Modeling Resolution of Resources Contention in Synchronous Data Flow Graphs

Marco Lattuada · Fabrizio Ferrandi

Received: date / Accepted: date

Abstract Synchronous Data Flow graphs are widely adopted in the designing of streaming applications, but were originally formulated to describe only how an application is partitioned and which data are exchanged among different tasks. Since Synchronous Data Flow graphs are often used to describe and evaluate complete design solutions, missing information (e.g., mapping, scheduling, etc.) has to be included in them by means of further actors and channels to obtain accurate evaluations. To address this issue preserving the simplicity of the representation, techniques that model data transfer delays by means of ad-hoc actors have been proposed, but they model independently each communication ignoring contentions. Moreover, they do not usually consider at all delays due to buffer contentions, potentially overestimating the throughput of a design solution.

In this paper a technique to extend Synchronous Data Flow graphs by adding ad-hoc actors and channels to model resolution of resources contentions is proposed. The results show that the number of added actors and channels is limited but that they can significantly increase the Synchronous Data Flow graph accuracy.

Keywords Synchronous Data Flow Graphs · Data Transfers · Buffers · Contention

M. Lattuada · F. Ferrandi
Politecnico di Milano - Dipartimento di Elettronica, Informazione e Bioingegneria
Via Ponzio 34/5
20133 Milano, Italy
E-mail: marco.lattuada@polimi.it, fabrizio.ferrandi@polimi.it

1 Introduction

The reduced time-to-market and the increased complexity of Multiprocessors Systems on Chip require usage of high level abstraction descriptions during early phases of embedded system design flow. Streaming processing (i.e., applying a set of elaborations in sequence on an input data stream) is becoming one of the most prominent paradigms in this type of design, since it characterizes most of the video and audio elaboration applications and allows to easily extract parallelism. Synchronous Data Flow (SDF) graphs [1] are one of the most common models of computation used to describe streaming applications. Being a well known and widely adopted formalism, several methodologies and tools to analyze and synthesize design solutions starting from them exist. The natively described information is limited: only the decomposition in tasks of an application and which data these tasks have to exchange are well described. Other important aspects, such as communication delays, which are necessary to analyze and evaluate a design solution, are not described. This missing information can be associated with the SDF graph, but its direct integration in the graph makes analyses and exporting of the solutions easier. The integration is usually implemented by adding new actors and channels to the initial SDF graph. For example, a scheduling solution can be integrated in a SDF graph [2] by adding actors and channels which force the sequential execution of the actors mapped on the same processing element, guaranteeing the execution order specified in the design solution. The computed throughput of the enriched graph in this way implicitly takes into account the scheduling solution, reducing the overestimation with respect to the throughput computed analyzing the initial SDF graph.

Even if data exchanged between actors are explicitly modeled by tokens, the details of the communication performances are not explicitly described by SDF graphs, which model them as instantaneous. To overcome this limitation, several methodologies (e.g., [3], [4]) have been proposed which detail inter-actors communications by adding new actors. These methodologies however focus only on the delay of the single communications and they do not correctly model communication interactions overestimating the throughput. Moreover, communication infrastructures (e.g., buses, networks on chip) are not the only resources which have to be considered in the implementation of a channel. In order to complete a data transfer, the destination buffer must have enough available space to receive the data. This space, thanks to the properties of SDF graphs, can be computed and optimized at design time. Like other resources (e.g., processing elements, networks), buffers can potentially be contended by different data transfers or actors. Ignoring these contentions is not possible because they can reduce the actual application throughput and make some design solutions infeasible.

In this paper a technique to model the delays due to resolution of resources contentions is proposed. The technique starts from a SDF graph, and after adding actors to model delays of the single channels, adds further actors and channels to serialize accesses to all the contended resources.

This paper is organized as follows. Section 2 presents the related works, Section 3 describes the proposed technique whose experimental evaluation is presented in Section 4 and finally Section 5 draws the conclusions of this paper.

2 Related Works

The enrichment of data flow graphs to include a design solution has been proposed in several works. For example, SynDEx [5] implements the *Algorithm Architecture Adequation (AAA)* methodology which combines information about the algorithm to be implemented (described by a data flow graph called *Software graph*) with information about the target architecture (described by a *Hardware graph*) to produce a data flow graph describing the details of the implementation solution. This tool however has some limitations: it is not open source and it has its own model of computation which does not allow schedulability analysis. To overcome these limits, Pelcat et al. reimplemented *AAA* in Preesm [6]. This tool can start its analysis from algorithms described by means of Synchronous Data Flow graphs, but these are transformed into Homogeneous SDF and then in Directed Acyclic Graph before computing scheduling. Moreover, schedule information is not directly embedded in the analyzed graph, but is provided in form of Gantt diagram and off-graph annotations.

