A Co-simulation Framework for MPSoC Run-Time Behavior Analysis in Early System Design

Anastasia Stulova, Jianjiang Ceng, Weihua Sheng, Jeronimo Castrillon, Rainer Leupers
Institute for Integrated Signal Processing Systems
RWTH Aachen University, Germany
{stulova, ceng, sheng, castrill, leupers }@iss.rwth-aachen.de

Abstract

Embedded systems architectures are characterized by high heterogeneity and diversity in both, hardware and software components. These properties are also valid for the operating systems (OS) services. A broad usage of MPSoC platforms as a hardware template opens a new dimension in the design of OS due to availability of numerous processing resources which should be properly utilized. In this paper we present a virtual OS model and its integration into the simulation framework for the early MPSoC software design evaluation.

1. Introduction

Ad-hoc, application specific OS services are widely used in embedded MPSoCs, mainly because they can bring significant improvements to the run-time efficiency. In this context, we present a framework for analysis of ad-hoc scheduling policies already at early system design phases. The importance of scheduling at run-time is mainly due to: (i) its high workload overhead[6] and (ii) the influence on the overall system execution. In order to efficiently manage the system resources available on MPSoC platforms, customized OS services and optimized task implementations are needed.

In this paper we present a framework for analysing the impact that customized OS services have on the MPSoC platform efficiency. The framework is developed as a part of a simulation model known as *High Level Virtual Platform* (HVP) [1]. HVP is intended to support the early MPSoC software design, when many system parameters are not yet precisely defined.

The rest of this paper is organized as follows. An overview of the related work is given in Section 2. In Section 3 we describe the proposed OS model and its integration into the HVP simulation flow. Case studies are presented in Section 4. Section 5 concludes this paper and gives insights into the future work.

2. Related Work

Our framework provides an environment where scheduling behavior can be co-simulated with the abstraction of an MPSoC platform while realistic applications runs on top of it. Among existing frameworks intended to monitor run-time behavior via simulations, we found limitations in the following aspects:

(i) In most frameworks [8, 3, 11] simulation model is very detailed with respect to the underlying hardware architecture, which at that time must be fully defined;

(ii) Some existing simulators used in early system design (such as [2] and [12]) model a particular OS or a range of them, without giving the possibility of customization or parameterization;

(iii) To our knowledge, there is no framework which allows to unrestrictedly model both static tasks mapping and dynamic approach with the possibility of task migrations among cores;

(iv) In many frameworks [9, 5] applications and run-time management developments are organized over different flows. To exercise schedulers the designer is forced to create artificial testbenches including hardware platform and tasks on top of it. As a drawback it introduces large overhead for OS developers.

3. A Virtual OS Model

Our work contributes with the methodology to analyze different scheduling policies. The framework has been implemented as a part of the high-level MPSoC simulator HVP. More details about this tool are given in our publication [1]. The strength of this approach is that it helps to

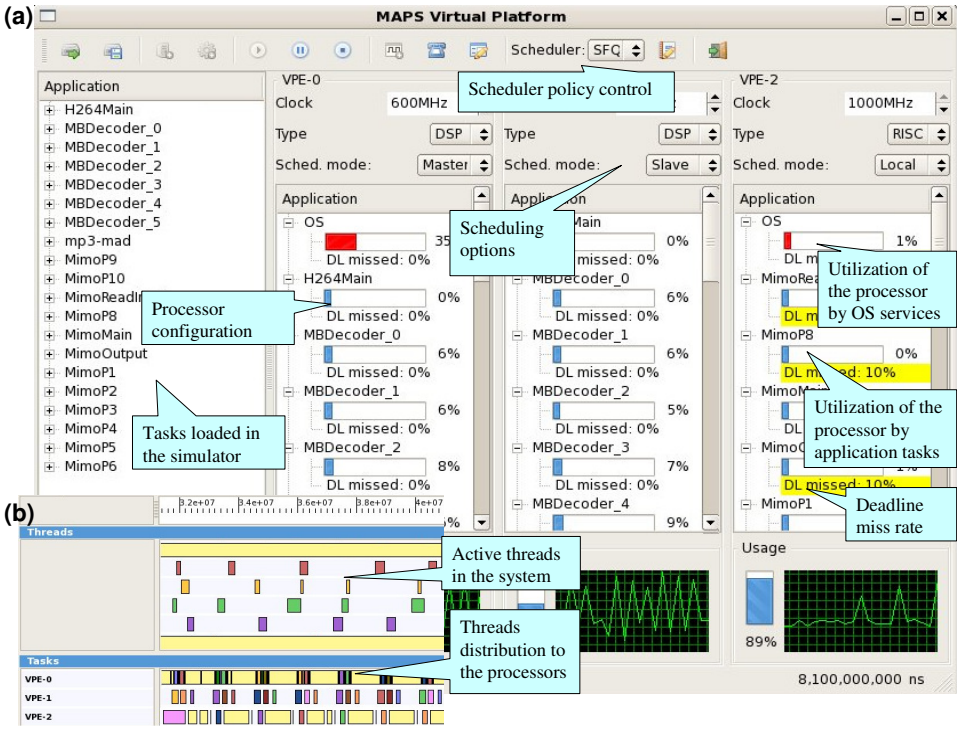

Figure 1. Scheduling analysis support: (a) displaying tasks and OS workload on processing resources via platform GUI; (b) scheduling traces visualization.

guide the software design process in very early stages, when many system parameters are not yet well defined. The application software development can be started much earlier than in the traditional flow and results are obtained faster.

