Generation and Calibration of Compositional Performance Analysis Models for Multi-Processor Systems

Wolfgang Haid, Matthias Keller, Kai Huang, Iuliana Bacivarov, Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, 8092 Zurich, Switzerland
Email: firstname.lastname@tik.ee.ethz.ch

***Abstract*—The performance analysis of heterogeneous multi-processor systems is becoming increasingly difficult due to the steadily growing complexity of software and hardware components. To cope with these increasing requirements, analytic methods have been proposed. The automatic generation of analytic system models that faithfully represent real system implementations has received relatively little attention, however. In this paper, an approach is presented in which an analytic system model is automatically generated from the same specification that is also used for system synthesis. Analytic methods for performance analysis of a system can thus be seamlessly integrated into the multi-processor design flow which lays a sound foundation for designing systems with a predictable performance.**

I. Introduction

Heterogeneous multi-processor systems are widely used in real-time systems. These systems provide a good trade-off between performance and flexibility which is required to implement today's complex applications for audio- and video-streaming, (array) signal processing, or packet processing. In the design of heterogeneous multi-processor systems, design space exploration is a central step because the different trade-offs in terms of performance, cost, size, and power need to be evaluated.

Performance analysis — the quantitative analysis of system properties for system analysis and/or system synthesis — plays a fundamental role in this process since it provides the performance data based on which the design decisions are made. Obviously, the performance analysis method used within a design space exploration cycle should be accurate and at the same time fast. Compositional performance analysis implemented in the frameworks of modular performance analysis (MPA) [1] and symbolic timing analysis for systems (SymTA/S) [2] is an analytic technique that achieves a good trade-off between these two goals. In particular, compositional performance analysis is a method that provides bounds on performance metrics, such as system throughput, delay, or resource utilization, and is, therefore, suited for the analysis of real-time systems. Furthermore, a large body of analysis and optimization methods based on compositional performance analysis is available that can be applied during design space exploration. Examples are sensitivity analysis [3], robustness optimization [4], or scheduling parameter optimization [5].

One of the main criticisms of compositional performance analysis (and other analytic methods, as well) is that the generation of an analysis model for a given system implementation is a notoriously difficult endeavor: First, many details about the system implementation itself are required. Second, a proficient knowledge about the modeling features of the used method is necessary. And third, the parameters for the analysis model need to be available.

Without a systematic way to deal with these aspects, model generation is a difficult task, indeed. To overcome this criticism, we thus propose a systematic approach for *model generation* — the generation of a system model for a given system implementation — and *model calibration* — the acquisition of the required model parameters.

In this paper, it is argued that model generation and calibration can be automated (for a certain class of systems). Following a top-down approach, it is shown how a compositional performance analysis model can be derived from the same specification that is used for system software synthesis. This enables the seamless integration of compositional performance analysis into the design space exploration cycle of multi-processor systems. The contributions of this paper can be summarized as follows:

- A formal description of the class of considered systems is given.
- We show how the generation of a compositional performance analysis model can be automatically performed in a multi-processor system design flow.
- It is shown how the required model parameters are automatically obtained.
- We implemented the proposed approach and use it in a case study to demonstrate its viability. The implementation has been developed as an extension of our framework for automated mapping of applications onto multi-processor systems [6] and is available online [7].

The remainder of the paper is organized as follows: The next section discusses related work. Section III gives an overview over the tackled problem and the used approach. In Section IV, the used notation and the used compositional analysis method, namely modular performance analysis (MPA) [1], are introduced, and the considered problem is formally

978-1-4244-4501-1/09/$25.00 ©2009 IEEE

defined. Section V and Section VI describe how MPA models can be created and calibrated based on a given system specification. Section VII illustrates the discussed aspects in a case study. Finally, Section VIII concludes the paper.

II. Related Work

The models most often generated from synthesizable specifications are simulations at different levels of abstractions. In particular, this applies for (real-time) multi-processor system design space exploration frameworks, such as Koski [8], Metropolis [9], Milan [10], or Sesame [11]. Also chip manufacturers provide cycle- or instruction-accurate simulators, so-called virtual platforms [12] [13] [14], to aid in the design and analysis of systems. Compared to simulation, the derivation of analytic models from synthesizable specifications has received much less attention. So far, work in this direction has been mainly done in the domain of (best-effort) networking systems where, for instance, layered queuing networks [15], Petri nets [16], or state-based formalisms [17] have been derived from synthesizable specifications, such as the ITU-T specification and definition language (SDL). In the domain of heterogeneous multi-processor systems, the UML-MAST tool can generate a model for holistic scheduling based on a system model described in the UML profile for schedulability, performance, and time (SPT) [18]. Since UML-SPT models are, in general, not synthesizable, the question how to automatically obtain the required model parameters remains open.