Both scheduling and mapping are instead directly described by the *Interprocessor Communication SDF Graphs* (IPC graphs) proposed by Bambha et al. [7]. This type of graphs is built starting from a Synchronous Data Flow representing the algorithm to be implemented: it describes the sequence of actor activations for each processing element and the communications among them. The graph is not an enriched version of the application SDF graph, but it is built from scratch: in this way some information (e.g., characteristics of the channels connecting actors mapped on the same processing element) of the analyzed application is lost. Moreover, also in this case, initial SDF graphs have to be transformed into Homogeneous ones, so size of the graphs to be analyzed can potentially explode.

Other works propose to integrate design solution directly in the Synchronous Data Flow graph to refine throughput analyses. For example, in [2] Damavandpeyma et al. propose a technique to include scheduling information in a SDF graph. This technique adds new channels and new actors to the original graph to prevent simultaneous execution of actors assigned to the same processing element. The throughput computed on the modified graph does not take into account the communication delays, so the performances provided by different types of interconnections (e.g., buses, networks on chip) cannot be easily evaluated on these enriched SDF graphs. This type of approach has been extended in [8] to model pipelined applications: additional actors are added to model the synchronization between the different stages of the pipeline.

Other techniques allow to integrate information about communication performances in the SDF graphs. In [3] the delay for sending a token on a connection between different processing elements is modeled by a single actor, whose execution time corresponds to the delay to be modeled. The original communication channel is replaced by this new actor and by two channels (ingoing to and outgoing from it). A self-edge is added to each new actor to enforce the sequentiality of tokens transmission. A more complex model is proposed by Holzenspies et al. [4]: each original channel is replaced by a cycle composed of three actors representing *receiving*, *processing* and *transmission* of tokens. The first and the last actor make reading and writing delays explicit, refining the analyzed model.

These techniques only model the delay of isolated communications. A more complex model is proposed by Bennour et al. in [9] where two different aspects of an architecture are considered. The first aspect is the

presence of processing elements with limited local memory: if the tokens to be consumed by an actor cannot be locally stored, an external memory has to be exploited. The induced data transfers are described by means of five new actors and some new channels. The second aspect is the interconnection infrastructure and the introduced delay in sending data from a processing element to another one. With respect to previously presented works, the communication is described in a more accurate way: three actors are added for each channel respectively modeling the sending latency, the transmission latency and the delay for waiting the Time Division Multiple Accesses slot. The problem of modeling the effects of communication resources contentions is not considered since the use of time slots prevents it.

The contention of communication resources is considered in [10]: the authors describe the scheduling as a constraint based problem where access in mutual exclusion to a communication resource is a constraint. However, the constraints are not directly included in the graph representation, making more difficult to integrate the methodology proposed by the authors in existing design frameworks.

3 Modeling Resolution of Resources Contention

The proposed methodology aims at refining analyses performed on SDF graphs by including in them information about how different accesses to the same resources have been serialized to resolve contentions. This further information not only increases the accuracy of the results obtained during evaluation of the design solutions, but preventing optimistic evaluations better guarantees that produced solutions respect time constraints. Moreover, it also allows to identify solutions which are actually infeasible because of the memory contentions. The analyses performed by considering this information in a separate way (i.e., without embedding it into the SDF graph) are possible, but can be much more complex. For examples, approaches like the ones presented in [6] and [7] require to transform the analyzed SDF graph in a Homogeneous SDF, potentially increasing in a very significant way the size of the graph to be analyzed and so the complexity of the rest of the flow [11].

The methodology can target different types of communication infrastructures, from simple shared buses to complex networks on chip. Different levels of detail can be adopted in the modeling (i.e., different number of actors and channels). Ideally the communication infrastructure should be described at the same level of detail of the rest of the system (i.e., algorithms and processing elements). More accurate communication models can increase the complexity of the SDF analyses without significantly increasing the overall accuracy because of the other approximations of the complete model. Nevertheless, communication and memory resources contentions cannot be completely ignored since their impact on the execution time can be very significant.

