Providing Accurate Event Models for the Analysis of Heterogeneous Multiprocessor Systems

Simon Schliecker, Jonas Rox, Matthias Ivers, Rolf Ernst
Institute of Computer and Communication Network Engineering
Technical University of Braunschweig
schliecker@ida.ing.tu-bs.de, rox@ida.ing.tu-bs.de, ivers@ida.ing.tu-bs.de,
ernst@ida.ing.tu-bs.de

ABSTRACT

This paper proposes a new method for deriving quantitative event information for compositional multiprocessor performance analysis. This procedure brakes down the complexity into the analysis of individual components (tasks mapped to resources) and the propagation of the timing information with the help of event models. This paper improves previous methods to derive event models in a multiprocessor system by providing tighter bounds and allowing arbitrarily shaped event models. The procedure is based on a a simple yet expressive resource model called the *multiple event busy time* which can be derived on the basis of classical scheduling theory — it can therefore be provided for a large domain of scheduling policies. Our experiments show that overestimation by previous methods can be reduced significantly.

Categories and Subject Descriptors C.4 [**Performance of Systems**]: Computer Systems Organization—*Performance attributes*
General Terms: Performance, Verification

1. INTRODUCTION

Formal performance verification is essential to safely verify the compliance of current multiprocessor systems with real-time constraints. With increasing system complexity and continued functional integration, system level performance issues are becoming more complex and the analysis must often be adapted to different systems. To counter this challenge, compositional approaches have been proposed that break down the complexity into the analysis of components (i.e. processors, busses, or memories) and a characterization of the events that are generated between theses components (so called event models).

A key problem of compositional analysis is to accurately derive the event models in the system, as this is the basis to determine the amount of interference any execution or communication experiences in the system. Commonly a task is assumed to be activated once by each event in an input event stream and to produce one event per execution at its output. Sharing resources among tasks inevitably increases the dynamism of events in the output event stream, which can be observed as a larger jitter. A model of the resource timing is required to derive the amount of dynamism imposed.

Previous research relies on different resource abstractions for this purpose. For example, the approach in [1] simply uses the task response time bounds. As these bounds can be determined with classical (single-)processor scheduling theory, this method allows to address a large set of scheduling and arbitration policies. However, relying on the response time bounds alone yields conservative results in cases where the worst case response time is only an effect of a transient overload situation. In [2], resources are modeled with more general service functions. This formalism has advantages when it comes to hierarchical scheduling techniques and delivers very accurate results. Since it is a load model, however, sequence dependent behavior, such as number of context switches, blocking effects including non-preemptive and collaborative scheduling, or mutual dependencies (as in round robin) cannot easily be covered.

In this paper, we combine the advantages. We propose a new abstraction of resource timing, namely the multiple event busy time model. This model is a direct extension of the classical busy window approaches such as [3] that symbolically derive worst case task sequences considering context switches, blocking, release offsets, or mutual dependencies. However, these methods only provide the busy window of a critical instant until the resource is idle for the first time. We therefore generalize this concept, by explictly considering the individual events in the busy window and also allowing for idle times between them in our proposed method for deriving event models at the task outputs.

In a last step, we show that this new method incorporates into a compositional analysis framework, allowing performance analysis of heterogeneous multiprocessor systems.

The paper is organized as follows: We present and evaluate the work related to our approach in Section 2. We formally introduce the multiple event busy time concept in Section 3.1. This concept allows us to derive the accurate event models in Sections 3.2 and 3.3. Section 4 shows that the convergence of the compositional multiprocessor performance analysis is still ensured when using the proposed event model calculation. Finally, the experiments in Section 5 show the quantitative advantage of the new method.

2. RELATED WORK

In general, a multiprocessor system consists of a set of *tasks* which can be a computation or communication with known minimum and maximum execution time. Tasks are activated by *events* and produce events when their execution is finished. Tasks are mapped to *resources* that arbitrate between the tasks mapped to it according to their *scheduling policy*, causing task activations to possibly interrupt each other. Tasks generally process the events of an event stream in-order. This typical assumption in scheduling

CODES+ISSS'08, October 19–24, 2008, Atlanta, Georgia, USA.

theory matches the design practice, prioritized events are modeled with separate event streams.

