Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 73 (2007) 207–224
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

Workload correlations in multi-processor hard real-time systems

Ernesto Wandeler *, Lothar Thiele

Computer Engineering and Networks Laboratory, Department of Information Technology and Electrical Engineering, ETH Zurich, 8092 Zurich, Switzerland

Received 1 July 2005; received in revised form 10 December 2005

Available online 24 May 2006

Abstract

Modern embedded systems that are integrated as multi-processor system on chips, are often characterized by the complex behaviors and dependencies between system components. Different events that trigger such systems may cause different execution demands, depending on their event type as well as on the task they are processed by, leading to complex workload correlations. For example in data processing systems, the size of an event's payload data will typically determine its execution demand on most or all system components, leading to highly correlated workloads. Performance analysis of such systems is often difficult, and conventional analysis methods have no means to capture the possible existence of workload correlations. This leads to overly pessimistic performance analysis results, and thus to expensive system designs with considerable performance reserves. We propose an abstract model to characterize and capture workload correlations present in a system architecture, and we show how the captured additional system information can be incorporated into an existing framework for modular performance analysis of embedded systems.

Keywords: Performance analysis; Real-time calculus; Workload correlations; Embedded systems; Multi-processor systems on chip

1. Introduction

Today's complex real-time embedded systems are often comprised of a combination of different hardware and software components that are triggered by a number of different incoming event streams, and that communicate via some communication network. Such systems are often integrated as a Multi-processor System on Chip (MpSoC) and many alternatives for partitioning, allocation and binding lead to a large design space.

During the system level design process of such a complex system, a designer is typically faced with the questions whether the timing properties of a certain implementation will meet the design requirements, what architectural element will act as a bottleneck, or what the on-chip memory requirements will be. One of the major challenges in the design process is therefore to analyze essential characteristics of a system architecture, such as end-to-end delays, or buffer requirements in an early design stage. This analysis is typically referred to as system-level performance analysis, and supports the choice of important design decisions before much time must be invested in detailed implementations.

* Corresponding author.
E-mail addresses: wandeler@tik.ee.ethz.ch (E. Wandeler), thiele@tik.ee.ethz.ch (L. Thiele).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2006.04.005

Obtaining tight results for the above-mentioned characteristics of a system design is not only difficult because of the heterogeneity and complexity of the complete system, but also because of the possibly complex behaviors and dependencies of the different components in the system. Different events may arrive on incoming event streams of a component, and they may create different resource demands, depending on their event type as well as on the task they are processed by. Due to this, there often exist complex correlations in such systems. We can thereby differentiate between correlations in the occurrence of different event types on incoming event streams, which we call event stream correlations (see [17]), and correlations of different resource demands that an event of a given type causes on different system components, which we call workload correlations. Highly correlated resource demands are for example typical in data processing systems, where the resource demand that an event creates on a component directly depends on the size of its payload data. In such a system, an event with a large payload will typically cause a large resource demand on most or all components it is processed on, and in turn, an event with a small payload will cause a small resource demand on most or all components.

The system-level analysis of such complex and heterogeneous systems with a high degree of internal correlations is currently mainly based on simulation, using for example SystemC, or trace-based simulation as in [10]. These techniques allow modeling and simulation of complex systems in any level of detail. However, in terms of completeness, simulation-based approaches do not allow to obtain worst-case results, because any concrete simulation-run can in general not guarantee to cover all corner cases. But most of all, simulation often suffers from long run-times and from a high set-up effort for each new architecture and mapping to be analyzed, and is therefore not very well suited for performance analysis in early design stages.

Other existing techniques use probabilistic models to describe real-time systems, see, e.g., [16]. Such models can capture statistical characteristics of complex components. The results obtained from an analysis with these probabilistic models will however again be of probabilistic nature. Hence, they are not applicable for the analysis of hard real-time systems.

In contrast to the above-mentioned methods, formal analytical methods allow to obtain the hard-bounded results that are needed for the analysis of hard real-time systems. A well-known technique to model real-time components are Timed Automata [1]. In [6], it was shown that timed automata can be used as task models for event-driven systems and that the schedulability problem in such a model can be transformed to a reachability problem for timed automata and is thus decidable. While we could theoretically model complex component behaviors in any level of detail using this technique, the limitation to synchronous communication in timed automata, but also the existence of different event types would require to explicitly model all buffers in a stream based multi-processor application with asynchronous communication. But a single buffer of size b that may hold events of e different types, already requires $e^{(b+1)} - 1$ states. Analysis of a multi-processor system with explicitly modeled buffers would therefore quickly lead to state-space explosion, turning the analysis effort for a complete system to be prohibitive.

In [14], an analytical framework for system-level performance analysis was proposed that is based on classical scheduling results from hard real-time system design. This framework uses a number of well-known abstractions to capture the timing behavior of event streams and provides additional interfaces between them. Recently, effort was put into the refinement of component interaction models in this framework [13]. These models however do not explicitly consider workload correlations yet. In [7], intra event and inter event stream contexts were presented that address the problem of correlations present in complex embedded systems. Intra event stream contexts consider the correlations of different event types on a single processor, while inter event stream contexts consider the correlations of different event streams on a single processor. None of the methods however consider the workload correlations on multiple consecutive processors presented in this work. A general disadvantage of the framework presented in [14], and hence of all methods based on it, is its confinement to a limited number of event stream timing behaviors. Due to this, the framework can only capture limited aspects of event streams that exhibit arbitrary timing behaviors.

On the other hand, powerful abstractions have been developed in the domain of communication networks, to model flow of data through a network. In particular the theoretical framework called Network Calculus [9] provides means to deterministically reason about timing properties of data flows in queuing networks. Real-Time Calculus [15] extends the basic concepts of Network Calculus to the domain of real-time embedded systems and in [2] a unifying approach to modular system level performance analysis with Real-Time Calculus has been proposed. It is based on a general event model, allows for hierarchical scheduling and arbitration, and can take computation and communication resources into account.

The framework presented in [2] has been successfully used in several case studies for hard real-time system analysis (see, e.g., [3]). However, since it is also not capable to capture the above mentioned correlations, that are often present in complex embedded systems, the results obtained from performance analysis with [2] are sometimes overly pessimistic. In [17], a new method was presented that extended the existing framework by allowing to characterize event stream correlations. And in this work, we present a new method that extends the same framework and that allows to characterize, capture and exploit workload correlations in the performance analysis of complex systems.

1.1. Contributions

- We provide an insight to the problem of performance analysis of distributed hard real-time systems with varying worst-case and best-case resource demands, and we point out the performance reserves that often exist in such systems due to workload correlations on different system components. These are typically not considered by existing performance analysis methods.
- We present a new abstract model that allows to characterize and capture workload correlations between different components in a system architecture, and we show how the captured information of workload correlations can be incorporated into the framework presented in [2]. Further, we present an analytic method to extract the workload correlation information from a typical system model.
- We show the applicability of the presented model and methods in a detailed case study of a Multi-processor System on Chip, where the analysis results obtained using the new methods are considerably improved compared to the results obtained using conventional methods.

2. Workload correlations—an application scenario

We first present an application scenario of a multi-processor system on chip (MpSoC) with workload correlations, that will serve as a case study throughout the paper, and that will help to visualize the effects and the importance of the presented methods. In Section 5, we provide the complete analysis of this application scenario, and discuss the obtained results.

An overview of the MpSoC in consideration is depicted in Fig. 1. The figure shows a two-processor system on chip, that must serve two event streams, both entering and leaving independently through the I/O-interfaces of the chip. Both event streams have hard-real time processing requirements.

Stream 1 is processed consecutively by *Task 1* and *Task 2*, that are running on *Processor I* and *Processor II*, respectively. In the long-term average, the events on this stream arrive periodically with a period of 4 ms. From time to time, bursts in the event arrival are possible on this stream, the full specification of *Stream 1* is provided in Example 1. The events arriving on *Stream 1* may be differentiated into two event types, with different execution demands. Events of type *A* have an execution demand of 20,000 cycles in *Task 1* and a demand of 15,000 cycles in *Task 2*. Events of type *B* on the other hand are less resource demanding and demand only 5000 cycles, both in *Task 1*

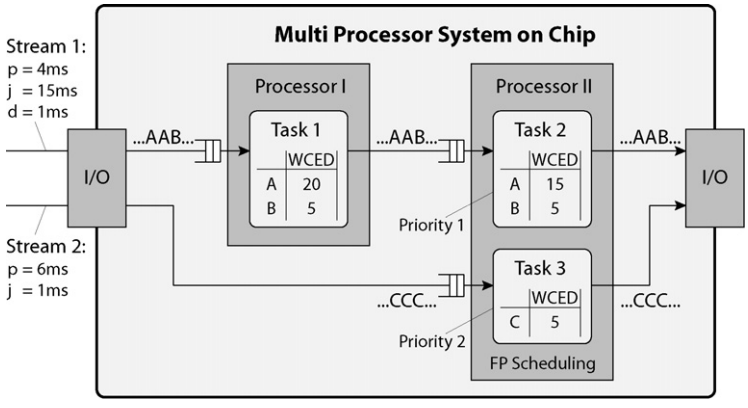

Fig. 1. A multi-processor system on chip with correlating execution demands. The execution demands are indicated in units of 1000 cycles.

as well as in *Task 2*. Further let us suppose that events of both types may arrive equally frequently and that we know nothing about possible correlations or arriving patterns of the two event types on the first event stream.

