Intra- and Inter-Processor Hybrid Performance Modeling for MPSoC Architectures

Frank E. B. Ophelders[1] Samarjit Chakraborty[2] Henk Corporaal[1]
[1]Department of Electrical Engineering, Technische Universiteit Eindhoven
[2]Department of Computer Science, National University of Singapore
f.e.b.ophelders@student.tue.nl, samarjit@comp.nus.edu.sg, h.corporaal@tue.nl

ABSTRACT

The heterogeneity of modern MPSoC architectures, coupled with the increasing complexity of the applications mapped onto them has recently led to a lot of interest in hybrid performance modeling techniques. Here, the idea is to apply different modeling and analysis techniques to different subsystems/components of an architecture/application. Such hybrid techniques often turn out to be more efficient and accurate compared to relying on a single analysis technique for the entire system. However, the challenge associated with this approach is to combine the different analysis results effectively to obtain conservative performance estimates for the entire system. In this paper we study a hybrid scheme where certain system components are simulated (e.g. using instruction set simulators), whereas others are analyzed using a formal technique called Real-Time Calculus (RTC). The main novelty of our approach stems from our use of this hybrid technique even for multiple tasks mapped onto a single processing element. In contrast to this, previous approaches relied on either full simulation or RTC-based analysis for an entire architectural component (e.g. a processor or a bus). The techniques we develop in this paper therefore allow for both intra- and inter-processor hybrid performance modeling and show how the different analysis results can be combined to efficiently obtain tight performance estimates for complex MPSoC architectures. We demonstrate the usefulness of this approach using an MPEG-2 decoder application that is partitioned and mapped onto two processing elements connected by FIFO buffers.

Categories and Subject Descriptors

C.3 [**Computer Systems Organization**]: Special-purpose and application-based systems—*Real-time and embedded systems*

General Terms

Algorithms, Performance, Design

Keywords

Performance analysis, Simulation, System-on-Chip

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

1. INTRODUCTION

Modern real-time systems are increasingly becoming distributed and heterogeneous. They consist of multiple processing elements, hardware accelerators, memory units and communication systems. Each of these components pose very diverse requirements and timing constraints. They also support different scheduling and resource arbitration policies and often interact with the physical world (via sensors and as actuators). As a result, timing and performance analysis of such system is an important and challenging problem.

There have been previous efforts to use multiple languages and formalisms for *specifying* such heterogeneous real-time and embedded systems (e.g. see [17]). However, the issue of composing different *analysis techniques* has not been sufficiently explored so far. Recently, there has been some progress on this front. For example, [6] outlined a scheme where some of the components of a system were analyzed using purely simulation-oriented techniques (based on SystemC models), while the remaining components were modeled and analyzed using a set of algebraic equations based on the so called Real-Time Calculus (RTC) [3] formalism. Such an hybrid approach provides a good tradeoff between the analysis time involved and the quality (tightness) of the results (e.g. timing/performance estimates) returned. While formal analysis techniques like RTC involve very short run/analysis times, the estimates returned by them can be overly optimistic or pessimistic if the component being analyzed implements a complex functionality which is difficult to model accurately. On the other hand, simulation-based techniques are relatively easy to use and often provide accurate analysis results. However, they usually involve large simulation times and implementation overheads. As a result, using either of these two approaches for an *entire* system can turn out to be unacceptable for large and complex systems consisting of multiple heterogeneous components.

The main contribution of [6] was an interface between the results returned by components which are simulated and those which are analyzed using RTC. While the RTC-based analysis represents the arrival patterns of data and event streams in an abstract fashion, the SystemC-based models require concrete input traces to trigger the simulation. To compose the two analysis techniques, one must be able to inter-convert the two models. Deriving abstract representations of arrival patterns from concrete traces (the SystemC-to-RTC conversion) is relatively straightforward (e.g., by using standard event models such as *periodic with jitter*). However, a transformation in the other direction, i.e., deriving a small set of representative traces from an abstract event model (the RTC-to-SystemC conversion) is a challenging problem. A similar approach was presented in [13], where certain system components were modeled using SDF graphs [5, 7] with the remaining ones being subjected to classical real-time schedulability analysis techniques.

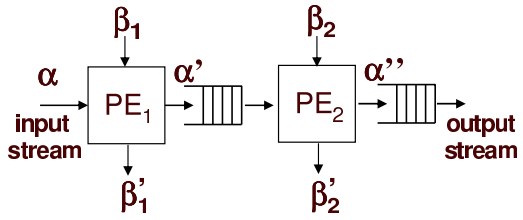


Figure 1: Modeling an MPSoC architecture.