The performance analysis problem is addressed by various compositional approaches that separate the problem into local component analyses and the modeling of event traffic between them. In Network Calculus [4] and the Real-Time Calculus [2] based on it the local resource behavior is modeled as the execution time provided to the processing of events of a certain stream within a time window of given size Δt. Such a resource service curve is depicted in Figure 1a, for minimum ($\beta^-(\Delta t)$) and maximum supplied service ($\beta^+(\Delta t)$). Appropriate service curves have been provided e.g. for static priority preemptive scheduling, EDF, TDMA, and others [2]. But specific resource service curves are difficult to derive when e.g. a preemption count is required, as for the calculation of context switch overhead or cache related preemption delay. The approach can also be computationally intensive as it derives output event models and remaining resource capacity by folding operations in continuous time domain. For this reason practical simplifications have been suggested (e.g. stepwise evaluation [5], finite models of event streams [6]).

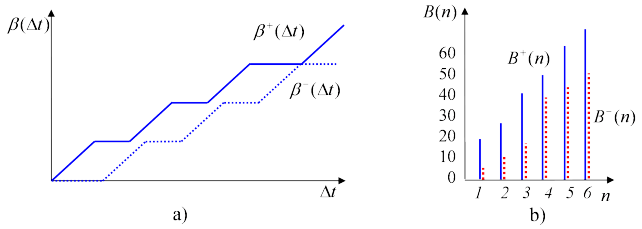

Figure 1: Models of Resource Service.

The opportunities of relying on simpler event and resource models are presented in Section 2.2.

2.1 Event Models

A key element of compositional performance analysis is expressing the traffic flow between different components with the help of *event models*.

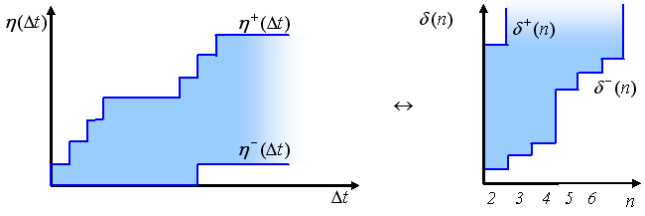

Figure 2: Event Model Representation.

In [2] and [1] event models describe the maximum and minimum number of events η^+ and η^- that may occur during a time interval of given size Δt. Figure 2 shows such an event model representation on the left. An event model can also be expressed by the distances of the contained events. This is shown on the right side of Figure 2. The functions $\delta^-(n)$ and $\delta^+(n)$ represent the minimum and maximum distance between the occurrence of any n events in the stream. The δ functions are therefore sensefully defined only for $n \geq 2$. Both the δ and the η representations can be converted to each other.

In this paper we will mainly utilize the δ representations. It has the strong advantage of being a discrete function of n, rather than the η representations, which are continuous. Thus they are more suitable to the actual numerical computation by allowing to investigate only a discrete set of relevant events.

Correct δ functions have some fundamental properties (such as superadditivity [4]), but no rules are imposed on how they are actually numerically described. For a compact representation, the *standard event models* in [7] rely on the three parameters event stream period $\mathcal{P}$, event stream jitter $\mathcal{J}$, and minimum distance between any two events d^{min}. The δ-function of e.g. a bursty event stream can then be expressed as follows:

$$n \geq 2 : \delta^+(n) = (n-1)\mathcal{P} + \mathcal{J} \quad (1)$$

$$\delta^-(n) = \max((n-1)d^{min}, (n-1)\mathcal{P} - \mathcal{J}) \quad (2)$$

The standard event models are appropriate for many common real-time setups (e.g. in automotive) but may inaccurately represent more complex event patterns. These may occur at the inputs to the system but also within a system where the sequential processing on different resources can significantly distort the event traffic [8]. Packetization and layered communication can aggravate this problem (as investigated in [9]). Finally, in the domain of systems-on-chip the modeling of accesses to a shared memory is of increasing importance [10][11]. Such accesses can be very irregular and their accurate treatment is key to timing validation in MpSoCs. In these cases, a more generic representation must be chosen. Our proposed analysis therefore supports arbitrary event models.

2.2 Compositional Performance Analysis

In [1] the analysis of individual resources is interleaved with the propagation of event models. The procedure is shown in Figure 3.

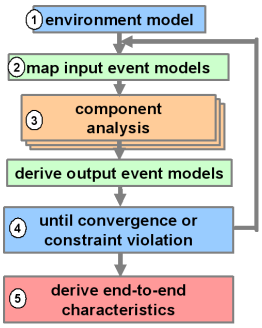

Figure 3: Compositional Analysis Loop.

