Towards Multi-application Workload Modeling in Sesame for System-level Design Space Exploration

Mark Thompson and Andy D. Pimentel

Computer Systems Architecture group, Informatics Institute
University of Amsterdam, The Netherlands
{thompson,andy}@science.uva.nl

Abstract. The Sesame modeling and simulation framework aims at early and thus efficient system-level design space exploration of embedded multimedia system architectures. So far, Sesame only supported performance evaluation when mapping a single application onto a (multi-processor) architecture at the time. But since modern multimedia embedded systems are increasingly multi-tasking, we need to address the modeling of effects of executing multiple applications concurrently in our system-level performance models. To this end, this paper conceptually describes two multi-application workload modeling techniques for the Sesame framework. One technique is based on the use of synthetic application workloads while the second technique deploys only real application workloads to model concurrent execution of applications. For illustrative purposes, we also present a preliminary case study in which a Motion-JPEG encoder application is executed concurrently with a small synthetic producer-consumer application.

1 Introduction

The increasing complexity of modern embedded systems has led to the emergence of system-level design [1]. A key ingredient of system-level design is the notion of high-level modeling and simulation in which the models allow for capturing the behavior of system components and their interactions at a high level of abstraction. As these high-level models minimize the modeling effort and are optimized for execution speed, they can be applied at the very early design stages to perform, for example, architectural design space exploration. Such early design space exploration is of eminent importance as early design choices heavily influence the success or failure of the final product.

In recent years, a fair number of system-level simulation-based exploration environments have been proposed, such as Metropolis [2], GRACE++ [3], Koski [4], and our own Sesame [5] framework. The Sesame modeling and simulation framework aims at efficient system-level design space exploration of embedded multimedia systems, allowing rapid performance evaluation of different architecture designs, application to architecture mappings, and hardware/software partitionings. Moreover, it does so at multiple levels of abstraction. Key to this flexibility is the separation of application and architecture models, together with an explicit mapping step to map an application model onto an architecture model.

So far, Sesame has only supported the mapping of a single application onto an architecture model at the time. But since modern multimedia embedded systems are in-

creasingly multi-tasking, we need to address the modeling of effects of executing multiple applications concurrently in our system-level architecture models. To this end, this paper presents two multi-application workload modeling techniques. One technique is based on the use of synthetic application workloads while the second technique deploys only real application workloads to model concurrent execution of applications. The presented techniques are currently being implemented in our Sesame framework. This implies that this paper mostly discusses concepts while detailed results will be published in a follow-up paper. However, for illustration purposes, we do present a preliminary case study in which a Motion-JPEG encoder application is executed concurrently with a small synthetic producer-consumer application.

The remainder of the paper is organized as follows. The next section provides an introduction to the Sesame modeling and simulation framework. Section 3 presents the two proposed multi-application workload modeling techniques for Sesame. This section also contains a discussion on how these workload modeling techniques can be used for the modeling of reactive application behavior. In Section 4, we present a small case study in which we model the concurrent execution of two applications. Section 5 describes related work, after which Section 6 concludes the paper.

2 Sesame

The Sesame modeling and simulation environment [5], illustrated in Figure 1, addresses the performance analysis of embedded multimedia system architectures. To this end, it recognizes separate application and architecture models, where an application model describes the functional behavior of an application and the architecture model defines architecture resources and captures their performance constraints. After explicitly mapping an application model onto an architecture model, they are co-simulated via trace-driven simulation. This allows for evaluation of the system performance of a particular application, mapping, and underlying architecture. Such an explicit separation between application and architecture opens up many possibilities for model re-use. For example, a single application model can be used to exercise different hardware/software partitionings or can be mapped onto a range of different architecture models.

For application modeling, Sesame uses the Kahn Process Network (KPN) model of computation [6], which nicely fits the targeted multimedia application domain [7]. KPNs are structured as a network of concurrent communicating processes, connected via unbounded FIFO channels. Reading from these channels is done in a blocking manner, while writing is non-blocking. The computational behavior of an application is captured by instrumenting the code of each Kahn process with annotations that describe the application's computational actions. The reading from and writing to Kahn channels represent the communication behavior of a process within the application model. By executing the Kahn model, each process records its computational and communication actions to generate its own trace of *application events*, which is necessary for driving an architecture model. There are three types of application events: the communication events READ and WRITE and the computational event EX(ECUTE). These application events typically are coarse grained, such as EX(*DCT*) or READ(*channel_id,pixel-block*).