Stream 2 is processed by *Task 3* that is running on *Processor II*. Events on *Stream 2* arrive periodically, with a period of 6 ms and a maximal jitter of 1 ms. The events on this event stream are all of type *C* and have an execution demand of 5000 cycles in *Task 3*.

On *Processor II*, *Task 2* and *Task 3* are scheduled using a preemptive fixed priority scheduler, where, according to the *Rate Monotonic* scheduling scheme [11], *Task 2* (processing the stream with period $p = 4$ ms) has the higher priority than *Task 3* (processing the stream with period $p = 6$ ms). Since processing *Task 3* has a lower priority than processing *Task 2*, the events on *Stream 2* may experience a processing delay due to the interference of *Task 2*.

Let the speed of *Processor I* in the system be fixed to 6 MHz. Our design problem will then be to determine upper bounds to the maximum delay that events on *Stream 2* experience during processing on the MpSoC, as a function of the processor speed of *Processor II*. We thereby define the maximum delay as the maximum time that elapses between the arrival of an event on *Stream 2* and the finishing time of the corresponding *Task 3* that processes the event.

In this MpSoC, we nicely see the correlations of the workloads that *Stream 1* is creating on the two processors: an event of type *A*, that creates a high demand of 20,000 cycles on *Processor I* again creates a high demand of 15,000 cycles on *Processor II*. On the other hand, an event of type *B* that only creates a demand of 5000 cycles on *Processor I*, creates also a low demand of 5000 cycles on *Processor II*.

More details of this application scenario are provided in the example sections of this work. The complete analysis, and the results obtained for the design problem are presented and discussed in Section 5.

3. Modular performance analysis of hard real-time systems

Figure 2 shows an overview of an approach to Modular Performance Analysis (MPA). The central idea of MPA is to construct a performance model of a system, that captures the properties of the different system elements, such that they can be used for performance analysis. The approach described here is based on Real-Time Calculus [15], that itself is based on the theoretical framework called Network Calculus [9]. Network Calculus provides means to deterministically reason about timing properties of data flows in queuing networks. In the here described approach to MPA, performance models are built using three basic abstract elements: arrival curves model the environment (incoming and outgoing event streams), service curves model the capacity of resources and abstract components define the semantics of task execution in the system. The following sections give a short introduction to these basic elements and to the analysis methods.

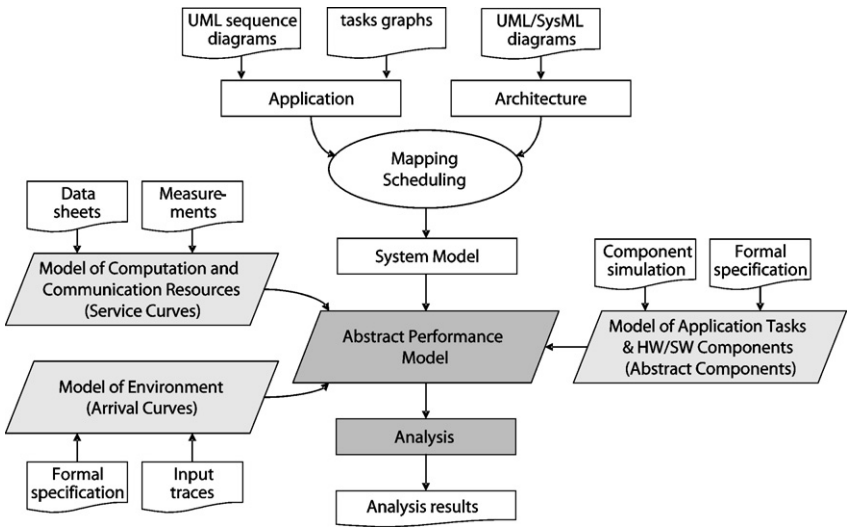

Fig. 2. Elements of Modular Performance Analysis and methods to obtain them.

3.1. Model of environment (Arrival Curves)

Every event on an *event stream*, has a time when it occurs. This timing information of an event stream is captured using the concept of upper and lower *arrival curves* [5]:

Definition 1 *(Arrival Curves).* Let $R[s,t)$ denote the number of events that arrive on an event stream in the time interval $[s,t)$. Then, R, $\bar{\alpha}^u$ and $\bar{\alpha}^l$ are related to each other by the following inequality:

$$\bar{\alpha}^l(t-s) \leqslant R[s,t) \leqslant \bar{\alpha}^u(t-s), \quad \forall s < t, \tag{1}$$

where $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$.

In this model, the timing information of standard event stream models like *periodic*, *periodic with jitter*, *periodic with bursts*, *sporadic* or of any event stream with a known analytical timing behavior can be represented by an appropriate choice of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ [2]. Moreover, it is also possible to determine the values of $\bar{\alpha}^u$ and $\bar{\alpha}^l$ corresponding to any given finite length event trace, obtained for example from observation or simulation.

Example 1. In literature, standard event arrival patterns are often specified by a parameter triple (p, j, d), where p denotes the period, j the jitter, and d the minimum inter-arrival distance of events in the modeled stream. Event streams that are specified using these parameters can directly be modeled by the following arrival curves:

$$\bar{\alpha}^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor, \tag{2}$$

$$\bar{\alpha}^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{d} \right\rceil \right\}. \tag{3}$$

In Fig. 3, the relation between these parameters and the corresponding arrival curves is graphically depicted, and Fig. 4 shows the arrival curves of the two input event streams *Stream 1* and *Stream 2* that trigger the MpSoC in Fig. 1.

3.2. Model of resources (Service Curves)

The capacity of a *resource* can be characterized using lower and upper *service curves* β^l and β^u [9]:

Definition 2 *(Service Curves).* Let $C[s,t)$ denote the number of processing or communication units available from a resource over the time interval $[s,t)$, then the inequality

$$\beta^l(t-s) \leqslant C[s,t) \leqslant \beta^u(t-s), \quad \forall s < t, \tag{4}$$

holds, where again $\beta^l(0) = \beta^u(0) = 0$.

The service curves of a resource can be determined using data sheets, using analytically derived properties or by using measuring.

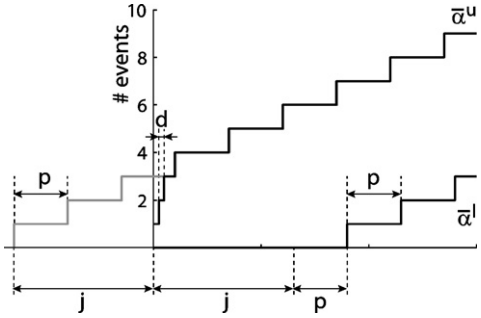

Fig. 3. Arrival curves from (p, j, d).

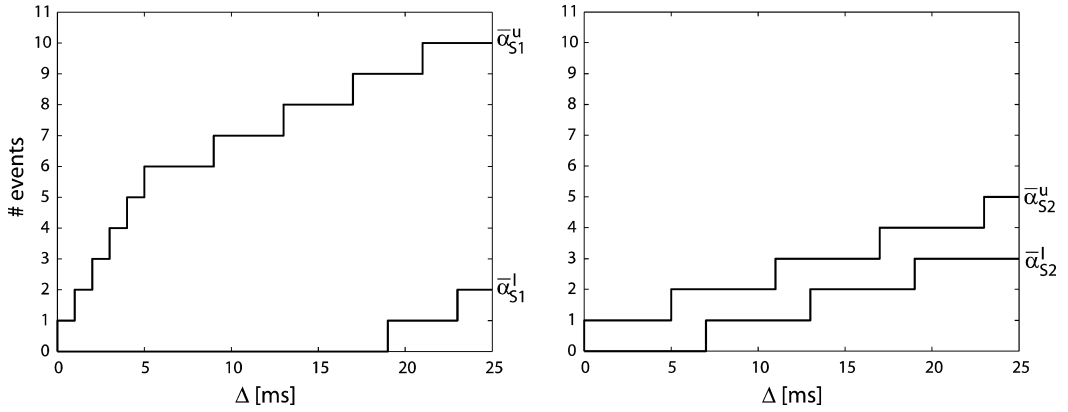

Fig. 4. The event stream model of *Stream 1* (left) and *Stream 2* (right) that trigger the MpSoC in Fig. 1.

Example 2. Both processors of the MpSoC in Fig. 1 are initially unloaded, i.e. fully available. We therefore model both as a pair of straight upper and lower service curves, with a slope corresponding to the processor speed. For example for *Processor I* with a processor speed of 6 MHz, $\beta^u_{\mathrm{PI}} = \beta^l_{\mathrm{PI}} = 6{,}000{,}000 \cdot \Delta$.

