Resource Manager for Non-preemptive Heterogeneous Multiprocessor System-on-chip

Akash Kumar, Bart Mesman, Bart Theelen and Henk Corporaal
Eindhoven University of Technology
5600MB Eindhoven, The Netherlands
Email: {a.kumar, b.mesman, b.d.theelen, h.corporaal}@tue.nl

Ha Yajun
National University of Singapore
10 Kent Ridge Crescent, Singapore
Email: elehy@nus.edu.sg

Abstract

Increasingly more MPSoC platforms are being developed to meet the rising demands from concurrently executing applications. These systems are often heterogeneous with the use of dedicated IP blocks and application domain specific processors. While there is a host of research done to provide good performance guarantees and to analyze applications for preemptive uniprocessor systems, the field of heterogeneous, non-preemptive MPSoCs is a mostly unexplored territory. In this paper, we propose to use a resource manager (RM) to improve the resource utilization of these systems. The basic functionalities of such a component are introduced. A high-level simulation model of such a system is developed to study the performance of RM, and a case study is performed for a system running an H.263 and a JPEG decoder. The case study illustrates at what control granularity a resource manager can effectively regulate the progress of applications such that they meet their performance requirements.

1. Introduction

Current developments in set-top box products for media systems show a need for integrating a (potentially large) number of applications or functions in a single device. An increasing number of processors are being integrated into a single chip to build Multiprocessor Systems-on-chip (MPSoC). The heterogeneity of such a system increases with the use of specialized digital hardware, application domain processors and other IP (intellectual property) blocks on a single chip using complex networks. Almost all the major companies in consumer electronics have already released heterogeneous multiprocessors e.g. Intel IXP2850, Philips Nexperia, TI OMAP and ST Nomadik, to name a few.

Non-preemptive operating systems are preferred over preemptive scheduling for a number of reasons [1]. In many practical systems, properties of device hardware and software either make the preemption impossible or prohibitively expensive. For embedded systems in particular, non-preemptive scheduling algorithms are easier to implement than preemptive algorithms and have dramatically lower overhead at run-time [1]. It is therefore important to investigate non-preemptive heterogeneous MPSoCs.

A heterogeneous MPSoC places high demands on the arbitration of computational resources (among other resources). A need has therefore arisen for a middle-ware or OS-like component for the MPSoC, as observed in [2]. In this paper, we propose the use of a *Resource Manager (RM)* for non-preemptive heterogeneous MPSoCs. We state the basic functionalities expected from such a component. While most research only focusses on schedulability analysis, here we also discuss the required protocol needed to realize a working system. Further, we present a simulation model developed using the modeling language POOSL [3] to validate this protocol. POOSL is a very expressive modeling language with a small set of powerful primitives and completely formally defined semantics. It furthermore serves as a basis for performance analysis. This setup can be used to study various trade-offs in design of an MPSoC. In this paper, we use the setup to study the trade-off between control overhead of monitoring and budget enforcement on the one hand, and performance of applications on the other.

The remainder of this paper is organized as follows. Section 2 discusses the relevant research that has been done. In Section 3 we present an example to highlight the problem in combining multiple applications' resource utilization. We also show how the resource manager can be useful when desired performance is to be met in multiple applications. Section 4 discusses the setup of the simulation model, and goes into the details of RM and the protocol. The results of a case study done with H263 and JPEG decoder are discussed in Section 5. Section 6 ends the paper with some conclusions and direction for future work.