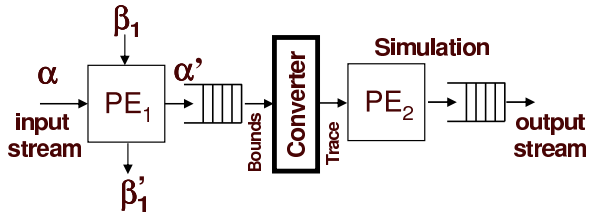


Figure 2: Inter-processor hybrid performance modeling.

Contributions of this paper: In this paper we extend the approach proposed in [6] so that the hybrid analysis technique is not only applied across different architectural components (e.g. a processor or a bus) but also across tasks mapped onto a single processing element. As a result, some of the tasks mapped onto the same processor are modeled using RTC, while the executions of the remaining tasks are simulated. To understand how this is achieved, some details of the RTC framework are necessary.

Consider a multiprocessor system-on-chip (MPSoC) architecture consisting of multiple processing elements (PEs) connected by FIFO buffers and communicating through some communication subsystem (e.g., shared buses). Further, consider an application that has been partitioned and mapped onto this architecture. An overview of such an architecture is shown in Figure 1. A data (e.g., video) stream enters this system and first gets processed at PE_1. This partially processed stream gets written into a buffer, which is then read by PE_2 for further processing. Finally, the fully processed stream gets written into an output buffer which is read by a playout device.

The RTC framework was proposed to analyze such setups in [3], which was further extended in subsequent papers (e.g. see [4, 15]). The key feature of this framework is that it allows a very general modeling of event streams and resource availability beyond the classical event models such as periodic, sporadic, periodic with jitter, etc. Another important feature — which is particularly relevant for performance analysis — is that, rather than recording the precise arrival times of events, RTC uses a *count-based abstraction* which specifies upper and lower bounds on the possible number of events that can arrive (or can get processed) within any pre-specified time interval length. This count-based abstraction captures in a very natural way, bursty arrival patterns of events/data, variable execution demands and irregular resource availability. RTC-based models use upper and lower bounds on the number of events that arrive at a component (processing or communication element) to get processed within any time interval of a specified length. Such bounds are represented as *functions* (denoted by α in Figure 1) and are used to estimate the workload to be supported by the component. Similar functions are also used to represent the *service* offered by the component (denoted by β in Figure 1). While such functional modeling allows for efficient analysis, the main drawback of this approach is that it can not model complex task processing or any state-based information. For example, a common feature such as "the processor stalls when the output buffer is full" cannot be modeled easily since the service offered by the processor in this case depends on the *state* (fill level) of the buffer.

To get around this problem, [6] proposed to model some of the PEs using RTC, while the others were simulated. Towards this, suppose that PE_1 in Figure 1 is analyzed using RTC, whereas PE_2 is analyzed using simulation (e.g. either based on a SystemC model or an instruction set simulator). For this, the so called *arrival curve* α is used to bound the arrival process of the data items at PE_1. The *service* offered by this PE is captured using a similar bound or *service curve* β_1. Formal definitions of arrival and service curves are given later in the paper. The arrival curve of the processed stream – denoted by α' – is computed from α and β_1 and serves as the input to PE_2. Now, since PE_2 is simulated, it needs a concrete input trace (i.e., arrival times of the data items at PE_2) to trigger the simulator. For this, a *bound-to-trace converter* is used to obtain one or more representative traces (which conform to α') that serve as an input to the simulator for PE_2. This is illustrated in Figure 2. A similar *trace-to-bound converter* is required when the output from a simulator feeds a component which is analyzed using RTC.

Note that using the above scheme, an *entire* architectural component is either modeled using RTC or is simulated. In this paper we extend this basic idea one step further to incorporate *intra-processor* hybrid performance modeling. Towards this, some of the tasks on a PE are modeled using RTC, while the execution of the others is simulated. What is now required is a converter from the service curve β to one or more processor availability traces, and vice-versa. In the following sections, we show how such converters are obtained and how our intra-processor hybrid performance modeling works in conjunction with the inter-processor hybrid modeling introduced in [6]. By inter-converting between arrival/service curves and traces we are able to obtain a good balance between the accuracy of the performance analysis results and the time taken to perform such analysis.

