Relaxed Subgraph Execution Model for the Throughput Evaluation of IBSDF Graphs

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

Relaxed Subgraph Execution Model for the Throughput Evaluation of IBSDF Graphs

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*—The Interface-Based Synchronous Dataflow (IBSDF) Model of Computation (MoC) is a hierarchical extension of the well-known Synchronous Dataflow (SDF) MoC. The IBSDF model extends the semantics of the SDF model by introducing a graph composition mechanism based on hierarchical interfaces. The IBSDF model introduces also execution rules to ease the analysis of the IBSDF graph such as evaluating the throughput; an essential key performance to evaluate when designing Digital Signal Processing (DSP) systems. However, respecting the execution rules may slow down the execution of IBSDF graphs, and so stop the applications to reach their maximum throughput. This article presents first how to speed-up the execution of an IBSDF graph by relaxing the execution rules. Second, a new method to compute the throughput of IBSDF graphs under a relaxed execution. Finally, a performance comparison between the proposed method and basic methods that rely on a transformation of the IBSDF graph to an equivalent non-hierarchical graph of potentially exponential size. The proposed method outperforms basic methods and makes it possible to evaluate the maximum throughput of large IBSDF graphs in less than 2 seconds.**

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, schedulability and throughput [5], [6]. The Interface-Based Synchronous Dataflow (IBSDF) Models of Computation (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. Additionally to interfaces, the IBSDF model define execution rules to ease the analysis of the graph.

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.

A basic method to compute the throughput of an IBSDF graph is to transfomr it to a non hierarchical graph (flattening the hierarchy) and use [8] and [9] methods for SDF graphs. [10] shows that this transformation results in a exponential large graph for which basic methods fail to compute the throughput. In [10] a method named Schedule-Replace (SR) was introduced to compute the IBSDF graph throughput without flattening its hierarchy. The SR method takes advantage of the execution rules and outperforms basic methods.

However, respecting the execution rules may slow down the execution of the IBSDF graph and stop the application to reach its maximum throughput. In this paper we show how to speed up the execution of the IBSDF graph by relaxing the execution rules without affecting the applications computation. By relaxing the execution rules, SR technique is not suited to compute the IBSDF graph throughput. Hence, we introduce a new method named Evaluate-Schedule-Replace (ESR) based on SR technique to compute the IBSDF graph throughput under a relaxed execution.

Section II presents basic definition of SDF properties and how to evaluate its throughput. The IBSDF model is presented in section III. In section IV, we present how to flatten the hierarchy of an IBSDF graph and use basic methods to compute its throughput. Section V presents the advantages of the Schedule-Replace technique. Section VI presents a relaxed execution for IBSDF graphs. Section VII introduce our new method. A performance comparison of the presented methods is presented in section VIII. Section IX concludes the paper.

II. Synchronous Dataflow (SDF) graph

A. Consistency and liveness

An SDF graph is an oriented graph $G(A, F)$. The vertices A, called actors, model the computations of the application. The edges F model the FIFOs that allow actors to exchange data tokens. The SDF graph G can be used in all steps of the MPSoC programming. The first step before computing the throughput of the application is to verify the consistency and the liveness of the SDF graph i.e. checking if the application is deadlock free. An SDF graph G 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 $\Gamma(G)$ represents the consumption and production rates of the actors. RV is the Repetition Vector (RV). The elements of RV represent the number of executions needed for each actor to restore the initial marking, that is to say the same number of data tokens in the FIFOs before the first execution of the application. An SDF graph G is said to be live when each actor $a \in A$ can be executed $RV(a)$ times (a graph iteration) without a lack of data tokens caused by an insufficient initial marking. In Figure 1, the graph $ABCD$ composed by the four actors A, B, C, and D represents a consistent SDF graph with an $RV = [1\ 2\ 2\ 2]$ and an initial marking $[1, 0, 0, 0, 1]$ for which the graph is live.

