On Composability of MPSoC Applications

Akash Kumar*, Bart Theelen*, Bart Mesman*, Henk Corporaal*[1]

*Eindhoven University of Technology, 5600MB Eindhoven, The Netherlands

ABSTRACT

Mapping and analyzing multiple applications on a multi-processor system is a complex problem. Analyzing the feasibility and resource utilization of all possible use-cases is often infeasible. Here we highlight the issue of composability, i.e. being able to analyze applications in isolation while still reason about the overall system behavior. We observe that arbitration plays an important role in this analysis. Two simple, yet commonly used arbitration mechanisms are compared, and the properties are highlighted that are important for such composability.

1 Introduction

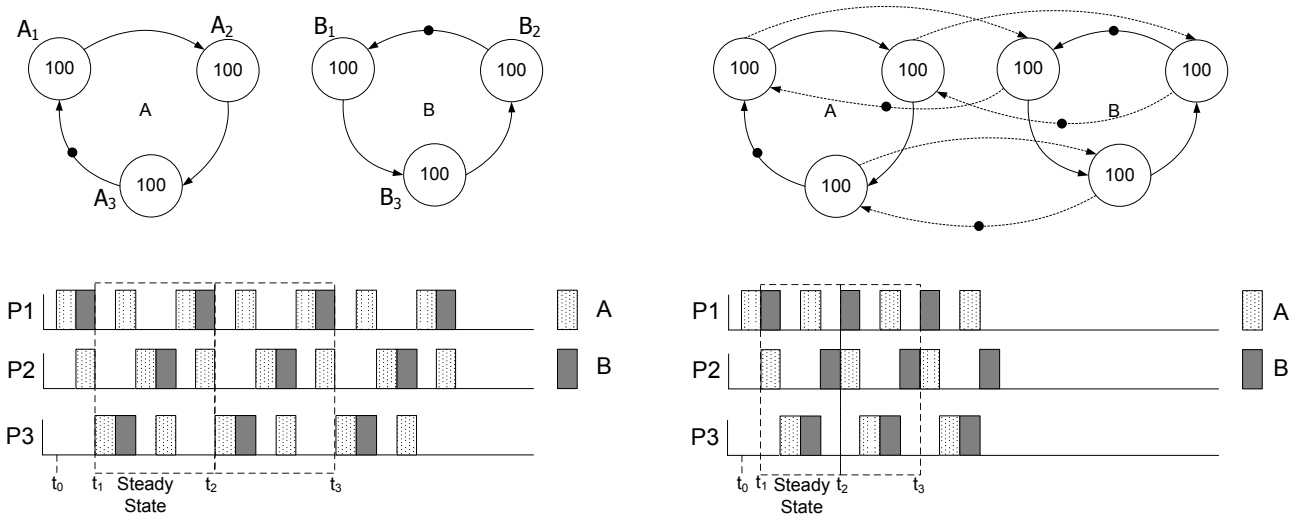
Typical Multi-Processor System-on-Chips (MPSoC) run multiple applications in parallel. An MPSoC-based mobile phone may execute for example an MP3 decoder to produce music, while the user also writes a text message concurrently to downloading a new ring tone in the background. The user should not experience significant quality drops or delays when activating or deactivating applications. These requirements imply the allocation of the available resources (like processors, memories and busses) to provide certain guarantees. Where schedulability analysis for traditional single-processor systems (which often support preemption) is well studied [LL73], the theory of rate-monotonic analysis and the likes do not apply for MPSoCs.

An MPSoC on which N applications may potentially run, has up to 2^N possible use-cases. Hence, evaluating the resource requirements for all use-cases is often too expensive. Instead, one would like to analyze each application in isolation (thereby reducing the analysis time to a linear function in N) and determine the overall performance of the system from the results for the individual applications in combination with used scheduling policy. In the context of concurrent systems, a stronger version of such compositionality is sometimes called *composability*, which states that the properties satisfied by the individual applications should remain satisfied by their parallel compositions [Sif01]. Composability could be achieved by means of *virtualization*, which involves reserving a (weighted) share of the resources for each application. A disadvantage is however the waste of resources when certain applications are inactive. This paper studies *how the exponential analysis complexity for MPSoCs can be reduced to linear complexity, without paying the overhead of complete virtualization*. To this end, we assume applications to be specified as a (Homogeneous) Synchronous Data Flow (SDF) graph [LM87], where vertices indicate separate tasks (also called actors) of an application and edges denote dependencies between them. In HSDF, an actor can only fire (execute) when a token (data item) is available on all its inputs. Using (H)SDF allows analysis of properties like absence of deadlock, throughput and memory requirements [SB00]. Assuming an MPSoC without support for preemption, we research the impact on the composability regarding throughput. Static (fixed) order and round robin (with skipping) schedules are compared.