Model calibration is a well-known technique in many modeling and simulation domains. In the context of multi-processor system design and analysis, calibration has been applied for low-level models, such as cycle-accurate simulation [19], as well as for high-level models, such as trace-based simulation [20] (in the Sesame framework), task graphs [8] (in the Koski framework), or analytic bus models [21], for instance. For the calibration of high-level models, usually virtual platforms are used to obtain the required parameters. The same approach is taken in this work, but compared to the related approaches, which require the (manual) instrumentation of code sequences or the execution of dedicated benchmark applications, the calibration is completely automated and does not require any instrumentation of code. This is achieved by using a modular system specification consisting of an application, an architecture and a mapping specification, and by using high-level API function calls as “hooks” to track the execution of the tools from which the model parameters are obtained.

The calibration of compositional performance analysis models has only been addressed to a very limited extent. The reason is that compositional performance analysis has been mainly applied during early design space exploration where estimation rather than calibration is used to determine model parameters [2] [4]. By using automated calibration, however, compositional performance analysis can also be used in later design phases: The presented case study shows that the analysis results for a calibrated model match well with the quantities observed in a real system implementation.

III. Problem and Approach

In this paper, the design space exploration setting depicted in Fig. 1 is considered. Given are the specifications of a parallel application and a multi-processor architecture. The mapping, that is, the binding and scheduling, of the application onto the architecture is subject to optimization. The application, architecture, and mapping specification together form a synthesizable system specification that completely describes one specific system implementation. In this paper, synthesis refers to the code generation and compilation of the software binaries for the different processors of the architecture. Design space exploration in the described setting aims at optimizing certain system properties by varying the binding of the application elements to computation and communication resources, and the scheduling/arbitration policies and parameters for each computation and communication resource.

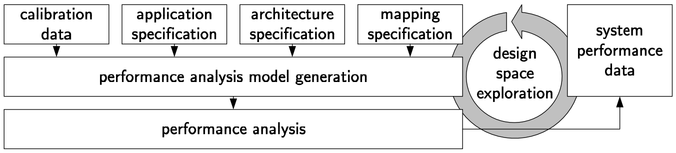

Fig. 1. Design space exploration using compositional performance analysis.

To be able to perform this exploration, a suitable method for performance analysis is required. A viable approach is to use compositional performance analysis. The reason is that compositional performance analysis is based on system abstractions that allow modeling different system implementations by combining pre-characterized analysis components, resulting in two key advantages: First, the abstractions are chosen in a way such that components need to be characterized only once and independent of other components. Second, if suitable parameters are chosen, the components can be treated as “black boxes” during analysis. An example for a component that possesses these properties is a component that models the execution of a piece of software on a processor by just considering its best-case execution time (BCET) and the worst-case execution time (WCET). These properties are in contrast to other methods, such as simulation-based methods or holistic techniques, where subsystems usually cannot be characterized or analyzed in isolation. This incurs a higher effort for characterization and/or analysis of systems, thereby slowing down the design space exploration cycle.

The specific problem considered in this paper is how a compositional performance analysis model can be generated and calibrated based on the synthesizable system specification. In particular, modular performance analysis (MPA) [1] is considered which is a framework for best-case/worst-case compositional performance analysis. MPA allows to derive bounds on system properties, making it suitable for the analysis of real-time systems. An implementation of MPA is freely available as a Matlab toolbox [22]. Due to the same basic abstractions underlying different methods for compositional performance analysis, the presented approach could also be implemented based on SymTA/S [2] or MAST [18], for instance.

IV. Problem Formalization

Like other analysis methods, best-case/worst-case compositional performance analysis methods are not equally well suited for analyzing different kinds of systems. In this paper, we therefore restrict ourselves to a certain class of systems which is formally described in this section. In particular, systems are considered where an application adhering to the synchronous data flow (SDF) [23] model of computation is executed on a distributed memory architecture. Note that the modeling scope of compositional performance analysis methods encompasses more general models of computation, such as Kahn process networks [24], and other computer architectures. Extensions of the described approach to model those systems incur additional modeling and calibration effort that goes beyond the scope of this paper. In the second part of this section, a formal description of MPA models is given.

A. System Specification

In general, a multi-processor system can be completely specified by an application, architecture, and mapping specification, as shown in Fig. 1. Following this general scheme, a formal specification of the class of considered systems is given in this section. In Fig. 2, an example system is shown to illustrate the used notation.

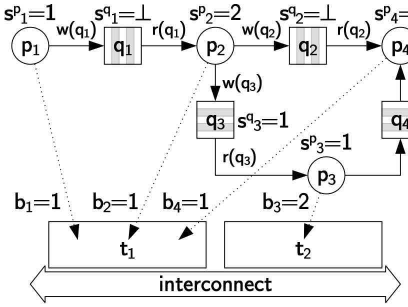

Fig. 2. Formal system specification.