2. Related Work

For traditional systems, with a single general-purpose processor supporting pre-emption, the analysis of schedulability of task deadlines is well known [4] and widely used. Non-preemptive scheduling has received considerably less attention. It was shown by Jeffay et al. [1] and further strengthened by Cai and Kong [8] that the problem of determining whether a given periodic task system is non-preemptively feasible even upon a single processor is already intractable. Also, research on multiprocessor real-time scheduling has mainly focused on preemptive systems [9, 10]. Recently, more work has been done on non-preemptive scheduling for multiprocessors [5]. Al-

Properties	Liu *et* al [4] 1973	Jeffay *et* al [1] 1991	Baruah [5] 2006	Richter *et* al [6] 2003	Hoes [7] 2004	Our method
Multiprocessor	No	No	Yes	Yes	Yes	Yes
Heterogeneous	N. A.	N. A.	No	Yes	Yes	Yes
Non-preemptive	No	Yes	Yes	Yes	Yes	Yes
Non-Periodic support	No	Yes	No	Yes	Yes	Yes
Utilization	High	High	Low	Low	Low	High
Guarantee	Yes	Yes	Yes	Yes	Yes	No

Table 1. Summary of related work (Heterogeneous property is not applicable for uniprocessor schedulers)

ternative methods have been proposed for analyzing task performance and resource sharing. A formal approach to verification of MPSoC performance has been proposed in [6]. Use of real-time calculus for schedulability analysis was proposed in [11]. Computing worst-case waiting time taking resource contention into account for round-robin and TDMA (requires preemption) scheduling has also been analyzed [7]. However, potential disadvantages of these approaches are that the analysis can be very pessimistic. This is best illustrated by means of the example shown in Figure 1. The example shows 3 application SDF [12] graphs - A, B, and C, with 3 actors each. Actors T_i are mapped on processing node P_i where T_i refers to A_i, B_i and C_i for $i = 1, 2, 3$. Each actor takes 100 time units to execute as shown.

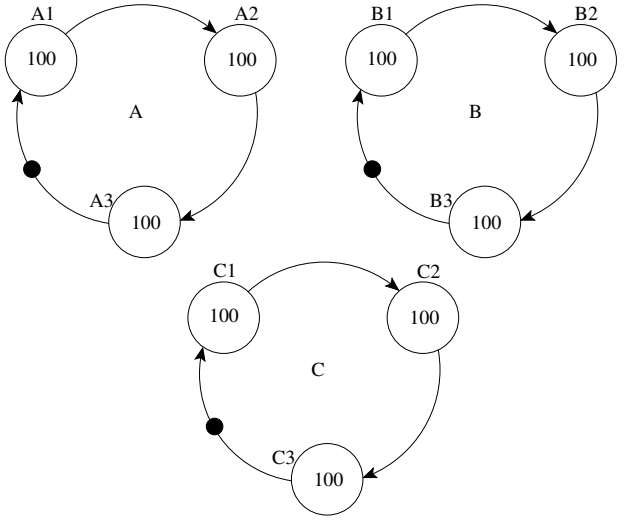

Figure 1. Example of a set of 3 application graphs.

Since 3 actors are mapped on the same node, an actor may need to wait when it is ready to be executed at a node. Modeling of worst case waiting time for round-robin according to [7] leads to a waiting time of 200 time units as shown in Figure 2. As can be seen, an extra node has been added for each 'real' node to depict the waiting time (WT_A_i). This suggests that each application will take 900 time units in the worst case to finish execution. A resource manager approach can nicely interleave the actors and each application will only require 300 time units, thereby achieving three times the performance guaranteed by analysis in [7].

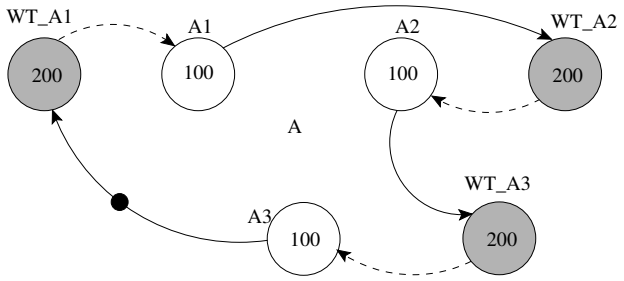

Figure 2. Modeling worst case waiting time for application A in Fig. 1.

Table 1 shows a comparison of various analysis techniques that have been presented so far in literature, and where our approach is different. As can be seen, all of the research done in multiprocessor domain provides low utilization guarantees. Our approach on the other hand aims at achieving high utilization by sacrificing hard guarantees.