3.3. Abstract components and Real-Time Calculus

Abstract components are the basic building blocks to construct a performance model of a system. They model the application tasks in a concrete system and define the semantics of how application tasks are executed on architectural elements. At the same time, they also build the basis for performance analysis.

A very common execution semantics for application tasks in embedded systems is greedy preemptive computation, where a component is triggered by the events of an incoming event stream. A fully preemptable task is instantiated at every event arrival to process the incoming event, and active tasks are processed greedily in FIFO order, while being restricted by the availability of resources.

In the presented framework, concrete components are modeled by abstract components with a set of internal relations that transform incoming arrival curves α and service curves β into outgoing arrival curves α' and service curves β', according to the execution semantics of the concrete component. For a component with a greedy preemptive computation semantics, these relations were first derived in [2] using Real-Time Calculus:

$$\alpha^{l\prime}(\Delta) = \min\Big\{ \inf_{0 \leqslant \mu \leqslant \Delta} \Big\{ \sup_{\lambda \geqslant 0} \big\{ \alpha^l(\mu + \lambda) - \beta^u(\lambda) \big\} + \beta^l(\Delta - \mu) \Big\}, \beta^l(\Delta) \Big\}, \tag{5}$$

$$\alpha^{u\prime}(\Delta) = \min\Big\{ \sup_{\lambda \geqslant 0} \Big\{ \inf_{0 \leqslant \mu \leqslant \lambda + \Delta} \big\{ \alpha^u(\mu) + \beta^u(\lambda + \Delta - \mu) \big\} - \beta^l(\lambda) \Big\}, \beta^u(\Delta) \Big\}, \tag{6}$$

$$\beta^{l\prime}(\Delta) = \sup_{0 \leqslant \lambda \leqslant \Delta} \big\{ \beta^l(\lambda) - \alpha^u(\lambda) \big\}, \tag{7}$$

$$\beta^{u\prime}(\Delta) = \max\Big\{ \inf_{\lambda \geqslant \Delta} \big\{ \beta^u(\lambda) - \alpha^l(\lambda) \big\}, 0 \Big\}. \tag{8}$$

In Real-Time Calculus, the maximum delay experienced by an event stream with an upper arrival curve α^u that is processed at an abstract component with greedy preemptive computation semantics and with a lower service curve β^l is bounded by the maximum horizontal distance between α^u and β^l [9]:

$$d_{\max} \leqslant \sup_{t \geqslant 0} \big\{ \inf \big\{ \tau \geqslant 0 \colon \alpha^u(t) \leqslant \beta^l(t + \tau) \big\} \big\}. \tag{9}$$

Analogously, the maximum buffer space $b_{\max}$ that is required to buffer this event stream is bounded by the maximum vertical distance between α^u and β^l [9]:

$$b_{\max} \leqslant \sup_{\lambda \geqslant 0} \big\{ \alpha^u(\lambda) - \beta^l(\lambda) \big\}. \tag{10}$$

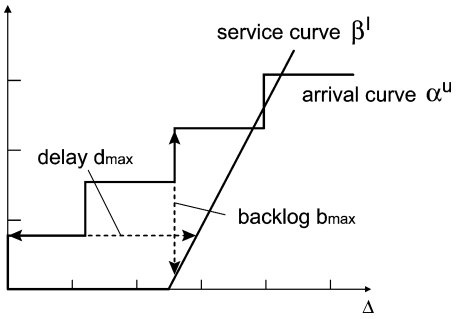

Fig. 5. Delay and backlog obtained from arrival and service curves.

In Fig. 5, the relations between α, β, $d_{\max}$ and $b_{\max}$ are depicted graphically.

We may connect abstract components into a network according to the system model that is obtained after mapping and scheduling the various applications on a system architecture. Event flows that exit abstract components (tasks) can be connected to an event flow input of another component and similarly, resource capacity that is not consumed by an abstract component can be connected to the resource input of a lower priority abstract component. Together with the models of all system resources (service curves) and incoming event streams (arrival curves), we obtain the performance model of a complete system.

3.4. The importance of workload transformations

As defined above, the arrival curves $\bar{\alpha}^l$ and $\bar{\alpha}^u$, used to model event streams, capture the number of events that arrive on an event stream in a given time interval. On the other hand, the service curves β^l and β^u denote the number of resource units (e.g. cycles) that are available on an architectural element in a given time interval. It becomes apparent, that in order to apply formulas (5)–(9), both the arrival curves and the service curves need to be expressed in the same base unit. For this, we can either transform the event based arrival curves $\bar{\alpha}^l$ and $\bar{\alpha}^u$ into resource-based arrival curves, denoted by α^l and α^u. Or alternatively, we can transform the resource-based service curves β^l and β^u into event-based service curves, denoted by $\bar{\beta}^l$ and $\bar{\beta}^u$. It is important to note, that these transformations must be conservative, such that the analysis still provides hard bounded results.

In the most simple case, where only one type of events is processed in a task, and where each event always creates exactly the same, constant resource demand (i.e. the worst-case demand equals the best-case demand), resource-based quantities can be transformed lossless into event-based quantities and vice versa. Event-based quantities simply need to be multiplied with the resource demand of a single event, and resource-based quantities need to be divided by the same number. In the latter case, for resource-based arrival curves that are transformed into event-based arrival curves, floor and ceil operators can be applied additionally:

$$\bar{\alpha}^u(\Delta) = \lceil \alpha^u / d \rceil, \tag{11}$$

$$\bar{\alpha}^l(\Delta) = \lfloor \alpha^l / d \rfloor. \tag{12}$$

In a more general case, this transformation is not as simple anymore, and we need more powerful methods to capture the relation between the number of events and the created resource demand in a task. Workload Curves, that were first introduced in [12] provide the required expressiveness:

Definition 3 *(Workload Curves).* Let $W(u)$ denote the total resource demand created in a task by u consecutive events of an incoming event stream. For every event sequence s, the lower workload curve γ^l and the upper workload curve γ^u satisfy the relation:

$$\gamma^l(v-u) \leqslant W[u,v) \leqslant \gamma^u(v-u), \quad \forall u < v. \tag{13}$$

Therefore, any sequence of e consecutive events, will create a total resource demand of at least $\gamma^l(e)$ and at most $\gamma^u(e)$ on an architectural element.

Example 3. We know nothing about possible correlations or arriving patterns of the different event types on the event streams triggering the MpSoC in Fig. 1. Therefore, the resource demand that e consecutive events could create, is only bounded by the worst (best) case, where all e events create the maximum (minimum) possible resource demand. This leads to the following workload curves for the different tasks in Fig. 1: $\gamma^l_{T1}(e) = e \cdot 5000$ cycles, $\gamma^u_{T1}(e) = e \cdot 20{,}000$ cycles, $\gamma^l_{T2}(e) = e \cdot 5000$ cycles, $\gamma^u_{T2}(e) = e \cdot 15{,}000$, and $\gamma^l_{T3}(e) = \gamma^u_{T3}(e) = e \cdot 5000$ cycles.

Using workload curves and their pseudo-inverse, arrival and service curves can be transformed from event-based to resource-based quantities and vice versa as follows:

$$\alpha^l(\Delta) = \gamma^l\big(\bar{\alpha}^l(\Delta)\big), \qquad \beta^l(\Delta) = \gamma^l\big(\bar{\beta}^l(\Delta)\big), \tag{14}$$

$$\alpha^u(\Delta) = \gamma^u\big(\bar{\alpha}^u(\Delta)\big), \qquad \beta^u(\Delta) = \gamma^u\big(\bar{\beta}^u(\Delta)\big), \tag{15}$$

$$\bar{\alpha}^l(\Delta) = \gamma^{u\,-1}\big(\alpha^l(\Delta)\big), \qquad \bar{\beta}^l(\Delta) = \gamma^{u\,-1}\big(\beta^l(\Delta)\big), \tag{16}$$

$$\bar{\alpha}^u(\Delta) = \gamma^{l\,-1}\big(\alpha^u(\Delta)\big), \qquad \bar{\beta}^u(\Delta) = \gamma^{l\,-1}\big(\beta^u(\Delta)\big). \tag{17}$$

3.5. Abstract components and workload transformations

We mentioned before that all workload transformations must be conservative in order to guarantee hard bounded analysis results. Mathematically expressed, the following relations (and similar relations for all other arrival and service curves) must hold with all workload curves:

$$\bar{\alpha}^u(\Delta) \leqslant \gamma^{l\,-1}\big(\gamma^u\big(\bar{\alpha}^u(\Delta)\big)\big), \tag{18}$$

$$\bar{\alpha}^l(\Delta) \geqslant \gamma^{u\,-1}\big(\gamma^l\big(\bar{\alpha}^l(\Delta)\big)\big). \tag{19}$$

And for general workload curves, where the upper curve γ^u does not equal the lower curve γ^l (i.e. the worst-case execution demand of all possible events does not equal the best-case execution demand), the right-hand side of (18) and (19) will normally be substantially larger than the left-hand side. This means, that while the application of one workload transformation (i.e. transforming from event-based to resource-based quantities, or the other way around) in general leads to realistic hard bounds (depending of course on the accuracy of the workload curves), a two-way workload transformation (from event based to resource-based quantities and back again to event-based quantities, or the other way around) often leads to overly pessimistic, yet still valid bounds.