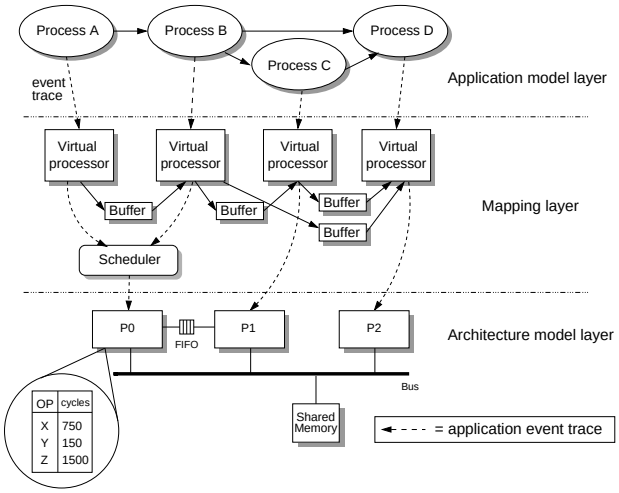

Fig. 1. The three layers in Sesame's modeling and simulation infrastructure.

An architecture model simulates the performance consequences of the computation and communication events generated by an application model. To this end, each architecture model component is parameterized with a table of operation latencies (illustrated for Processor 0 in Figure 1). The table entries could, for example, specify the latency of an EX(*DCT*) event, or the latency of a memory access in the case of a memory component.

To map Kahn processes from an application model onto architecture model components and to support the scheduling of application events when multiple Kahn processes are mapped onto a single architecture component (e.g., a programmable processor), Sesame provides an intermediate *mapping layer*. This layer consists of virtual processor components and FIFO buffers for communication between the virtual processors. There is a one-to-one relationship between the Kahn processes in the application model and the virtual processors in the mapping layer. This is also true for the Kahn channels and the FIFO buffers in the mapping layer, except for the fact that the latter are limited in size. Their size is parameterized and dependent on the modeled architecture. A virtual processor in the mapping layer reads in an application trace from a Kahn process and dispatches the events to a processing component in the architecture model. Communication channels – i.e., the buffers in the mapping layer – are also mapped onto the architecture model. In Figure 1, for example, buffers can be placed in shared memory or can use the point-to-point FIFO channel between processors 0 and 1. Accordingly, the architecture model accounts for modeling the communication (and contention) behavior at transaction level, including arbitration, transfer latencies and so on.

The mechanism used to dispatch application events from a virtual processor to an architecture model component guarantees deadlock-free scheduling of the application events from different event traces [5]. In this mechanism, EX(ecute) events are always immediately dispatched by a virtual processor to the architecture model component

that models their timing consequences. Scheduler components in the mapping layer (see Figure 1) allow for scheduling the dispatched application events from different virtual processors that are destined for the same (shared) architecture model resource. Communication events, however, are not immediately dispatched to the underlying architecture model. Instead, a virtual processor that receives a communication event first consults the appropriate buffer at the mapping layer to check whether or not the communication is safe to take place so that no deadlock can occur. Only if it is found to be safe (i.e., for read events the data should be available and for write events there should be room in the target buffer), then communication events may be dispatched. As long as a communication event cannot be dispatched, the virtual processor blocks. This is possible because the mapping layer executes in the same simulation as the architecture model. Therefore, both the mapping layer and the architecture model share the same simulation-time domain. This also implies that each time a virtual processor dispatches an application event (either computation or communication) to a component in the architecture model, the virtual processor is blocked in simulated time until the event's latency has been simulated by the architecture model.

Essentially, the mapping layer can be considered as an *abstract RTOS model*, in which the virtual processors are abstract representations of application processes that are executing, ready for execution or blocked for communication. Here, the scheduler components in the mapping layer provide the scheduling functionality of the RTOS.