The methodology does not support modeling of run-time decisions, like the dynamic routing of the different communications on a network on chip. However, since one of the main properties of the SDF graphs is the static schedulability of actors, it seems natural to extend this property also to channels. In a similar way different types of buffer implementations are supported, from processing elements with local single port memory to complex memory architectures where each buffer implementation is independent.

The input of the proposed methodology is a SDF graph representing the application to be implemented and a complete design solution which must include three types of information:

- mapping and scheduling of the actors, i.e., to which processing element each actor has been assigned and in which order the actors mapped to the same processing element have to be executed;
- mapping and scheduling of the channels, i.e., to which communication device each channel has been assigned and in which order the data of channels mapped to the same device have to be transferred;
- allocation of buffers, i.e., to which (part of) device memory each buffer has been assigned.

How the design solution to be represented is generated is out of the scope of this paper. The output of the proposed methodology is the initial SDF graph enriched with all the information about the design solution.

The proposed methodology is described by Algorithm 1 where:

- each channel c_i is characterized as $c_i = \{s_i, t_i, p_i, d_i, i_i\}$ where s_i is the actor source, t_i is the target actor, p_i is the number of tokens produced by the source actor, d_i is the number of tokens consumed by the target actor, and i_i is the number of initial tokens of the channel;
- `AddActor(a,t)` adds an actor a whose execution time is t to the SDF graph;
- `AddChannel(c, s, t, p, d, i)` adds a channel c connecting s to t; p and d are the number of tokens produced and consumed by s and t respectively, i is the number of initial tokens;
- `AddScheduling(pe)` models the scheduling of a processing element pe using the technique presented in [2];

Algorithm 1: Code Analysis.

```
1  foreach c_i = {s_i, t_i, p_i, d_i, i_i} ∈ C do
2      if Mapping(s_i)≠Mapping(t_i) then
3          Remove(c_i)
4          AddActor(ac_i,Time(c_i,Mapping(c_i)))
5          AddChannel(nc1_i, s_i, ac_i, p_i, 1, 0)
6          AddChannel(nc2_i, ac_i, t_i, 1, d_i, i_i)
7      end
8  end
9  foreach processing element p_i do
10     AddScheduling(p_i)
11 end
12 foreach communication devices n_i do
13     AddScheduling(n_i)
14 end
15 foreach memory location m_j do
16     foreach writing w_i of m_j do
17         r_i =PreviousReading(m_j, w_i)
18         if GetActor(w_i) can be executed before GetActor(r_i)
           then
19             Serialize(GetActor(r_i), GetActor(w_i))
20         end
21     end
22 end
```

- `Mapping(a)` returns on which processing element an actor a is mapped;
- `Time(c,n)` returns the communication time for a single token of a channel c when implemented on a communication device n;
- `PreviousReading(m,w)` given a memory location m and a writing w returns the last reading which precedes w; if w is the first writing of the iteration, return the last reading of the (previous) iteration;
- `GetActor(o)` given a memory operation o returns the actor which performs it;
- `Serialize(a,b)` adds the actor and the channels necessary to force the postponing of execution of actor b after the ending of execution of actor a.

The following main phases can be identified in the algorithm:

1. *Communication modeling* (lines 1 - 8): this step models the delays of single communications in absence of resource contentions.
2. *Processing resources contention modeling* (lines 9 - 11): this step models the delays due to contention in accessing processing elements.
3. *Communication resources contention modeling* (lines 12 - 14): this step models the delays due to contention in exploiting communication infrastructure.
4. *Memory resources contention modeling* (lines 15 - 22): this step models the delays due to contention in exploiting buffers.

In the following each step of the algorithm will be described in detail and its application to an example will be shown. For the sake of simplicity, it is assumed that the target architecture considered for the example is composed of three homogeneous processors ($CPU1, CPU2, CPU3$),

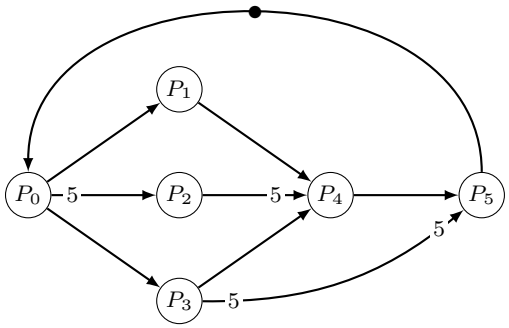

Fig. 1 Example of SDF graph.