In this paper we address the problem of the MPSoC runtime behavior evaluation, mainly dedicated to OS designers. In particular we introduced into the current HVP runtime management flow possibility to customize the scheduling policy and ability to express a broad range of different scheduling solutions while keeping generality of the tool. Scheduling behavior can be written in C or C++ (which are very familiar to embedded system designers). To be plugged into the simulation platform the scheduler must be compliant to the specified interfaces. The interfaces define a set of C functions and data structures.

Scheduling strategy is described by the *sched_update* function. Each call of this function returns the scheduling matrix *sched_matr* and the time slice vector *ts_vect*. Given that N_t is the number of tasks and N_v is the number of Virtual Processing Elements (VPEs) in the platform, *sched_matr* is a matrix of bits with dimension $N_t \times N_v$. Each bit set to 1 corresponds to task dispatching to specific VPE. In *ts_vect* time slice values for the next tasks activation can be specified. It is a vector of size N_t (one value for each task). Value 0 is used for non-time sliced tasks.

According to the introduced notation the scheduling function has the following format:

sched_update(sched_evnt) $\rightarrow$ *(sched_matr, ts_vect).*

The input parameter *sched_evnt* used by the function denotes an event in the simulator which might cause tasks rescheduling. Among these events are: task state transitions, end of time slice interval, parameter of VPE change. The source of the event determines its nature. The scheduling update function described above is invoked on every arrival of the scheduling event. It is described by a 4-tuple *(evnt_type, time_stamp, task, vpe)*, where the first value corresponds to the event nature. The second element is a time stamp of event occurrence. The last two parameters specify a source task and/or VPE.

Custom scheduling policies can be attached to the simulator by reimplementing the *sched_update()* function. Such implementations can be compiled to dynamic libraries using a host compiler and linked to the platform at run-time via graphical user interface (GUI) or a configuration file.

Various scheduling policies classified in [10] can be implemented using this generic interface. A platform user can specify preemptive and non-preemptive, with static mapping and migrative, generic and containing the real-time features scheduling schemes.

The scheduler has access to several internal data structures of the simulator via *task* and *vpe* elements of the *sched_evnt* entity. They contain a full description of different tasks and VPE characteristics (for example, a task name, a deadline, a period, a priority; a VPE name, a type and a clock frequency). This is essential as such parameters are evaluated by the scheduling policy in order to make decisions about tasks dispatching.

We model OS timing overhead as a combination of three components: a context switch time, a migration penalty and a scheduling time. Most of them can be parametrized by a constant value via GUI or a configuration file. The context switch represents the time consumed in *save* and *load* operations required to begin tasks execution and only included in the case swapping in the tasks execution occurs. The migration penalty corresponds to the additional time due to task context transmissions across different cores (only considered if a task is dispatched from one core to another). To model the time required to take scheduling decisions two approaches are proposed:

(i) The *time annotation* mechanism similar to the one presented in [7]. This way is more convenient if the existing custom scheduling solutions are to be plugged into the platform and some initial hardware templates are given. Therefore timing information is known or can be measured from available implementations;

(ii) Using the *source code instrumentation* as described in the HVP tool chain [1]. In this case the time for newly defined scheduling can be modeled because exact numbers about execution latencies can be hardly specified by the developer.

Our main motivation is to support the OS developer to make correct decisions about the scheduler design while providing a more realistic environment for the application developer to reason about application behavior at the early design stage. For these purposes various information is presented to the designer via a graphical workbench which enables to quickly investigate candidate solutions (Figure 1). For example for real-time analysis task deadline miss rates are available. To inspect OS overhead the utilization of processors by the scheduler can be monitored. Moreover, scheduling traces can be graphically followed. Every trace contains dynamic information about task migrations among different processors and latencies of their execution (including OS services). Knowledge of the OS latency is important for system performance monitoring and verification.

4. Case study

We performed a case study on a test bench with three applications: H.264 (video decoder), GSM (speech encoder), and AES (encryption). These applications are frequently used in the embedded multimedia and can potentially run concurrently on a single device. GSM and H.264 were parallelized and characterized by throughput constraints (as hard and soft real-time respectively). The applications set up is shown in Table 1. We let the scheduler handle such multi-applications scenario on a platform with three RISC processors running at 1 GHz clock speed. The monitored characteristics are the OS overhead, the violation of real-time constraints (the deadline miss rate), the available processor bandwidth for non real-time AES application and the average processing elements (PEs) utilization. To model the timing behavior of different scheduling implementations, the annotation mechanism was used and the time measurements of scheduling executions were performed on the Instruction Set Simulator.

Table 1. Applications scenario set up.