Before concluding this section, we would like to point out that our approach is orthogonal to the previous efforts on full-system performance analysis, for example, using holistic schedulability analysis [1, 10, 11, 12, 14]. Although these efforts were also directed towards analyzing systems with multiple resources and scheduling policies, the underlying analysis techniques used for the different components were similar and were mostly based on classical event models. Our goal, on the other hand, is to combine two fundamentally different performance models for system-level analysis of complex MPSoC architectures.

Organization of this paper: In the next section, we briefly outline the RTC framework. This is followed by a description of the converters that are used for the proposed hybrid performance modeling. Finally, a case study is presented in Section 4, followed by a discussion on possible extensions of this work.

2. REAL-TIME CALCULUS

As mentioned in the previous section, an event or data stream in RTC is described using upper and lower *arrival curves*, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$. These curves provide upper and lower bounds on the number of events that are seen on the event stream within any time interval of length Δ. In particular, there are at most $\alpha^u(\Delta)$ and at least $\alpha^l(\Delta)$ events within the time interval $[t, t+\Delta)$ for all time instances t. If $R(t)$ denotes the number of events that arrive at a resource during the time interval $[0, t]$ then such a stream is bounded by $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, where

$$\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta),\ \forall t \geq 0,\ \forall \Delta \geq 0$$

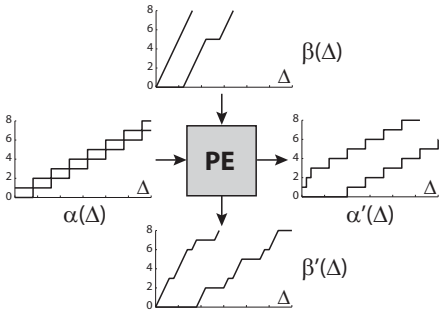


Figure 3: An abstract performance model in RTC.

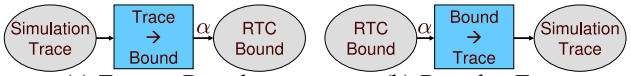
Analogously, a resource is characterized using upper and lower service curves, $\beta^u(\Delta)$ and $\beta^l(\Delta)$, which provide upper and lower bounds on the available resource in any time interval of length Δ. The unit of resource depends on the nature of the resource, e.g., processing cycles (computation) or bytes (communication), which can be transformed to the same unit as the arrival curves (number of events). The processing semantics of a processing/communication component (e.g. a processor or a bus) can be captured using an RTC component as shown on Figure 3. Here, an arrival curve $\alpha(\Delta)$ enters a processing element and is processed using the service curve $\beta(\Delta)$. The output is an arrival curve $\alpha'(\Delta)$ (which denotes the arrival pattern of the *processed events*) and the *remaining resource* is expressed as the service curve $\beta'(\Delta)$. Internally, the RTC component is specified by a set of functions, that relate the incoming arrival and service curves to the outgoing arrival and service curves. These functions are dependent on the processing semantics of the component (see e.g. [3]). Similar representations exist for resource scheduling disciplines such as EDF, TDMA, and GPS, when multiple streams or tasks are being processed by the component. The outgoing arrival curves might serve as input to another component, denoting further processing of the event stream (as shown in Figure 1). Similarly, the outgoing service curve represents the service that is available to other tasks running on the same component. Various performance properties can be computed analytically using this performance model such as end-to-end delays and buffer requirements. These methods have been implemented as a toolbox which is based on Matlab and Java (see [16]).

3. DESIGNING CONVERTERS

In this section we will describe methods to design converters between bounds specified by arrival and service curves (i.e., α and β) and concrete traces of event/data and processor availability. As described in Section 1, such converters will then enable us to design hybrid performance models. For combining RTC-based performance models with simulation-oriented models, [6] proposed the following converters.

- Trace-to-Bound event-based converter, as shown in Figure 4(a). Such a converter converts concrete traces of event/data streams – that were obtained from a simulation – into bounds represented by an arrival curve α.

- Bound-to-Trace event-based converter, as shown in Figure 4(b). Such a converter is used to generate one or more representative event/data traces from a bound specified by an arrival curve α. Typically such an α would be obtained from an RTC-based analysis and the generated trace would be used to trigger a simulator.



(a) Trace to Bound (b) Bound to Trace

Figure 4: Event-based converters.

