Throughput Evaluation of DSP Applications based on Hierarchical Dataflow Models

Hamza Deroui, Karol Desnos, Jean-François Nezan, Alix Munier-Kordon

Throughput Evaluation of DSP Applications based on Hierarchical Dataflow Models

Hamza Deroui, Karol Desnos and Jean-François Nezan
IETR, INSA Rennes
CNRS UMR 6164, UEB
Rennes, France
Email: hderoui, kdesnos, jnezan@insa-rennes.fr

Alix Munier-Kordon
Sorbonne Universites
UPMC, UMR 7606, LIP6
Paris, France
Email: alix.munier@lip6.fr

***Abstract*—Synchronous Dataflow (SDF) is the most commonly used dataflow Model of Computation (MoC) for the specification of Digital Signal Processing (DSP) systems. The Interface-Based SDF (IBSDF) model extends the semantics of the SDF model by introducing a graph composition mechanism based on hierarchical interfaces. Computing the throughput of an application is essential when designing DSP systems. This article introduces and assesses new methods to compute the throughput of DSP applications specified with IBSDF graphs. First, a basic method inspired from the state-of-the-art techniques that relies on a transformation of the IBSDF graph to an equivalent non-hierarchical graph of potentially exponential size. Second, a new technique that takes advantage of the hierarchy semantics of the IBSDF MoC to speed-up the throughput evaluation without any conversion. The proposed technique makes it possible to compute the throughput of large IBSDF graphs in a few milliseconds, where the basic method fails to produce a result.**

I. Introduction

Multiprocessor Systems-on-Chips (MPSoCs) are chips especially designed to support the specifications of complex applications in terms of computing power, power consumption, size and per-unit cost. The programming of MPSoCs is more and more complex due to the increasing number of Processing Elementss (PEs) and their heterogeneity.

The use of dataflow graphs gains popularity for the design and the programming of MPSoC [1]–[3]. Synchronous Dataflow (SDF) [4] is the first and the most studied dataflow model to describe applications in this context. SDF graph $G = \langle A, F \rangle$ decomposes an application into a set of actors A interconnected by a set of First-In First-Out queues (FIFOs) F to exchange data tokens. An actor is a computational entity, whose internal behavior is described using a traditional programming language, called host code. Each actor consumes (resp. produces) a fixed number of data-tokens on its input FIFOs (resp. output FIFOs) at each execution.

The popularity of SDF graphs is due to their decidability, which enables the use of compile-time analyses to verify key properties of applications, such as consistency (deadlock-freeness), schedulability and throughput [5], [6]. The expressivity limitation of SDF graphs and the increasing complexity of Digital Signal Processing (DSP) applications lead to the introduction of new dataflow Models of Computation (MoCs). The Interface-Based SDF (IBSDF) MoCs [7] extends the SDF MoCs with a hierarchy mechanism that enables the specification of the internal behaviour of actors with a SDF subgraph instead of host code. The hierarchy mechanism of IBSDF graph is based on interfaces that insulate each subgraph from its upper graph in term of schedulability.

In the design of real-time signal processing applications, the throughput is one of the required properties to be evaluated as early as possible by the developer. Very fast evaluation of this property is mandatory for real-time feedback to the developer during the application development, for the mapping/scheduling of the application on MPSoCs, and for the MPSoC Design Space Exploration (DSE) i.e. the research of the best hardware for a specific application.

This paper formalizes and assesses new methods to evaluate the throughput of IBSDF graphs. The first method is inspired from the state-of-the-art throughput evaluation methods for SDF graphs, based on a conversion of the IBSDF graph to a non hierarchical graph. Next, our main contribution is a new method named Schedule-Replace technique that take advantage of the semantics of the IBSDF MoC to compute its throughput without any conversion.

Section II presents basic definition of SDF properties and how to evaluate its throughput. The Hierarchical SDF (HSDF) and the IBSDF graph are presented in section II. In section III, we presents the conversion of an IBSDF graph to a non hierarchical graph and discuss how it can be avoided to compute the throughput with a Schedule-Replace technique. A performance comparison of the presented methods is presented in section IV. Section V concludes the paper.