3 Multi-application workload modeling

As mentioned before, Sesame has up to now only supported the mapping of a single application onto an architecture model at the time. Modern multimedia embedded systems are however increasingly multi-tasking. Therefore, we need to address the modeling of effects of executing multiple applications concurrently in our system-level architecture models. To this end, we propose two multi-application workload modeling techniques. One technique, which we will discuss first, is based on the use of synthetic application workloads while the second technique deploys only real application workloads to model concurrent execution of applications.

3.1 Synthetic multi-application workload modeling

Multi-application modeling using synthetic application workloads is illustrated in Figure 2. Note that the FIFO buffers between virtual processors are not depicted in Figure 2 for the sake of simplicity. On the left-hand side, a Sesame system-level model with a single, primary application is shown. The three processes in this application are mapped onto two processing cores (P0 and P1) in the underlying architecture. Since processes A and B are mapped onto the same resource, a scheduler named Local-Scheduler (or L-Scheduler) is used for scheduling the workloads (i.e., application events) from both processes. However, a second level of scheduling hierarchy is added by introducing so-called Global-Schedulers (or G-Schedulers). These global schedulers are basically equivalent to local schedulers in terms of functionality but instead of intra-application events they schedule application events from different applications. Evidently, the local

and global schedulers can also deploy different scheduling policies. When, for example, the interleaving of processes inside an application is statically determined at compile time, the local scheduler can model this by 'merging' the events from the event traces according to this given static schedule. At the same time, the global scheduler can schedule application events from different applications in a dynamic fashion based on, for example, time slices, priorities, or a combination of these two. Here, we would like to note that although the schedulers support preemptive scheduling, this can only be done at the granularity of application events. The simulation of a single application event is atomic and thus cannot be preemted in Sesame. Furthermore, we currently do not model any overheads caused by the context switching itself (e.g., OS overhead, cache misses, etc.). This is considered as future work.

In synthetic multi-application modeling, the application events external to the primary application (see Figure 2) are generated by a stochastic event generator. Hence, this event generator mimics the concurrent execution of one or more application(s) besides the primary application. Based on a stochastic application description, which will be discussed later on, the application generator generates traces of EX(ecute), READ and WRITE application events and issues these event traces to special virtual processors, indicated by VP_S in Figure 2. Multiple instances of these event generators, each with their own stochastic application description, can be used to model concurrent execution of more than two applications.

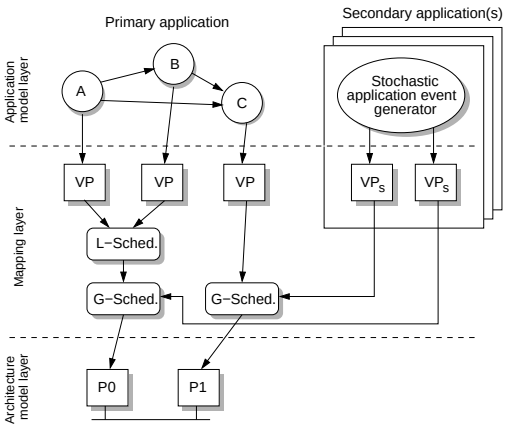

Fig. 2. Multi-application modeling using synthetic application workloads.

The virtual processors (VP_S) used for the trace events from the stochastic event generator are special in the sense that they, unlike normal virtual processors, are not connected to each other according to the application topology (see Section 2). Rather than explicitly modeling communication synchronizations, a VP_S models synchronization behavior stochastically. To illustrate the interactions between the event generator, a VP_S and a global scheduler of a system-level model, consider Figure 3. The figure shows these interactions in the case an "EX(A) , EX(B) , READ , WRITE " event sequence is generated by the event generator. At (simulation) time t_0, the EX(A) event is consumed by the VP_S. The VP_S immediately forwards this event to the global scheduler it is connected to, and waits for an acknowledgment from the scheduler. After the EX(A) event has been scheduled for execution on the architectural resource (taking T(sched) time units) and the actual execution (taking T(A) time units), control is returned to the VP_S by sending it an acknowledgment. Hereafter, the VP_S can consume another application event again. In the case of the example in Figure 3, the VP_S