each one with a local memory of 64KB, connected by a bus. The application example is described by the SDF graph of Fig. 1 (rates equal to 1 have been omitted): execution time of each actor is $15ms$, the transmission of a token on the bus requires $2.5ms$ and all the tokens are 8KB large. The mapping solution which has to be included in the SDF graph of Fig. 1 is: P_0, P_1, P_4 and P_5 on $CPU1$, P_2 on $CPU2$ and P_3 on $CPU3$. The scheduling solution, expressed as the *Periodic Static Order Schedule (PSOS)* of each processing element, is: $PSOS_{CPU1} = \{P_0, P_1, P_4, P_5\}$, $PSOS_{CPU2} = \{{P_2}^5\}$, $PSOS_{CPU3} = \{P_3\}$. Since the considered architecture contains only one bus, all the communications are mapped on it in this order: (P_0, P_2), (P_0, P_3), (P_3, P_5), (P_3, P_4), (P_2, P_4).

3.1 Communication modeling

This step of the proposed methodology (lines 1-8) aims at finalizing the detailed description of the considered solution in absence of resources contentions. Before this, the channels have to be classified in intra-processor and inter-processors. The formers connect actors mapped on the same processing element, so their communication delays can be considered null and they will be ignored in this phase of the methodology flow. The latters connect actors mapped on different processing elements, so they introduce communication delays in the application. For the sake of simplicity the approach proposed in [3] is applied to model the delay introduced by a communication. More complex approaches like the ones proposed in [4], [9] can however be integrated in the proposed methodology. Given a channel $c_i = \{s_i, t_i, p_i, d_i, i_i\}$, if s_i and t_i are mapped on different processing elements, c_i is removed from the SDF graph and replaced with two channels and one actor. Since the added actor represents the communication, its execution time represents the communication delay, so its actual value depends

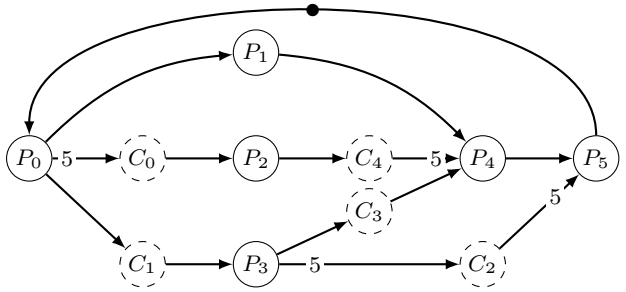

Fig. 2 Example of SDF graph with Communication Actors.

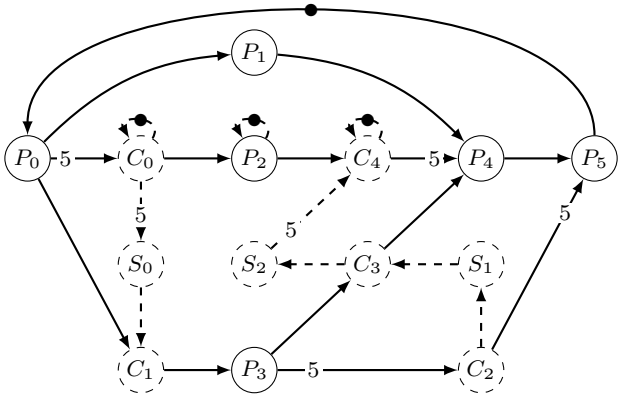

Fig. 3 Examples of SDF graph including modeling of resolution of communication resources contentions.

on the particular communication device on which it has been mapped. The delay to be considered is the transmission of a single token on the communication infrastructure when this is free (i.e., in absence of contention).

When mapping solution proposed in the previous section is applied to the example of Fig. 1, channels $(P_0, P_2), (P_0, P_3), (P_2, P_4), (P_3, P_4)$ and (P_3, P_5) become inter-processors channels. To model the delays, five actors C_0, C_1, C_2, C_3, and C_4 are added: the resulting SDF graph is shown in Fig. 2.

3.2 Processing resources contention modeling

In this phase of the proposed methodology (lines 9-11) the already pre-processed SDF graph describing the application is enriched by applying the methodology described in [2]. This methodology analyzes the SDF graph looking for actors mapped on the same processing element which can potentially be executed at the same time and adds the opportune actors and channels to force their serial execution. The order of execution to be forced is specified by the scheduling solution. The methodology assumes that all the instances of the same actor are mapped on the same processing element. If the mapping solution does not satisfy this assumption, the actors mapped on different processing elements are replicated so that all the instances of a replica are mapped on a single processing element. The result is a SDF graph where execution of actors mapped on the same processing element is serialized by the added actors and channels.