3. Motivating Example

In this paper, we assume applications to be specified as a (Homogeneous) Synchronous Data Flow ((H)SDF) graph [12], where vertices indicate separate tasks (also called actors) of an application, and edges denote dependencies between them. SDF is widely used; it is very suitable to express concurrency in applications, and is therefore useful to analyze multiprocessor systems. It also allows deriving a static schedule and timing properties at design time. Two such SDF graphs are shown in Figure 3. Each graph has three actors scheduled to run on three different processors (mapping is same as in Fig 1). However, the flow of applications A and B is reversed with respect to each other to create more resource contention. The execution time of each actor of B is reduced to 99 to create a situation where A experiences a worst-case waiting time. If each application is running in isolation, the achieved period would be 300 and 297 time units for A and B respectively. Clearly, when the two are running together on the system, due to contention they may not achieve this performance. In fact, since every actor of B always finishes just before A, it always gets the desired resource and A is not able to achieve the required

performance; while B on the other hand achieves a better performance than necessary. The corresponding schedule is shown in Figure 4.

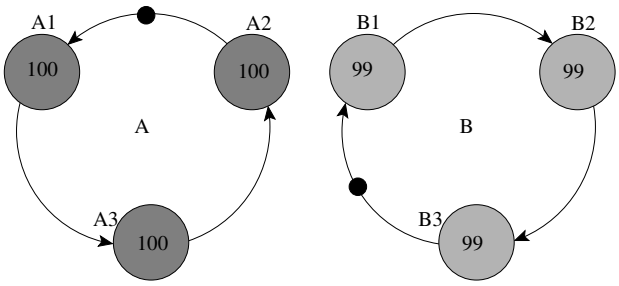

Figure 3. Two applications running on same platform and sharing resources.

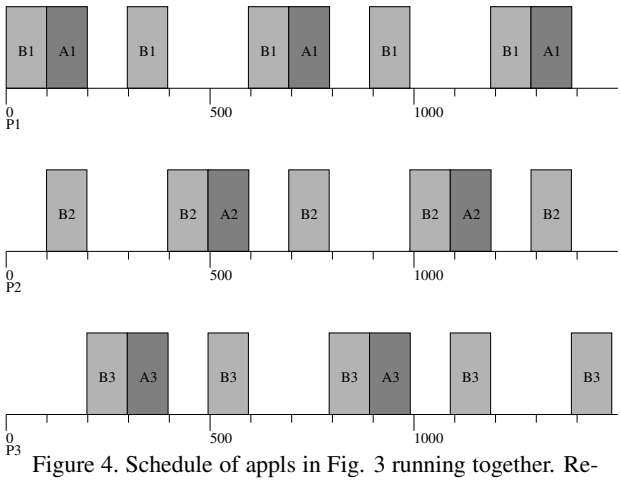

Figure 4. Schedule of appls in Fig. 3 running together. Required performance is one iteration in 450 cycles.

If the applications are allowed to run without intervention from the RM, we observe that it is not possible to meet the performance requirements; the resulting schedule for executing A and B, which is highly unpredictable at compile time, yields a throughput of B that is twice the throughput of A. However, if B could be temporarily suspended, A will be able to achieve the required throughput. A resource manager can easily provide such a control and ensure that desired throughput for both A and B is obtained.

We also see in the example that even though each application only uses a third of each processing node, thereby placing a total demand of two-third on each processing node, the applications are not able to meet their required performance. A compile-time analysis of all possible use-cases can alleviate this problem by deriving how applications would affect each other at run-time. However, the potentially large number of use-cases in a real system makes such analysis infeasible [13]. A resource manager can shift the burden of compile-time analysis to run-time monitoring and intervention when necessary.

4. System Setup

4.1. Model Overview

An MPSoC consists of three kinds of resources - computation, communication and storage. We ignore in this paper the contention for communication and storage resources as computation resources already demonstrate the complexity of the scheduling problem; and management of other resources can be added later in a similar fashion. Figure 5 shows an overview of the model. It consists of a user-interface, local application managers (one for each application), a resource manager, and the computation platform. The user-interface simulates input from and output to the user; for example, in case of mobile phone, input can come from a keypad or joystick, while the output can be in the form of screen display or sound.