II. Background

A. Consistency and Schedulability of SDF graph

Before computing the throughput of an SDF graph, the consistency of the graph must be verified. An SDF graph is said to be consistent when it can be executed without causing an infinite accumulation of data tokens in a bounded memory storage. In [4], the consistency is checked by solving the matrix equation $\Gamma * RV = 0$ where the topology matrix Γ represents the consumption and production rates of actors. RV is the Repetition Vector (RV). The elements of RV represents the number of executions needed for each actor to restore the initial marking: the data tokens already present in the FIFOs before a first execution of the graph. A graph iteration is

completed when each actor $a \in A$ is executed $RV(a)$ times without a lack of data tokens caused by an insufficient initial marking. In Figure 1, the graph composed by the three actors A, B, and C represents a consistent SDF graph for which the repetition vector is $RV = [2\ 3\ 3]$.

Once the consistency of a SDF graph is verified, a schedule to define the starting time of each actor execution is constructed. The As Soon As Possible (ASAP) schedule [8] is the most used schedule. It consists of executing actors as soon as there is enough data tokens on their input FIFOs. It allows the graph to reach its maximum throughput. The periodic schedule introduced in [9] for SDF graphs consists of defining a periodic execution for each actor. The advantage of periodic schedules σ is that the starting time $S^{\sigma}_{\langle a,k\rangle}$ of all executions k of an actor $a \in A$ are defined only by the starting time of the first execution $S^{\sigma}_{\langle a,0\rangle}$ and the execution period w^{σ}_{a} of a, such that:

$$\forall k \in \mathbb{N}^*, \quad S^{\sigma}_{\langle a,k\rangle} = S^{\sigma}_{\langle a,0\rangle} + (k-1)\cdot \omega^{\sigma}_{a}$$

However, the fact that SDF graphs may not expose all the parallelism of an application, the periodic schedule of an SDF graph does not guarantee a maximum throughput.

In contrast, the Single-Rate Synchronous Dataflow (srSDF) graph [6] exposes all the parallelism of an application by duplicating $RV(a)$ times each actor $a \in A$. Thus, a periodic schedule of the srSDF graph results in a maximum throughput execution. An algorithm for the conversion of SDF graph to srSDF graph is described in [6]. Figure 2a shows the equivalent srSDF graph of the SDF graph ABC of the Figure 1.

B. Throughput Evaluation of SDF graph

As the state-of-the-art throughput evaluation methods, all presented methods do not consider constraints on the number of PEs.

In [6], the maximum throughput of an SDF graph is defined as one divided by the Maximum Cycle Mean (MCM) of its equivalent srSDF graph. Efficient algorithms for calculating MCMs are compared in [10]. However, converting an SDF graph to a srSDF graph may results in an exponentially large graph which makes the MCM computation a havy task.

In [11], the throughput is computed with a method based on maxplus algebra. It consists of solving the Maximum Cycle Ratio (MCR) of the equivalent Linear Constraint Graph (LCG) of the SDF graph. The LCG is a conversion similar to the srSDF conversion. It expresses the SDF graph as a linear time-invariant maxplus system with less actors and edges than the srSDF graph.

In [8], a method to compute the maximum throughput without any conversion is introduced. The method simulates an ASAP schedule of the SDF graph which results in a transient phase followed by a periodic phase. The throughput is then computed as the duration of one iteration in the periodic phase. This method is efficient when the execution does not start with a long transient phase.

For large SDF graphs, computing the maximum throughput with the previous methods may failed due to the large size. Hence, an approximation of the maximum throughput can

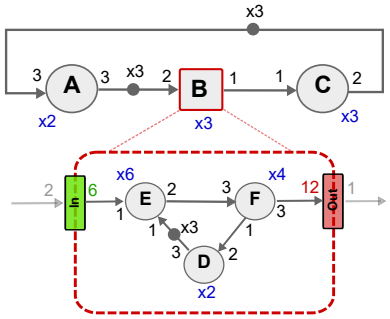


Fig. 1: Example of IBSDF graph in which actor B is described by an SDF subgraph.

be computed with the periodic schedule using the following equation:

$$Th^{\sigma^*}_{\max}(G) = \frac{1}{\max_{a_i \in A}\{\omega^{\sigma^*}_{a_i}\}}$$

where σ^* is the optimal periodic schedule [9] of the SDF graph which can be computed either by a mathematical model or by a polynomial time algorithm [10].

In this paper, [8] and [9] will be our base methods for the analysis of the IBSDF graph throughput.

C. Throughput Evaluation of HSDF graph

The HSDF MoC [5] was the first model to add a hierarchy mechanism to SDF graph. In the HSDF MoC, the hierarchy consists of encapsulating a set of SDF actors into a composite SDF actor that can be connected with other SDF actors. As described in [12], the HSDF MoC is not compositional: a composite SDF actor cannot be represented as an atomic SDF actor without loss of information that can lead to rate inconsistency or deadlock.