The only added channel for modeling the schedule in the example is the self-edges on actor P_2.

3.3 Communication resources contention modeling

Next step of the proposed methodology (lines 12-14) concerns modelization of resolution of communication resources contentions. The same approach adopted to model the scheduling on processing elements is adopted also for communication devices. It is not necessary to force the order of all the communications: some of them can be already serialized because of implicit dependences induced by original and previously added actors and channels. The execution time of actors added during this step is 0 since they do not model the implementation of any particular activity, but only force the sequential execution of data transfers.

In this intermediate step the communication actors have to be considered differently from the processing actors since the formers can be mapped on more than one resource. Duplication of communication actors to model this aspect cannot be adopted since it does not guarantee the contemporaneous usage of all the resources and so it does not guarantee the feasibility of the solution. For example, suppose to have an architecture with two buses connected by a bridge and suppose that the implementation of a channel uses both of them. If the usage of each bus would be represented by a separate actor, the second actor could be scheduled not immediately after the first. In this case a buffer would be required on the bridge to store the data after the first half of the communication, but this buffer can be not available on the target architecture, making the computed solution infeasible. In the proposed methodology, to guarantee the synchronization and the correct schedulability, the communication actors mapped on more than one resource are not duplicated, but each their instance is assigned to the Periodic Static-Order Schedules of all the resources on which they are mapped. In this way scheduling of communication actors is correctly modeled and there is no need to differentiate them from processing actors in the next step and in the produced Synchronous Data Flow graph.

Fig. 3 shows the results of applying all the methodology steps described until now to the example of Fig. 1. The self-edge of P_2 is added to model the scheduling of $CPU2$, the self-edges of C_0 and C_4 are added to model the impossibility of transferring tokens in parallel on the same channel. Actor S_0 is added to postpone the execution of communication C_1 after C_0; the other two actors (i.e., S_1 and S_2) are added to serialize the execution of C_2, C_3 and C_4 (channel (S_1, C_4) can be omitted because of transitivity). Note that there is no need to force the execution of C_2 after C_1 since this is already imposed by P_3.

3.4 Memory resources contention modeling

The last phase of the proposed methodology (lines 15-22) consists of integrating in the SDF graph information about resolution of memory devices contentions. One of the main advantages of adopting SDF graphs to represent design solutions consists of the possibility of statically scheduling the execution of the different actors and so of statically computing the corresponding throughput and buffer size boundaries. However, since memory can be a significantly limited resource in an embedded multiprocessor system, it is possible that in a design solution different buffers share the same physical memory locations. For this reason, computational and communication devices are not the only resources contended by different actors and channels. A design solution has to resolve the possible contentions of memory devices, otherwise an overestimation of the real throughput can occur or the implemented solution can produce erroneous results.

In the following it is assumed as most general case that all the input and all the output data of each actor can be allocated in a (distributed) shared memory. Usage of private memories potentially simplifies the problem since they reduced the possible resource contentions.

For each pair of actors a_i and a_j such that a_i can be executed before a_j (even not consecutively and even assigned to different resources), a design solution must satisfy the following conditions to guarantee its correct implementation:

- input and output data of a_i do not share any memory locations with input data of a_j or
- input data of a_i and input data of a_j share some memory locations, but a_i has already been executed when input data of a_j are written or
- output data of a_i and input data of a_j share some memory locations, but data produced by a_i have already been read when input data of a_j are written.

If none of the previous conditions is satisfied, the input data of a_j can overwrite other data that have not yet been read or can be overwritten before their reading potentially introducing computation errors. To avoid these issues, the possible contentions of memory resources have to be resolved in a design solution and information about the resolutions has to be included in the enriched SDF graph. Considering the statical schedulability of a SDF graph, the resolution of a contention simply consists of imposing the sequence of accesses to the contended memory locations such that at least one of the previous conditions holds. This sequence will be modeled in the enriched SDF graph by forcing the writing of a memory location to be executed after all its previous readings. Note that a reading can never be executed before the corresponding writing because of the dependence in the initial SDF graph. The actors and channels added in this phase can introduce deadlocks in the produced Synchronous Data Flow graph. This deadlock signals the infeasibility of the considered design solution due to a deadlock in acquisition of resources by two processing actors: design solution will have to be modified (e.g., by changing memory allocations or actor scheduling) to make it actually implementable.