Application	Parallelized	RT req.	Throughput
GSM	6 tasks	Hard	8000 samples/s
H.264	8 tasks	Soft	40 ms/frame
AES	no	none	-

Some results are summarized in Table 2. In our experiment four scheduling schemes were explored. Initially we configured scheduler to run locally at each processor, at first with the Round-Robin (RR) policy and after with Earliest Deadline First (EDF). Both partitioned applications were mapped onto two VPEs and sequential AES was assigned to another VPE. After that we analyzed two centralized schedulers: EDF and hierarchical scheduling with different qualities of service for different applications (as presented in [4]). We configured this scheduling to give the highest priority to tasks of the GSM application. Between the remaining H264 and AES applications priority is given to the first one. Tasks belonging to the same application are always arbitrated according to the EDF policy. It can be seen from Table 2 that the centralized task management performs more effective PEs utilization compared to the static mapping assignment. As a result it provides higher bandwidth to the non-real time AES application. In these cases the maximum utilization (up to 100%) of PEs is limited by the OS latency. Moreover, hierarchical scheduling configured as described above provides the best service guarantee to the real-time constraint applications (the deadline miss rate decreases especially for the hard real-time GSM application) due to its high customization to the applications scenario.

Table 2. Statistics of the applications execution on the platform with different scheduling configurations.

Scheduling	Local all RR	Local all EDF	Centr. EDF	Centr. Hierarch.
GSM max. DL misses(%)	50	12	10	0
H.264 max. DL misses(%)	15	8	7	9
avrg. AES exec. bandwidth(%)	52	47	63	68
avrg. OS overhead(%)	10	35	27	17
avrg. PEs utilization(%)	51	53	79	80

5. Conclusions

In this paper we presented the abstract OS model and its integration into the simulation platform. Our approach unifies the applications and OS development for embedded MPSoC platforms. The OS model was designed to be generic enough to model a wide range of scheduling types. On top of that the retargetability properties were addressed, which help to easily configure OS scheduling and the timing model. In contrast to many similar frameworks our approach is more oriented on the tooling support. We provide various statistics to developers to speed up the work on OS and software development.

In the future we plan to evaluate the accuracy of our approach compared to the real hardware MPSoC platforms or very detailed cycle accurate simulation engines. Additionally, the timing model of tasks migration can be further refined as it is essential for the migrative scheduling design.

References

[1] J. Ceng, W. Sheng, J. Castrillon, A. Stulova, R. Leupers, G. Ascheid, and H. Meyr. A high-level virtual platform for early MPSoC software development. In *Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '09, pages 11–20, New York, NY, USA, 2009. ACM.

[2] T. Furukawa, S. Honda, H. Tomiyama, and H. Takada. A hardware/software cosimulator with RTOS supports for multiprocessor embedded systems. In *Proceedings of the 3rd international conference on Embedded Software and Systems*, ICESS '07, pages 283–294, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] P. Gerin, X. Guérin, and F. Pétrot. Efficient implementation of native software simulation for MPSoC. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 676–681, New York, NY, USA, 2008. ACM.

[4] P. Goyal, X. Guo, and H. M. Vin. Readings in multimedia computing and networking. pages 491–505, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[5] M. A. Hassan, K. Sakanushi, Y. Takeuchi, and M. Imai. Enabling RTOS simulation modeling in a system level design language. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 936–939, New York, NY, USA, 2005. ACM.

[6] T. Kamiuchi, H. Nakanishi, and K. Hayashi. Operating system structure model for real-time systems. In *Object-Oriented Real-Time Dependable Systems,1996. Proceedings of WORDS '96., Second Workshop on*, pages 120 –124, feb 1996.

[7] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A modular simulation framework for spatial and temporal task mapping onto multi-processor SoC platforms. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*, DATE '05, pages 876–881, Washington, DC, USA, 2005. IEEE Computer Society.

[8] M. Krause, D. Englert, O. Bringmann, and W. Rosenstiel. Combination of instruction set simulation and abstract RTOS model execution for fast and accurate target software evaluation. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, CODES+ISSS '08, pages 143–148, New York, NY, USA, 2008. ACM.

[9] R. Le Moigne, O. Pasquier, and J.-P. Calvez. A generic RTOS model for real-time systems simulation with systemc. In *Proceedings of the conference on Design, automation and test in Europe - Volume 3*, DATE '04, pages 30082–, Washington, DC, USA, 2004. IEEE Computer Society.

[10] V. Nollet, D. Verkest, and H. Corporaal. A safari through the MPSoC run-time management jungle. *J. Signal Process. Syst.*, 60(2):251–268, Aug. 2010.

[11] Y. Yi, D. Kim, and S. Ha. Fast and time-accurate cosimulation with OS scheduler modeling. *Autom. Embedded Syst*, 8:211–228, 2003.

[12] S. Yoo, I. Bacivarov, A. Bouchhima, Y. Paviot, and A. A. Jerraya. Building fast and accurate SW simulation models based on hardware abstraction layer and simulation environment abstraction layer. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, DATE '03, pages 10550–, Washington, DC, USA, 2003. IEEE Computer Society.