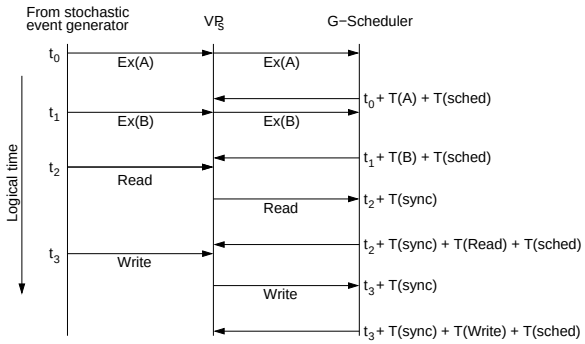

Fig. 3. Interaction between Virtual Processor (VP_S) and G(lobal)-Scheduler in synthetic multi-application modeling.

now consumes the EX(B) event which is handled in an identical fashion as the EX(A) event. However, VP_S handles the READ and WRITE events, which are consumed at times t_2 and t_3 respectively, in a slightly different way. Instead of directly forwarding these events to the global scheduler, like is done with EX events, VP_S now first models a synchronization latency. This latency refers to the time the read and write transactions need to wait for data or room in the buffer from/to which is read/written. The synchronization latency, indicated by T(sync) in Figure 3, is a stochastic parameter of VP_S, as discussed below.

Table 1. Parameters for the synthetic application workload generation.

Stochastic event generator parameter	Description
A_{Ex}	Set of possible Ex(ecute) application events
P_{Ex_i}, with $\sum_{i \in A_{Ex}} P_{Ex_i} = 1$	Probabilities of the different events in A_{Ex}
r_{comp}:r_{comm}	Computation to communication ratio
r_{read}:r_{write}	Read to write ratio
M	Set of possible message sizes
P_{M_i}, with $\sum_{i \in M} P_{M_i} = 1$	Probabilities of the different message sizes
NP	Number of communication ports
P_{port_i}, with $\sum_{i=0}^{NP} P_{port_i} = 1$	Probabilities of the different port usages
VP_S parameter	Description
$Sync_{Read}$	Mean synchronization latency for reads
σ_{Read}	Standard deviation of read latencies
$Sync_{Write}$	Mean synchronization latency for writes
σ_{Write}	Standard deviation of write latencies

Table 1 lists the parameters used by the stochastic event generator as well as a VP_S. These parameters can be specified both globally – describing the behavior for all traces (for the event generator) or ports (for a VP_S) – and on a per-trace/per-port basis. Descriptions on a per-trace/per-port basis overrule global descriptions, in the case there is an overlap of both types of descriptions. The parameter A_{Ex} specifies the set of possible EX events that can be generated. For example, $A_{Ex} = \{DCT, VLE\}$ specifies that EX(DCT) and EX(VLE) events can be generated. P_{Ex_i} describe the probabilities of the events in A_{Ex}. The ratio's r_{comp}:r_{comm} and r_{read}:r_{write} specify the computation to communication ratio and read to write ratio, respectively. So, for example, by increasing the r_{comp}:r_{comm} ratio, the application behavior can be made more computationally or communication intensive. The parameter M specifies the set of possible message sizes that can be used in communications. In multimedia applications, application data is often communicated in fixed data chunks (e.g. pixel blocks) from one application phase to the other. P_{M_i} specify the probabilities of the different message sizes. NP denotes the number of communication ports for which read and write transactions can be generated. P_{port_i} are the probabilities of the different port usages. Again, all of the above parameters can be specified globally (valid for all event traces) or on a per-trace basis.

The VP_S parameters $Sync_{Read}$ and $Sync_{Write}$ specify the mean synchronization latency for read and write transactions, respectively. σ_{Read} and σ_{Write} contain the standard deviations of the two aforementioned means. By default, a VP_S uses an Erlang distribution to determine synchronization latencies. These VP_S parameters can again be specified globally (valid for all communication ports of a VP_S) or on a per-port basis.