3.1 Event-based converters

The conversion from simulation traces (event streams) to event models specified by arrival curves is clearly simpler than a conversion in the other direction. After simulating a component, the output event traces can be used to derive $R(t)$. As in Section 2, $R(t)$ denotes the number of output events seen over the time interval $[0, t]$, and uniquely represents an event trace. Given $R(t)$, we can now compute the functions $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ as follows.

$$\alpha^u(\Delta) = \max_{t \geq 0}\{R(t+\Delta) - R(t)\} \quad (1)$$

$$\alpha^l(\Delta) = \min_{t \geq 0}\{R(t+\Delta) - R(t)\} \quad (2)$$

Given any Δ, α^u and α^l may be computed by sliding a window of length Δ over the trace $R(t)$ and recording the maximum and minimum number of data/events seen within this window. However, since the number of events in the traces we used were fewer than the number of time interval lengths Δ over which we computed α^u and α^l – for the sake of computational efficiency – we applied the "sliding window" procedure to instead compute ${\alpha^l}^{-1}(k)$ and ${\alpha^u}^{-1}(k)$ (i.e., the inverse of α^u and α^l). Here, the window length k represented the number of data items or events in the trace and the functions ${\alpha^l}^{-1}(k)$ and ${\alpha^u}^{-1}(k)$ denote the maximum and minimum time interval lengths respectively that can contain k events. The functions α^u and α^l can then be computed from ${\alpha^u}^{-1}$ and ${\alpha^l}^{-1}$ in a straightforward manner.

The event-based Bound-to-Trace converter is more complicated. Such a converter generates a trace of time stamps at which events arrive. This generated trace should not violate the upper or the lower arrival curve. Further, it should represent the worst-case scenario (e.g., a bursty arrival pattern) since the RTC framework is used for deterministic worst-case analysis. Towards this, the main idea proposed in [6] is to use an adapted "ON/OFF" traffic source (see [2] for more details on such traffic sources). Such a generator works as follows – if the generator is in the ON state, events are generated as soon as allowed by the upper arrival curve. Similarly, if the generator is in the OFF state events are only generated if the lower arrival curve would be violated otherwise. The generation algorithm consists of the following steps.

1. Determine time stamp T at which to switch state;
2. Generate events in the current state while time $t < T$;
3. Switch state and go to step 1;

The time distribution between the ON and OFF states is determined using a Weibull distribution [9], which results in the desired "fractal" behavior that is used in several traffic generators (see [2, 8]). The Weibull cumulative distribution function of a random variable X is given by:

$$P\{X \leq x\} = 1 - e^{-(x/\mathcal{B})^{\mathcal{A}}}, \forall x \geq 0 \quad (3)$$

It has two parameters $\mathcal{A} \geq 0$ and $\mathcal{B} \geq 0$. The parameter $\mathcal{A}$ is known as the *shape* parameter, while $\mathcal{B}$ is called the *scale* parameter. Random switch times can be generated by Eqn. (4), which results from Eqn. (3) by inverse transform sampling. Eqn. (4) uses a random number u from a uniform distribution in the interval (0,1].

Then the random number x (representing the switch time) is given by:

$$x = \mathcal{B}(-ln(u))^{(1/\mathcal{A})} \quad (4)$$

Traces generated by this ON/OFF traffic source exhibit a worst-case bursty behavior since events are generated as soon as allowed by the upper bound α^u in the ON state. This generation scheme also guarantees that the resulting trace never violates the given upper (α^u) or lower (α^l) bounds.

A set of generated traces using the above scheme is shown in Figure 10. The trace $R_1(t)$ was generated using $\mathcal{A} = 1000$ and $\mathcal{B} = 10000$; the trace $R_2(t)$ was generated with $\mathcal{A} = 1$ and $\mathcal{B} = 1 \times 10^8$; and trace $R_3(t)$ was obtained using the same parameters as in $R_2(t)$. The difference between the latter two traces is caused by the start state – our traffic generator randomly chooses whether to start in the ON or the OFF state. Using different parameters it is possible to generate different traces. Note that trace $R_1(t)$ was generated using shorter ON/OFF periods – as a result the trace is bursty and closely follows the upper bound α^u. The distribution for $R_2(t)$ and $R_3(t)$ yields longer ON/OFF periods and as a result these traces turn out to be different from $R_1(t)$.

3.2 Resource-based converters

In this section we further extend the hybrid performance modeling scheme by introducing *Resource-based converters*. Such converters are used to inter-convert bounds given by β^u or β^l and processor availability traces (i.e., idle cycle traces of a processor or communication resource).