For the sake of brevity, the details about the buffers mapping of the example has not been reported. Because of the limited size of the memory of CPU_1 where P_4 and P_5 are mapped, in all the possible buffer mappings the target buffers of channels (C_2, P_5) and (C_4, P_4) share some memory locations (the sum of their required memory is larger than the available). Usage of these memory locations have to be serialized to resolve their contention. Since C_2 is scheduled before C_4, the former will write data before the latter. P_5 has to read data written by C_2 before C_4 overwrites them, so P_5 has to be executed before C_4. The proposed methodology adds the actor M_1 to model this order as shown in Fig. 4 where the final SDF graph is presented.

This graph contains a cycle without initial tokens composed of C_4, P_4, P_5 and M_1 which introduces a deadlock: the produced SDF graph correctly models the actual infeasibility of the design solution. Note that transforming the solution by changing the execution order of C_2 after C_4 makes it implementable.

4 Experimental Evaluation

To evaluate the proposed methodology, this has been implemented in a C++ prototype which exploits a simple list-based heuristic [12] to generate the design solution to be represented. SDF^3 [13], a tool for the analysis and the transformation of SDF graphs, has been

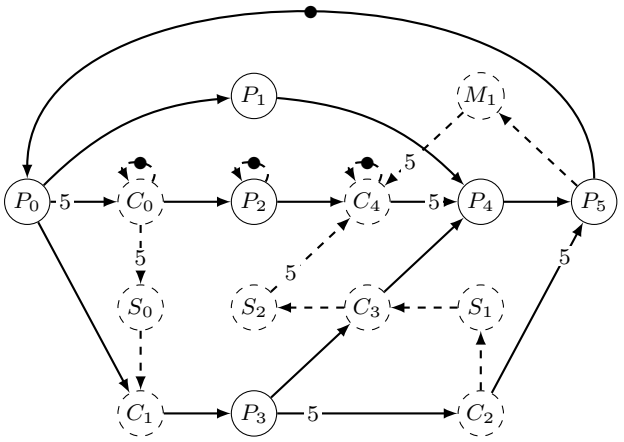

Fig. 4 Examples of SDF graph including modeling resolutions of all resources contentions.

exploited to compute the different throughputs after the graph transformations.

The methodology has been applied to the set of SDF graphs (distributed with SDF^3) modeling real applications: H263 Decoder [14] and Encoder [15], Modem Filter [16], MP3 Decoder [14], MP3 Playback [17], Sample-Rate Converter [16] and Satellite Receiver [18]. The characteristics of the analyzed SDF graphs are reported in Table 1.

Different target architectures have been considered. For the sake of simplicity and to highlight the effects of modeling communication infrastructure contention, all the considered architectures are composed of four homogeneous processors, but differ in the communication infrastructure which connects them and in the number of modules of local memories:

- *Crossbar*: there is a dedicated channel for each pair of processors; the local memory of each processor is infinite;
- *Network on Chip*: there are two parallel communication channels; each processor has two memory modules;
- *Shared Bus*: there is a unique communication channel; each processor has one memory module.

The communication time is modeled as linearly dependent on the size of the token and it is the same for all the architectures: in this way the effects of the communication resources contentions can be isolated and measured. Since the benchmarks require buffers with very different sizes, the memory modules have been differently sized for each analyzed benchmark (i.e., the target architectures for each benchmark are different) in order to obtain significant and realistic results: the size of a memory module is the smallest which allows to implement the selected mapping and scheduling solution.

For each benchmarks four SDF graphs are built:

- *Without Communication Contention*: it is the initial SDF graph enriched with scheduling actors modeling Periodic Static-Order Schedules of processing elements and communication actors modeling communication delays;
- *Crossbar*: it is the *Without Communication Contention* SDF graph enriched with actors modeling serialization of communications between each pair of processing elements;
- *Network on Chip*: it is the *Without Communication Contention* SDF graph enriched with actors modeling serialization of accesses to the Network on Chip and resolution of memory devices contentions;
- *Shared Bus*: it is the *Without Communication Contention* SDF graph enriched with actors modeling serialization of accesses to the shared bus and resolution of memory devices contentions.

The resolution of memory devices contentions has been modeled only in the last two graphs since in the first two presence of infinite memory is assumed.

