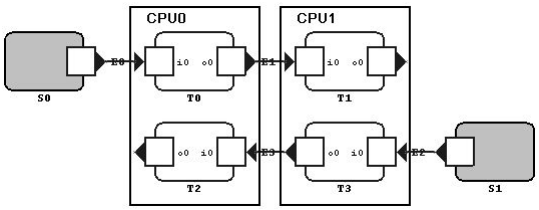

Figure 1. Example of a system modeled with SymTA/S

Figure 1 shows an example of a system modeled with SymTA/S. The system consists of 2 resources, each with 2 tasks mapped on it. *S0* and *S1* are the sources of activating events at the external system inputs. *CPU0* and *CPU1* are both assumed to be priority scheduled. Task *T2* has the highest priority on *CPU0* while *T1* has the highest priority on *CPU1*. The possible timing of activating events is captured by so-called *event models*, which are explained in detail in [19] and [20].

The compositional performance analysis methodology used in SymTA/S alternates local scheduling analysis and event model propagation during system-level analysis. This technique requires the possible timing of output events in

order to determine the activation of the next scheduling component. The event models used in SymTA/S allow to specify simple rules to obtain output event models that can be described with the same set of parameters as the input event models. The output event model period obviously equals the activation period. The output event model jitter is obtained by adding the difference between maximum and minimum response times (the response time jitter) to the activating event model jitter (equation 1).

$$\mathcal{J}_{out} = \mathcal{J}_{act} + (R_i - r_i) \quad (1)$$

The maximum and minimum response times, R_i and r_i, are calculated by scheduling analysis as a function of all activating event models for tasks mapped on that resource, and the worst-case respectively best-case core execution times of those tasks.

In the following we explain the compositional analysis approach using the system example in Figure 1. Initially, only the activating event models of $T0$ and $T3$ are known. At this point the system cannot be analyzed, because on every resource an activating event model for one task is missing. We need to calculate the response times on $CPU0$ to be able to analyze $CPU1$. On the other hand, we cannot analyze $CPU0$ before analyzing $CPU1$. We call this problem a *cyclic scheduling dependency*.

One solution to this problem is to initially propagate all external event models along all system paths until an initial activating event model is available for each task [17]. This approach is safe, since, on one hand, scheduling cannot change an event model period. On the other hand, scheduling can only *increase* an event model jitter [23]. Since a smaller jitter interval is contained in a larger jitter interval, the minimum initial jitter assumption is safe.

After propagating external event models, global system analysis can be performed. A global analysis step consists of two phases [20]. In the first phase local scheduling analysis is performed for each resource and output event models are calculated. In the second phase, all output event models are propagated. It is then checked if the first phase has to be repeated because some activating event models are no longer *up-to-date*, meaning that a newly propagated output event model is different from the output event models that was propagated in the previous global analysis step. Analysis completes if either all event models are up-to-date after the propagation phase, or if an abort condition, e. g. the violation of a timing constraint has been reached.

4. Sensitivity Analysis in SymTA/S

In this section we give an overview about the sensitivity analysis framework used in SymTA/S which is based on a combination of a binary search technique and the compositional analysis model presented in Section 3.2. First, we will describe the metrics and the objectives we use for the sensitivity analysis, then we will give some mathematical issues regarding the applicability of the binary search algorithm in the SymTA/S analysis model. Finally, we will present the algorithm implementation for different analysis metrics.

4.1. Metrics

The system designer usually wants to know the flexibility of the system behavior with respect to changes in the SW-part as well as in the HW-part. Variations in the implementation of different application parts, functional extensions, or changes of requirements such as external activation patterns are only some examples that can turn a previously conforming system into one that violates performance constraints. These and many other variations are only too common in a realistic design-flow. Therefore, it is desirable to have permanent control of the system reserves throughout the design process.

In this paper, we consider the following metrics for sensitivity analysis:

1. Maximum permissible increase of the core execution time of a task without violating system constraints or system schedulability.
2. Minimum speed factor of a resource without violating system constraints or system schedulability.

These objectives are currently implemented in SymTA/S. Other reasonable metrics for the sensitivity analysis would be:

1. Variations of different external event model parameters, e. g. periods or jitters.
2. Variations of internal event models by means of traffic shaping.

It is important to emphasize that, contrary to previous work, we focus on the effects of local variations on global system properties which are constrained. In particular, we consider end-to-end deadlines, workload on different resources, maximum permissible output jitter and maximum available memory for communication buffers.