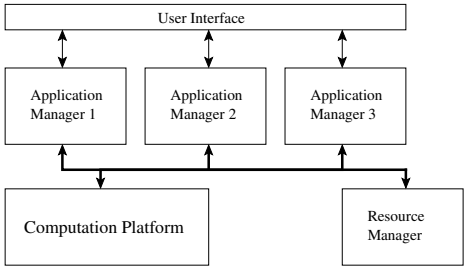

Figure 5. Overview of the system setup.

4.2. Resource Manager

The resource manager is responsible for two main functions, namely *admission control* and *resource budget enforcement*, which are explained in the following two subsections.

4.2.1 Admission Control

One of the main uses of a resource manager is admission control for new applications. In a typical high-end multimedia system, applications are started at run-time, and determining if there are enough resources in the system to admit new application is non-trivial. Since applications are not always *composable*, it is difficult to predict the resource utilization when multiple applications are allowed to run in the system. (*Composability* is defined as being able to analyze applications in isolation while still being able to reason about their overall behavior [13].)

When the resource manager receives a new application request, it checks if there are enough resources available for all the applications to achieve their desired performance using a *predictor*. The predictor module in this setup simply adds the processor utilization for each processing node and checks if it is less than 100%. However, more research needs to be done for a more intelligent predictor. An idea could be to reduce the 100% to $x\%$ where x is adaptive depending on the number of misses in the past, for example.

4.2.2 Resource Budget Enforcement

This is one of the most important functions we expect a resource manager to do. When multiple applications are running in a system (often with dynamically changing execution times), it is quite a challenge to schedule all of them such that they meet their throughput constraints. Using a *static-order* or *round-robin* approach for scheduling is neither scalable, nor adaptive to dynamism present in a system [13]. A resource manager, on the other hand, can monitor the progress of applications running in the system and enforce the budgets requested at run-time.

Clearly, the monitoring and enforcement also has an overhead associated with it. Granularity of control is therefore a very important consideration when designing the system. We would like to have as little control as possible while achieving close to desired performance. This is investigated further in Section 5.

Task migration

Task migration is not considered in this paper, but it could be useful in cases when a particular actor can be scheduled on multiple nodes. A technique to achieve low-cost task migration in heterogeneous MPSoC has been proposed in [14].

4.3. Protocol for Communication

Figure 6 shows an example interaction diagram between various modules in the design. The user-interaction module sends a signal to the respective application manager when any application is to be started. The resource and application managers are responsible for actually running the applications and interacting with the computation platform. The platform module can consist of a number of processors and each processor has a scheduler which can for example be first-come-first-serve (FCFS), round-robin or round-robin-with-skipping [13]. Other scheduler types can be easily added in the model.

In Figure 6, the user-interface module sends a request to start application X (1). The resource manager checks if there are enough resources for it, and then admits it in the system (2). Applications Y and Z are also started respectively soon after as indicated on the figure (3-6). However, when Z is admitted, Y starts to deteriorate in performance. The resource manager then sends a signal to suspend X (7) to the platform because it has slack and Y is then able to meet the desired performance. When X resumes (10), it is not able to meet its performance and Y is suspended (13) because now Y has slack. When the applications are finished, the result is transmitted back to the user-interface (12, 16 and 18). In streaming applications, the result may need to be continuously sent to the user-interface. We also see an example of application A being rejected (9) since the system is possibly too busy, and A is of lower priority than applications X, Y and Z.