3.2 Realistic multi-application workload modeling

In our second multi-application workload modeling technique, we realistically model the concurrent execution of multiple applications. That is, multiple Kahn application models are actually executed concurrently, as shown in Figure 4, and produce realistic

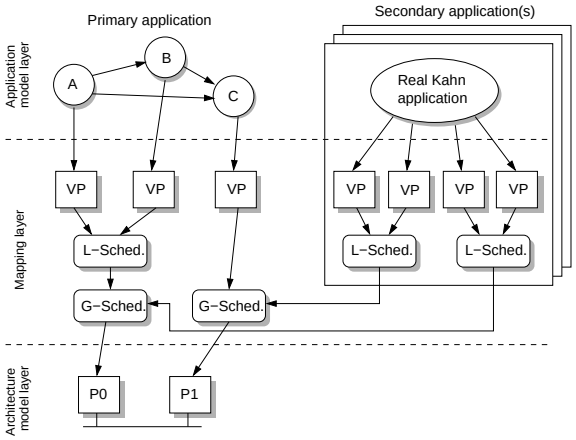

Fig. 4. Multi-application modeling using realistic application workloads.

event traces that are again scheduled on the underlying architectural resources using the global schedulers. In contrast to synthetic workload modeling, the secondary KPNs use normal virtual processors in the mapping layer. Hence, synchronization behavior in the parallel applications is modeled explicitly for all participating KPN applications (i.e., there is no difference between primary and secondary applications). This implies that, when considering Figure 3, the T(sync) now refers to the actual synchronization times between application processes. Moreover, the secondary KPNs also require L-schedulers to 'merge' (i.e. schedule) event traces when multiple application tasks are mapped onto a single architecture resource. Naturally, the policies of the L-schedulers can vary between the different KPN applications taking part in the system simulation. When considering Figure 3, we now have T(sched) = T(L-sched) + T(G-sched) for all participating KPNs.

3.3 Modeling reactive behavior

Because of its deterministic behavior, the KPN model of computation is relatively unsuited for modeling reactive application behavior, such as the occurrence of interrupts (e.g., a user presses a button on the TV's remote control after which teletext is started as a picture-in-picture application on the screen). Several researchers have proposed extensions to the KPN model of computation to resolve this [7–9]. Our two multi-application workload modeling techniques support the modeling of reactive behavior *between applications*, which could each be specified as a regular KPN. This can be achieved in a transparent manner by adding a 'SLEEP(N)' application event, which basically indicates that an application process is not active during a period of N time units. More specifically, a SLEEP event causes a virtual processor to sleep (i.e. block in virtual time) for the specified period. This event would not be simulated by the underlying architecture model. Evidently, the SLEEP events provide the opportunity to freeze the issuing of application events for a while, which basically mimics sporadic or periodic execution behavior of applications. To give an example in the case of synthetic multi-application modeling, the (stochastic) application event generator could model periods of inactivity (i.e., generating SLEEP events for all application processes) alternated with periods of application activity (i.e., generating EX, READ , and WRITE events). Clearly, this approach would allow us to assess a variety of different scenarios or use cases [10]. However, further research is needed to gain more insight about the qualitative and quantitative aspects of this modeling mechanism.

4 A preliminary case study

For illustrative purposes, we performed a small experiment using the multi-application workload modeling support that has already been realized in Sesame. More specifically, we modeled two Kahn applications that execute concurrently. The first (and primary) application is a Motion-JPEG (M-JPEG) encoder, and the other one is a synthetic 'producer-consumer' application transferring data from producer to consumer. The M-JPEG application encodes 8 consecutive 128x128 resolution frames, while the