Application. In this work, systems are considered where the application can be represented as an SDF graph. Due to the semantics of the SDF model of computation, the function of each single process is independent of a particular timing (as long as the execution order is correct). In addition, the SDF model of computation separates computation from communication. These properties have made SDF a popular model of computation for streaming applications executing on parallel, distributed systems. Apart from the implementation viewpoint, SDF has also advantages from the compositional performance analysis viewpoint. During model calibration, one can leverage the fact that, in the first instance, the execution time of each process does not depend on other processes (see Section V). During model generation and analysis, one can leverage the modular structure of SDF applications and reflect it in the analysis model (see Section VI).

An SDF application can be represented as a directed connected graph $\mathcal{A} = (P, Q)$. Each vertex $p \in P$ represents one of the $|P|$ processes and each directed edge between two processes p_i and p_j represents a (directed) first-in first-out queue between two processes. In SDF, the number of tokens produced during one execution of a producer process can differ from the number of tokens required for one activation of a consumer process. We speak of a data rate transition in this case, and denote the number of tokens in a queue q_j required for the activation of a process by $r(q_j)$ and the number of tokens written to a queue q_k upon completion of a process by $w(q_k)$.

Architecture. The described approach targets message-oriented distributed memory architectures frequently found in multi-processor systems. This organization assumes multiple software stacks running on heterogeneous processor subsystems (tiles).

The set of tiles is denoted by $\mathcal{T}$. Each tile $t_k \in \mathcal{T}$ has its own local program and data memory such that it can, in principle, operate independently from the other subsystems. The communication between different tiles is generally made through message passing. At the interface between a tile and the interconnect, a direct memory access (DMA) controller is located that can buffer messages and transmit them to the destination according to the interconnect arbitration policy without requiring processing resources on the tile itself.

Clearly, the method proposed in this paper is best suited for architectures where tight bounds on parameters can be obtained. Especially for the BCET and WCET of a process, this is rather difficult for many single-processor architectures which integrate speculative components, such as caches, branch prediction units, and pipelines. In multi-processor systems, however, subsystems are often kept comparatively simple. In the Sony/Toshiba/IBM Cell BE, for instance, the eight synergistic processing elements do not have a cache. Similarly, the Atmel DIOPSIS 940 used for the case study integrates a DSP subsystem without a cache. This reduces the variation of the BCET/WCET for a process, and also the influence on the BCET/WCET of other processes running on the same processor.

Mapping. The mapping of the application onto the architecture consists of the binding and the scheduling information. The binding of processes to processors is represented by a vector $b = (b_1, \ldots, b_{|P|}) \in \{1, \ldots, |\mathcal{T}|\}^{|P|}$ where $b_i = k$ if process p_i is assigned to tile t_k. Depending on the the binding of the processes, the message queues between processes are implemented differently. If the processes reside on the same tile, the queue is implemented in the local data memory of the tile which can be directly accessed by the two processes in a mutual exclusive manner. If the processes reside on different tiles, however, the data need to be transferred over the bus using the DMA controller. In addition to the binding information, a vector $s = (s^p, s^q)$ specifies the scheduling parameters for each process and queue. Preemptive fixed priority (FP) and time division multiple access (TDMA) scheduling are considered. Depending on the used scheduling policy, $s^p = (s_1^p, \ldots, s_{|P|}^p)$ and $s^q = (s_1^q, \ldots, s_{|Q|}^q)$ denote either the priority or the slot assignment according to which a process or data transfer is scheduled. Other scheduling policies, such as earliest deadline first, first-come first-serve, or generalized processor sharing, could be used as well since they belong to the modeling scope of MPA.

B. Modular Performance Analysis Model

In MPA, systems are modeled by a network of analysis components. The MPA model shown in Fig. 3, for instance, represents the system in Fig. 2. An MPA model can be represented by a graph $\mathcal{M} = (C, D)$ where the set of vertices C consists of the components and the set of edges D represent event streams (abstracted by arrival curves α) and resource streams (abstracted by service curves β). An abstract component referred to as greedy processing component (GPC) is used to model the execution of a process on a single resource, see Fig. 3. Processes with multiple inputs can be modeled by using abstract components that model the activation scheme, such as AND, OR, or combinations thereof [25] [26]. In the context of SDF applications, processes are activated when data are available on all input queues which can be modeled by an abstract AND component (AAC). To model the communication of data in distributed systems, interconnects are modeled by processing resources and the transmission of data by GPCs.

Using the mathematical framework of real-time calculus [1], system models as the one shown in Fig. 3 can be analyzed. Local quantities, such as the delay or the queue size of a specific process, are derived by performing local operations on the arrival and service curves of the corresponding component. Based on the local results, global quantities, such as end-to-end delays, system throughput, or total memory requirements, can be computed, afterwards.