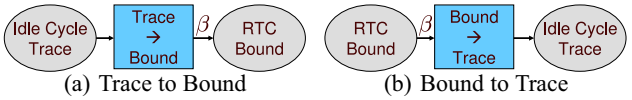
Recall from Section 2 that the service availability of a resource is given by the *service curves* $\beta^u(\Delta)$ and $\beta^l(\Delta)$. For now, let us assume that β^u and β^l denote upper and lower bounds on the number of processor (or bus) *cycles* provided by a resource (and not the number of data or events processed by it). Let $C(t)$ denote the number of processor cycles available over the time interval $[0, t]$. Then, as in the case of arrival curves, the service curves β^u and β^l can be computed from $C(t)$ as follows.

$$\beta^u(\Delta) = \max_{t \geq 0}\{C(t+\Delta) - C(t)\} \quad (5)$$

$$\beta^l(\Delta) = \min_{t \geq 0}\{C(t+\Delta) - C(t)\} \quad (6)$$

For any resource, an *idle cycle trace* is a trace of intervals $[s_i, f_i]$, $i = 0, 1, \ldots$, where the resource is available between time intervals s_i and f_i for task processing. Procedures for using an idle cycle trace can easily be incorporated within a cycle-accurate simulator (e.g, one based on SimpleScalar or SystemC) by instructing the simulator only to increase its cycle count within the specified idle cycle intervals. We have used a SystemC-based simulator which computes an output event stream and output idle cycle trace, given a specified input. The input consists of (i) a set of tasks, (ii) the execution requirement for each task, (iii) the arrival time stamps of events or the triggering pattern of each task, (iv) the scheduling policy according to which the tasks are scheduled, and (v) an idle cycle trace denoting the availability of the resource. Note that the processing time of different events by a task might be different. Hence, the input execution requirement of a task, to the simulator, consists of a trace of processor cycle requirements (i.e. number of cycles) for each event to be processed by the task. The simulator reads the above input and processes events only during the idle cycles specified by the input idle cycle trace. As soon as an event is processed, it is written to the output event stream. When the processor is idle, it adds idle cycles to the *output idle cycle trace*.

The above-mentioned simulator is used in conjunction with the following converters.



(a) Trace to Bound (b) Bound to Trace

Figure 5: Resource-based converters.

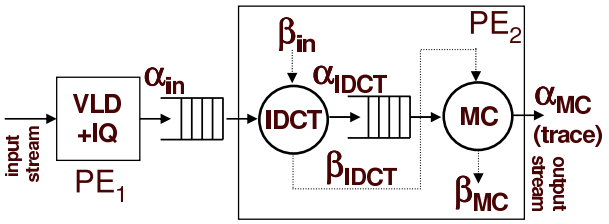


Figure 6: MPEG-2 decoder mapped onto an MPSoC architecture.

- Trace-to-Bound resource-based converter, as shown in Figure 5(a). It converts an idle cycle trace resulting from our simulator into service curves β^u and β^l for RTC-based analysis.
- Bound-to-Trace resource-based converter, as shown in Figure 5(b). It converts β^u and β^l obtained from an RTC-based analysis into idle cycle traces that serve as an input to our simulator.

As with the event-based Trace-to-Bound converter, converting idle cycle traces into service curves for RTC-based analysis is much simpler than the conversion in the reverse direction. Assume that two tasks T_1 and T_2 are executing on a single processing element; T_1 has a higher priority than T_2 and there are no tasks with priority levels in between. Further, assume that the execution of T_1 is simulated (using the SystemC simulator we described above) and T_2 is analyzed using RTC. Clearly, the remaining processor cycles after processing T_1 will be available to T_2. Based on the idle cycle trace of the processing element after processing T_1 (generated by our simulator), we can construct the function $C(t)$, which represents the number of cycles available over the time interval $[0, t]$. From $C(t)$, we can compute β^u and β^l using Eqns. (5) and (6).

As in the case of event-based converters, generating idle cycle traces from service curves is more difficult. Here, we again used a technique similar to the Weibull distribution-based ON/OFF traffic generator. From β^u and β^l, we generate one or more functions $C(t)$ using different values of the parameters in the Weibull distribution. From each such $C(t)$ it is now straightforward to generate an idle cycle trace, since $C(t)$ represents a concrete availability pattern of a resource.

4. ILLUSTRATIVE CASE STUDY

In this section, we demonstrate the usefulness of our proposed hybrid approach. Towards this, we will use an MPSoC architecture onto which an MPEG-2 decoder has been partitioned and mapped. As shown in Figure 6, PE_1 is used to run two tasks (VLD and IQ) and PE_2 also runs two tasks (IDCT and MC). First, we demonstrate that the bounds resulting from our hybrid approach are tighter than those obtained using RTC. More importantly, the resulting bounds do not violate the bounds returned by RTC. Second, we show that the maximum buffer fill-level estimates obtained from our hybrid approach are also significantly tighter than those obtained using RTC (which being a worst-case analysis framework, always returns conservative estimates).