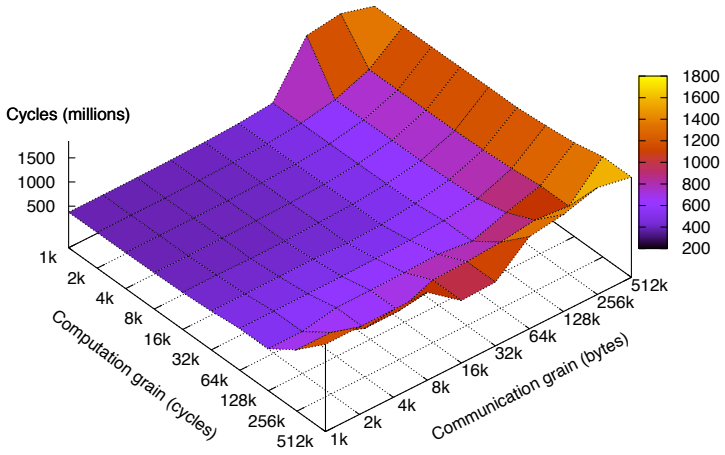

Fig. 5. Estimated execution times of concurrent execution of M-JPEG and producer-consumer applications. The latter is parameterized in both computation and communication grainsize.

producer-consumer application is parameterizable in both computational and communication load. That is, the producer iteratively models a parameterizable computing latency after which it sends a parameterizable chunk of data to the consumer. In our system-level model, both applications are mapped onto a multi-processor SoC, containing 4 processors with distributed memory and connected through a crossbar switch. We applied a simple round-robin policy for scheduling tasks from both applications at the G-schedulers (see Section 3.2).

Figure 5 shows the estimated system-level execution times (combined for both applications) when varying the computation and communication grainsizes of the producer-consumer application. As can be seen from Figure 5, the results show a quite predictable behavior, which helps to gain trust in our multi-application modeling method. That is, the system performance is only marginally affected for small computation and communication grains of the producer-consumer application. But after a certain threshold, the producer-consumer application starts to dominate the system performance (computation-wise, communication-wise, or both). As a next step, we plan to actually validate these results using the ESPAM system-level synthesis framework [11], which would allow us to compare our simulation results against an actual system implementation.

5 Related work

The modeling of (parallel) workloads for the purpose of performance analysis is a well-established research domain, both in terms of realistic and synthetic workload modeling (see e.g. [12–14]). A recent focus area is, for example, statistical simulation for micro-architectural evaluation [15]. In this technique, a stochastic program description, which is a collection of distributions of important program characteristics derived from execu-

tion profiles, is used to generate synthetic instruction traces. These synthetic traces are subsequently used in trace-driven processor and/or memory-hierarchy simulations. Another area in which synthetic workload modeling has recently received a lot of attention is network workload modeling for network-on-chip simulations [16–18].

In [19, 20], multimedia application workloads are described and characterized analytically using so-called variability characterization curves (VCCs) for system-level performance analysis of multi-processor systems-on-chip. These VCCs allow for capturing the high degree of variability in execution requirements that is often present in multimedia applications.

A fair number of research efforts addressed the high-level modeling of a RTOS to be used in system-level models for early design space exploration [21–23]. Rather than focusing on how to model multi-application workloads, these efforts mainly address abstract modeling of RTOS functionality, efficient simulation of this functionality, and refinement of these abstract RTOS models towards the implementation level.

6 Conclusions

In this paper, we addressed the extension of our Sesame modeling and simulation framework to support the modeling of multi-tasking between applications for the purpose of system-level performance analysis. To this end, we proposed two mechanisms for modeling multi-application workload behavior: one based on synthetic workload modeling and the other using only real application workloads. In addition, we indicated how reactive behavior at application granularity could be modeled. All presented methods are currently being implemented in Sesame. Using a small preliminary case study, however, we were already able to show an example of multi-application workload modeling using two applications. Future work needs to study the scope of application scenarios or use cases that can be modeled with these techniques. For example, it should be investigated to what extent the synthetic workload modeling technique allows for capturing the variability in execution requirements that is typically present in multimedia applications. Possibly, this technique could be extended with variability characterization curves such as proposed in [19, 20].