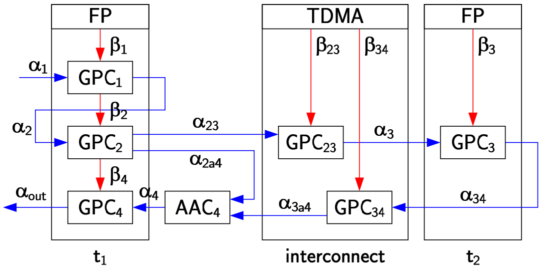

Fig. 3. MPA model of multi-processor system shown in Fig. 2.

C. Problem Statement

As already stated informally before, the considered problem is how an MPA model can be generated and calibrated based on a synthesizable system specification. Using the notation introduced above, the problem of model generation can be defined as follows: Given a system specification $\mathcal{S} = \{\mathcal{A}, \mathcal{T}, (b, s)\}$ consisting of an application specification $\mathcal{A}$, an architecture specification $\mathcal{T}$, and a mapping specification (b, s), generate the according MPA model $\mathcal{M} = (C, D)$. The problem of model calibration can be defined as follows: Given an implementation of the system $\mathcal{S}$ and the corresponding MPA model $\mathcal{M} = (C, D)$, determine the parameters of the components C and bounds on the input event and resource streams.

V. Analysis Model Calibration

The previous section introduced the basic abstractions used in MPA: GPCs, arrival curves, and service curves. Using these abstractions, the structure of the application, the data-flow through the application, the architectural resources, and the mapping of the application to the architecture can be modeled. To completely describe a system, however, a model needs to be parameterized accordingly. The parameters required in the context of MPA are summarized in Table I.

TABLE I
Model parameters required for MPA.

Entity	Parameter	Unit	Source
process p	best-/worst-case execution time BCET(p), WCET(p)	cycles/act.	low-level sim.
queue q	minimal/maximal token size $N_{\min}(q)$, $N_{\max}(q)$	bytes/access	functional sim.
	write rate, read rate $w(q)$, $r(q)$	1	functional sim.
processor	clock frequency	cycles/s	HW data-sheet
	best-/worst-case CPU utilization of run-time environment	cycles/s	low-level sim.
	best-/worst-case context switch time	cycles/s	low-level sim.
interconnect	throughput	bytes/s	HW data-sheet
environment	system input (arrival curve)	bytes/s	system spec.

Unfortunately, there is not a single source where all parameters could be obtained from. Hence, different sources need to be tapped:

- *Functional simulation:* Timing-independent parameters, such as the minimal and maximal token size can be obtained by functional simulation. The write and read rates of queues can be determined by purely functional simulation, as well.
- *Low-level simulation:* Architecture and mapping dependent parameters, such as the BCET/WCET of processes on processors, and the best-case and worst-case processor utilization of the run-time environment can be obtained by simulation using a virtual platform.
- *Hardware data sheets:* The performance of computation and communication resources in terms of clock frequency or throughput can be obtained from the architecture hardware data sheet.

- *System specification:* Besides functional requirements, the system specification usually contains also the non-functional requirements a system needs to fulfill. Information about the timing behavior of the system input can thus be obtained from the system specification.

The data available from hardware data sheets and the system specification, namely the clock frequency of the hardware resources and the timing behavior of the input, directly translate into service curves and arrival curves, respectively.

In the following, it is described how to obtain the remaining model parameters listed in Table I by simulation. Note that simulation is, in general, only suited to obtain estimates of best-case and worst-case quantities unless it can be guaranteed that the simulation actually exhibits the best-case and worst-case behavior. Nevertheless, for determining the best-case and worst-case estimate of a parameter, simulation is frequently the only practical possibility. For BCET/WCET of a process, for instance, formal methods and tools for determining safe bounds exist [27], but applying them is increasingly difficult due to complex processor architectures. Additionally, the models required for formal BCET/WCET analysis are only available for a few processors. We therefore use simulation — virtual platforms are widely available — for calibration, being aware that models calibrated using these data only qualify for the analysis of soft real-time systems.

A. Functional Simulation

To determine architecture-independent application parameters, functional simulation of the application is used. The term functional simulation is used to denote a program that executes a given SDF application on a standard PC. Fig. 4 shows a typical software architecture of a functional simulation based on the SystemC library: According to the application specification, a SystemC thread is instantiated for each SDF process. To implement the SDF queues, SystemC channels are used. The execution of the application is controlled by a simple data-driven scheduler.

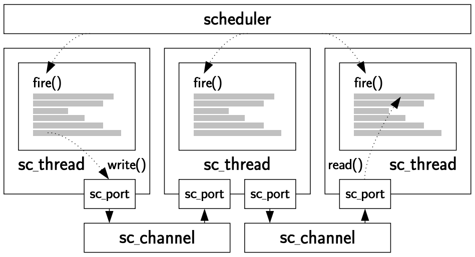

Fig. 4. Structure of functional simulation.