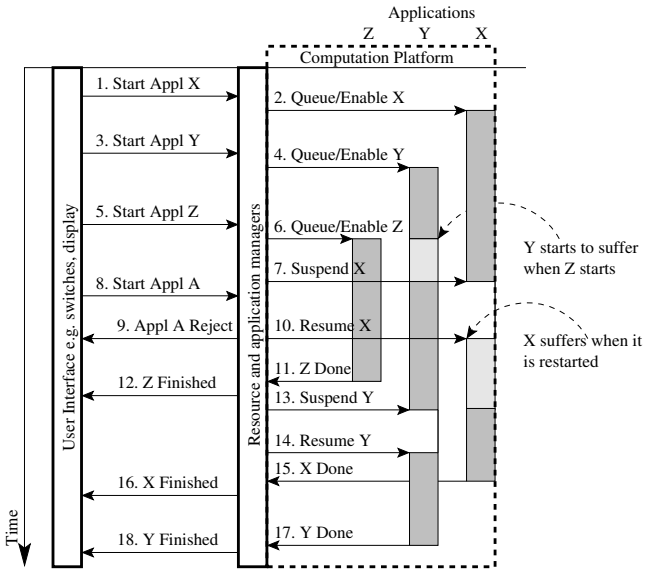

Figure 6. Interaction diagram between various modules in the system-setup.

4.4. Suspending Applications

Suspension of an application is not to be confused with pre-emption. In our model, we do not allow actors to be pre-empted, but an application can be suspended after completing any actor. Performance is specified as a desired throughput that is to be obtained for each application. The sample period of the resource manager is also specified. The resource manager checks the progress of all applications after every sample period - defined as *sample points*. If any application is found to be running below the desired throughput, the application which has the most slack (or the highest ratio of achieved to desired throughput) is suspended. Suspension and re-activation occur only at these sample points.

5. Performance Evaluation

We have developed a three-phase prototype tool-flow to automate the analysis of application examples. The first phase concerns specifying different applications (as SDF graphs), the processors of the MPSoC platform (including their scheduler type) and the mapping. For each application, desired throughput is specified together with the starting time for the application. After organizing the information in an XML specification for all three parts, a POOSL model of the complete MPSoC system is generated automatically. The second phase relies on the POOSL simulator, which obtains performance estimations, like the application throughput and processor utilization. It also allows generation of trace files that are used in the final phase to generate schedule diagrams and graphs like those presented in this paper.

This section presents results of a case study regarding

the mapping of H263 and JPEG decoder SDF models (described in [7] and [15] respectively) on a three-node MPSoC. An FCFS scheduling policy was used in all the cases presented below. Table 2 shows the load on each processing node due to each application. The throughput requirement of applications was chosen such that both applications place equal demands on resources. The results were obtained after running the simulation for 100M cycles.

	H263	JPEG	Total
Proc 1	0.164	0.360	0.524
Proc 2	0.4	0.144	0.544
Proc 3	0.192	0.252	0.444
Total	0.756	0.756	1.512
Throughput Required	3.33e-6	5.00e-6	

Table 2. Load (in proportion to total available cycles) on processing nodes due to each application

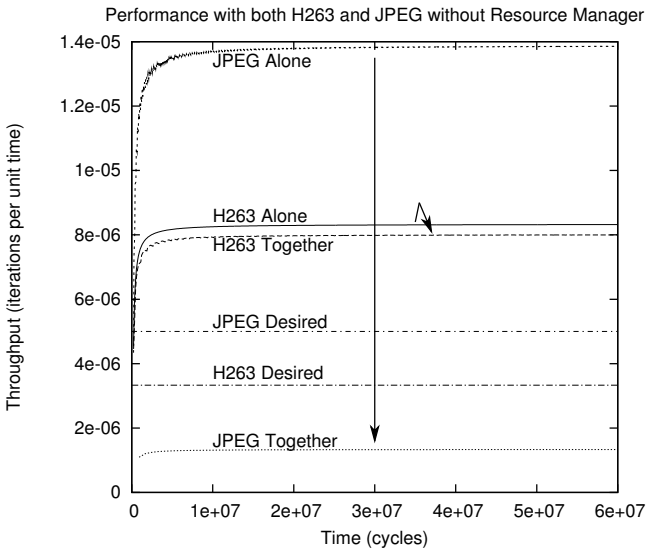

Figure 7. Progress of H263 and JPEG when they run on the same platform — in isolation and executing concurrently.