B. Throughput evaluation

There are many methods to compute the throughput of SDF graphs, which can be found in [6], [8], [9], [11], [12]. Most of these methods are based on two techniques: simulating a schedule, and analysing the graph structure. In this paper we are interested in the two methods [8] and [9] based on the As Soon As Possible (ASAP) schedule and the Periodic schedule.

*1) **ASAP Schedule [8]**:* Is the most used schedule. It consists in executing actors as soon as there is enough data tokens on their input FIFOs. The ASAP schedule results in a transient phase followed by a periodic phase in which the application reaches its maximum throughput. Computing the throughput with [8] method consists of simulating an ASAP schedule of the SDF graph until it reaches a periodic phase. Next, computing the throughput as :

$$Th(G) = \frac{Nb\ of\ graph\ iterations\ in\ one\ period}{The\ duration\ of\ one\ period}$$

*2) **Periodic Schedule [9]**:* It consists in defining a periodic execution for each actor. Hence, the starting time $S^{\sigma}_{\langle a,t\rangle}$ of all executions t of an actor $a \in A$ are defined 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:

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

An optimal periodic schedule σ^* is defined based on the structure of the SDF graph. Then, the throughput is computed as $Th^{\sigma^*}(G) = 1/\max_{a\in A}\{\omega^{\sigma^*}_{a}\}$. This method is suitable for large SDF graphs than the ASAP based method.

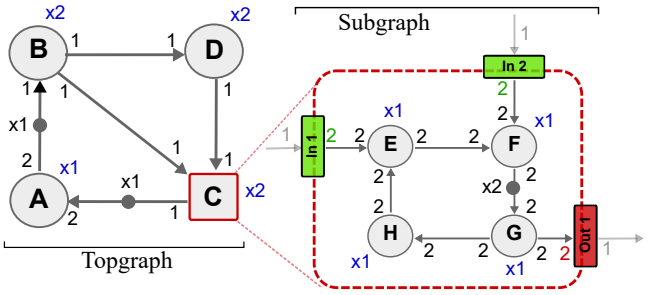


Fig. 1: Example of IBSDF graph in which actor C is described by an SDF subgraph of 4 actors.

III. Interface-Based SDF (IBSDF) graph

A. The interface-based hierarchy

The IBSDF MoC [7] is a hierarchical extension of the SDF MoC. In the IBSDF graph, the internal behaviour of actors can be specified either with host code, as it is usually done in SDF graphs, or with IBSDF subgraphs. 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 will duplicate a data token from the upper level of hierarchy if the subgraph requires several 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 to produce the number of data tokens required by the output interfaces, subactors may be executed n times their minimum repetition factor.

Figure 1 shows an example of IBSDF graph in which the actor C is a hierarchical actor described by an SDF subgraph of 4 actors. Executing one iteration of the subgraph results in consuming and producing 2 data tokens on its interfaces. In consequence, input interfaces $In1$ and $In2$ duplicate 2 times the data tokens received from their parent actor C. To respect the production rate of the actor C, the output interface $Out1$ will transmit only one data token even if it receives 2 from the actor G at each iteration of the subgraph.

B. Execution rules for IBSDF graphs

The data tokens duplication/ignorance behaviour of the interfaces guarantees an IBSDF graph composition that is entirely consistent if each SDF graph in the hierarchy is consistent. Additionally, [7] defines three execution rules to ease the liveness evaluation and the schedulability of the IBSDF graph. These execution rules are as follows:

- **Rule 1**: A subgraph can start its execution only when enough data tokens are available on all input FIFOs of its parent actor.
- **Rule 2**: Output interfaces can transmit data tokens in the output FIFOs of their parent actor only when the subgraph finishes its iteration.

- **Rule 3**: A subgraph executes only one iteration at a time for each firing of its parent actor.

With the execution rules described above, a hierarchical actor behaves like a regular actor consuming and producing data tokens. Hence, each subgraph can be evaluated and scheduled independently from the hierarchy. Furthermore, if each subgraph in the hierarchy is live, it is sufficient for proving the liveness of the entire IBSDF graph.