4.2. Binary search technique

As mentioned in Section 2, different approaches were proposed for the sensitivity analysis of different system parameters. However, most of them can perform only a local resource analysis, as they are bounded by local constraints (tasks deadlines). Due to a fast increase of system complexity and heterogeneity, the current distributed systems usually have to satisfy global constraints rather than local ones.

End-to-end deadlines or global buffer limits are an example of such constraints. Hence, the formal approaches used for the sensitivity analysis at resource level cannot be transformed and applied at the system level, as this implies huge effort and less flexibility.

Binary search is known as a simple and fast searching algorithm used to determine a specific value within an *ordered* set of data. Our analysis framework is based on a binary search algorithm that determines the sensitivity of the system behavior with respect to changes of different system parameters. As already mentioned in Section 4.1, the task core execution time is an important objective for the sensitivity analysis. Changes of its values directly affects a set of system parameters, like buffer sizes, response times, workload on different resources. Due to system-level dependencies, the effects of these changes are further propagated in the whole system.

Suppose that the worst-case core execution time of task *T1* in Figure 1 increases. As *T1* has the highest priority on *CPU1*, the worst-case response time of *T3* will also increase and, consequently, its output event model jitter as well. Since the activation timing of *T2* is determined by the output event model of *T3*, the scheduling on *CPU0* will be affected, too. The different activating event model for task *T2* may change the worst- and best-case response times of *T0*. Hence, the change of only one system parameter turned into a long sequence of changes of other system properties.

We use binary search to determine the maximum value of a system parameter that still leads to a conforming system configuration, i.e. all constraints are satisfied and all tasks in the system are schedulable.

We first vary worst-case task execution times. The worst-case response time R_i of any task Ti in the system shown in Figure 1 can be determined using the response time analysis presented in [1].

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \qquad (2)$$

where C_i represents the worst-case core execution time of task i, and T_i is its activation period. From Equation 2 it results that R_i is a function depending on the worst-case execution time of task i and of all tasks with a priority higher than task i. It is obvious that R_i is a monotonically increasing function of C_i and C_j, where $j \in hp(i)$. Similar argumentations can be given for other scheduling analysis techniques.

As J_{act} and r_i are not functions of C_i, it follows from Equation 1 that the output jitter J_{out} is also a monotonically increasing function of C_i. An increasing jitter may lead to an increasing response time of the connected task or can determine a larger buffering delay in case of traffic shaping.

Since the maximum latency of a path is defined as the sum of the worst case response times of all tasks along that path plus the buffering delay, the paths latencies are also monotonically increasing functions of a specific task core execution time. This guarantees that any value smaller than the maximum value found will also lead to a conforming system configuration.

4.3. Algorithm implementation

A system can be in 2 different states with respect to its current configuration and the constraints:

1. The system status is *success* if all constraints are satisfied and all tasks are schedulable.
2. The system status is *fail* if at least one constraint is violated or at least one task is not schedulable.

In both cases it is desired to determine the value of a specific system parameter that represents the bound between the *success*-status and *fail*-status.

Algorithm 1 determines the sensitivity of the system with respect to changes of a specific task core execution time. If the system status is *success*, then the search interval is determined by the current WCET value and the value corresponding to a maximum load allowed on the resource.

The current load (utilization) on a resource is [10]:

$$Load_{current} = \frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_i}{T_i} + ... + \frac{C_n}{T_n} \qquad (3)$$

If we increase the load of task i by Δt_i, then the new load is:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + ... + \frac{C_i + \Delta t_i}{T_i} + ... + \frac{C_n}{T_n} \leq Load_{max} \qquad (4)$$

which must be smaller than the maximum load allowed on the resource, $Load_{max}$. Hence,

$$\Delta t_i \leq T_i \times (Load_{max} - Load_{current}) \qquad (5)$$

If the system status is currently *fail*, then the search interval is determined by the current WCET value and 0. The system can remain in *fail*-status even if the algorithm reduces the WCET to 0.

During each iteration of Algorithm 1, a global analysis loop is performed by SymTA/S until the system is stable, i. e. the system parameters remain unchanged. After each iteration of the algorithm, the proper half of the previous search interval is taken, and the task WCET is set to the new search interval middle value. If the system status is *fail*, then a smaller value is required, and thus, the first half of the interval becomes the new search interval. Otherwise, a larger value is required and the second half of the interval is investigated. The algorithm terminates when the search interval size becomes smaller than a predefined value ε.