As a consequence, we must take care to use as few workload transformations as possible during performance analysis, and in particular we should omit two-way workload transformations whenever possible. Figure 6 shows a schema how to use a minimal number of workload transformations in the analysis of an abstract component. The bright shaded data flows are resource based quantities, while the dark shaded data flows are event-based quantities. In

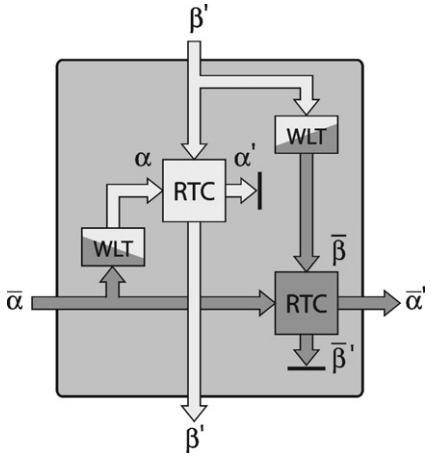

Fig. 6. Resource-based (bright) and event-based (dark) data flows in the analysis of an abstract component.

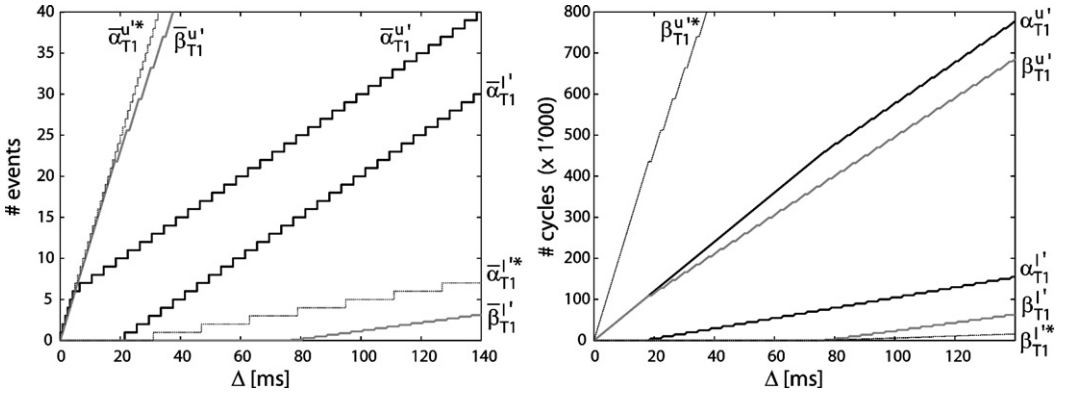

Fig. 7. The event-based (left) and resource-based (right) output curves of *Task 1* in the MpSoC in Fig. 1.

the two blocks labeled *WLT*, the workload transformations are applied according to (14)–(17), and in the two blocks labeled *RTC*, (5)–(8) are used to compute outgoing arrival and service curves in the respective base unit.

In the analysis schema shown in Fig. 6, (14)–(17) are applied to the resource-based curves α and β (in the bright block labeled *RTC*) as well as to their event-based counterparts $\bar{\alpha}$ and $\bar{\beta}$ (in the dark block labeled *RTC*). Both sets of curves, α and β as well as $\bar{\alpha}$ and $\bar{\beta}$, model the same properties of a system, but both use a different method of abstraction (resource units vs. events), and both may therefore capture different aspects of the same system properties (see Example 4). However, in the analysis schema shown in Fig. 6, the outgoing resource-based arrival curve α' as well as the outgoing event-based service curve $\bar{\beta}'$ are discarded after the analysis, together with the different aspects of system properties modeled by them. It would of course be possible to obtain an event-based outgoing arrival curve $\bar{\alpha}'$ from a workload transformation of the outgoing arrival curve α' (and similar we could obtain a β' from $\bar{\beta}'$). But due to the two-way transformation step that would be introduced by this, this curve will normally be overly pessimistic, as we will see in the following example.

Example 4. Figure 7 shows the event-based (left graph) and resource-based (right graph) outgoing arrival and service curves of Task 1 in the MpSoC in Fig. 1. The event-based arrival curves $\bar{\alpha}_{\mathrm{T1}} = \bar{\alpha}_{\mathrm{S1}}$ (see Example 1) and the resource-based service curves $\beta_{\mathrm{T1}} = \beta_{\mathrm{PI}}$ (see Example 2) served as initial input for the computation. With this input, all event-based and resource-based outgoing arrival and service curves were computed according to the analysis schema described above and depicted in Fig. 6. Additionally, the back-transformations of α'_{T1} and $\bar{\beta}'_{\mathrm{T1}}$ were computed by applying (14)–(17): $\beta'^{*}_{\mathrm{T1}} = \gamma_{\mathrm{T1}}(\bar{\beta}'_{\mathrm{T1}})$ and $\bar{\alpha}'^{*}_{\mathrm{T1}} = \gamma^{-1}_{\mathrm{T1}}(\alpha'_{\mathrm{T1}})$.

Let us first discuss the event-based upper outgoing arrival curve $\bar{\alpha}^{u'}_{\mathrm{T1}}$. Intuitively, we can say that the maximum possible event stream output (restricted by $\bar{\alpha}^{u'}$) will occur when the maximum possible event stream input (restricted by $\bar{\alpha}^{u}$) occurs together with the maximum possible resource availability (restricted by β^{u}). According to (6), the minimum possible resource availability (restricted by β^{l}) also plays a role (it restricts the maximum initial buffer fill level), but we will not consider it for the discussion here. From β^{u}_{T1} we know that in the best case there are 6000 cycles available per millisecond to process incoming events on *Processor I*. And since the less resource demanding event *B* on *Stream 1* only requires 5000 cycles to be processed by *Task 1*, we know that we could in the best case process 1.2 events per millisecond. This is expressed in $\bar{\beta}^{u}_{\mathrm{T1}}$. Because of this high event-based processing capacity (based on the case where all arriving events are of the less resource demanding type *B*), the maximum possible event stream output ($\bar{\alpha}^{u'}_{\mathrm{T1}}$) is only restricted by the availability of triggering input events ($\bar{\alpha}^{u}_{\mathrm{T1}}$), as we see in Fig. 7.

Let us now concentrate on the resource-based upper outgoing arrival curve $\alpha^{u'}_{\mathrm{T1}}$. The worst possible resource demand is created by the input event stream when all arriving events are of the more resource demanding type *A*. In this case, a total execution demand of $6 \cdot 20{,}000$ cycles $= 120{,}000$ cycles is created during a burst of only little more than 5 ms. But in the same time, a maximum of only $6 \cdot 6000$ cycles $= 36{,}000$ cycles are available for processing, and therefore most of the arriving events will be buffered in the input buffer to *Task 1*. In this phase, the output event stream rate is restricted by the availability of resources. However, after a burst, the events on the input event stream will again arrive periodically, creating a maximum resource demand of 20,000 cycles every 4 ms. And since in the

same period a maximum of 24,000 cycles are available for processing, the input buffer of *Task 1* will eventually be emptied again after a burst, leading into a phase where the maximum output event stream rate is restricted only by the availability of triggering input events. From the change of slopes of $\alpha_{T1}^{u'}$ (and $\beta_{T1}^{l'}$) at $\Delta \approx 77$ ms in Fig. 7, we know that the first phase, where the output event stream is restricted by the availability of resources, is upper-bounded by 77 ms.

When we are using (17) to transform $\alpha_{T1}^{u'}$ back to $\bar{\alpha}_{T1}'^{*}$, we must assume that with every investment of 5000 cycles we could in the best case generate an event of type B on the output event stream. But by taking this assumption, we neglect the fact that for the computation of $\alpha_{T1}^{u'}$, we assumed that all events are of type A. This decoupling of worst-case and best-case events leads to the very pessimistic upper bound $\bar{\alpha}_{T1}'^{*}$. Similar thoughts can explain the pessimism of all other back-transformed curves in Fig. 7.

From the results shown and discussed in Example 4, we clearly see that the resource-based outgoing arrival curve α' of an abstract component may carry valuable information (as for example the presence of a phase where the maximum available resources restrict the output event stream rate in Example 4) that may not be present in the event based outgoing arrival curve $\bar{\alpha}'$, or in the back-transformed curve $\bar{\alpha}'^{*}$.

But we also see that back-transforming the resource-based outgoing arrival curve α' into an event-based curve $\bar{\alpha}'^{*}$ is overly pessimistic, such that $\bar{\alpha}'^{*}$ does not contain much or any information of interest anymore. In the existing framework, α' is therefore typically discarded, as depicted in Fig. 6.