IV. Throughput evaluation by flattening

This section presents a basic method to compute the throughput of an IBSDF graph using SDF graphs methods. The method consists in transforming the IBSDF graph to a non hierarchical graph.

A. srSDF / HSDF conversion

The Single-Rate Synchronous Dataflow (srSDF) graph is a conversion of the SDF graph, which exposes all the parallelism of an application by duplicating $RV(a)$ times each actor $a \in A$. The Homogeneous SDF (HSDF) graph is a simplified srSDF graph in which all the consumption and production rates are set to 1. An algorithm for converting an SDF graph to a srSDF / HSDF graph is described in [6]. Figure 2a shows the equivalent srSDF graph of the SDF graph $ABCD$ of Figure 1.

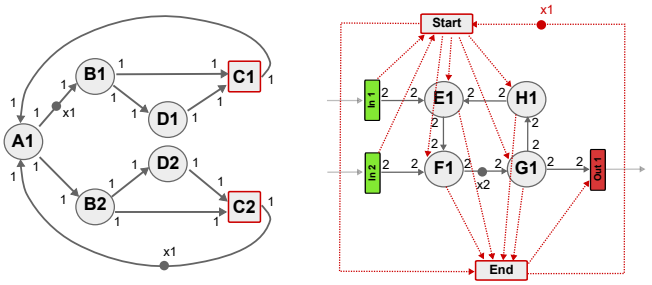
B. Flattening the IBSDF graph hierarchy

Transforming an IBSDF graph into an equivalent non hierarchical graph (i.e flattening the hierarchy) is a preliminary step to evaluate the IBSDF graph throughput with state-of-the-art methods of SDF graphs. The transformation consists in converting the IBSDF graph to an equivalent flat srSDF graph. To obtain the equivalent flat srSDF graph, the two following steps are repeated for each level of the hierarchy beginning by the top level up to the bottom one:

- Step 1: Convert the SDF topgraph to a srSDF graph.
- Step 2: Replace each instance of a hierarchical actor by the equivalent srSDF graph of its SDF subgraph.

During the srSDF conversion of the subgraphs, extra actors $Start$ and End, and extra edges are added to ensure the compliance with the three execution rules of the IBSDF graph. The $Start$ actor enforces *Rule 1* by imposing a precedence relationship between all input interfaces of the subgraph on one side, and all subgraph actors on the other side. The End actor enforces *Rule 2* by imposing a precedence relationship between all subgraph actors on one side, and all output interfaces of the subgraph on the other side. *Rule 3* is enforced by adding precedence relationships between the $Start$ and the End actor. Figure 2b shows the resulted srSDF subgraph of figure 1 after adding the extra actors and the extra edges.

Figure 3a shows the equivalent flat srSDF graph of the IBSDF graph of figure 1. The flat srSDF graph is obtained by first, converting the topgraph to a srSDF graph (fig 2a). Next, by replacing $C1$ and $C2$, the two instances of actor C, by its srSDF subgraph with extra actors and extra edges (fig. 2b). The flat srSDF graph contains 23 actors and 50 edges.



(a) The srSDF graph version of the IBSDF topgraph of Fig. 1.

(b) The srSDF subgraph with extra actors and extra edges.

Fig. 2: srSDF graph version of the topgraph and subgraph of fig. 1.

C. Throughput evaluation by a basic method

A basic method to evaluate the IBSDF graph throughput is to compute the maximum throughput of its equivalent flat srSDF graph using ASAP schedule [8] and periodic schedule [9] based methods for SDF graphs.

*1) **ASAP (Flat srSDF) [8]:*** It consists in simulating the ASAP schedule of the equivalent flat srSDF graph until it reaches the periodic phase. Figure 3b shows the ASAP schedule of the equivalent flat srSDF graph in figure 3a. One iteration of the flat srSDF graph takes 7 times unit. Hence, the IBSDF graph throughput is $1/7$ iteration per times unit.