First the environmental input event models representing the minimum and maximum amount of events that the system is exposed to are specified (1). All other input event models within the system are initialized with optimistic guesses, which are iteratively refined during the analysis procedure. These event models are supplied to the individual components (2), where they are used for local analysis (3). This local analysis provides timing information for each task mapped to the resource.

In [1], the basic metric expressing the timing of a component are the tasks' worst (and best) case response times. This simple metric has been the focus of numerous research in single processor scheduling theory such as [12][3]. A common procedure for its derivation is symbolic simulation of a critical instant scenario. Various extensions have been proposed to improve the analysis results (such as offsets [13] or variable task execution times [14]) and consider realistic scheduling behavior (e.g. cache related preemption delay in preemptive scheduling [15] or the FlexRay bus protocol [16]).

Based on the component's timing, the output event models are

calculated. These output event models can in turn be input event models to other components, or outputs to the environment. The output event models are compared to those used in the previous analysis iteration (4). If all are the same the analysis has converged, otherwise the corresponding local analyses are repeated with the refined inputs.

All event models can only become more generic with each iteration [7], meaning that each iteration contains the previous models. Thus, the complete procedure is monotonic. The analysis is complete if either all event streams converge toward a fix-point, or if an abort condition, e.g. the violation of a timing constraint has been reached.

3. DERIVING OUTPUT EVENT MODELS

The core element of the compositional analysis is the procedure of deriving a task's output event model from the task's input event model and local resource model. We call this procedure "propagation" of event models. In [1], the propagation was achieved by reproducing the standard event model at the input of the task at its output, but with an increased jitter and possibly different minimum distance. The jitter increase is determined by the difference between the task's worst case and the best case response time. In our case, by relying on the multiple event busy time introduced below, we have the opportunity to calculate more precise output event models and to support arbitrary event models.

3.1 Multiple Event Busy Time Model

This section elaborates the concept of the busy period that is used in most response time analyses that use the windowing technique to determine a task's worst case response time. In [3], the busy period is the time interval from the occurrence of a "critical instant" until the resource is idle for the first time. One of the task activations within this busy period then experiences the worst case response time.

As a generalization of this concept, we formally define the *multiple event busy time* function. The multiple event busy time function represents the amount of time necessary to process a certain number of events that arrive within the same busy window. For example, $B^+(1)$ is the maximum busy window inflicted by a single event that arrives after the previous was finished. $B^+(2)$ is the maximum busy window size that is spanned by two events, where the second arrives before the first is finished. The minimum busy time B^- can be defined correspondingly.

Definition 1 (Multiple Event Busy Time)**.** *The n-event busy time $B_i^+(n)$ ($B_i^-(n)$) of a task i is given by the maximum (minimum) time it may take i to process n events, if all but the first of the n events arrive before the preceding is finished.*

For illustration consider the example schedule in Figure 4. The first activation of task T3 experiences a critical instant scenario for static priority preemptive scheduling: all higher priority tasks are activated at the same time and as early as possible thereafter. This leads to a worst case busy time of $B_{T3}^+(1) = 15$, which is the sum of the involved core execution times. The next activation (arriving at time 11) is processed subsequently, and finished no later than $B_{T3}^+(2) = 22$. The series of such derived busy times can be plotted, as depicted in Figure 1b.

By the above definition, the busy time contains all effects that can delay the finishing of the task activations. In particular, this includes the task's execution times, the interference by other tasks mapped to the same resource as well as the scheduler's decisions of the execution order. If present, inter-task communication, context switch overhead and other delays must also be covered.

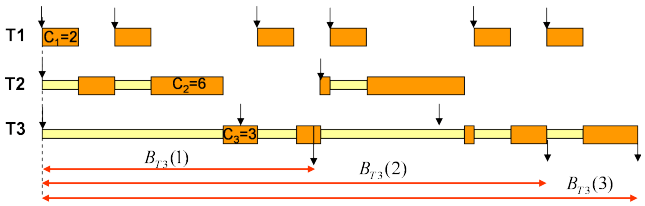

Figure 4: Multiple Event Busy Times.