Figure 7 illustrates the design flow in our case study. From Figures 6 and 7, it may be noted that PE_1 is always simulated, whereas PE_2 is subjected to both RTC-based analysis and our proposed hybrid analysis technique. Finally, the results obtained from these two

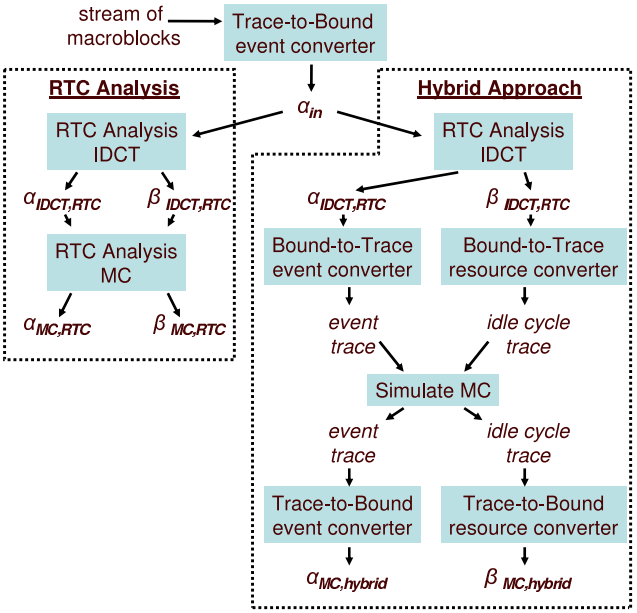


Figure 7: Design flow in our case study.

schemes are compared. In our hybrid modeling, the IDCT task on PE_2 is modeled using RTC and the MC task is simulated (see Figure 7).

Using an event-based Trace-to-Bound converter, the timing properties of the output from PE_1 (which is a stream of partially decoded macroblocks) – that is obtained from simulation – is represented using α_{in}. The RTC framework now uses this α_{in} and β_{in} (service offered by PE_2) to compute $\alpha_{IDCT,RTC}$ and $\beta_{IDCT,RTC}$. $\alpha_{IDCT,RTC}$ denotes bounds on the arrival process of the output stream after being processed by the IDCT task, and $\beta_{IDCT,RTC}$ denotes bounds on the *remaining service* from PE_2 after processing the IDCT task. Note that we have so far assumed α to be represented in terms of *number of events*, whereas β is represented in terms of *number of cycles*. These two bounds may be reconciled by using the number of processor cycles required to process each event. The details on how this can be done may be found in [15].

Now, the task MC is (i) modeled using RTC with $\alpha_{IDCT,RTC}$ and $\beta_{IDCT,RTC}$ as inputs, and (ii) simulated using multiple concrete traces generated from $\alpha_{IDCT,RTC}$ and $\beta_{IDCT,RTC}$ using event-based and resource-based converters. In case (ii) the output traces from the simulator are converted back to arrival and service curve bounds, which are then compared with the output bounds obtained from case (i). The resulting bounds in case (i) are denoted by $\alpha_{MC,RTC}$ and $\beta_{MC,RTC}$, and those in case (ii) by $\alpha_{MC,hybrid}$ and $\beta_{MC,hybrid}$.

For our experiments, we used MPEG-2 video clips with 1.5 Mbps bitrate and 352×420 pixels resolution. Our analysis was done at the macroblock granularity, i.e., we assumed the video stream to be made up of a stream of macroblocks which trigger the different decoder tasks. Finally, the video clips were played out at the rate of 14 fps and both PE_1 and PE_2 were MIPS-like processors running at 100 MHz (both the processors were moderately loaded from the MPEG-2 decoder tasks).

Figure 8 shows $\beta_{IDCT,RTC}$, along with three different idle cycle traces (C_1 C_2 and C_3) obtained using a resource-based Bound-to-Trace converter. Figure 9 shows bounds on the remaining service from PE_2 after processing both IDCT and MC. Here, $\beta_{MC,RTC}$ denotes the bounds resulting from RTC, and $\beta_{MC,hybrid}$ denotes those obtained using our hybrid approach. It may be noted that the latter bounds are "enclosed" by the former, thereby indicating that