In [2], a clustering technique based on SDF composition theorem to transform a consistent SDF graph into an HSDF graph without creating deadlocks is proposed. The clustering technique allows to hierarchically schedule the graph which reduces the complexity of scheduling SDF graphs onto multiple processors.

The throughput evaluation of HSDF graphs is not addressed in the literature because of the non-compositionality of the graph. In this context the throughput is evaluated after converting the HSDF graph to an SDF graph.

D. Interface-Based SDF (IBSDF) model

The Interface-Based SDF (IBSDF) MoC [7] is a hierarchical extension of the SDF MoC. In the IBSDF MoC, the internal behaviour of actors can be specified either with host code, as it is with SDF, or with IBSDF subgraph. As presented in [7], for each input (resp. output) port of a hierarchical actor, an input (resp. output) interface is added in the associated subgraph. The purpose of interfaces is to transmit data tokens to and from a subgraph and to insulate levels of hierarchy in terms of consistency and schedulability analysis. To achieve this purpose, input interfaces may duplicate the same data tokens received from the upper level of hierarchy if the subgraph

requires more data tokens to complete an iteration. Similarly, output interfaces receiving more data tokens than necessary will only transmit to the upper level of hierarchy the number of data tokens defined by the hierarchical actor. In order to consume all data tokens available on the input interfaces and produce the number of data tokens required by the output interfaces, the SDF subgraph may execute several iterations. Figure 1 shows an example of IBSDF graph in which actor B is a hierarchical actor. The input interface duplicates 3 times the data tokens received from its parent actor B and the output interface transmits one data token while it receives 12 times the data tokens needed. In this context, the sub-actors E, F, and D are executed twice their minimum repetition factor.

Execution rules for the IBSDF graph to ensure the insulation property of the MoC are defined in [7]. An iteration of a subgraph cannot start if its input interfaces are not full. Similarly, the output interfaces cannot transmit data tokens to upper level if the subgraph iteration is not complete.

III. Throughput Evaluation of IBSDF graph

In this section we present first how to compute the throughput of IBSDF graphs with a srSDF conversion based method. Next, we show how to take advantage of the interface-based hierarchy to compute the throughput without any conversion.

A. Flat srSDF conversion based methods

A basic method to compute the throughput of an IBSDF graph is to transform it to a flat srSDF graph with no hierarchy i.e. flattening the hierarchy, and use state-of-the-art methods for SDF graphs to compute the throughput.

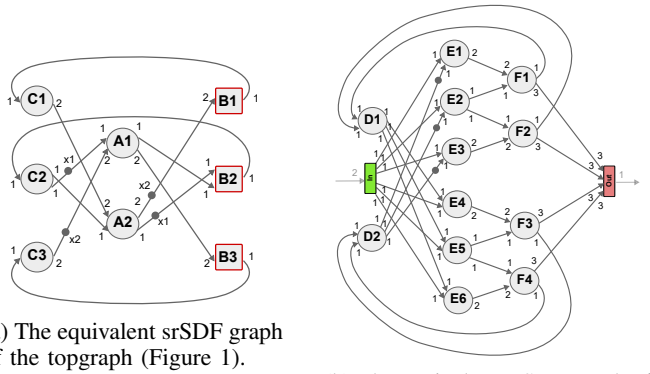
Converting an IBSDF graph to an equivalent flat srSDF graph consists of converting first the SDF topgraph, the graph of the upper level in the hierarchy, to srSDF graph and then replace each instance of a hierarchical actor with the equivalent srSDF graph of its SDF subgraph. This process is repeated for each level of the hierarchy to obtain the global flat srSDF graph. During the conversion, some extra edges are added to ensure the IBSDF execution rules. The conversion process may result in an exponentially large graph that makes the throughput evaluation an heavy task.

The equivalent flat srSDF graph of the IBSDF graph in Figure 1 is obtained by converting the topgraph to an srSDF graph as shown in Figure 2a, then the three instances of B are replaced with the srSDF graph of B subgraph shown in Figure 2b. The global srSDF graph contains 47 actors ($2A + 3 \times (1In + 2D + 6E + 4F + 1out) + 3C$) and 170 edges.