To obtain the architecture-independent parameters $r(q)$, $w(q)$, $N_{\max}(q)$, and $N_{\min}(q)$ for each queue q, one can simply use the $read()$ and $write()$ calls as hooks to monitor the execution. This monitoring does not require any application-specific code instrumentation because it can be implemented once in the generic $read()$ and $write()$ methods.

B. Low-Level Simulation

Low-level simulation on a virtual platform is used to determine the architecture-dependent parameters, namely the BCET/WCET of each process and the CPU resources consumed by the run-time environment (operating system and hardware abstraction layer). The approach to obtain these parameters is similar to the one used for functional simulation in the sense that generic routines can be monitored, eliminating the need for code instrumentation. This is possible because the structure of the software stack running on a real platform is similar to the one shown in Fig. 4. In a real-time operating system compliant to the POSIX 1003.1b standard, for instance, operating system threads and messages queues would be used to implement the processes and queues of the SDF application.

The BCET/WCET of each process is used to parameterize the corresponding GPC in the MPA model. The run-time environment overhead and the maximum context switch time are reflected in the service curve of a resource. In Fig. 7 in Section VII, for instance, one can see that the run-time overhead on the RISC processor has been bound to 5% and the context switch time to 8500 processor cycles.

Finally, note that the monitoring itself should not affect the observed quantities. In the implementation used for the case study, the CoWare Virtual Platform Analyzer [12] is used. This virtual platform features a Tcl-scripting interface that allows non-intrusive tracking of the execution of the hardware platform.

C. Calibration for Design Space Exploration

As just shown, the parameters for a single analysis model can be obtained by carrying out a functional and a low-level simulation of the corresponding system. During mapping design space exploration, however, this approach is prohibitively slow, mainly due to the long run-time of the low-level simulation (see Table III in Section VII).

A viable strategy is to collect all parameters before the design space exploration, however. The purely functional parameters are independent of the mapping, so functional simulation needs to be executed only once for a given application. The architecture- and mapping-dependent parameters (BCET/WCET of processes, run-time overhead, context switch times) can be obtained by executing a set of "calibration mappings" on the virtual simulation platform. After executing the set of calibration mappings, for each process a table of BCETs/WCETs similar to the one depicted in Fig. 5 exists. Subsequently, during design space exploration the corresponding parameters can be looked up in this table.

VI. Analysis Model Generation

Even when all data are available, generating an analysis model in an automatic manner is a nontrivial task: On the one hand, there is not a simple 1:1 relationship between the elements in the system specification and the components and their interaction in the analysis model. On the other hand, creating an actual implementation of the analysis model using a specific analysis framework requires the consideration of further technicalities.

To tackle this problem, the generation of an analysis model is split up into two steps, as shown in Fig. 5. In the first step, an analysis model (in XML format) is generated based on the system specification which consists of a network of processing components connected by event streams and the mapping information of the components. In this model, just the abstractions of processing components and event streams are used which are common to compositional performance analysis frameworks, such as MPA and SymTA/S. In the second step, the model is refined using the framework-specific abstractions and the according code is generated. In the case of MPA, this incurs the abstraction of event streams by arrival curves, the abstraction of processing components by GPCs, and the abstraction of component mapping by service curves. The subsequent code generation step targets the MPA Matlab toolbox [22].

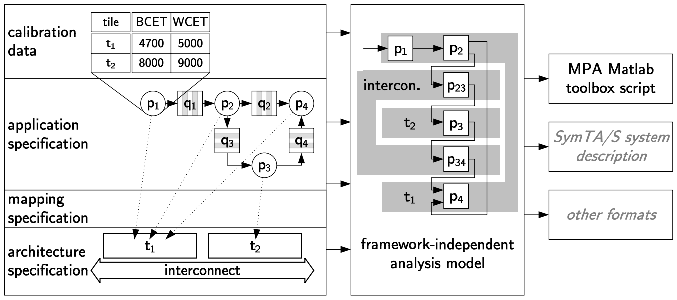

Fig. 5. MPA model generation from synthesizable system specification.

The advantage of having an intermediate analysis model is that both its front-end and its back-end can be extended independently. By introducing an additional code generation back-end for SymTA/S, for instance, the same model could be used as source to generate code for SymTA/S *and* MPA. Likewise, additional front-ends for transforming different kinds of system specifications into the framework-independent analysis model can be easily implemented.

The generation of the analysis model of a system works as follows: For each process in the SDF application and for each queue that is bound to the shared interconnect, a processing component is instantiated. Corresponding to the edges in the SDF graph, event streams are then instantiated between the corresponding processing components. In addition to the application, the model contains all the calibration data for the application, a description of the architecture and its calibration data, and the mapping information.