In the following section, we present a new method, that allows to consider the information contained in α' in the analysis of succeeding performance components.

4. Workload correlations and performance analysis of multi-processor systems

As we have seen in Section 3.3, we may interconnect abstract components to a network, to build the performance model of a complete multi-processor system. In the existing framework, we may however only use event-based arrival curves $\bar{\alpha}$ and resource-based service curves β to interconnect abstract components. The reason for this restriction is that outgoing resource-based arrival curves of one component are not directly related to ingoing resource-based arrival curves of another component, since the same event causes in general a different resource demand in two components, when it is processed by two different tasks. Similar thoughts can be made for event-based service curves.

As a direct consequence of this restriction, existing methods have no means to use the information captured by the outgoing resource based arrival curve α' of one component in the analysis of a succeeding component. To overcome this problem, we will now present a new method that will allows to directly transform the outgoing resource-based arrival curve α' of one abstract component into an ingoing resource-based arrival curve of another abstract component, by exploiting the knowledge of workload correlations existing between these components. No two-way transformation step will be needed for this, and therefore much of the information captured in α' will be preserved.

Figure 8 shows the performance model of the MpSoC of Fig. 1, where the outgoing resource-based arrival curve of *Task 1* is directly connected to the ingoing resource-based arrival curve of *Task 2*, using the new workload correlation based interconnection method.

The central part of this interconnection path is the box labeled *WCC*, where the resource-based arrival curves are transformed, by the use of what we will call Workload Correlation Curves (WCC). Workload Correlation Curves allow to analytically characterize and capture existing workload correlations between two components:

Definition 4 *(Workload Correlation Curves).* Suppose r_{T1} resource units are invested in a task T_1 to process incoming events and generate outgoing events. Further, let $W_{T1\to T2}(r_{T1})$ denote the total execution demand that is created in a task T_2 by the arrival of the events that were generated on the output of the preceding task T_1 by the investment of r_{T1} resource units. Then, for any number of invested resources r_{T1} in T_1, the lower workload correlation curve $\delta_{T1\to T2}^{l}(r)$ and the upper workload correlation curve $\delta_{T1\to T2}^{u}(r)$ satisfy the relation:

$$\delta_{T1\to T2}^{l}(v-u) \leqslant W_{T1\to T2}[u,v) \leqslant \delta_{T1\to T2}^{u}(v-u), \quad \forall u < v. \tag{20}$$

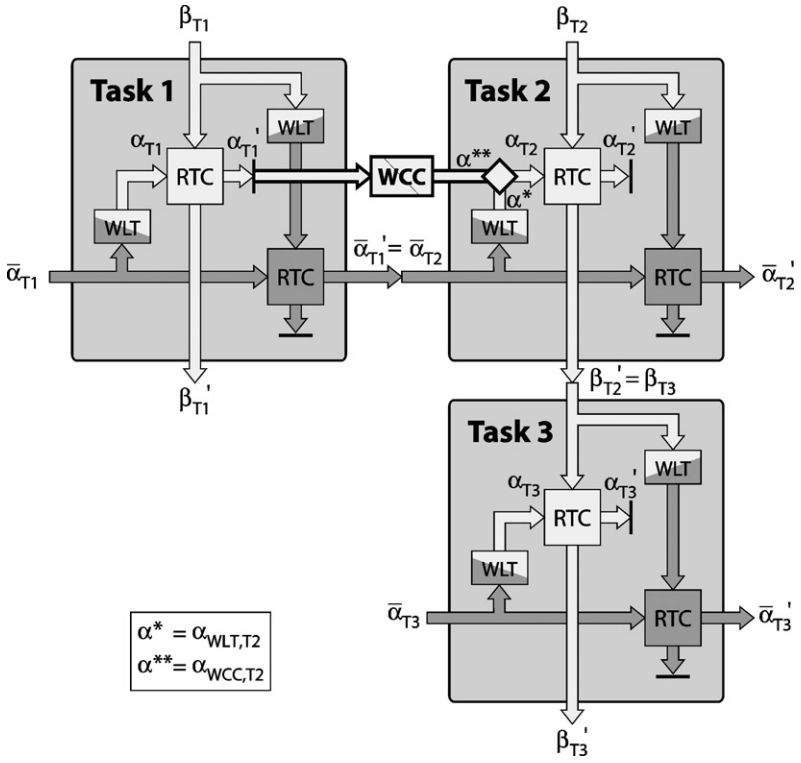

Fig. 8. The performance model of the MpSoC of Fig. 1, with the new component interconnection method highlighted.

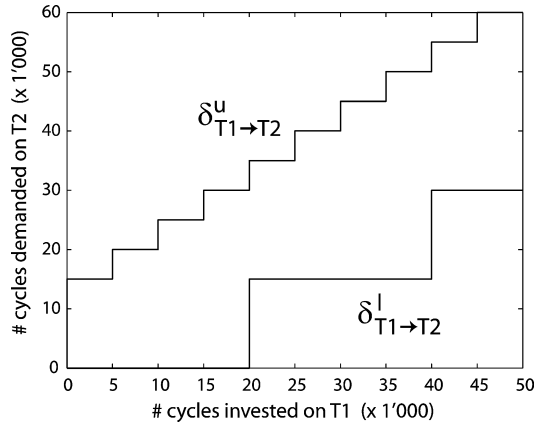

Fig. 9. The workload correlation curves between *Task 1* and *Task 2* in the MpSoC in Fig. 1.

Therefore, whenever r_{T1} resources are invested in a task T_1, the outgoing events that are generated by this resource investment will create a total resource demand of at least $\delta^l_{T1\to T2}(r_{T1})$ and at most $\delta^u_{T1\to T2}(r_{T1})$ resource units in a succeeding task T_2.

Example 5. Figure 9 shows the workload correlation curves between *Task 1* and *Task 2* of the MpSoC in Fig. 1. With the availability of a single cycle (the 20,000th), *Task 1* can generate an event of type *A* on its output event stream, leading to an execution demand of 15,000 cycles in *Task 2*. But after this initial event of type *A*, the constant processing of events of type *B* (assuming that the input buffer of *Task 1* never underflows) leads to the maximum workload on *Task 2*. This is because with events of type *B*, every cycle invested in *Task 1* generates in average a demand of 1 *cycle* in *Task 2*, while with events of type *A*, every invested cycle in *Task 1* only generates an average

demand of $15{,}000/20{,}000 = 0.75$ cycles in *Task 2*. On the other hand, the constant processing of events of type *A* leads to the minimum workload on *Task 2*. Section 6 describes in detail how to compute the curves shown in Fig. 9.

Using workload correlation curves, outgoing resource-based arrival curves of a task T_1 can be transformed directly into ingoing resource-based arrival curves of a task T_2:

$$\alpha^l_{\mathrm{WCC,T2}}(\Delta) = \delta^l_{\mathrm{T1}\to\mathrm{T2}}\big(\alpha^{l'}_{\mathrm{T1}}(\Delta)\big), \tag{21}$$

$$\alpha^u_{\mathrm{WCC,T2}}(\Delta) = \delta^u_{\mathrm{T1}\to\mathrm{T2}}\big(\alpha^{u'}_{\mathrm{T1}}(\Delta)\big). \tag{22}$$

In parallel, as we see in Fig. 8, we can still obtain the ingoing resource-based arrival curves $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WLT,T2}}$ from a workload transformation of the ingoing event-based arrival curves, by applying (14) and (15) to $\bar{\alpha}^l_{\mathrm{T2}}$ and $\bar{\alpha}^u_{\mathrm{T2}}$. And since all so obtained ingoing resource-based arrival curves, $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WLT,T2}}$ as well as $\alpha^l_{\mathrm{WCC,T2}}$ and $\alpha^u_{\mathrm{WCC,T2}}$, are hard bounds, we can combine the curves by taking the pairwise maximum and minimum, respectively:

$$\alpha^l_{\mathrm{T2}}(\Delta) = \max\big(\alpha^l_{\mathrm{WLT,T2}}(\Delta), \alpha^l_{\mathrm{WCC,T2}}(\Delta)\big), \tag{23}$$

$$\alpha^u_{\mathrm{T2}}(\Delta) = \min\big(\alpha^u_{\mathrm{WLT,T2}}(\Delta), \alpha^u_{\mathrm{WCC,T2}}(\Delta)\big). \tag{24}$$

5. Case study

In this section we analyze the MpSoC presented in Section 2. We will in particular compute upper bounds to the maximum delay that events of *Stream 2* experience during processing on the MpSoC, as a function of the processor frequency f_{PII} of *Processor II*. Note that in the system analysis presented in this section, we neglect the delay that events experience by the I/O-interfaces and by the on-chip communication network.