The right part of Table 1 shows the number of added actors and channels with respect to the *Without Communication Contention* SDF graphs. Table 2 shows the computed throughput for each architecture and its difference with respect to the throughput computed ignoring resolution of communication and memory devices contentions. All the throughputs have been computed with SDF^3 since direct profiling or simulation of the applications is not possible (the actual implementations of the applications are not provided). It can be easily inferred that by adding a limited number of actors and channels it is possible to describe in a more detailed way the analyzed solution, obtaining much more accurate throughput results. The impact of communications and memory accesses serialization due to resources contentions can indeed be very significant and heavily depends on the communication and memory architectures. For example, the results on Sample-Rate Converter and Satellite Receiver show how this aspect can introduce a performance overhead of more than 50%.

Even admitting a complex network architecture which guarantees an high level of parallel communication, like in the case of *Crossbar* interconnection, the resources contentions cannot be ignored. Multiple parallel communications at a time are allowed but only if they do not share the same pair of processing elements. On the contrary, the communications between the same pair of processing elements (e.g., transfer of multiple tokens on

Table 1 Size of the analyzed SDF graphs.

	Without Communication Contention		Number of Added Actors and Channels					
			Crossbar		NoC		Shared Bus	
Benchmark	Actors	Channels	Actors	Channels	Actors	Channels	Actors	Channels
H.263 Decoder	7	13	0	+6	0	+6	+2	+11
H.263 Encoder	10	17	0	+5	+2	+11	+4	+17
Modem Filter	20	43	0	+4	+1	+7	+3	+12
MP3 Decoder (block level)	31	55	0	+17	+3	+26	+7	+36
MP3 Decoder (granule level)	31	55	0	+17	+4	+28	+5	+32
MP3 Playback	8	16	0	+4	+2	+10	+4	+15
Sample-Rate Converter	11	21	0	+6	+1	+9	+2	+12
Satellite Receiver	44	93	0	+5	+3	+14	+7	+22

Table 2 Effects of communication resources contentions on application throughput. *Without Communication Contention* is the throughput computed on the SDF graph which does not model communication resource contention; *Crossbar*, *Network on Chip* and *Shared Bus* are the throughputs computed on the graphs which model the resolution of resources contentions on the analyzed target architectures and their difference with respect to *Without Communication Contention* throughput.

	Throughput						
Benchmark	Without Communication Contention	Crossbar		Network on Chip		Shared Bus	
H.263 Decoder	$3.01 \cdot 10^{-6}$	$3.01 \cdot 10^{-6}$	0.00%	$3.01 \cdot 10^{-6}$	0.00%	$3.01 \cdot 10^{-6}$	0.00%
H.263 Encoder	$7.44 \cdot 10^{-7}$	$7.44 \cdot 10^{-7}$	0.00%	$7.44 \cdot 10^{-7}$	0.00%	$7.11 \cdot 10^{-7}$	3.99%
Modem Filter	$2.63 \cdot 10^{-2}$	$2.12 \cdot 10^{-2}$	19.39%	$2.12 \cdot 10^{-2}$	19.39%	$1.70 \cdot 10^{-2}$	35.36%
MP3 Decoder (block level)	$2.73 \cdot 10^{-7}$	$2.67 \cdot 10^{-7}$	2.25%	$2.67 \cdot 10^{-7}$	2.25%	$2.14 \cdot 10^{-7}$	19.85%
MP3 Decoder (granule level)	$2.74 \cdot 10^{-7}$	$2.68 \cdot 10^{-7}$	2.19%	$2.68 \cdot 10^{-7}$	2.19%	$1.93 \cdot 10^{-7}$	29.56%
MP3 Playback	$5.90 \cdot 10^{-6}$	$5.90 \cdot 10^{-6}$	0.00%	$5.90 \cdot 10^{-6}$	0.00%	$4.74 \cdot 10^{-6}$	19.66%
Sample-Rate Converter	$1.04 \cdot 10^{-3}$	$3.40 \cdot 10^{-4}$	67.31%	$3.40 \cdot 10^{-4}$	67.31%	$3.40 \cdot 10^{-4}$	67.31%
Satellite Receiver	$9.47 \cdot 10^{-4}$	$7.66 \cdot 10^{-4}$	19.11%	$5.24 \cdot 10^{-6}$	44.66%	$4.16 \cdot 10^{-6}$	56.07%

the same channel) have to be serialized decreasing the application throughput (like in Modem Filter, Sample-Rate Converter and Satellite Receiver). The throughput of other applications is instead influenced by modeling resolution of resources contentions only if this limits the number of parallel communications. Depending on their parallelism degree, their throughputs start to decrease when number of parallel communications is reduced to 2, (*Network on Chip*) like Satellite Receiver, or when it is reduced to 1 (*Shared Bus*), like Modem Filter and MP3 Decoder. The resolution of memory accesses contentions further reduce the throughput by making infeasible some of the design solutions initially computed by the list based heuristic. Finally, there are some applications (H.263 Decoder and H.263 Encoder) where modeling of resolution of resources contentions do not decrease significantly the computed throughput since communication takes much less than processing.