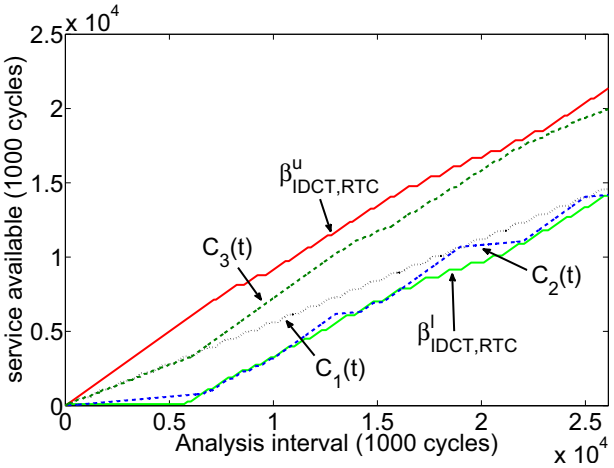


Figure 8: Service availability for MC ($\beta_{IDCT,RTC}$), along with three idle cycle traces (C_1, C_2, C_3).

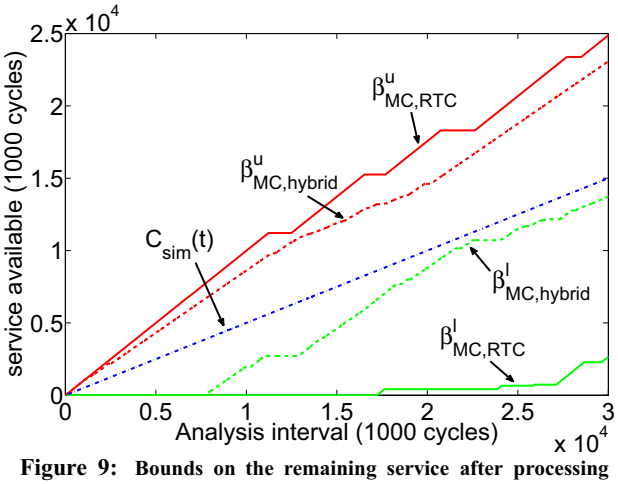


Figure 9: Bounds on the remaining service after processing both IDCT and MC.

our hybrid scheme yields much tighter bounds compared to those obtained by RTC alone. However, it should be noted that since our hybrid scheme has simulation in the design loop, the resulting bounds are not guaranteed to represent the best/worst case scenario. However, for application domains like multimedia processing these bounds might be more meaningful than the conservative bounds returned by a pure RTC-based analysis. C_{sim} in Figure 9 shows one idle cycle trace that was obtained as an output from the simulator.

Figure 10 shows three different concrete traces R_1, R_2 and R_3, that were generated from $\alpha_{IDCT,RTC}$ by the event-based Bound-to-Trace converter. A number of traces like these were used to trigger the simulator modeling the processing of the task MC. The output of the simulator, which were traces of fully decoded macroblocks, were then used to compute $\alpha_{MC,hybrid}$. The corresponding bounds, obtained purely using RTC, are denoted by $\alpha_{MC,RTC}$. Figure 11 shows that the bounds given by $\alpha_{MC,hybrid}$ are completely enclosed by those from $\alpha_{MC,RTC}$, thereby once again showing that the RTC-based bounds are more conservative than those we obtain using our hybrid approach. Note that the bounds obtained by the hybrid approach do not violate those obtained from RTC, thereby establishing the correctness of the approach. Figure 11 also shows a typical output trace that was obtained from our SystemC-based simulator.

Finally, Table 1 shows the estimated maximum fill levels of the playout buffer that sits between the MPSoC architecture shown in Figure 6 and the playout device. The decoded video is written out

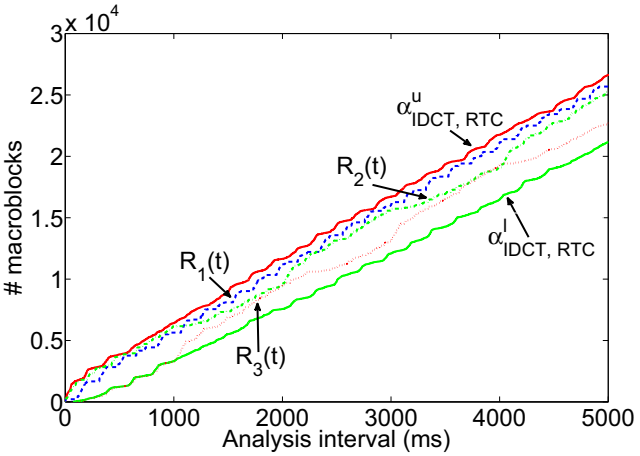


Figure 10: $\alpha_{IDCT,RTC}$ bounds the number of macroblocks arriving at the task MC.