Once the IBSDF graph is converted, the throughput is computed by the ASAP based method [8] or by the Periodic Schedule [9]. For both methods the resulting throughput is maximum since the flat srSDF graph is used. The LCG conversion based method [11] is not usable since the graph is already converted to a flat srSDF graph.

B. Schedule-Replace method

The Schedule-Replace technique is based on constructing an ASAP schedule of the IBSDF graph in a bottom-up approach and computes the throughput as follows:



(a) The equivalent srSDF graph of the topgraph (Figure 1).

(b) The equivalent srSDF graph of the subgraph of B (Figure 1).

Fig. 2: The global flat srSDF graph of the IBSDF graph of the Figure 1 is obtained by replacing the instances $B1$, $B2$, and $B3$ of actor B with the srSDF graph version of its subgraph.

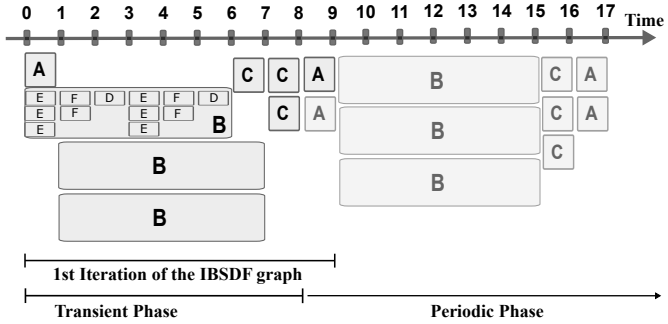


Fig. 3: ASAP schedule of IBSDF graph of the Figure 1, in which the duration of regular actors set to one unit time. The duration of actor B is equal to 6 time units.

1) Starting from the bottom level of the hierarchy up to the topgraph, for each level:
   a) Compute the duration of the hierarchical actors by scheduling their subgraph using [8] technique.
   b) Replace each hierarchical actor with a regular actor with the same duration computed in step 1a.
   c) Move to the upper level and repeat step 1a and 1b.
2) Convert the resulting SDF graph to a srSDF graph and add a self-loop edge for actors originally hierarchical to ensure the IBSDF execution rules.
3) Compute the throughput of the srSDF graph with [9].

Based on the IBSDF execution rules defined in section II-D, an execution of a hierarchical actor is defined as the execution of a complete iteration of its subgraph. It means scheduling an iteration of a subgraph in the hierarchy allows to measure the execution duration of its parent actor in the upper level.

Thanks to the compositionality feature of the interface-based hierarchy, each IBSDF subgraph can be scheduled independently. Hence, at each level of hierarchy, the duration of each hierarchical actors is defined by scheduling its subgraphs. Applying this principle, starting from the bottom level of the hierarchy up to the topgraph, results in an ASAP schedule of IBSDF graphs. Figure 3 shows the ASAP schedule of the

TABLE I: Performance comparison between the Schedule-Replace technique and the srSDF conversion based methods.

IBSDF Graph			srSDF	Exec.Time		
Name	Levels	Actors	Actors	ASAP [8]	Periodic [9]	Sched.-Rep.
Crypto	2	10	34	**4 ms**	8 ms	43 ms
Large FFT	2	10	267	**29 ms**	48 ms	46 ms
LTE	4	18	250	**22 ms**	32 ms	47 ms
Stereo	2	41	1604	3676 ms	151 ms	**51 ms**
Graph 1	3	15	503	493 ms	67 ms	**47 ms**
Graph 2	5	20	17727	>5 min	3060 ms	**46 ms**
Graph 3	6	24	84440	>5 min	14600 ms	**43 ms**
Graph 4	5	150	653289	>5 min	234000 ms	**69 ms**
Graph 5	8	240	39 E 10	-	-	**70 ms**
Graph 6	10	100	31 E 15	-	-	**73 ms**

IBSDF graph of Figure 1 in which, each execution of actor B is represented as one bloc abstracting its subgraph execution.

In terms of time analysis, a hierarchical actor can be replaced with a regular actor once its duration is defined. The replacement does not affect the ASAP schedule since the duration abstract the subgraph execution. Furthermore, replacing all the hierarchical actors of the topgraph allows to abstract the complete execution of the IBSDF graph in one SDF graph (step 1). The equivalent srSDF graph of the resulting SDF graph has the same execution time as the original IBSDF graph and so, the same throughput (step 2-3).