5 Conclusion

Evaluation of the performances of a streaming application can be easily obtained by computing the throughput of the corresponding SDF graph. In order to get accurate results just by analyzing this graph, information about the mapping, the scheduling, the communication and the buffers allocation of a design solution has to be integrated as much as possible in it. State of the art techniques well model independent transfers of data, but they do not take into account possible resources contentions.

In this paper, a methodology to include this information in the analyzed SDF graph is proposed. Experimental results show how the few actors and channels added to the SDF graph allow to significantly improve the accuracy of its analyses.

References

1. E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235 – 1245, sept. 1987.
2. Morteza Damavandpeyma, Sander Stuijk, Twan Basten, Marc Geilen, and Henk Corporaal. Schedule-Extended Synchronous Dataflow Graphs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol.32, no.10, pp.1495,1508, Oct. 2013
3. S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 777–782, New York, NY, USA, 2007. ACM.
4. P.K.F. Holzenspies, G.J.M. Smit, and J. Kuper. Mapping streaming applications on a reconfigurable mpsoc platform at run-time. In *System-on-Chip, 2007 International Symposium on*, pages 1 –4, nov. 2007.
5. Y. Sorel. Massively parallel computing systems with real time constraints: the Algorithm Architecture Adequation methodology". In *Massively Parallel Computing Systems, 1994.*, vol., no., pp.44,53, 2-6 May 1994
6. Maxime Pelcat, Jonathan Piat, Matthieu Wipliez, Slaheddine Aridhi, and Jean-François Nezan. An Open Framework for Rapid Prototyping of Signal Processing Applications. *EURASIP J. Embedded Systems*, January 2009.

7. N. K. Bambha, V. Kianzad, M. Khandelia, and S. S. Bhattacharyya. Intermediate representations for design automation of multiprocessor dsp systems. *Design Automation for Embedded Systems*, vol. 7, no. 4, pp. 307–323, 2002.
8. M. Lattuada, and F. Ferrandi. Modeling pipelined application with Synchronous Data Flow graphs *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII)*, pp.49,55, 15-18 July 2013.
9. I. Bennour, D. Sebai, and A. Jemai. Modeling sw to hw task migration for mpsoc performance analysis. In *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2010 5th International Conference on*, pages 1 –6, march 2010.
10. Jun Zhu, I. Sander, and A. Jantsch. Constrained global scheduling of streaming applications on mpsocs. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 223 –228, jan. 2010.
11. A.-H Ghamarian, M. C W Geilen, S. Stuijk, T. Basten, A.J M Moonen, M.J.G. Bekooij, B.D. Theelen, and M.R. Mousavi. Throughput Analysis of Synchronous Data Flow Graphs. *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pp.25,36, 28-30 June 2006.
12. Edward Ashford Lee and David G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, January 1987.
13. S. Stuijk, M. Geilen, and T. Basten. *sdf*^{3}: Sdf for free. In *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*, pages 276 – 278, june 2006.
14. Sander Stuijk, Marc Geilen, and Twan Basten. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *IEEE Trans. Comput.*, 57(10):1331–1345, October 2008.
15. Hyunok Oh and Soonhoi Ha. Fractional rate dataflow model and efficient code synthesis for multimedia applications. *SIGPLAN Not.*, 37(7):12–17, June 2002.
16. Shuvra S. Bhattacharyya, Praveen K. Murthy, and Edward A. Lee. Synthesis of embedded software from synchronous dataflow specifications. *J. VLSI Signal Process. Syst.*, 21(2):151–166, June 1999.
17. Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 658–663, New York, NY, USA, 2007. ACM.
18. S. Ritz, M. Willems, and H. Meyr. Scheduling for optimum data memory compaction in block diagram oriented software synthesis. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 4, pages 2651 –2654 vol.4, may 1995.