Analysis	Max. buffer fill level (in macroblocks)
Hybrid 1	924
Hybrid 2	945
RTC	7408
Simulation	585

Table 1: Estimated maximum buffer fill levels.

into this buffer, which is read out by the playout device (display). This buffer is not shown in Figure 6. "Hybrid 1" and "Hybrid 2" in Table 1 refer to two hybrid analysis cases with different trace generators. "RTC" refers to the estimate obtained using a pure RTC-based analysis, and "Simulation" refers to the maximum buffer fill level recorded with a single video clip when the full system is simulated. It is easy to see that the results obtained using "Hybrid 1" and "Hybrid 2" are much closer to those obtained using simulation, but the analysis time involved was much shorter. The pure RTC-based analysis – although provides guaranteed worst-case estimates – is much more conservative.

From the above results, it is easy to see that our proposed hybrid performance modeling approach provides a very competitive alternative to full system simulation. A purely simulation-oriented analysis requires significantly higher simulation times. The hybrid approach on the other hand involves lower analysis times – because certain system components are now mathematically analyzed using RTC – but provides results which are relatively close to those obtained from simulation. However, it should once again be noted that unlike the results obtained through RTC, those obtained using our hybrid approach do not come with any provable guarantees. We believe that for application domains such as multimedia processing – where simulation-oriented approaches are widely practised today – this lack of provable guarantees is acceptable.

5. CONCLUDING REMARKS

In this paper we have proposed a hybrid performance modeling scheme for MPSoC architectures. Using this scheme some of the system components are mathematically analyzed using the RTC framework, while the others are simulated. In contrast to previous approaches, our scheme also allows this hybrid scheme to be applied to different tasks mapped onto the same processing element. This hybrid scheme is a viable alternative to performance analysis through full system simulation. It cuts down simulation time significantly, but provides results which are close to those obtained using pure simulation.

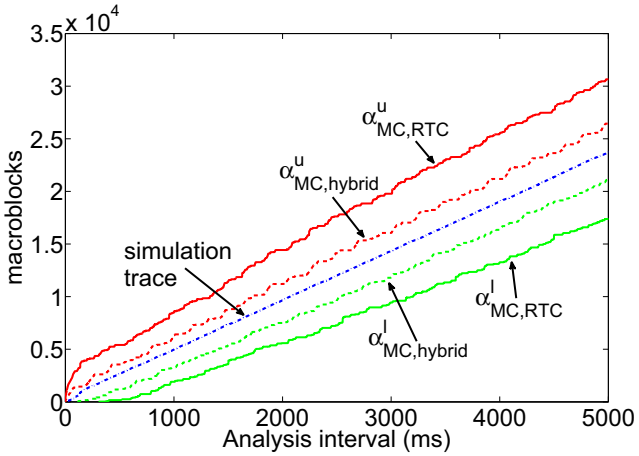


Figure 11: Bounds on the arrival process of the fully decoded video stream (α_{MC}).

Our intra-processor hybrid modeling scheme was shown to work when the different tasks on a processing element are scheduled according to a fixed-priority scheduling policy. It is currently not clear how to extend this to other scheduling policies like EDF. Further work needs to be done to generalize this scheme to cover a wider variety of scheduling policies.

Acknowledgements: We thank Lei Ju for many helpful discussions and for providing us the results of his preliminary study on this problem.

6. REFERENCES

[1] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design & Test of Computers*, 15(1):71–82, 1998.

[2] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Perform. Eval. Rev.*, 26(1):151–160, 1998.

[3] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

[4] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. Interface-based rate analysis of embedded systems. In *RTSS*, 2006.

[5] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. Bekooij, B. D. Theelen, and M. R. Mousavi. Throughput analysis of synchronous data flow graphs. In *ACSD*, 2006.

[6] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *DATE*, 2006.

[7] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[8] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic. *SIGCOMM Comput. Commun. Rev.*, 23(4):183–193, 1993.

[9] D. N. Prabhakar Murthy, M. Xie, and R. Jiang. *Weibull Models*. Wiley Series in Probability and Statistics, 2003.

[10] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *CODES*, 2002.

[11] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *DATE*, 2002.

[12] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4):60–67, 2003.

[13] S. Schliecker, S. Stein, and R. Ernst. Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis. In *DATE*, 2007.

[14] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3), 1994.

[15] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[16] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[17] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI: a system model for heterogeneously specified embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):379 – 389, 2002.