This busy time has been implicitly used in many previous scheduling analyses that rely on the windowing technique such as [12, 3, 17, 18]. During the calculation of the worst case response time, finishing times of different task activations are calculated — in a worst case scenario this corresponds to busy times as defined above. Thus, in this case the calculation of the busy times comes at no additional computational costs. Despite its obvious similarity to previous use of the busy period, the above definition is different in mainly the following ways:

- The multiple event busy time does not depend on the actual activation pattern of the investigated task (T3 in the example). This fully decouples the resource model from the event model, making it highly useful for the system level analysis.
- The multiple event busy time does not imply non-idleness (as the busy period), although this will be the case for any work conserving scheduler.

To demonstrate the feasibility of the concept, Lemma 1 provides the multiple event busy time for static priority preemptive scheduling of independent tasks (on the basis of [3]).

Lemma 1. *The multiple event busy time $B_i^+(n)$ for a task i under static priority preemptive scheduling with independent tasks is given by*

$$B_i^+(n) = n \cdot C_i + \sum_{j \in hp(i)} (\eta_j^+(B_i^+(n)) \cdot C_j \qquad (3)$$

where
C_i, C_j is the maximum core execution time of task i, j.
$hp(i)$ is the set of tasks with higher priority than i.
$\eta_i^+(\Delta t)$ is the maximum number of events that lead to an activation of task i in a time window of size Δt.

As $B_i^+(n)$ is used on both sides of Equation 3 no direct solution is available. However, the right hand side is monotonic with respect to $B_i^+(n)$, and thus the fixpoint can be found through iteration.

Similarly, more sophisticated windowing based analyses can be straight-forwardly extended to produce the busy time function. This allows to consider e.g. shared resources, task preemption costs, task offsets [13], variable task execution times [14], or accesses to shared memories [19].

3.2 Minimum Distances

As described above, a task's output event model was derived in previous work mainly on the basis of its worst case response time. However an event's arrival and its resulting response time are actually correlated.

For an example, let all events be numbered according to the sequence of their occurrence — events occurring later receive higher numbers. Now, consider finding the minimum distance between the production of an event 0 and some preceding event -1. In any case event 0 will be processed no sooner than $B^-(1)$ after its arrival. So the problem can be reduced to finding the maximum finishing time of event -1. This again depends on the state of the resource

at the time of -1's arrival. If it is not busy processing preceding events, -1 will be finished no later than $B^+(1)$ after it has arrived (see Scenario a in Figure 5). If a previous event -2 is still being processed, event -1 automatically contributes to the ongoing busy interval. This interval is then finished e.g. no later than $B^+(2)$ after the arrival of the preceding event -2 (Scenario 5b). Thus, the maximum finishing time is actually the maximum over the end times of the busy intervals started by all previous events.

The following theorem states our derivation of the minimum distances at the output of a task for arbitrary event models and busy time functions. We consider a task i with input event model δ_i and output event model δ_{i+1}.

Theorem 1. *Given a task i with maximum busy time function $B_i^+(n)$ and minumum busy time $B_i^-(1)$. When the minimum distance between n arriving events is bounded by $\delta_i^-(n)$ then the minimum distance between n events at the output of this task is given by $\delta_{i+1}^-(n)$ as follows:*

$$\begin{aligned} \delta_{i+1}^-(n) &= \max[0, \\ &\quad \min_{k\in K}\{\delta_i^-(n+k-1) - B_i^+(k)\} + B_i^-(1)] \\ K &= \{k \in \mathbb{N}^+ \mid \delta_i^-(k+1) \leq B_i^+(k)\} \end{aligned}$$

Proof. Let the arrival time of an event n at the resource to which task i is mapped be $a_i(n)$ and the time at which the resulting task activation produces an event be $e_i(n)$.

The distance between any n events at the output can never be smaller than the minimum time between the production of an event m and the production time of an event q that has been produced $n-1$ events earlier ($q = m - n + 1$). Let $e_i^-(m)$ be the earliest possible production time of event m and $e_i^+(q)$ the latest production time of q. This minimizes the distance of their production. Furthermore, event m is always produced after event q, due to in order processing.

$$\delta_{i+1}^-(n) = \max[0, e_i^-(m) - e_i^+(m-n+1)] \tag{4}$$

For ease of presentation we perform some simplifications without loss of generality: First, we rename events such that $m = 0$. Then, let the input events that have led to the production of the events 0 and $-n+1$ have arrived at times $a_i(0)$ and $a_i(-n+1)$ respectively. Furthermore, let the time at which event 0 arrives be time 0, i.e. $a_i(0) = 0$. Finally, we omit the index i identifying the task under consideration.