2 Composability Problem

In this section we present an example which demonstrates why the resource requirements for actors can not be simply added. Figure 1 shows an example of two applications A and B (with three actors each) mapped on a three processor system. We assume that actors A_i and B_i are mapped onto the processor P_i for $i = 1, 2, 3$. Each actor takes 100 time units to execute and a dot on an edge indicates the availability of an initial token. Because of the dependencies within the applications, only one actor of A and B can be executing at a time and hence, A and B both have a throughput of $\frac{1}{300}$. For determining the throughput of more complex task graphs, one could also apply the HSDF analysis technique of the Maximum Cycle Mean (MCM) [SB00]. The throughput of an HSDF equals

[1]E-mail: {a.kumar,b.d.theelen,b.mesman,h.corporaal}@tue.nl



(a) Schedule for round-robin with skipping.

(b) Schedule for round-robin w/o skipping (static order).

Figure 1: Scheduling two applications using round robin with skipping and without skipping.

the reciprocal of its MCM (being 300 time units for both A and B). We furthermore observe that each application induces a load of $\frac{1}{3}$ on each processor and hence when running both, the maximum achievable processor utilization is $\frac{2}{3}$.

However, the assumed mapping leads to contention regarding the processors, which may result in not being able to deliver the performance for each application individually as analyzed above. A conservative way of determining the processor utilization is to take the worst case waiting time for the actors into account, which can be computed based on the *critical instant* as defined by Liu and Layland [LL73]. The critical instant for a task is defined as an instant at which a request for that task will have the largest response time. Since we only have two actors mapped on each processor, the maximum waiting time for each actor is 100 time units, and the response time for each actor is therefore, at most 200 time units. The MCM for each graph individually is then 600 time units. The load due to each application becomes $\frac{1}{6}$, leading to a total of $\frac{1}{3}$ on each processor. This forms the lower bound on processor utilization using first-come-first-serve (FCFS) or round-robin with skipping.

Figure 1(a) illustrates scheduling A and B when using round robin (RR) with skipping. The first contention between application A and B occurs at t_0, when both A_1 and B_1 are ready to execute on P_1. As shown, A wins the arbitration, while B_1 must wait. Another contention occurs at t_1 for processor P_3, and then for P_2 followed by P_1. The schedule assumes that A wins every arbitration. The schedule soon settles into a periodic regime taking 600 time units, in which A completes two iterations and B completes only one. Hence, A will achieve a throughput of $\frac{1}{300}$ (as desired) while the throughput of B is only $\frac{1}{600}$. In case B would have won every arbitration, the situation is reversed and B would execute twice as many times as A. In either of the cases, the processor utilization is only $\frac{1}{2}$ (instead of the expected $\frac{2}{3}$).

Figure 1(b) shows a schedule when using round robin without skipping, which is equivalent to a static order schedule. Such static order schedules can be expressed in (H)SDF by adding dependencies as illustrated in Figure 1(b). For this schedule, both A and B take 400 time units for completing one iteration. As a result, the throughput for both A and B equals $\frac{1}{400}$, while the processor utilization is again only $\frac{1}{2}$. The throughput results are confirmed by applying the MCM technique on the HSDF graph of the two applications together, which equals 400 time units (consider for example, the cycle $A2 \rightarrow A3 \rightarrow B3 \rightarrow B2 \rightarrow A2$). The problem with applying the MCM technique is however the requirement to consider the HSDF graph that represents all applications and the chosen static order schedule together. When the number of applications increases, the number of possible HSDF graphs that can be constructed to depict the whole system increases exponentially. There are, for example, $((g-1)!)^a$ unique possible static order schedules for g applications consisting of a actors each, which may all have a different MCM. A similar complexity concerns the size of these HSDF graphs.

As can be seen the resource utilization and realized throughput of each application varies with the scheduling mechanism chosen. In Table 1 we summarize these results. The table shows how many

	Dependency Considered			Scheduler used		
	None	Intra-application	Inter-application	Static order	RR A pref	RR B pref
Application A	5,000	3,333	1,666	2,500	3,333	1,666
Application B	5,000	3,333	1,666	2,500	1,666	3,333
Total	10,000	6,666	3,333	5,000	5,000	5,000
Processor Utilization	1	2/3	1/3	1/2	1/2	1/2

Table 1: Accounting of resource utilization: Iterations for each application for 1,000,000 time units

Property		**Static Order**	**Round Robin with skipping**
Design time overhead	Calculating Schedules	- -	++
Run-time overhead	Memory requirement	-	++
	Scheduling overhead	+	+
Predictability	Throughput	++	- -
	Resource Utilization	+	-
New job admission	Admission criteria	++	- -
	Deadlock-free guarantee	-	++
	Reconfiguration overhead	-	+
Dynamism	Variable Execution time	-	+
	Handling new use-case	- -	++

Table 2: Properties of Scheduling Strategies

iterations of each applications are executed in one million time units. The first set of columns are obtained by analyzing the applications in isolation, and not actually scheduling them. Three cases are shown.

- None: No dependency is considered. Since each processor has a total load of 200 time units, each application can finish 5,000 iterations.
- Intra-application: The dependency within the application is considered. The MCM of each application is 300 time units, and can therefore, finish 3,333 iterations. It should be mentioned that it is only because the total load on the processor is $\frac{2}{3}$.
- Inter-application: When contention due to other applications is taken into account, we use the worst-case estimate to arrive at the MCM of 600, thereby leading to only 1,667 iterations.