Figure 7 shows performance of the two applications when they are run in isolation on the platform and also when they are run concurrently. In this figure, the resource manager does not interfere at all, and the applications compete with each other for resources. As can be seen, while the performance of H263 drops only marginally (depicted by the small arrow in the graph), a huge drop is observed in JPEG performance (big arrow in the graph). In fact, we see that even though the total load on each processing node is close to 50%, JPEG throughput is much lower than desired. Figure 8 shows how a resource manager interferes and ensures that both are able to meet their minimum specified throughput. In this figure the resource manager checks every 5 million cycles whether applications are performing as desired. Every time it finds that either JPEG or H263 is performing below the desired throughput, it suspends the other application. Once the desired throughput is reached, the suspended application is re-activated. We observe that the RM effectively interleaves three infeasible schedules (JPEG Alone, H263 Alone, and H263/JPEG Together, in Fig. 7) that yields a feasible overall throughput for each application. (In *Alone*, only one application is active and therefore, those schedules are infeasible for the other application.)

	Specified	No RM	RM sampling period		
			5,000k	2,500k	500k
JPEG	500	133	541	520	620
H263	333	800	554	574	504
Proc 1	0.524	1.00	0.83	0.85	0.90
Proc 2	0.544	0.56	0.71	0.71	0.72
Proc 3	0.444	0.46	0.55	0.55	0.56
Total	1.512	2.02	2.09	2.11	2.18

Table 3. Iteration count of applications and utilization of processors for different sampling periods for 100M cycles.

Figure 9 shows applications' performance when the sample period of resource manager is reduced to 500,000 cycles. We observe that progress of applications is 'smoother' as compared to Figure 8. The 'transition phase' of the system is also shorter, and the applications soon settle into a 'long-term average throughput', and do not vary significantly from this average. This can be concluded from the almost horizontal curve of achieved throughput. It should be mentioned that this benefit comes at the cost of increasing monitoring from the resource manager, and extra overhead in reconfiguration (suspension and reactivation).

Table 3 shows the iteration count for each application specified, achieved without and with intervention from the RM. The first two columns clearly indicate that JPEG executes only about one-fourth of the required number of iterations, whereas H263 executes twice the required iteration count. The next three columns demonstrate the use of our RM to satisfy the required throughput for both the applications. The last row indicates that the utilization of resources increases with finer grain of control from the RM.

6. Conclusions and Future Work

In this paper, we propose a resource manager (RM) for non-preemptive heterogeneous MPSoCs. Although the scheduling of these systems has been considered in the literature, the actual resource management in the context of concurrently executing applications is still unexplored area. Theoretically, compile-time analysis of all possible use-cases can provide performance guarantees, but the potentially large number of use-cases in a real system makes such analysis infeasible [13]. Our resource manager shifts

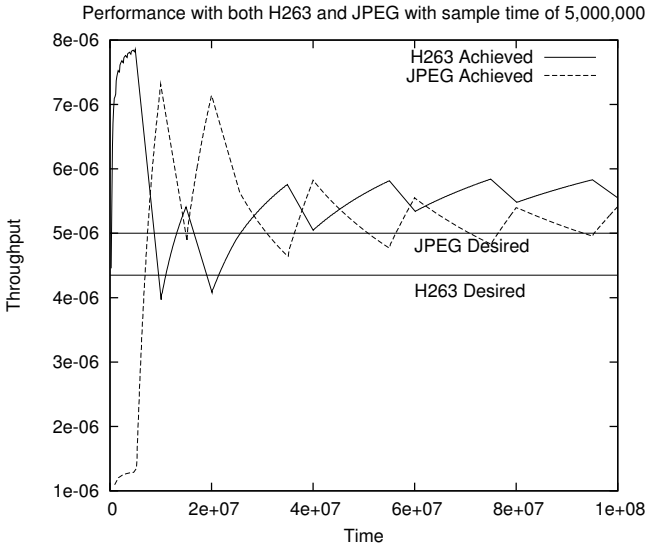

Figure 8. With a resource manager, the progress of applications is closer to desired performance.