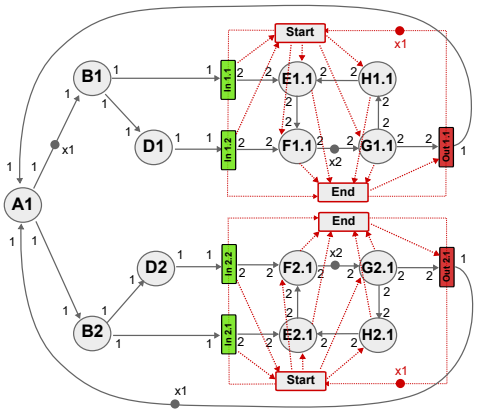
*2) **Periodic (Flat srSDF) [9]:*** In the case of srSDF graphs, all the actors have the same execution period denoted K when applying a periodic schedule. Hence, the throughput is equal to $1/K$. The value of K is computed as the value of the Maximum Cost-to-Time Ratio (MCR) cycle [13] of the HSDF graph version of the srSDF graph. For the flat srSDF graph in figure 3a, the MCR value of its HSDF graph is $7/1$; the value of the cycle $\{$A1, B1, D1, In2, Start, G1.1, H1.1, E1.1, F1.1, End, Out1$\}$. Thus, $K = 7$ and so the IBSDF graph throughput equals to $1/K = 1/7$ iteration per times unit.

V. Schedule-Replace (SR) technique

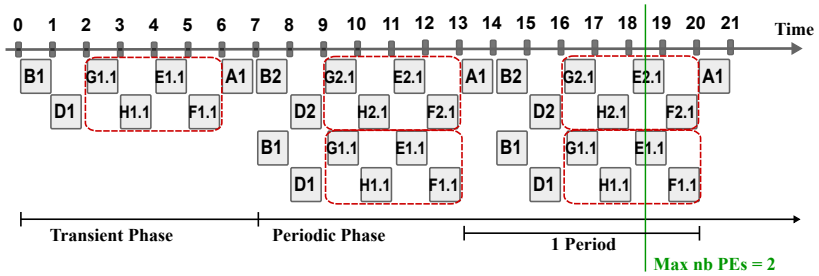
Computing the IBSDF graph throughput with the basic methods [8] and [9] requires to flatten the hierarchy. [10] shows that the flattening process results in an exponentially large flat srSDF graph. Which makes the throughput computation a hard task for the basic methods. [10] introduced a Schedule-Replace (SR) technique that takes advantage of the interface-based hierarchy and the execution rules to compute the IBSDF graph throughput without flattening its hierarchy.

The Schedule-Replace (SR) technique consists of first, scheduling an iteration of a subgraph to measure its duration. Next, replacing the hierarchical parent actor by a regular actor with the same duration of its subgraph. This process is repeated through the hierarchy in a bottom-up approach to abstract the complete execution of the IBSDF graph in its topgraph. Then, computing only the topgraph throughput is equivalent to evaluate the throughput of the IBSDF graph.

The Schedule-Replace (SR) [10] technique do not require any complex srSDF conversion and so outperforms the basic methods [8] and [9] that rely on flattening the hierarchy.

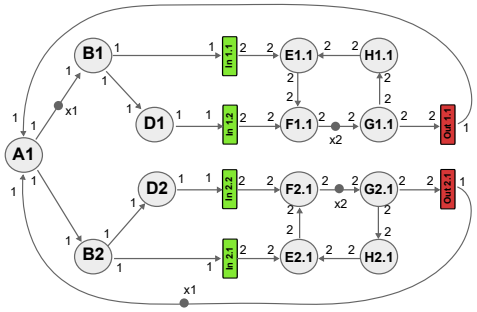


(a) The equivalent flat srSDF graph of the IBSDF graph of Figure 1, with extra edges.