For the system performance analysis, we use the performance model of the MpSoC, shown in Fig. 8. To model the processing capacity of the initially unloaded *Processor II*, we use $\beta_{\mathrm{T2}} \equiv \beta_{\mathrm{PII}} = f_{\mathrm{PII}} \cdot \Delta$. The event-based arrival curves $\bar{\alpha}_{\mathrm{T1}} \equiv \bar{\alpha}_{\mathrm{S1}}$ and $\bar{\alpha}_{\mathrm{T3}} \equiv \bar{\alpha}_{\mathrm{S2}}$ (see Example 1) and the resource-based service curves $\beta_{\mathrm{T1}} \equiv \beta_{\mathrm{PI}}$ (see Example 2) serve as further system inputs.

5.1. Analytic bound on the maximum delay

According to (9), we need α^u_{T3} and β^l_{T3} to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II* of the MpSoC.

We can immediately compute α^u_{T3} by applying (15) to $\bar{\alpha}^u_{\mathrm{T3}}$.

To obtain β^l_{T3}, we need in a first step to compute the outgoing arrival curves $\alpha^{u'}_{\mathrm{T1}}$ and $\bar{\alpha}^{u'}_{\mathrm{T1}}$ of *Task 1*. We compute these curves according to the analysis schema described in Section 3.5, using (5)–(8) and (14)–(17). The resulting outgoing curves are shown in Fig. 7 and are discussed in Example 4.

According to the performance model shown in Fig. 8, the event-based upper outgoing arrival curve of *Task 1* can directly be interconnected to the event-based upper ingoing arrival curve of *Task 2*: $\bar{\alpha}^u_{\mathrm{T2}} = \bar{\alpha}^{u'}_{\mathrm{T1}}$.

To compute the resource-based upper ingoing arrival curve of *Task 2*, we first conventionally apply (15) to $\bar{\alpha}^u_{\mathrm{T2}}$, leading to $\alpha^u_{\mathrm{WLT,T2}}$. Applying (22) to $\alpha^{u'}_{\mathrm{T1}}$ will in turn lead to $\alpha^u_{\mathrm{WCC,T2}}$, and computing the minimum of $\alpha^u_{\mathrm{WLT,T2}}$ and $\alpha^u_{\mathrm{WCC,T2}}$ will according to (24) finally lead to α^u_{T2}. Similarly, the resource-based lower ingoing arrival curve of *Task 2* can according to (23) be computed as the maximum of $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^l_{\mathrm{WCC,T2}}$, where $\alpha^l_{\mathrm{WLT,T2}}$ and $\alpha^l_{\mathrm{WCC,T2}}$ are obtained by applying (14) to $\bar{\alpha}^l_{\mathrm{T2}}$ and (21) to $\alpha^{l'}_{\mathrm{T1}}$, respectively. All these variants of resource-based ingoing arrival curves of *Task 2* are shown in Fig. 10.

Then, to compute the resource-based lower outgoing service curve of *Task 2*, we apply (7) to α^u_{T2} and β^l_{T2}, leading to $\beta^{l'}_{\mathrm{T2}}$, that can be interconnected to β^l_{T3}. This allows finally to use (9) to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*:

$$d_{\mathrm{T3}} \leqslant \sup_{t \geqslant 0}\big\{\inf\big\{\tau \geqslant 0\colon \alpha^u_{\mathrm{T3}}(t) \leqslant \beta^l_{\mathrm{T3}}(t+\tau)\big\}\big\}.$$

To see the impact of the new presented methods to the obtained analysis results, we do the same performance analysis using the conventional analysis methods. There, we apply (7) directly to $\alpha^u_{\mathrm{WLT,T2}}$ and β^l_{T2}, leading to

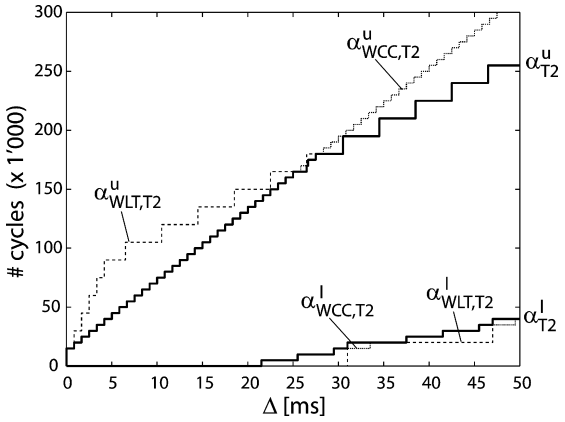

Fig. 10. All variants of resource-based input arrival curves of *Task 2* in the MpSoC in Fig. 1.

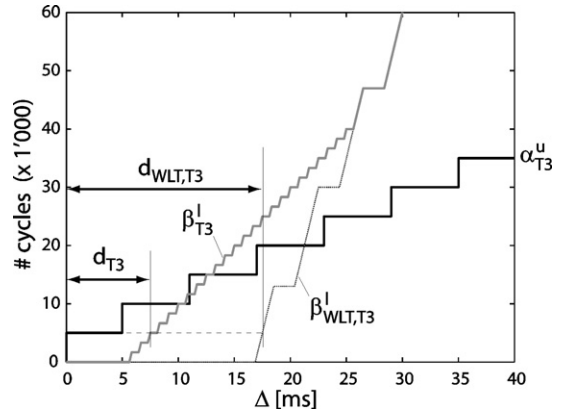

Fig. 11. The resource-based upper ingoing arrival curve together with both, the conventional and the new variant of the resource-based lower ingoing service curves of *Task 3* in the MpSoC in Fig. 1. (For $f_{\mathrm{PII}} = 8$ MHz.)

$\beta^{l'}_{\mathrm{WLT,T2}} \equiv \beta^{l}_{\mathrm{WLT,T3}}$. We then use again (9) to compute an upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*:

$$d_{\mathrm{WLT,T3}} \leqslant \sup_{t \geqslant 0}\left\{\inf\left\{\tau \geqslant 0\colon \alpha^u_{\mathrm{T3}}(t) \leqslant \beta^l_{\mathrm{WLT,T3}}(t+\tau)\right\}\right\}.$$

In Fig. 11, the resource-based upper ingoing arrival curve α^u_{T3} is shown together with the resource-based lower ingoing arrival curve $\beta^l_{\mathrm{WLT,T3}}$ that was obtained using conventional analysis methods, and the resource-based lower ingoing arrival curve β^l_{T3} that was obtained using the new presented methods that allow to incorporate existing workload correlations into performance analysis.

5.2. Experimental results

We used the above described analysis to compute the upper bound to the maximum delay that an event will experience during processing by *Task 3* on *Processor II*, for processor frequencies f_{PII} ranging from 6 to 25 MHz, in steps of 1 MHz. The obtained results for the analysis using the conventional methods as well as for the analysis with the new presented methods are visualized in the graph in Fig. 12.

As a reference, Fig. 12 also depicts the maximum delays that were observed in a SystemC simulation, as well as the exact worst-case delays that an event will experience during processing by *Task 3*. The exact worst-case delays were computed manually by identifying and analyzing the worst-case traces for *Stream 1* and *Stream 2*.

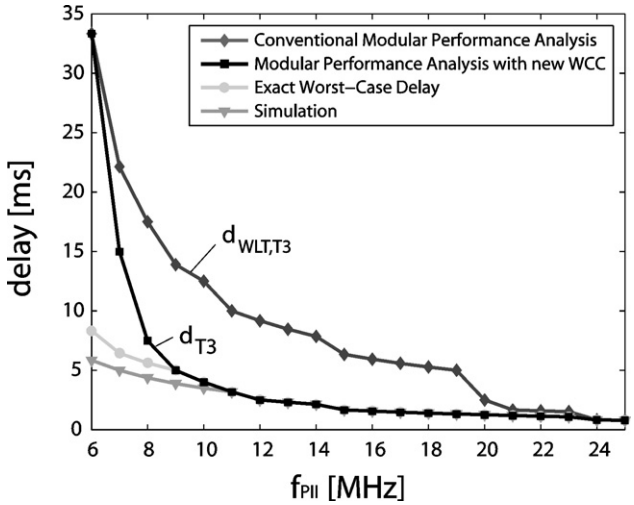

Fig. 12. Comparison of the analytic upper bound to the maximal delay experienced by an event during processing by *Task 3* in the MpSoC in Fig. 1, when analyzed using conventional methods (upper curve), and when analyzed using the new presented methods that consider workload correlations (lower curve).

From the results shown in Fig. 12, we see that the use of the new presented analysis methods lead to considerably tighter analytic bounds for the investigated design problem. The obtained results clearly point out the performance reserves that often exist in complex systems due to workload correlations, and that are usually not considered by existing performance analysis methods. For example, while we need to run *Processor II* at a processor speed of 14 MHz to guarantee an upper bound to the maximum delay of 8 ms with existing performance analysis methods, we can guarantee the same bound already at a processor speed of 8 MHz with the new presented analysis methods. This corresponds to a processor speed reduction of more than 42%. Or similarly, when we look at the delay guarantee for a processor speed of 8 MHz, we see that we can guarantee a maximum delay of 17.05 ms with existing performance analysis methods, while we can guarantee a maximum delay of only 7.5 ms with the new presented analysis methods. This corresponds to a delay guarantee improvement of 56%.