A similar algorithm (2) is used to determine the minimum speed of a resource that still leads to a *success*-status

Algorithmus 1 Task execution time variation

Require: current WCET: C
current resource load: $Load_R$
maximum load: $Load_{max}$
algorithm precision: ε
task activation period: T
Ensure: maximum WCET allowed: C_{max}

1: $C_{init} = C$
2: $Load_{current} = Load_R$
3: $Load_{prev} = 0$
4: analyze system;
5: **if** $(system\_success)$ **then**
6: $low = C$
7: $high = C + T \times (Load_{max} - Load_R)$
8: **else**
9: $low = 0$
10: $high = C$
11: **end if**
12: **while** $(Load_{current} - Load_{prev}) > \varepsilon$ **do**
13: $middle = (high + low)/2$
14: $C = middle$
15: analyze system;
16: **if** $(system\_success)$ **then**
17: $low = middle$
18: **else**
19: $high = middle$
20: **end if**
21: $Load_{prev} = Load_{current}$
22: $Load_{current} = Load_R$
23: **end while**
24: $C = C_{init}$
25: $C_{max} = low$

of the system. The core execution times of all tasks mapped on a resource are inverse functions of the resource speed, i.e. reducing the resource speed by a factor sf will increase the WCETs of all tasks on that resource by the same factor and vice versa. If we modify the resource speed by sf, Equation 3 becomes:

$$\frac{1}{sf}\sum_{i=1}^{n}\frac{C_i}{T_i} \leq Load_{max} \tag{6}$$

The new load obviously has to be smaller than the maximum load allowed on the resource. Hence,

$$sf \geq \frac{Load_{current}}{Load_{max}} \tag{7}$$

Similarly, the designer can define a minimum load and the maximum value for sf. This factor corresponds to the maximum clock frequency at which the resource can operate.

$$\frac{1}{sf}\sum_{i=1}^{n}\frac{C_i}{T_i} \geq Load_{min} \tag{8}$$

and thus,

$$sf \leq \frac{Load_{current}}{Load_{min}} \tag{9}$$

If the system status is initially *success*, then the interval between $\frac{Load_{current}}{Load_{max}}$ and the current speed factor is selected for analysis. Otherwise the interval determined by the current speed and the maximum speed factor is investigated.

Algorithmus 2 Resource speed variation

Require: current speed factor: $Speed_R$
current resource load: $Load_R$
maximum load: $Load_{max}$
algorithm precision: ε
minumum load: $Load_{min}$
Ensure: minimum speed allowed: $Speed_{min}$

1: $Speed_{init} = Speed_R$
2: $Load_{init} = Load_R$
3: analyze system;
4: **if** $(system\_success)$ **then**
5: $low = Load_{init}/Load_{max}$
6: $high = Speed_R$
7: **else**
8: $low = Speed_R$;
9: $high = Load_{init}/Load_{min}$
10: **end if**
11: **while** $(high - low) > \varepsilon$ **do**
12: $middle = (high + low)/2$
13: $Speed_R = middle$
14: analyze system;
15: **if** $(system\_success)$ **then**
16: $high = middle$
17: **else**
18: $low = middle$
19: **end if**
20: **end while**
21: $Speed_R = Speed_{init}$
22: $Speed_{min} = low$

The ε value that appears in both algorithms represents the precision of the computation. It determines how tightly the result approximates the exact searched value.

5. System Example

The system in Figure 2 shows a hypothetical system modeled in SymTA/S consisting of a micro-controller (*uC*), a digital signal processor (*DSP*) and dedicated hardware (*HW*), all connected via an on-chip bus (*BUS*). The physical system is monitored by 3 sensors (*Sensor_1 - Sensor_3*),

which produce data sporadically as a reaction to irregular system events. This data is registered by an OR-activated monitor task (*Monitor*) on the *uC*, which decides how to update the control algorithm. This information is sent to task *Update* on the *DSP*, which writes the updated controller parameters into shared memory.