Now, we can safely assume that event 0 could not be produced earlier than $a(0) + B^-(1)$.

$$e(0) \geq a(0) + B^-(1) \tag{5}$$

Thus, the problem is now reduced to find the maximum production time of event $-n+1$. The production time of this event, $e(-n+1)$, is given by the time $a(-n+1)$ at which the corresponding input event has arrived plus the amount of time B that was required to process it.

$$e(-n+1) \leq a(-n+1) + B \tag{6}$$

Firstly, events that belong to the same event stream are constrained by the corresponding event model, particularly the minimum distances δ_i^-. For example, the minimum and maximum arrival times $a(n)$ and $a(m)$ of two events n and m (with $n < m$) are by definition constrained as follows:

$$\begin{aligned} &a(m) \geq a(n) - \delta^-(m-n+1) \quad \wedge \\ &a(m) \leq a(n) - \delta^+(m-n+1) \quad \forall n, m \in \mathbb{N}, n < m \end{aligned} \tag{7}$$

Thus, the event $-n+1$ can arrive no later than

$$a(-n+1) \leq a(0) - \delta^-(n) = -\delta^-(n). \tag{8}$$

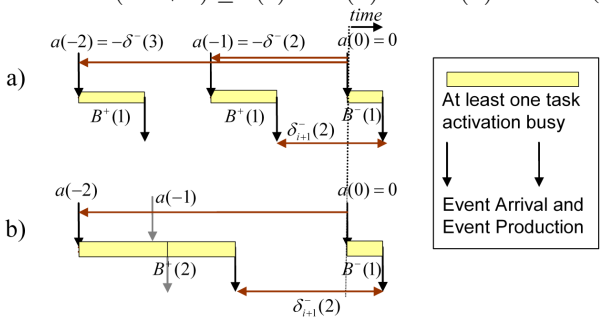

Figure 5: a) Busy times of preceding events may not overlap b) may overlap.

Secondly, the processing time B in Equation 6 depends on the state of the component at the arrival time of the input event. If at time $a(-n+1)$ no previous events from the same event stream are being processed then the processing of event $-n+1$ will be finished after the worst case busy time of a single event ($B^+(1)$). (This is illustrated in Figure 5a for $n = 2$. Here, event -1 arrives after event -2 is finished.) Thus:

$$e(-n+1) \leq a(-n+1) + B^+(1) \tag{9}$$

If the arrival of input event $-n+1$ *does* fall into the busy period of a previous event, it is still guaranteed that its processing is finished after the busy period that it now contributes to is over (see Figure 5b where event -1 falls into the busy period of event -2, and both activations are finished no later than $a(-2) + B^+(2)$).

More generally, given event $-n+1$ falls into the unfinished busy period of the event that has arrived k events before, $a(-n+1) < e(-n+1-k)$, this busy period has started no later than $a(-n+1-k)$ and the corresponding event must be produced no later than at time $a(-n+1-k)+B^+(k+1)$. From the definition of the busy time follows that all events that have arrived before event $-n+1$ (including $-n+1$) must at that time be finished.

Thus, we can state

$$\forall k \in \mathbb{N}^+ : a(-n+1) < e(-n+1-k) \tag{10}$$

$$\Rightarrow \quad e^+(-n+1) \leq a(-n+1-k) + B^+(k+1) \tag{11}$$

Note that equation 9 turns out to be a special case of equation 11 with $k = 0$.

Thus we have a set of inequalities bounding the maximum exit time of event $-n+1$. But the inequalities 11 only hold for those events for which the condition 10 is fulfilled. Except for $k = 0$, it is not possible to say in advance which conditions are fulfilled (i.e. how many events are still unprocessed at the time of $-n+1$'s arrival). In any case, condition 10 can only be fulfilled when the minimum distance between the arrival of event $-n+1$ and event $-n+1-k$ is smaller than the maximum busy time that can be started by event $-n+1-k$. This reduces condition 10 to the following set:

$$\begin{aligned} K &= \{k \in \mathbb{N}^+ \mid a(-n+1) < e(-n+1-k)\} \\ &= \{k \mid \delta^-(k+1) < B^+(k)\} \end{aligned} \tag{12}$$

Summarizing, for all k, for which the condition in the set of equation 12 is true, event $-n+1$ may fall into the busy period of k events before, and equation 11 bounds its finishing time. In particular, if $-n+1$ does not fall into the busy period of any preceding event, equation 9 bounds its busy time. No other cases are possible.