Looking at the results in Fig. 12, one might wonder why the analysis with WCCs does not lead to improved results for low processor speeds (e.g. 6 MHz). When we look at Fig. 10, we see that without the use of WCCs, $\alpha^u_{\mathrm{WLT,T2}}$ would limit the input to *Task 2*, consisting of a steep burst at the beginning and a less steep long-term load afterwards. With WCCs however, $\alpha^u_{\mathrm{WCC,T2}}$ reduces the burst load on *Task 2* within short time intervals. In the long term however ($\Delta \geqslant 30$ ms) the load on *Task 2* is still limited by the long-term slope of $\alpha^u_{\mathrm{WLT,T2}}$. If we now reduce the speed of *Processor II*, it eventually gets so slow that the available cycles per time interval are less than the arriving resource demand of the reduced burst that is specified by $\alpha^u_{\mathrm{WCC,T2}}$, i.e. the processor gets too slow to even process the reduced burst without buffering. In this case, remaining resources to process *Task 3* only exist in time intervals that are longer than the interval length when $\alpha^u_{\mathrm{WCC,T2}}$ and $\alpha^u_{\mathrm{WLT,T2}}$ cross for the last time in Fig. 10, i.e. for $\Delta \geqslant 30$ ms. Then however the remaining resources only depend on the long term slope of $\alpha^u_{\mathrm{WLT,T2}}$, and hence WCCs do not have any influence on the remaining resources anymore.

6. Computing workload correlation curves

In this section, we present a method to analytically compute the workload correlation curves between two components. The presented method is based on a slightly reduced version of a task model that was first introduced in [18]. While we use this reduced version for the sake of simplicity, the method could also easily be extended to work with the full version of the task model introduced in [18].

6.1. Task model

We model the execution demand behavior of a task by a state-transition graph that relates incoming events to best-case and worst-case resource demands as well as to outgoing events.

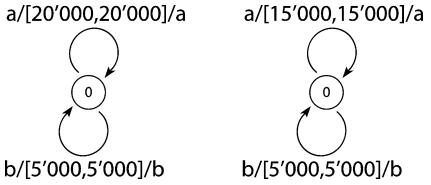

Fig. 13. The task models of *Task 1* (left) and *Task 2* (right) of the multi-processor system on chip in Fig. 1.

Definition 5 *(Task Automaton).* A task automaton F_T is a tuple $(S, S^0, \Sigma_I, \Sigma_O, D, T)$, where S is a set of states, $S^0 \subseteq S$ is a set of initial states, Σ_I is the set of accepted incoming event types and Σ_O is the set of generated outgoing event types. Further, D is a demand function $D : S \times \Sigma_I^* \times \Sigma_O^* \times S \to [\mathbb{Z}_{\geqslant 0}, \mathbb{Z}_{\geqslant 0}]$. This function determines the upper and lower bound of the resources needed by a task in order to process an incoming event, generate an outgoing event and change its internal state. Finally, $T \subseteq S \times \Sigma_I \times \mathbb{Z}_{\geqslant 0} \times \mathbb{Z}_{\geqslant 0} \times \Sigma_O \times S$ is a set of transitions.

For the following discussion, we use the notation $s \xrightarrow{\sigma_I/[d_l,d_u]/\sigma_O} s'$ do denote a transition with $\sigma_I \in \Sigma_I$, $d_l \in \mathbb{Z}_{\geqslant 0}$, $d_u \in \mathbb{Z}_{\geqslant 0}$ and $\sigma_O \in \Sigma_O$, that starts at s and ends at s'.

A task always starts in an initial state, and if $s \xrightarrow{\sigma_I/[d_l,d_u]/\sigma_O} s'$ and if the task is triggered by the incoming event σ_I, the task has a resource demand of at least d_l and at most d_u cycles to emit an event σ_O and change its state from s to s'.

We can get the needed information to model a task from a formal specification of the task or from data sheets. Moreover, it may also be possible to obtain this information from program analysis of the task code.

Example 6. Figure 13 shows the task models of the two simple tasks *Task 1* and *Task 2* running on *Processor I* and *Processor II* of the multi processor system on chip shown in Fig. 1.

Both tasks are modeled using only a single state. Upon arrival of an event *A*, *Task 1* has an execution demand of 20,000 cycles to process the event and to generate again an event *A* on its output. But the processing of an event *B* only requires 5000 cycles and leads again to an event *B* on the output of *Task 1*.

In *Task 2* on the other hand, the processing of an event *A* only requires 15,000 cycles, while the processing of an event *B* again requires 5000 cycles.

The task models shown in Fig. 1 are of the most simple type of task automata we could think of, where F_T consists of only a single state with a self-loop for every accepted event type. The presented task model however also allows to model much more complex task behaviors including for example the modeling of caching effects from deterministic caches, see [18]. And with the full version of the task model introduced in [18], it is even possible to model for example arbitrary up- and down-sampling of event rates in tasks. An example of such a more complex task automaton is provided in [18].

To reduce the size of the different intermediate automata that will be needed to compute the workload correlation curve of two task automata, we may divide the various resource demands denoted on the transitions by the greatest common divisor (GCD) of all resource demands. The resulting task automata are depicted in Fig. 14.

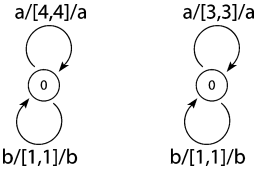

Fig. 14. The task models of *Task 1* (left) and *Task 2* (right) of the multi-processor system on chip in Fig. 1, after dividing the resource demands by the GCD.

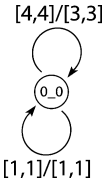

Fig. 15. The workload correlation automaton of the two task models shown in Fig. 13.

6.2. Workload correlation automaton

The first step towards computing the workload correlation curve of two task automata, is to build a so-called workload correlation automaton F_{WC}. This workload correlation automaton is built from the product automaton of the two task automata, where we have a transition between two states if and only if two transitions existed in the initial task automata, with the output event type of the first task automaton transition equaling the input event type of the second task automaton transition (we say that the first task automaton and the second task automaton synchronize on such transitions):

$$
\begin{aligned}
S_{\mathrm{WC}} &= S_{\mathrm{T1}} \otimes S_{\mathrm{T2}},\\
S^0_{\mathrm{WC}} &= S^0_{\mathrm{T1}} \otimes S^0_{\mathrm{T2}},\\
T_{\mathrm{WC}} &= \big\{\big((u,v),[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}],[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}],(u',v')\big) \mid\\
&\qquad \big(u,\sigma_{I,\mathrm{T1}},[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}],\sigma_{O,\mathrm{T1}},u'\big) \in T_{\mathrm{T1}}\\
&\qquad \wedge \big(v,\sigma_{I,\mathrm{T2}},[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}],\sigma_{O,\mathrm{T2}},v'\big) \in T_{\mathrm{T2}}\\
&\qquad \wedge \sigma_{O,\mathrm{T1}} = \sigma_{I,\mathrm{T2}}\big\}.
\end{aligned}
$$

After building all transitions, we can remove all states and state trajectories that are not reachable from an initial state anymore. This finally leads to the workload correlation automaton F_{WC} that is described by the tuple $(S_{\mathrm{WC}}, S^0_{\mathrm{WC}}, T_{\mathrm{WC}})$.

Example 7. Figure 15 shows the workload correlation automaton of the two task models shown in Fig. 13.

6.3. Workload correlation curve

From the workload correlation automaton F_{WC}, we can compute the upper workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}$ following a four-step procedure.