The next set of columns is obtained by scheduling applications by hand using different schedulers. We can thus conclude that overall resource utilization can not simply be added.

3 Resource Manager

This section summarizes the properties of an arbitration strategy that makes it suitable for a *resource manager*. A resource manager is a separate task running on the system that monitors and enforces usage of resources. Admission of a new job also falls in the scope of resource manager. Table 2 shows a summary of various performance parameters that are important for resource manager. The two arbitration mechanisms compared in the paper are evaluated on these performance measures. (A complete description can be found in [KMC+06].) From the table, we conclude that round-robin with skipping satisfies most of our criteria, and is hence quite suitable for a resource manager in an MPSoC. It however, suffers from lack of predictability since it is difficult to estimate resource usage using analytical tools. In the next section, we describe a prototype tool flow developed that overcomes this limitation.

4 Prototype Tool Flow

Section 2 illustrated the composability problem by means of a small example (Figure 1), which could easily be analyzed by hand. Automated analysis of bigger case studies using static order schedules could be performed using existing SDF analysis tools like the one in [SGB06] (using the MCM technique). To investigate composability for other scheduler types than static order, another tool was

needed. We therefore developed a three-phase prototype tool flow that relies on the Y-chart approach [dKSvdW+00] and the modeling language POOSL [POO]. The first phase concerns specifying the different applications (as SDF graphs), the processors of the MPSoC platform (including their scheduler type) and the mapping. After organizing the information in an XML specification for all three parts, a POOSL model of the complete MPSoC system is generated automatically. The generation relies on the approach in [TGB+06] for modeling the applications, while the mapping and processors are modeled according to the approach of [FdHVC06]. Currently, the POOSL model generation tool supports FCFS, round-robin without skipping (static order) and round-robin with skipping as options for the scheduler types, while existing modeling patterns for other scheduler types (such as a priority-based scheduler with preemption) could easily be integrated as well. The second phase relies on the simulator for POOSL models, which obtains estimation results for performance metrics like the application throughput and processor utilization with a predetermined accuracy. It also allows generation of trace files that can be used in the final phase to generate schedule diagrams like those presented in this paper.

Using the prototype tool flow, we investigated a few simple examples with various scheduler types to validate the tool itself. We tried an example with three applications - A, B and C; each with three actors, similar to the ones shown in Figure 1. We set the run-time of each actor in A to be 100, in B to be 80 and in C to be 50 time-units. Initial investigations show that round-robin with skipping and FCFS fair better than strict round-robin. In some simulations, we also found that the example in Figure 1 achieves only a processor utilization of 33% (and not 50%) when a static order is used. The difference is caused by the time at which applications are introduced in the system.

5 Conclusions and Future Work

In this paper, we have introduced the composability problem and shown that combining resource usage is non-trivial. We also summarized properties and requirements for resource arbitration for a multi-processor based system-on-chip. Furthermore, using these requirements, we have compared two simple arbitration mechanisms. We also developed a tool-flow to automatically analyze bigger case studies and investigate performance of different schedulers. We would now like to evaluate the arbitration mechanism on real MPSoC hardware implementation. An FPGA infrastructure has already been developed in order to make architectural explorations and do quick design iterations.

References

[dKSvdW+00] E. A. de Kock, W. J. M. Smits, P. van der Wolf, J.-Y. Brunel, W. M. Kruijtzer, P. Lieverse, K. A. Vissers, and G. Essink. Yapi: application modeling for signal processing systems. In *DAC '00: Proceedings of the 37th conference on Design automation*, New York, NY, USA, 2000. ACM Press.

[FdHVC06] O. Florescu, M. de Hoon, J. Voeten, and H. Corporaal. Probabilistic Modelling and Evaluation of Soft Real-Time Embedded Systems. In *Proceedings of Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2006.

[KMC+06] A. Kumar, B. Mesman, H. Corporaal, J. van Meerbergen, and Y. Ha. Global analysis of resource arbitration for mpsoc. In *Ninth Euromicro Conference on Digital System Design*. Euromicro, 2006.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[LM87] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous dataflow programs for digital signal processing. *IEEE Transactions on Computers*, Feb 1987.

[POO] http://www.es.ele.tue.nl/poosl.

[SB00] S. Siram and S.S. Bhattacharyya. *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.

[SGB06] S. Stuijk, M.C.W. Geilen, and T. Basten. SDF3: SDF for Free. In *Proceedings of the International Conference on Application of Concurrency to System Design*. IEEE Computer Society Press, 2006.

[Sif01] J. Sifakis. Modeling Real-Time Systems; Challenges and Work Directions. In *Proceedings of the First International Workshop on Embedded Software*. Springer-Verlag, 2001.

[TGB+06] B.D. Theelen, M.C.W. Geilen, T. Basten, J.P.M. Voeten, S.V. Gheorghita, and S. Stuijk. A Scenario-Aware Data Flow Model for Combined Long-Run Average and Worst-Case Performance Analysis. In *Proceedings of the International Conference on Formal Methods and Models for Co-Design*. IEEE Computer Society Press, 2006.