Any one of these scenarios is possible during runtime. A conservative assumption is now to take the maximum of all scenarios for which condition 12 is true. Thus, $e(-n+1)$ can be bounded by

$$e(-n+1) \leq \max_{k \in K}\{a(-n+1-k) + B^+(k)\} \quad (13)$$

Together with equation 8 this can be expressed as follows:

$$e(-n+1) \leq -\min_{k \in K}\{\delta^-(n+k-1) - B^+(k)\} \quad (14)$$

Finally, together with equation 4 the theorem follows.

□

Finally, to predict the minimum output event distances more accurately, it is worthwhile to reconsider the case where the resource can be transiently over-occupied. Theorem 1 will in this case often return an overly conservative 0 as the minimum distance between a small number of events. The minimum execution time C_i^- of the processing task i can be used to ameliorate this prediction. If all events from the same event stream are processed in order by the same task, the events at the output will be produced at least with a distance C_i. More generally, the busy time $B^-(n)$ delivers the minimum time to finish n coinciding activations. This is exploited in the following theorem.

Theorem 2. *At the output of a task i the distance between any n produced events is never smaller than*

$$\delta_{i+1}^-(n) \quad = \quad B_i^-(n-1) \quad (15)$$

Proof. In order for 2 events to be observable at the output of task i, task i must have been activated by two events. Each of these task activations will produce exactly one event during its execution. Both executions can directly succeed each other. Assuming that the events are produced at the end of the task execution, the minimum distance between the 2 produced events is $\delta_{i+1}^-(2) = B_i^-(1)$.

Any further events to be observed at the output require another task activation. Every task activation requires at least $B_i^-(1)$ to execute. A third activation can therefore begin its execution no sooner than $B_i^-(1)$ after the previous. Thus, the third event may not be produced $\delta_{i+1}^-(3) = B_i^-(2)$ after the first. This reasoning can be continued for further events. □

3.3 Maximum Distances

The above calculated minimum distance between events is of major importance to calculate the worst case load on a particular resource. But for many setups, e.g. in control loops, the best case is of equal importance. For its derivation it is necessary to provide the minimum load, which is given by the maximum distance between events. This is provided in the following theorem. Similar to the previous consideration about minimum distances, the key idea is to minimize the finishing time of one event and maximize the finishing time of a successive event.

Theorem 3. *Given a task i with a maximum busy time function $B_i^+(n)$ and a minimum busy time $B_i^-(1)$. When the maximum distance between n arriving events is bounded by $\delta_i^+(n)$ then the maximum distance between n events at the output is bounded by $\delta_{i+1}^+(n)$ as follows:*

$$\delta_{i+1}^+(n) = \max_{k \in K}\{\delta_i^+(n-k+1) + B_i^+(k)\} - B_i^-(1) \quad (16)$$

Proof. The proof of this Theorem follows along the lines of Theorem 1: Event 0 can not be processed earlier than $B_i^-(1)$. The finishing time of a successive event n is maximized, if it arrives as late as possible. Then let all intermediate events also arrive as late as possible and span the largest possible busy windows, causing the maximum disturbance to event n. The difference between the production of the first and the n-th event is the maximum event distance. The full proof is ommited for brevity. □

CPU1					CPU2			
task	event model t_i, b, t_o	cet	priority		task	event model t_i, b, t_o	cet	priority
T1	1,3,20	[1,3]	high	→	T3	derived	[2,5]	high
T2	random	[0.25, 3.5]	low	→	T4	derived	[1,4]	low

Table 1: Parameters of Example System

4. ANALYSIS CONVERGENCE

Above, we have extended the analysis procedure of 2.2 to allow the treatment of more expressive event and resource models. An important part of the analysis is the applicability to multiprocessor systems with cyclic functional task dependencies (i.e. data dependencies) and non-functional dependencies (as introduced by resource sharing). A cyclic dependency in the system leads to cyclic dependencies for the analysis. We therefore revisit the question of analysis convergence in the following rationale.

The analysis convergence relies on the following key factors: each analysis is monotonic (i.e. given analysis A and event models M_1, M_2 $M_1 \geq M_2$ implies $A(M_1) \geq A(M_2)$, where $\geq$ on event models means literally 'more generic'), furthermore a minimal increment for each analysis can be found (i.e. if event models are updated infinitely often, at least one must reach infinite load).