Step 1. On the transitions of the workload correlation automaton F_{WC}, we retain the lower resource demand of the first task T1 and the upper resource demand of the second task T2, and discard the other information:

$$
s \xrightarrow{[d_{l,\mathrm{T1}},d_{u,\mathrm{T1}}]/[d_{l,\mathrm{T2}},d_{u,\mathrm{T2}}]} s' \quad\Rightarrow\quad s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s'.
$$

Step 2. We then transform the so obtained automaton into an automaton where every transition corresponds to a created demand of one resource unit on the first task T1 and where the weights on the transition correspond to the created demand on the second task T2. For this, we first replace every transition $s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s'$ with $d_{l,\mathrm{T1}} > 0$ with a state trajectory according the following rule:

$$
s \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s' \quad\Rightarrow\quad \underbrace{s \xrightarrow{0} s_{(1)} \xrightarrow{0} \cdots \xrightarrow{0} s_{(d_{l,\mathrm{T1}}-1)} \xrightarrow{d_{u,\mathrm{T2}}} s'}_{d_{l,\mathrm{T1}}\ \text{transitions}}.
$$

After this, we replace all transitions $s' \xrightarrow{d_{l,\mathrm{T1}}/d_{u,\mathrm{T2}}} s''$ with $d_{l,\mathrm{T1}} = 0$ as follows:

$$
s \xrightarrow{w} s' \xrightarrow{0/d_{u,\mathrm{T2}}} s'' \quad\Rightarrow\quad s \xrightarrow{w+d_{u,\mathrm{T2}}} s''.
$$

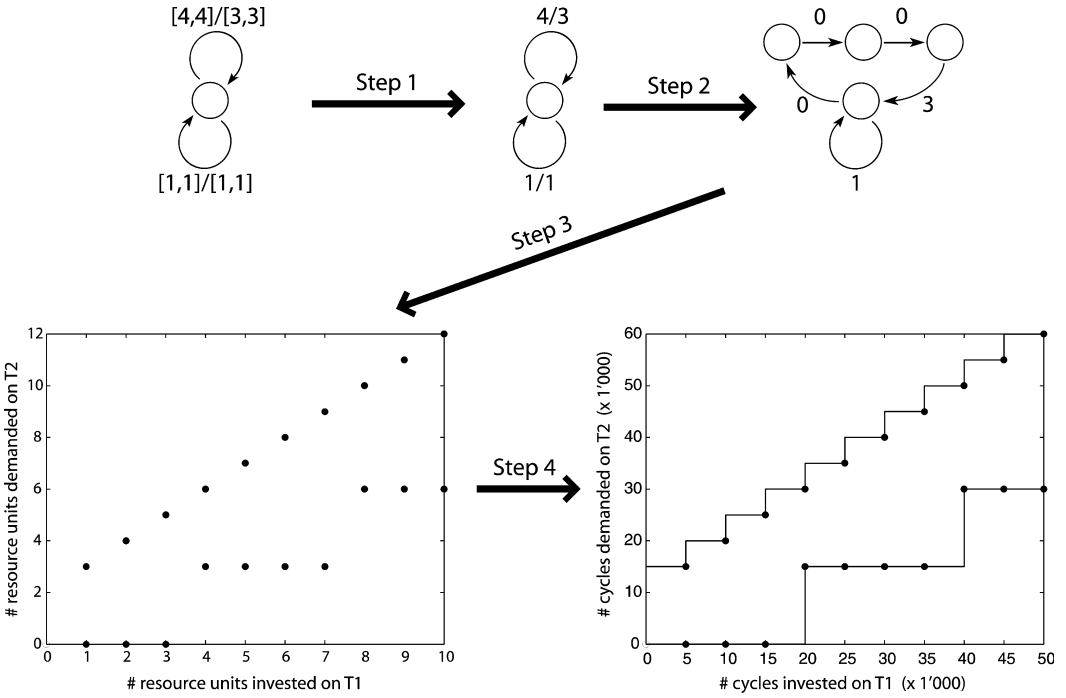

Fig. 16. The four-step procedure to compute the workload correlation curve for the workload correlation automaton of Fig. 15.

Step 3. In this automaton, we interpret the execution demand d as the weight of a transition. The weight $w^u(e)$ of the maximum-weight path with length e then equals the value of the upper workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(e)$. Techniques to efficiently find maximum-weight paths of any length in a weighted graph are described in [4] and [8] and an efficient algorithm with a computational complexity of $O(e|S||T|)$ was presented in [17].

Step 4. We now define the workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(x)$ for all values $x \in \mathbb{R}_{\geqslant 0}$ as:

$$\delta^u_{\mathrm{T1}\to\mathrm{T2}}(x) = \min_{e\geqslant x,\, e\in\mathbb{N}^+} \left\{\delta^u_{\mathrm{T1}\to\mathrm{T2}}(e)\right\}. \tag{25}$$

And finally, we need to multiply all values on both, the x- and the y-axis by the GCD of the initial workload demands, by which we initially divided the demands in the task automata.

To compute the lower workload correlation curve $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(e)$, we must follow the same four-step procedure as described above, but in *Step 1* we need to retain the upper resource demand $d_{u,\mathrm{T1}}$ of the first task T1 and the lower resource demand $d_{l,\mathrm{T2}}$ of the second task T2. In *Step 3*, the value of the lower transfer curve $\delta^l_{\mathrm{T1}\to\mathrm{T2}}(e)$ then equals the weight $w^l(e)$ of the minimum-weight path with length e in the automaton obtained in *Step 2*. In *Step 4*, the lower workload correlation curve $\delta^u_{\mathrm{T1}\to\mathrm{T2}}(x)$ is defined for all values $x \in \mathbb{R}_{\geqslant 0}$ as:

$$\delta^l_{\mathrm{T1}\to\mathrm{T2}}(x) = \max_{e\leqslant x,\, e\in\mathbb{N}^+} \left\{\delta^l_{\mathrm{T1}\to\mathrm{T2}}(e)\right\}. \tag{26}$$

And finally, all values need to be multiplied by the GCD of the initial workload demands.

Example 8. Figure 16 depicts the above-defined four-step procedure for the workload correlation automaton of Fig. 15.

7. Conclusions

We introduced a new abstract model that allows to characterize and capture existing workload correlations between different components in a system architecture, and we presented an analytic method to extract this abstract workload

correlation model from a typical system specification. We further presented the needed methods to incorporate the new model, and the system information captured by it, into an existing framework for modular performance analysis of embedded systems. We have shown the applicability of the presented model and methods in a detailed case study of a multi-processor system on chip, where a considerable improvement of certain analytic worst-case bounds was achieved compared to conventional analysis methods. The improved analysis results clearly point out the performance reserves that often exist in complex embedded systems due to workload correlations that are usually not considered by existing analysis methods.

Acknowledgments

The authors thank Alexander Maxiaguine for the interesting discussions, Simon Perathoner for providing the SystemC simulation results, and the anonymous reviewers for providing feedback and comments to the paper. This research has been funded by the Swiss National Science Foundation (SNF) under the Analytic Performance Estimation of Embedded Computer Systems project 200021-103580/1, and by ARTIST2.

References

[1] R. Alur, D.L. Dill, A theory of timed automata, Theoret. Comput. Sci. 126 (2) (1994) 183–235.
[2] S. Chakraborty, S. Künzli, L. Thiele, A general framework for analysing system properties in platform-based embedded system designs, in: Proc. 6th Design, Automation and Test in Europe, DATE, March 2003, pp. 190–195.
[3] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, P. Sagmeister, Performance evaluation of network processor architectures: Combining simulation with analytical estimation, Comput. Netw. 41 (5) (April 2003) 641–665.
[4] G. Cohen, D. Dubois, J.P. Quadrat, M. Voit, A linear-system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing, IEEE Trans. Automat. Control AC-30 (3) (1985).
[5] R. Cruz, A calculus for network delay, IEEE Trans. Inform. Theory 37 (1) (1991) 114–141.
[6] C. Ericsson, A. Wall, W. Yi, Timed automata as task models for event-driven systems, in: Proc. of RTCSA '99, IEEE Press, 1999.
[7] M. Jersak, R. Henia, R. Ernst, Context-aware performance analysis for efficient embedded systems design, in: Proc. 7th Design, Automation and Test in Europe, DATE, 2004.
[8] R.M. Karp, A characterization of the minimum cycle mean in a digraph, Discrete Math. 23 (1978) 309–311.
[9] J.Y. Le Boudec, P. Thiran, Network Calculus—A Theory of Deterministic Queuing Systems for the Internet, Lecture Notes in Comput. Sci., vol. 2050, Springer-Verlag, 2001.
[10] C. Lee, M. Potkonjak, W.H. Mangione-Smith, Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems, in: International Symposium on Microarchitecture, 1997, pp. 330–335.
[11] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in hard real-time environment, J. ACM 20 (1) (1973) 46–61.
[12] A. Maxiaguine, S. Künzli, L. Thiele, Workload characterization model for tasks with variable execution demand, in: Proc. 7th Design, Automation and Test in Europe, DATE, 2004.
[13] R. Racu, K. Richter, R. Ernst, Calculating task output event models to reduce distributed system cost, in: GI/ITG/GMM Workshop, February 2004.
[14] K. Richter, M. Jersak, R. Ernst, A formal approach to MpSoC performance verification, IEEE Comput. 36 (4) (April 2003) 60–67.
[15] L. Thiele, S. Chakraborty, M. Naedele, Real-time calculus for scheduling hard real-time systems, in: Proc. IEEE International Symposium on Circuits and Systems, ISCAS, vol. 4, 2000, pp. 101–104.
[16] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, J.W.-S. Liu, Probabilistic performance guarantee for real-time tasks with varying computation times, in: Proceedings of the IEEE Real-Time Technology and Applications Symposium, IEEE Comput. Soc., 1995, pp. 164–173.
[17] E. Wandeler, A. Maxiaguine, L. Thiele, Quantitative characterization of event streams in analysis of hard real-time applications, Real-Time Syst. 29 (2–3) (March 2005) 205–225.
[18] E. Wandeler, L. Thiele, Abstracting functionality for modular performance analysis of hard real-time systems, in: Proc. of Asia South Pacific Design Automation Conference 2005, ASP-DAC, January 2005, pp. 697–702.