IV. Experimental Results

A. Experimental Setup

To evaluate the performance of our method to compute the throughput of IBSDF graphs, we compared the Schedule-Replace technique with the srSDF conversion followed by the state-of-the-art throughput evaluation methods [8], [9].

For each method we have measured the running time to compute the throughput including the srSDF conversion when relevant. Experimental results are summarized in Table I for two categories of graphs. The first category is a set of real DSP applications modeled as IBSDF graphs, available in [13]. The second category is a set of synthetics IBSDF graphs generated randomly using a newly developed generator based on Turbine tool [14]. The ASAP based method [8] used is the open source implementation of SDF^3 [15]. A mathematical model to compute the optimal period solved with the mathematical programming solver Gurobi [16] is used for the Periodic Schedule [9]. All methods were tested on one core of an Intel i5-6300 processor clocked at 2.40 GHz, and with 8GB of RAM. The Schedule-Replace method is implemented in the open-source Preesm framework [1].

B. Results

As Table I shows, the number of actors grows exponentially with the srSDF conversion. In fact, it was not possible to convert the last two synthetics graphs with the available RAM and so, to compute their throughput with [8] and [9] methods.

For small graphs, [8] is faster than [9] and Schedule-Replace technique. The results confirm that [9] is suitable for larger graph than [8]. However, the execution time of both methods [8] and [9] increases exponentially as the number of levels and the size of the srSDF graph grow.

Since the Schedule-Replace technique does not rely on the srSDF conversion, it computes the throughput of all the graphs in a few milliseconds. For the Stereo-Matching DSP application, the Schedule-Replace technique is 3 times faster than [9] and 70 times faster than [8].

V. Conclusion

We have introduced a new method named Schedule-Replace technique to compute the throughput of IBSDF graphs without any complex graph conversion. Experiments show that new method outperforms methods based on srSDF conversion for a fast evaluation of the throughput of large IBSDF graphs. With this work, the developer is able to analyze in real-time the performance of DSP applications during the development process and thus accelerate the MPSoCs DSE.

A future work is to extend the Schedule-Replace technique for throughput evaluation under constrained PE or memory resources.

References

[1] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J.-F. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore dsp programming," in *Embedded Design in Education and Research Conference (EDERC)*, Sept 2014, pp. 36–40.

[2] J. L. Pino, S. S. Bhattacharyya, and E. A. Lee, *A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs*. Electronics Research Laboratory, College of Engineering, University of California, 1995.

[3] M. A. Aguilar, R. Leupers, G. Ascheid, and L. G. Murillo, "Automatic parallelization and accelerator offloading for embedded applications on heterogeneous mpsocs," in *Design Automation Conference*, jun 2016.

[4] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, Sept 1987.

[5] S. S. Battacharyya, E. A. Lee, and P. K. Murthy, *Software Synthesis from Dataflow Graphs*. Kluwer Academic Publishers, 1996.

[6] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2nd ed. CRC Press, Inc., 2009.

[7] J. Piat, S. Bhattacharyya, and M. Raulet, "Interface-based hierarchy for synchronous data-flow graphs," in *IEEE Workshop on Signal Processing Systems. SiPS*, 2009, pp. 145–150.

[8] A. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Sixth International Conference on Application of Concurrency to System Design. ACSD*, 2006, pp. 25–36.

[9] A. Benabid-Najjar, C. Hanen, O. Marchetti, and A. Munier-Kordon, "Periodic schedules for bounded timed weighted event graphs," *IEEE Transactions on Automatic Control*, vol. 57, no. 5, pp. 1222–1232, 2012.

[10] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 4, pp. 385–418, Oct. 2004.

[11] R. de Groote, J. Kuper, H. Broersma, and G. J. M. Smit, "Max-plus algebraic throughput analysis of synchronous dataflow graphs," in *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, Sept 2012, pp. 29–38.

[12] S. Tripakis, D. Bui, M. Geilen, B. Rodiers, and E. A. Lee, "Compositionality in synchronous data flow," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 3, p. 83, 2013.

[13] "Preesm." [Online]. Available: http://preesm.sourceforge.net/website/

[14] B. Bodin, Y. Lesparre, J.-M. Delosme, and A. Munier-Kordon, "Fast and efficient dataflow graph generation," in *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '14. ACM, 2014, pp. 40–49.

[15] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF for free," in *Sixth International Conference on Application of Concurrency to System Design. ACSD*, 2006, pp. 276–278.

[16] "Gurobi Optimization." [Online]. Available: http://www.gurobi.com/