The event model of [7] has a single parameter critical in the analysis iteration: the maximum jitter. It is the only parameter changed during system analysis. We replace this concept with a more expressive vector describing the event model at multiple points. The notion of 'more generic' can be however easily extended to our vector description. Furthermore the minimal increment is still justified, as the system is entirely discrete and the smallest measures of the system is some minimum execution sequence or at least a 'clock tick'. Finally, correct analyses are always monotonic as defined above. Thus, the requirements to ensure convergence are still conserved.

5. EXPERIMENTS

We have conducted a set of experiments to show the validity and benefit of the presented approach. First, consider a simple example system consisting only of two tasks mapped to a static priority preemptive resource. Table 5 shows the setup, and Figure 6 shows the calculated output event models of the low priority task T2. The black square-tagged curves show the bounds on the event model derived by the classical method of adding the response time jitter (EM_{RT}), while the inner triangle-tagged curves show the result of the procedure proposed in this paper (EM_{BT}). It can be seen that the new method predicts tighter bounds on the resulting number of events.

To quantify the improvement, we introduce three metrics: First, the maximum deviation in the number of predicted events. In the example, the original method calculates the number of events by more than 30% larger than the new method. In a time window slightly smaller than 10 time units, the original analysis predicts 8 events, while only 5 may occur. Second, the maximum deviation of the predicted event distances. In the example the distance between 8 events was predicted to be at least 9 time units, while the lower bound provided by the new method was actually 17 time units. Thus the original method has overestimated by almost 90%. These two metrics focussed on the worst case deviation, in order to capture the overall accuracy, we also compare the "tightness" of the

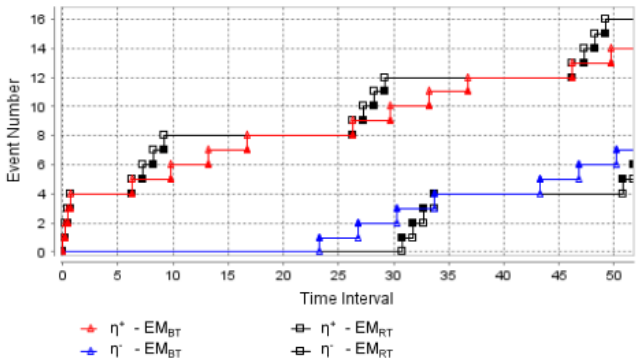

Figure 6: Example Output Event Models Propagated

two event models. For this, we calculate for each the area between the upper and lower event curve. A smaller area means a better prediction. In this example, the improvement in area was around 10%.

The increased precision of the proposed approach is due to the correlation of individual event distances and their respective worst case latencies. The loss of precision can be expected to increase if more than one resource is involved. Reduced event model accuracy leads to a degradation of successive response time estimates, eventually causing rejection of systems that are actually schedulable.

To further investigate the average benefit, we model a small two processor system with two parallel processing chains (see Table 5, task T1 sends data to T3, task T2 sends data to T4). We generated 1000 sets of random event models with periodic bursts at the input of T2 (Parameter ranges: Inner period $1 \leq t_i \leq 4$, burst size $1 \leq b \leq 10$, outer period $t_o = t_i * (b + 1 + r) \mid 0 \leq r \leq 80$). About 10% of the generated event models caused a resource to be overloaded and were discarded. In all other cases we compared the resulting event models at the output of T4. Total analysis time in our implementation was 2 minutes.

The distribution of the maximum deviation of predicted event distances is shown in Figure 7a. In most of the experiments, the original propagation method derived around 30% to 40% larger values, and in the extreme cases even 400%, which shows the benefit of using the new method.