References

1. Keutzer, K. et al.: System level design: Orthogonalization of concerns and platform-based design. IEEE Trans. on CAD of Integrated Circuits and Systems **19** (2000)
2. Balarin, F. et al.: Metropolis: An integrated electronic system design environment. IEEE Computer **36** (2003)
3. Kogel, T. et al.: Virtual architecture mapping: A SystemC based methodology for architectural exploration of system-on-chip designs. In: Proc. of the Int. workshop on Systems, Architectures, Modeling and Simulation (SAMOS). (2003) 138–148
4. Kangas, T. et al.: UML-based multi-processor SoC design framework. ACM Trans. on Embedded Computing Systems **5** (2006) 281–320
5. Pimentel, A.D., Erbas, C., Polstra, S.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans. on Computers **55** (2006) 99–112

6. Kahn, G.: The semantics of a simple language for parallel programming. In: Proc. of the IFIP Congress 74. (1974)
7. de Kock, E.A. et al.: Yapi: Application modeling for signal processing systems. In: Proc. of the Design Automation Conference (DAC). (2000) 402–405
8. Geilen, M., Basten, T.: Reactive process networks. In: Proc. of the 4th ACM International Conference on Embedded Software (EMSOFT). (2004) 137–146
9. van Dijk, H.W., Sips, H.J., Deprettere, E.F.: Context-aware process networks. In: Proc. of the Int. Conf. on Application-specific Systems, Architectures, and Processors (ASAP). (2003) 6–16
10. Gheorghita, S., Basten, T., Corporaal, H.: Application scenarios in streaming-oriented embedded system design. In: Proc. of the Int. Symposium in System-on-Chip. (2006)
11. Nikolov, H., Stefanov, T., Deprettere, E.: Multi-processor system design with ESPAM. In: Proc. of the Int. Conf. on HW/SW Codesign and System Synthesis (CODES-ISSS'06). (2006) 211–216
12. Kotsis, G.: A systematic approach for workload modeling for parallel processing systems. Parallel Computing **22** (1997) 1771–1787
13. Feitelson, D.: Workload modeling for performance evaluation. In Calzarossa, M.C., Tucci, S., eds.: Performance Evaluation of Complex Systems: Techniques and Tools. LNCS, Springer (2002) 114–141
14. Skadron, K., Martonosi, M., August, D.I., Hill, M.D., Lilja, D.J., Pai, V.S.: Challenges in computer architecture evaluation. Computer **36** (2003) 30–36
15. Eeckhout, L., Nussbaum, S., Smith, J.E., Bosschere, K.D.: Statistical simulation: Adding efficiency to the computer designer's toolbox. IEEE Micro **23** (2003) 26–38
16. Varatkar, G., Marculescu, R.: On-chip traffic modeling and synthesis for MPEG-2 video applications. IEEE Trans. on Very Large Scale Integration Systems **12** (2004) 108–119
17. Thid, R., Sander, I., Jantsch, A.: Flexible bus and NoC performance analysis with configurable synthetic workloads. In: Proc. of the Conference on Digital System Design. (2006) 681–688
18. Mahadevan, S., Angiolini, F., Storgaard, M., Olsen, R.G., Sparso, J., Madsen, J.: A network traffic generator model for fast network-on-chip simulation. In: Proc. of the Conference on Design, Automation and Test in Europe (DATE). (2005) 780–785
19. Liu, Y., Chakraborty, S., Ooi, W.T.: Approximate VCCs: a new characterization of multimedia workloads for system-level MpSoC design. In: Proc. of the conference on Design Automation (DAC). (2005) 248–253
20. Maxiaguine, A., Zhu, Y., Chakraborty, S., Wong, W.F.: Tuning SoC platforms for multimedia processing: identifying limits and tradeoffs. In: Proc. of the Int. conference on Hardware/software codesign and system synthesis (CODES-ISSS). (2004) 128–133
21. Gerstlauer, A., Yu, H., Gajski, D.D.: RTOS modeling for system level design. In: Proc. of the Conference on Design, Automation and Test in Europe (DATE). (2003) 10130
22. Hessel, F., da Rosa, V.M., Reis, I.M., Planner, R., Marcon, C.A.M., Susin, A.A.: Abstract RTOS modeling for embedded systems. In: Proc. of the 15th IEEE International Workshop on Rapid System Prototyping (RSP'04). (2004) 210–216
23. Lavagno, L., Passerone, C., Shah, V., Watanabe, Y.: A time slice based scheduler model for system level design. In: Proc. of the Conference on Design, Automation and Test in Europe (DATE). (2005) 378–383