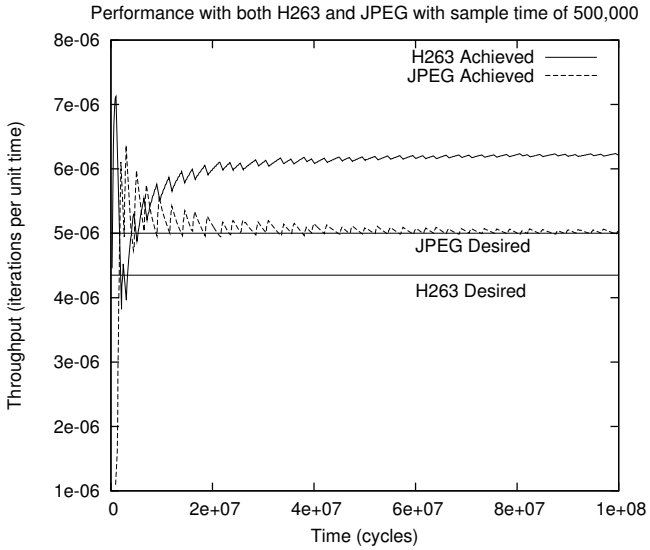

Figure 9. Increasing granularity of control makes the progress of applications smoother

the burden of compile-time analysis to run-time monitoring and intervention when necessary.

A high-level simulation model has been developed using POOSL methodology to realize RM. A case study with an H263 and a JPEG decoder demonstrates that RM intervention is essential to ensure that both applications are able to meet their throughput requirements. Further, a finer grain of control increases the utilization of processor resources, and leads to a more reactive and stable system.

Future research will focus on incorporating more intelligent prediction schemes for admission control. An FPGA implementation of an MPSoC is already realized, and we will use it to run our proposed RM on, to handle more realistic cases and to measure the overhead. Further, we would like to extend the model to include the communication and storage resources as well.

References

[1] K. Jeffay, D. F. Stanat, and C. U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of 12th IEEE Real-Time Systems Symposium*, pages 129–139, 1991.

[2] W. Wolf. The future of multiprocessor systems-on-chips. In *Proceedings of the 41st DAC '04*, pages 681–685, 2004.

[3] Available from: http://www.es.ele.tue.nl/poosl.

[4] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[5] S. Baruah. The non-preemptive scheduling of periodic tasks upon multiprocessors. *Real-Time Systems*, 32(1):9–20, 2006.

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

[7] R. Hoes. Predictable Dynamic Behavior in NoC-based MPSoC. Available from: www.es.ele.tue.nl/epicurus/, 2004.

[8] Y. Cai and M. C. Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica*, 15(6):572–599, 1996.

[9] S. Davari and S. K. Dhall. An on line algorithm for real-time tasks allocation. *IEEE Real-time Systems Symposium*, 1986.

[10] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. An on line algorithm for real-time tasks allocation. *Algorithmica*, 15:600–625, 1996.

[11] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of ISCAS 2000 Geneva.*, volume 4, pages 101–104, Geneva, Switzerland, 2000.

[12] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, 1987.

[13] A. Kumar, B. Mesman, H. Corporaal, J. van Meerbergen, and Y. Ha. Global analysis of resource arbitration for MPSoC. In *Proceedings of Ninth Euromicro Conference on Digital System Design*. Euromicro, 2006.

[14] V. Nollet, P. Avasare, J-Y. Mignolet, and D. Verkest. Low cost task migration initiation in a heterogeneous mp-soc. In *Proceedings of DATE '05*, pages 252–253. IEEE Computer Society, 2005.

[15] E.A. de Kock. Multiprocessor mapping of process networks: a JPEG decoding case study. In *Proceedings of 15th Intl. Symp. on System Synthesis, 2002.*, pages 68–73. IEEE Computer Society, 2002.