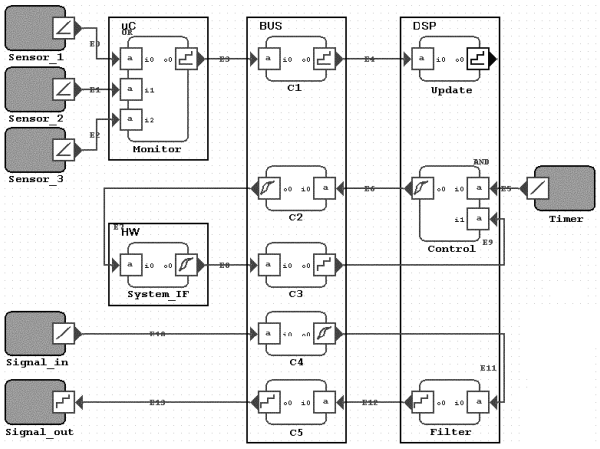

Figure 2. System Example

The *HW* acts as an interface to a physical system. It runs one task (*System_IF*) which issues actuator commands to the physical system and collects routine sensor readings. *System_IF* is controlled by controller task *Control*, which evaluates the sensor data and calculates the necessary actuator commands. *Control* is activated by a periodic timer (*Timer*) and by the arrival of new sensor data (AND-activation).

The *DSP* additionally executes a signal-processing task (*Filter*), which filters a stream of data arriving at input *Signal_in*, and sends the processed data via output *Signal_out*. All communication (with the exception of shared-memory on the *DSP*) is carried out by communication tasks *C1 - C5* over the on-chip *BUS*.

Computation and communication tasks have core execution times as listed in Table 1. All numbers are given in abstract time units.

We assume the event models given in Table 2 at system inputs. In case of sporadic events, the period parameter $\mathcal{P}$ denotes the maximum period or, in other words, the minimum distance between events.

In order to function correctly, the system has to satisfy the path latency constraints and the maximum jitter constraint at *Signal_out* listed in Tables 3(a) and 3(b).

In the following, we assume that the *DSP* as well as the *BUS* are scheduled according to a static priority preemptive policy. We want to determine the sensitivity of the system to local changes in task exceution times and resource speeds for a given priority assignment. The priority assignments on both *DSP* and *BUS* are given in Table 4.

task	core execution time
Monitor	[10,12]
System_IF	[15,15]
Filter	[12,15]
Update	[5,5]
Control	[20,23]
C1	[4,4]
C2	[4,4]
C3	[4,4]
C4	[8,8]
C5	[4,4]

Table 1. Tasks core execution times

input	event model
Sensor_1	sporadic, $\mathcal{P}_{Sensor\_1} = 1000$
Sensor_2	sporadic, $\mathcal{P}_{Sensor\_2} = 750$
Sensor_3	sporadic, $\mathcal{P}_{Sensor\_3} = 600$
Signal_in	periodic, $\mathcal{P}_{in} = 60$
Timer	periodic, $\mathcal{P}_{Timer} = 70$

Table 2. System input event models

#	path	maximum latency
1	*Sensor_i* $\rightarrow$ *Update*	100
2	*Signal_in* $\rightarrow$ *Signal_out*	80
3	cycle(e.g. *Control* $\rightarrow$ *Control*)	140

(a) Path latency constraints

#	output	event model jitter
4	*Signal_out*	$\mathcal{J}_{out,max} = 40$

(b) Maximum jitter constraint at Signal_out

Table 3. System constraints

6. Sensitivity Analysis

In this section, we perform sensitivity analysis of the system described in Section 5. In the first set of experiments, we determine the maximum valid increase of the WCET for each task while keeping all other values constant. Then we

task	priority	mapping
Monitor	1	*uC*
System_IF	1	*HW*
Update	1	*DSP*
Filter	2	*DSP*
Control	3	*DSP*
C1	5	*BUS*
C2	3	*BUS*
C3	4	*BUS*
C4	1	*BUS*
C5	2	*BUS*

Table 4. Tasks priority assignment

vary the speed of one resource at a time in order to find the minimum resource speed scaling factor. In the second set of experiments, we focus on algorithm complexity as a function of computation precision and different bounds of the resource load.

6.1. Task core execution time

Table 5 shows the results corresponding to sensitivity analysis of task core execution times. The third column contains the maximum value allowed for task execution times if we change only one task at a time. We performed the analysis with a precision (ε) of 10^{-2} and a maximum load bound of 98%.

task	current WCET	max WCET
C1	4.00	12.32
C2	4.00	14.00
C3	4.00	14.00
C4	8.00	15.00
C5	4.00	14.00
Monitor	12.00	23.00
System_IF	15.00	38.00
Update	5.00	9.66
Filter	15.00	22.00
Control	23.00	37.00

Table 5. Maximum values of tasks WCETs

Figure 3 depicts the results presented in table 5. The darker region in the bar diagrams represents the reserves corresponding to each task core execution time. In this particular experiment, we observe that the reserves of the tasks mapped on the same resource are about the same percentage of the current WCET values. This is an indication that it is worth trying to clock down the resource, consequently scaling all tasks core execution times with a common factor (Section 6.2).

