An MPSoC Design Approach for Multiple Use-cases of Throughput Constrainted Applications

Ahsan Shabbir[1]
a.shabbir@tue.nl

Sander Stuijk[1]
s.stuijk@tue.nl

Akash Kumar[1,2]
akash@nus.edu.sg

Henk Corporaal[1]
h.corporaal@tue.nl

Bart Mesman[1]
b.mesman@tue.nl

[1]Eindhoven University of Technology, The Netherlands, [2]National University of Singapore, Singapore

ABSTRACT

Modern multimedia systems must support a variety of different use-cases. Multi-processors Systems-on-Chip (MPSoCs) are used to realize these systems. A system designer has to dimension the size of an MPSoC such that the performance constraints of the applications are satisfied in all use-cases. In this paper, we present an approach to design MPSoCs that can meet the throughput constraints of a set of applications while minimizing the resource requirements.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems

General Terms

Design, performance

Keywords

Dataflow, design-space-exploration, throughput constraints

1. INTRODUCTION

Multi-processor Systems-on-Chip (MPSoCs) are increasingly used in multimedia systems to meet the performance and energy constraints of modern applications. The different combinations in which applications are active on the system are known as *use-cases*. The MPSoC should be dimensioned in such a way that it can support all possible use-cases while minimizing the cost of the hardware platform. This requires an approach to synthesize a hardware platform and to map applications onto this platform. Several techniques have been developed to address this problem (e.g. [1, 3]). These techniques assume that applications are modeled as acyclic task graphs. This model-of-computation (MoC) is not very suitable for modeling streaming, pipelined applications. The synchronous dataflow MoC [2] allows modeling of pipelining. A design approach that uses this MoC can potentially save resources as compared to a design approach based on acyclic task graphs. A design approach based on this MoC has been introduced in [4]. This design approach, however, does not consider multiple use-cases. In this paper, we introduce a novel MPSoC design approach to generate an MPSoC that is able to support a set of use-cases in which each use-case consists of a number of throughput-constrained applications. Applications are modeled with synchronous dataflow graphs (SDFGs) [2] which allows us to exploit pipelining and hence save resources. Experimental results show that our technique is able to find an MPSoC with fewer resources as compared to a technique [4] that does not consider multiple use-cases.

CF'11, May 03–05, 2011, Ischia, Italy.
ACM 978-1-4503-0698-0/11/05.

2. DESIGN APPROACH

The design approach proposed in this paper is illustrated in Figure 1. In this example, we must design an MPSoC to run applications A and B. These applications are modeled with an SDFG. The nodes in an SDFGs are called *actors* and they model the computation that needs to be performed by the application. The execution time of the actors is annotated inside the nodes. Actors can only communicate to each other by sending data items, called *tokens*, through their *edges*. An essential property of SDFGs is that every time an actor *fires* (executes) it consumes the same amount of tokens from its input edges and produces the same amount of tokens on its output edges. These amounts are called the *rates*. In our example, all rates are equal to one. Assume that both applications have a throughput constraint of 0.005 iterations/time-unit. In this case, a new firing of each actor should on average start every 200 time-units.

Our algorithm starts by estimating the minimum number of processors needed to meet the throughput constraints of the applications. Minimum number of processors are found by multiplying the repetition vector entry of an actor with its execution time and summing it over all the actors. The result is multiplied with the throughput constraint of the application. For our running example, the algorithm estimate that a minimum of two processors is required. Next, an initial mapping is created by mapping the actors of the applications to the various processors (step 1 in Figure 1). The algorithm tries to do this in such a way that the processor loads are balanced. Subsequently, the mapping is evaluated in all use-cases. If an application does not satisfy its throughput constraints, then some of its actors are moved to a processor having minimal load (steps 2-6 in Figure 1). This process is repeated for all actors of the application failing its throughput constraints. If after moving all actors once, the application still does not meet its throughput constraint, the algorithm adds a new processor (step 7 in Figure 1) and