Subsequently, the MPA Matlab toolbox script can be generated: First, the system input and the resource capabilities are expressed by arrival curves and service curves. Second, for each processing component in the framework-independent analysis model, a GPC is instantiated. For GPCs with multiple inputs, an AAC is instantiated. Third, the resource streams are instantiated to reflect the binding and scheduling of GPCs. To model preemptive FP scheduling, GPCs are chained in the order of their priority: Only resources not consumed by a high-priority GPC are available to GPCs with lower priority. To model TDMA scheduling, dedicated service curves according to the TDMA cycle length and slot length are instantiated. Algorithm 1 shows the pseudo-code for this third step.

Algorithm 1 Modeling shared resources in MPA.

```
for all (tiles $t_i \in \mathcal{T}$) do                ▷ instantiate service curves
    if (scheduling policy($t_i$) == TDMA) then
        for all ($p_j \in P : b(p_j) == i$) do
            instantiate $\beta_j$(freq., slot len($s_j^p$), cycle len($s_j^p$))
        end for
    else if (scheduling policy($t_i$) == FP) then
        instantiate $\beta_{t_i}$(freq., overhead, switch time)
    end if
end for
for all (tiles $t_i \in \mathcal{T}$) do ▷ link service curves with GPCs
    for all ($p_j \in P : b(p_j) == i$) do
        if (scheduling policy($t_i$) == TDMA) then
            connect $\beta_j$ to GPC$_j$
        else if (scheduling policy($t_i$) == FP) then
            if ($s_j^p == 1$) then
                connect $\beta_{t_i}$ to GPC$_j$
            else
                instantiate $\beta_j = \beta'(\text{GPC}_k : s_k^p == s_j^p - 1)$
                connect $\beta_j$ to GPC$_j$
            end if
        end if
    end for
end for
```

Finally, a greedy algorithm is used to generate the Matlab script itself: Component by component, the corresponding Matlab commands are added to the script if all the input variables are available. To resolve cyclic dependencies that occur, for instance, if the priorities of t_1 in Fig. 3 were reversed, fixed point iteration is applied. The correctness of this approach is formally proved in [28].

VII. Case Study

In this section, the analysis model generation and calibration of an application executing on an Atmel DIOPSIS 940 is considered. The Atmel DIOPSIS 940 is a multi-processor system-on-chip integrating an ARM9 RISC processor and a mAgic DSP. For the reasons mentioned in Section IV, the cache of the ARM9 processor was disabled (the DSP does not have a cache). The application is an audio streaming application, namely wave field synthesis (WFS). By using an array of loudspeakers and audio beamforming techniques, WFS allows to reproduce an acoustic sound field whose perceived origin is not restricted to the position of physical loudspeaker boxes [29]. The SDF graph of the application for rendering a sound source using 16 loudspeakers is shown in Fig. 6. The *source* process reads a (mono) audio signal and the *control* process reads the beamforming coefficients from a digital interface. The signal processing takes place in the *compute* processes whereby each process synthesizes the signals for 8 channels. The computed signals are communicated to the *loudspeaker* process which drives the D/A converters. The processing is done in tokens of 32 samples, each represented as a single-precision 32-bit floating point number. The sampling rate is 48 kHz, leading to the data rates indicated in Fig. 6.

The design flow to implement the application has been presented in [30]. In particular, the model generation and calibration framework was implemented as an extension to the distributed operation layer [6] which is available for download [7]. For calibration, an instruction-accurate simulation of the system based on the CoWare Virtual Platform Analyzer [12] has been used.

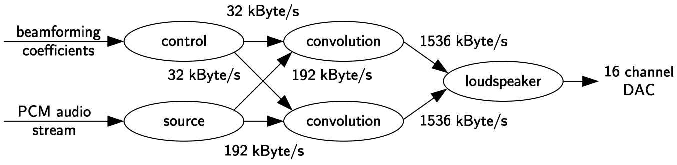

Fig. 6. SDF graph of wave field synthesis application.

First, we discuss the accuracy of the generated performance analysis model by considering a mapping of the WFS application onto the Atmel DIOPSIS 940. Fig. 7 shows the generated analysis model, including all the required parameters according to Table I obtained from specifications, data sheets, and by automated calibration.

Table II compares several bounds derived by MPA using this model to the actual (average-case behavior) quantities observed during simulation of the system. The differences between the derived bounds and the actually observed quantities are in a range that is typical for compositional performance analysis. Differences in the same range have been observed for several systems in [31], for instance. The main reason for these differences is that the analysis refers to best-case and worst-case behavior which usually does not occur in the system.

TABLE II
BOUNDS OBTAINED BY MPA COMPARED TO (AVERAGE) QUANTITIES OBSERVED IN SIMULATION.

Quantity	MPA	Simulation
total buffer requirements	$<$ 5272 Bytes	4760 Bytes
delay (source → loudspeaker)	$<$ 1.19 ms	0.83 ms
RISC utilization	31.4 % – 37.5 %	34.1 %
DSP utilization	50.2 % – 56.6 %	52.1 %

To improve the results, advanced techniques could be applied, such as a more detailed characterization of the workload of processes [32], or the analysis of execution demand correlations between processes [33]. Despite the extensions required to implement these techniques, the presented model generation and calibration approach would conceptually not change.