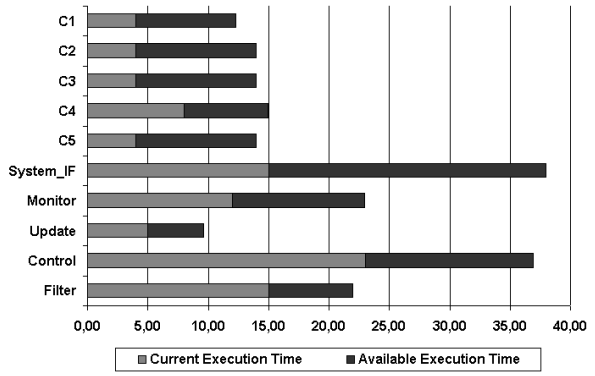

Figure 3. Sensitivity analysis of tasks WCETs

6.2. Resource speed factor

We perform similar experiments in order to determine the minimum speed factor of a resource. Table 6 shows that the speed of *HW* and *uC* can be decreased by about 60% and 50% respectively, as both resources are running only one task. The analysis was performed with a precision of 10^{-2} and a maximum load bound of 100%. The current speed values correspond to a speed factor of 1. The values presented in the third column are the relative values of the current speed factor.

resource	current speed	min speed
HW	1.00	0.40
DSP	1.00	0.78
BUS	1.00	0.67
uC	1.00	0.53

Table 6. Minimum values of resource speed factors

The bar diagrams in Figure 4 show the minimum speed with respect to current resource speed.

6.3. Algorithm complexity vs. precision of calculation

In this section, we perform a set of experiments in order to determine the algorithm run-time when varying the

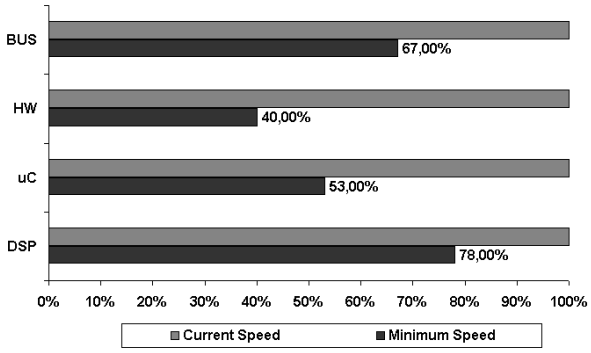

Figure 4. Sensitivity analysis of resource speed factors

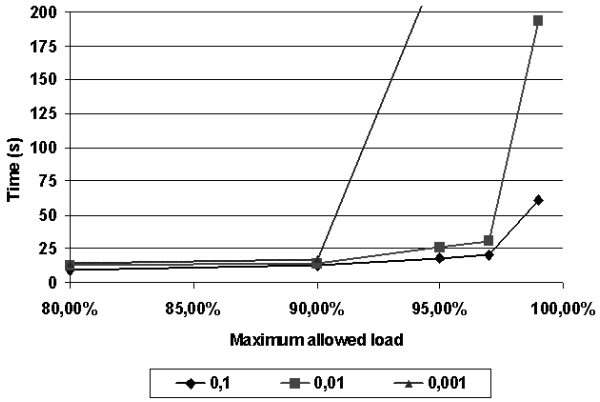

Figure 5. The run-time of the sensitivity analysis algorithm

precision of computation and the resource load bounds.

As already mentioned in Section 4.3, the ε value selects the algorithm precision, and the $Load_{max}$ and $Load_{min}$ values specify the load bounds allowed on a resource. In our experiments we vary the ε value between 10^{-3} and 10^{-1}. The $Load_{max}$ varies between 80% and 100%. Table 7 shows the run-time of the binary search algorithm (values are expressed in ms) for different combinations of the predefined values.

	Maximum allowed load				
ε	80%	90%	95%	97%	99%
0,1	9	13	18	21	61
0,01	12	15	26	31	193
0,001	14	18	236	-	-

Table 7. Sensitivity analysis: algorithm run-time (in seconds)

Figure 5 shows the values presented in Table 7.