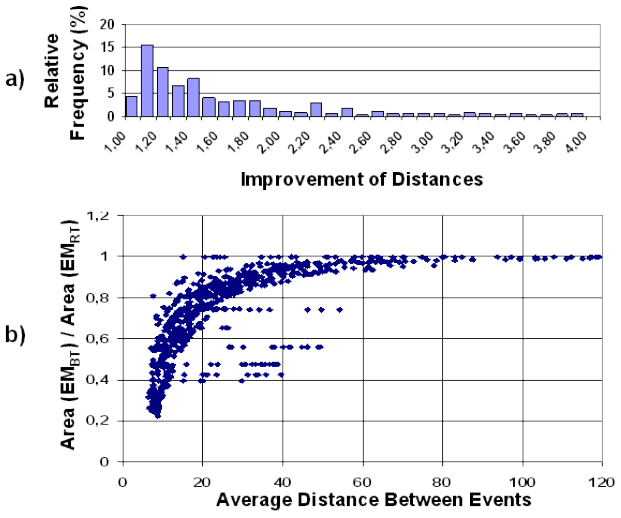

Figure 7: Comparing Quality of Obtained Event Models

Also the overall tightness of the resulting event models improves significantly: In our experiments, the event model area could on average be reduced by 35%. The amount of improvement is dependant on the load imposed on the processors. In Figure 7b the change in event model area is plotted against the average distance between events, which linearly influences the processor load. In systems with large event distances (i.e. low utilization), the simple propagation mechanism already captures the behavior accurately. As the load increases, the new event model propagation provides significantly tighter bounds with less than one quater of the original area.

6. CONCLUSION

In this paper we have proposed an efficient and accurate methodology to derive traffic estimates and path latencies in a multiprocessor system. The resource behavior is modeled using the discrete multiple event busy time function, which can be derived on the basis of classical response time analysis. Through this abstraction our method is suitable to consider arbitrary input event models in large variety of different heterogeneous scheduling policies. The experiments have demonstrated the applicability and improvement over previous methods.

7. REFERENCES

[1] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. In *IEE Proceedings Computers and Digital Techniques*, 2005.

[2] S. Chakraborty, S. Kunzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. *Proc. 6th Design, Automation and Test in Europe (DATE)*, 2003.

[3] KW Tindell, A. Burns, and AJ Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[4] J.Y. Le Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer, 2001.

[5] S. Chakraborty and L. Thiele. A New Task Model for Streaming Applications and Its Schedulability Analysis. *In Proc. Design, Automation and Test in Europe (DATE)*, 2005.

[6] Ernesto Wandeler. *Modular Performance Analysis and Interface-based Design of Embedded Systems*. PhD thesis, Swiss Federal Institute of Technology, 2006.

[7] Kai Richter. *Compositional Scheduling Analysis Using Standard Event Models*. PhD thesis, Technical University of Braunschweig, 2004.

[8] Simon Perathoner, Ernesto Wandeler, and Lothar Thiele et al. Influence of different system abstractions on the performance analysis of distributed real-time systems. *Design Automation for Embedded Systems*, 2008.

[9] Jonas Rox and Rolf Ernst. Construction and Deconstruction of Hierarchical Event Streams with Multiple Hierarchical Layers. *In Proc. 20th Euromicro Conference On Real-Time Systems*, 2008.

[10] S. Schliecker, M. Ivers, and R. Ernst. Memory Access Patterns for the Analysis of MPSoCs. *IEEE North-East Workshop on Systems*, 2006.

[11] K. Albers, F. Bodmann, and F. Slomka. Hierarchical Event Streams and Event Dependency Graphs: A New Computational Model for Embedded Real-Time Systems. *In Proc. 18th Euromicro Conference on Real-Time Systems*, 2006.

[12] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390, 1986.

[13] J.C. Palencia and M.G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. 19th IEEE Real-Time Systems Symposium*, 1998.

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

[15] J. Staschulat, S. Schliecker, and R. Ernst. Scheduling Analysis of Real-Time Systems with Precise Modeling of Cache Related Preemption Delay. *In Proc. 17th Euromicro Conference on Real-Time Systems*, 2005.

[16] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing Analysis of the FlexRay Communication Protocol. *In Proc. 18th EuroMicro Conference on Real-Time Systems, Dresden*, 2006.

[17] C. Li, R. Bettati, and W. Zhao. Response time analysis for distributed real-time systems with bursty job arrivals. *Proceedings of IEEE ICPP*, 1998.

[18] Razvan Racu, Li Li, Rafik Henia, Arne Hamann, and Rolf Ernst. Improved Response Time Analysis of Tasks Scheduled under Preemptive Round Robin. *Intl. Conf. on Hardware Software Codesign and System Synthesis*, 2007.

[19] S. Schliecker, M. Ivers, and R. Ernst. Integrated analysis of communicating tasks in MPSoCs. *Intl. Conf. on Hardware Software Codesign and System Synthesis (CODES)*, 2006.