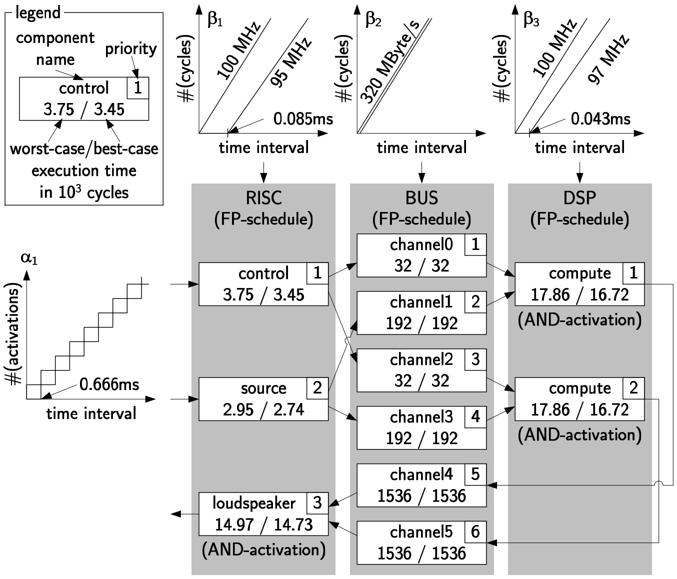

Fig. 7. MPA model resulting from automated generation and calibration.

Finally, Table III list the durations of the single steps to generate, calibrate, and evaluate the MPA model for one (arbitrary) mapping. Calibration is the step that takes by far the most time, whereas model generation and performance analysis is a matter of seconds. One can draw the following conclusions:

- Evaluating a system's performance using instruction-accurate simulation is obviously much slower compared to using a compositional performance analysis model. This justifies our proposed approach to use instruction-accurate simulation only for calibration, and perform design space exploration using an analytic model.
- Even though calibration is completely automated, it takes a considerable time. Thus, manual calibration at the same level of detail would be a difficult and time-consuming endeavor. Similar observations can be made with respect to model generation. This supports our argument that automated model generation and calibration are essential when compositional performance analysis is applied for analyzing real systems, and not just during early design phases.

TABLE III
DURATION OF ANALYSIS MODEL GENERATION AND CALIBRATION, MEASURED ON A 2 GHZ AMD ATHLON XP 2800+ MACHINE.

Step		Duration
model calibration (one-time effort)	functional simulation generation	35 s
	functional simulation	3 s
	synthesis (generation of binary)	176 s
	simulation on virtual platform	1300 s
	log-file analysis and back-annotation	50 s
model generation		0.5 s
performance analysis based on generated model		2 s

VIII. Conclusion

In this paper, the automated generation of analysis models for best-case/worst-case compositional performance analysis has been considered. By using the system specification that is used for system synthesis as the basis for the generated model and by calibrating the generated model using low-level simulation, an analysis model can be generated that faithfully models the real system. The gap between high-level analysis model and system implementation is thereby kept small, such that reasonably accurate best-case/worst-case bounds for system properties can be obtained. Methods based on compositional performance analysis that have been mainly used in early design phases can thus be applied in the entire design flow of heterogeneous multi-processor systems.

Acknowledgments

The work described in this paper has been carried out as part of the European project SHAPES. For more information visit http://www.shapes-p.org.

References

[1] S. Chakraborty, S. Künzli, and L. Thiele, "A General Framework for Analyzing System Properties in Platform-Based Embedded System Design," in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar. 2003, pp. 190–195.

[2] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis — The SymTA/S Approach," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, Mar. 2005.

[3] R. Racu, A. Hamann, and R. Ernst, "Sensitivity Analysis of Complex Embedded Real-Time Systems," *Real-Time Systems*, vol. 39, no. 1–3, pp. 31–72, Aug. 2008.

[4] A. Hamann, R. Racu, and R. Ernst, "Multi-Dimensional Robustness Optimization in Heterogeneous Distributed Embedded Systems," in *Proc. Real Time and Embedded Technology and Applications Symposium (RTAS)*, Bellevue, WA, United States, Apr. 2007, pp. 269–280.

[5] E. Wandeler and L. Thiele, "Optimal TDMA Time Slot and Cycle Length Allocation for Hard Real-Time Systems," in *Proc. 11th Asia and South Pacific Conf. on Design Automation (ASP-DAC)*, Yokohama, Japan, Jan. 2006, pp. 479–484.

[6] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, "Mapping Applications to Tiled Multiprocessor Embedded Systems," in *Int'l Conf. on Application of Concurrency to System Design (ACSD)*, Bratislava, Slovak Republic, Jul. 2007, pp. 29–40.