We observe that for algorithm precisions smaller than 10^{-2} and load bounds above 97%, the run-time of the sensitivity analysis algorithm drastically increases. This is due to a large number of system analysis steps performed close to 100% load. At these load values on one hand the run-time of local scheduling analysis algorithms increases, as the number of consecutive activations that have to be considered grows. On the other hand, the number of system-level analysis steps increases in order to calculate the system-level impact of growing output jitters. However, a resource load above 97% is not realistic in practice due to variations of the system clock frequency or other distorting factors, and because designers usually want to maintain a larger headroom. Therefore, we feel that a load-bound around 95% is realistic in practice. For such a value, our algorithms are efficient.

7. Conclusion

In this paper we presented a sensitivity analysis framework based on a binary search technique. The sensitivity analysis determines the system reserves and describes the system flexibility during design phase while guaranteeing the satisfiability of system constraints.

We derived algorithms for the sensitivity analysis of task core execution times and resource speeds and we presented a set of experiments that demonstrated the applicability of these algorithms in a networked multi-processor system example. We are currently working on similar algorithms for other sensitivity analysis metrics and we investigate system models with additional context information.

References

[1] N. C. Audsley, A. Burns, M. F. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Journal of Real-Time Systems*, 8(5):284–292, 1993.

[2] F. Cottet and J. Ph. Babau. An iterative method of task temporal parameter adjustment in hard real-time systems. In *Proc. of the 2nd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '96)*, Montreal, Canada, October 1996.

[3] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design space exploration and system optimization with SymTA/S -

Symbolic Timing Analysis for Systems. In *Proc. 25th International Real-Time Systems Symposium (RTSS'04)*, Lisbon, Portugal, December 2004.

[4] Arne Hamann, Rafik Henia, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. SymTA/S - Symbolic Timing Analysis for Systems. *http://www.symta.org/*.

[5] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Proceeding 40th Design Automation Conference*, Annaheim, USA, June 2003.

[6] M. Jersak, R. Henia, and R. Ernst. Context-aware performance analysis for efficient embedded system design. In *Proc. of Design, Automation and Test in Europe (DATE'04)*, Paris, France, March 2004.

[7] M. Jersak, K. Richter, and R. Ernst. Performance analysis for complex embedded applications. *International Journal of Embedded Systems, Special Issue on Codesign for SoC*, 2004.

[8] Daniel I. Katcher, Hiroshi Arakawa, and Jay K. Strosnider. Engineering and analysis of fixed priority schedulers. *Software Engineering*, 19(9):920–934, 1993.

[9] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings Real-Time Systems Symposiom*, pages 166–171, IEEE Computer Society Press, 1989.

[10] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[11] Live Devices, ETAS Group. RTA-OSEK in detail. *http://en.etasgroup.com/products/rta/*.

[12] A.K. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[13] J. C. Palencia and M. G. Harbour. Schedulablilty analysis for tasks with static and dynamic offsets. In *Proc. of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Spain, 1998.

[14] Traian Pop, Petru Eles, and Zebo Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Tenth International Symposium on Hardware/Software Codesign (CODES'02)*, Estes Park, Colorado, USA, May 2002.

[15] Sasikumar Punnekkat, Robert Davis, and Alan Burns. Sensitivity analysis of real-time task sets. *ASIAN*, pages 72–82, 1997.

[16] John Regehr. Scheduling tasks with mixed preemption relations for robustness to timing faults. In *Proc. of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Austin,Texas, December 2002.

[17] K. Richter. *Compositional Performance Analysis*. PhD thesis, Technical University of Braunschweig, 2004.

[18] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *Proc. of Design, Automation and Test in Europe Conference (DATE'02)*, Paris, France, March 2002.

[19] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4), April 2003.

[20] K. Richter, R. Racu, and R. Ernst. Scheduling analysis integration for heterogeneous multiprocessor SoC. In *Proceedings 24th International Real-Time Systems Symposium (RTSS'03)*, Cancun, Mexico, December 2003.

[21] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proceeding 39th Design Automation Conference*, New Orleans, USA, June 2002.

[22] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS 221, Department of Computer Science, University of York, UK, 1994.

[23] K. Tindell, A. Burns, and A. Wellings. An extendible approach for analysing fixed priority hard real-time systems. *Journal of Real-Time Systems*, 6(2):133–152, Mar 1994.

[24] Steve Vestal. Fixed-priority sensitivity analysis for linear compute time models. *IEEE Transactions on Software Engineering*, 20(4), april 1994.

[25] R. Yerraballi, R. Mukkamala, K. Maly, and H. A. Wahab. Issues in schedulability analysis of real-time systems. In *Proc. of 7th Euromicro Workshop on Real Time Systems (EUROMICRO-RTS'95)*, pages 87–92, June 1993.