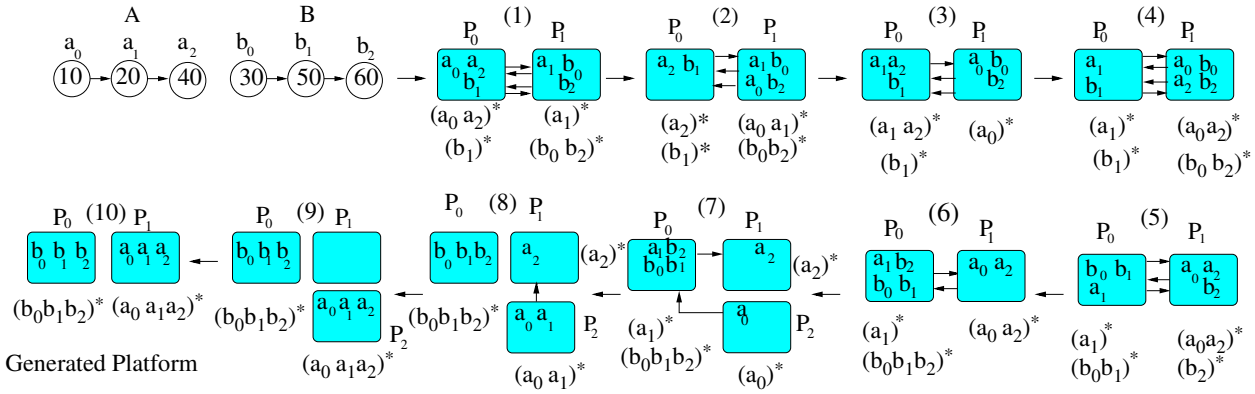

Figure 1: Illustrative example.

Table 1: Resource used by [4] and our algorithm

Algorithm	[4]					Ours (with use-cases)				Ours (without use-cases)					
	actors			edges		actors		edges		actors				edges	
Applications	p0	p1	p2	#FIFOs	memory	p0	p1	#FIFOs	memory	p0	p1	p2	p3	#FIFOs	memory
H.263 decoder	1	2	1	3	1194	4	0	0	1196	0	0	0	4	0	1196
H.263 encoder	3	1	1	4	304	5	0	0	304	5	0	0	0	0	304
mp3 decoder	5	3	6	7	19	13	1	1	18	0	1	13	0	1	18
Total	3 proc			14 FIFOs	1518 bytes	2 proc		1 FIFO	1500 bytes	4 proc				1 FIFOs	1518 bytes

transfers actors to this processor. Once all applications meet their throughput constraints (step 7), the algorithm tries to minimize the number of FIFO connections between the processors (step 8-9 in Figure 1). This process may involve the re-mapping of actors in order to reduce the number of required connections. Step 9 of our example shows that by moving all actors from application A to processor P_2, the number of FIFOs can be reduced. Finally, our algorithm tries to assign the edges in the application graphs to the FIFOs in such a way that as many FIFO as possible can be re-used between use-cases. The output of our algorithm is an MPSoC platform and schedules of the actors onto the processors. The actors from same application are scheduled using a static order schedule. The actors from different applications are scheduled using a round robin arbiter.

3. EXPERIMENTS

In this section, we present a comparison between our technique and the technique presented in [4]. We use three applications, a H.263 decoder, a H.263 encoder, and an MP3 decoder, to make this comparison. Using these three applications, we created four different use-cases. In the first three use-cases only one application is active. In the fourth use-case, the H.263 encoder and decoder are active, but the MP3 decoder is inactive. Hence, there is no use-case in which all three applications are active simultaneously.

There are three important differences between our technique and the technique presented in [4]. (1) The technique from [4] maps the applications onto a given MPSoC while our approach generates an MPSoC. (2) We exploit the use-cases to save resources. The technique from [4] assigns separate resources to each application. Hence, it assumes that all applications are active simultaneously. In our case study, this situation (i.e. use-case) does not occur. (3) The technique from [4] assumes a platform with preemptive scheduling whereas we target a platform that uses non-preemptive scheduling.

Table 1 shows the resources consumed by the two techniques. Columns (2-4) show the number of actors of each application mapped onto the various processors (p_i). The table also shows the number of connections through the interconnect (#FIFOs) and the total buffer size used by each application (memory). The results show that our approach, when considering use-cases, requires 33% fewer processors, 93% fewer interconnect FIFOs and 1% less memory as compared to the technique from [4]. These resource savings come from the fact that we exploit the property that not all applications are active simultaneously. The last columns (labeled 'Ours (without use-cases)') shows the results when we would not consider use-cases (i.e. we would assume that all applications are active simultaneously). The results show that in this situation, more resources are consumed as compared to [4]. The reason for this high resource usage is the fact that non-preemptive systems have worse response times as compared to pre-emptive systems.

4. CONCLUSIONS

In this paper, we present a novel approach to generate MPSoCs that satisfies the throughput constraints of multiple applications in all possible use-cases. Experimental results show that the concept of use-cases allows our approach to design systems with fewer resources as compared to a technique [4] which does not consider use-cases.

5. REFERENCES

[1] Hyunok, O., et al. Hardware-software co-synthesis of multi-mode multi-task embedded systems with real-time constraints. In *CODES '02*, p. 133–138.
[2] Lee,E.A., et al. Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing. In *IEEE Transactions on Computers '87*, p. 24–35.
[3] Shin, Y., et al. Schedulability-driven performance analysis of multiple mode embedded real-time systems. In *DAC '00*, p. 495–500.
[4] Stuijk, S.,et al. Multiprocessor resource allocation for throughput-constrained Synchronous Dataflow Graphs. In *DAC '07*, p. 777–782.