[7] I. Bacivarov, W. Haid, and K. Huang, "Distributed Operation Layer," http://www.tik.ee.ethz.ch/∼shapes.

[8] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, and T. D. Hämäläinen, "UML-Based Multiprocessor SoC Design Framework," *ACM Trans. Embedded Comp. Systems*, vol. 5, no. 2, pp. 281–320, May 2006.

[9] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: An Integrated Electronic System Design Environment," *Computer*, vol. 36, no. 4, pp. 45–52, Apr. 2003.

[10] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems Using Symbolic Search and Multi-Granular Simulation," in *Proc. of the Joint Conf. on Languages, Compilers and Tools for Embedded Systems (LCTES/SCOPES)*, Berlin, Germany, Jun. 2002, pp. 18–27.

[11] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and Automated Multiprocessor System Design, Programming, and Implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 3, pp. 542–555, Mar. 2008.

[12] "CoWare Virtual Platform Analyzer," http://www.coware.com.

[13] "Virtio," http://www.virtio.com.

[14] "Vast," http://www.vastsystems.com.

[15] D. Petriu, C. Shousha, and A. Jalnapurkar, "Architecture-Based Performance Analysis Applied to a Telecommunication System," *IEEE Trans. Softw. Eng.*, vol. 26, no. 11, pp. 1049–1065, Nov. 2000.

[16] M. Woodside, *From Annotated Software Designs (UML SPT/MARTE) to Model Formalisms*, ser. LNCS 4486. Berlin Heidelberg, Germany: Springer-Verlag, 2007, pp. 429–467.

[17] A. Viehl, T. Schönwald, O. Bringmann, and W. Rosenstiehl, "Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design," in *Proc. Design, Automation and Test in Europe (DATE)*, Munich, Germany, Mar. 2006, pp. 242–247.

[18] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano, "MAST: Modeling and Analysis Suite for Real Time Applications," in *Proc. Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, Jun. 2001, pp. 125–134.

[19] B. Black and J. P. Shen, "Calibration of Microprocessor Performance Models," *Computer*, vol. 31, no. 5, pp. 59–65, May 1998.

[20] A. D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of Abstract Performance Models for System System-Level Design Space Exploration," *Journal of Signal Processing Systems*, vol. 50, no. 2, pp. 99–114, Feb. 2008.

[21] M. Ruggiero, A. Guerri, D. Bertozzi, M. Milano, and L. Benini, "A Fast and Accurate Technique for Mapping Parallel Applications on Stream-Oriented MPSoC Platforms with Communication Awareness," *Int'l Journal of Parallel Programming*, vol. 36, no. 1, pp. 3–36, Feb. 2008.

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

[23] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[24] G. Kahn, "The Semantics of a Simple Language for Parallel Programming," in *Proc. IFIP Congress*, North Holland Publishing Co, 1974.

[25] W. Haid and L. Thiele, "Complex Task Activation Schemes in System Level Performance Analysis," in *Proc. Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*, Salzburg, Austria, Oct. 2007, pp. 173–178.

[26] M. Jersak, K. Richter, and R. Ernst, "Performance Analysis for Complex Embedded Systems," *Int'l Journal of Embedded Systems*, vol. 1, no. 1–2, pp. 33–49, 2005.

[27] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools," *ACM Trans. on Embedded Computing Systems*, vol. 7, no. 3, pp. 36:1–36:53, Apr. 2008.

[28] B. Jonsson, S. Perathoner, L. Thiele, and W. Yi, "Cyclic Dependencies in Modular Performance Analysis," in *Proc. Int'l Conf. on Embedded Software (EMSOFT)*, Atlanta, GA, USA, Oct. 2008, pp. 179–188.

[29] A. J. Berkhout, D. de Vries, and P. Vogel, "Acoustic Control by Wave Field Synthesis," *J. Acoust. Soc. Am.*, vol. 93, no. 5, pp. 2764–2778, May 1993.

[30] P. S. Paolucci, A. A. Jerraya, R. Leupers, L. Thiele, and P. Vicini, "SHAPES: A Tiled Scalable Software Hardware Architecture Platform for Embedded Systems," in *Proc. Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES/ISSS)*, Seoul, South Korea, Oct. 2006, pp. 167–172.

[31] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. G. Harbour, "Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems," in *Proc. Int'l Conf. on Embedded Software (EMSOFT)*, Salzburg, Austria, Oct. 2007, pp. 193–202.

[32] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative Characterization of Event Streams in Analysis of Hard Real-Time Applications," *Real-Time Systems*, vol. 29, no. 2, pp. 205–225, Mar. 2005.

[33] E. Wandeler and L. Thiele, "Characterizing Workload Correlations in Multi Processor Hard Real-Time Systems," in *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, San Francisco, CA, USA, Mar. 2005, pp. 46–55.