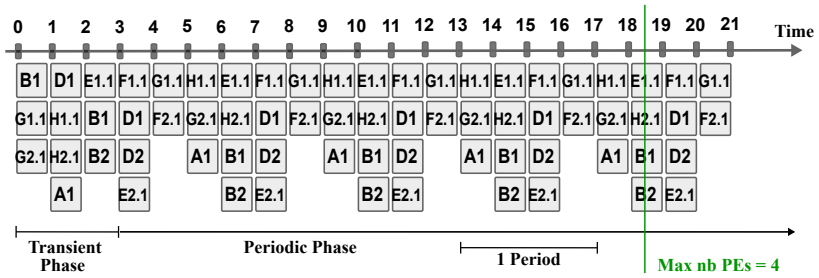


(b) ASAP schedule of the equivalent flat srSDF graph of figure 3a.

Fig. 3: ASAP schedule of the IBSDF graph of figure 1, respecting the execution rules.



(a) The equivalent Relaxed srSDF graph of the IBSDF graph of Figure 1, without extra edges.



(b) ASAP schedule of the equivalent relaxed srSDF graph of figure 4a.

Fig. 4: ASAP schedule of the IBSDF graph of figure 1 under a relaxed execution.

VI. Relaxed execution for IBSDF graphs

Th execution rules allows to ease the analysis of the IBSDF graph, especially the liveness evaluation. However, they may slow down the execution of the IBSDF graph by constraining the execution of subgraph actors. Hence, the execution rules may stop the application to reach its maximum throughput.

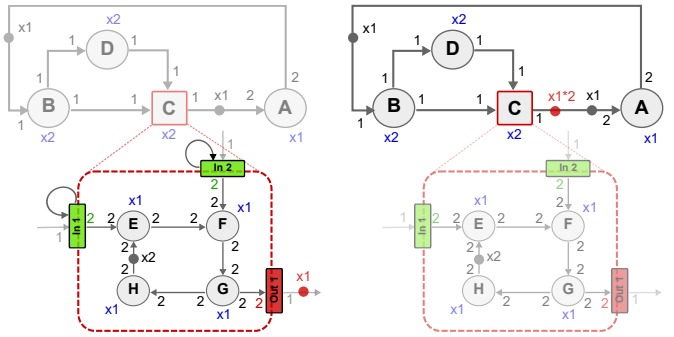
Otherwise, the execution rules can be relaxed once the consistency and the liveness of the IBSDF graph are verified. By relaxing the execution rules, each subgraph actor is free to execute as soon as there is enough data tokens on its input FIFOs including input and output interfaces. In this way, applications can reach their maximum throughput.

The execution rules are relaxed by deleting the extra actors and the extra edges from the subgraphs. Figure 4a shows the equivalent relaxed srSDF of the IBSDF graph of figure 1. Figure 4b shows the ASAP schedule of the relaxed srSDF graph. As this two figures show, subgraph actors $G1.1$ and $G2.1$ are free to execute at $t0$ because $Rule\ 1$ is relaxed. Similarly by relaxing $Rule\ 2$, the output interfaces $Out1.1$ and $Out2.1$ are free to transmit data tokens immediately after the execution of $G1.1$ and $G2.1$. And so, actor $A1$ can be executed right after that at $t1$. Relaxing $Rule\ 1$ allows the input interface $In2.1$ to transmit immediately the data tokens received from actor $B2$ to the subgraph actor $E2.1$. The input interface $In2.1$ does not need to wait until the input interface $In2.2$ receives data tokens from actor $D2$. Thus as the schedule shows, subgraph actor $E2.1$ and actor $D2$ begin to execute simultaneously after the execution of actor $B2$ at $t2$.

When the relaxed srSDF graph reaches its periodic phase, a graph iteration takes a period of 4 times unit (fig 4b). Which means the throughput of the IBSDF graph is $1/4$ iterations per times unit. Comparing to the previous execution (fig. 3b), relaxing the execution rules speeds up the application by a factor of $7/4 = 1.75$ without changing the application design.

Relaxing the execution rules allows to fully exploit the parallelism of the application and use more PEs if needed. Comparing the two schedules of figures 3b and 4b, the relaxed execution allows to use up to 4 PEs in parallel instead of 2.

Since subgraph interfaces can start to execute separately at different times, hierarchical actors cannot be replaced by regular actors. As consequences, the SR technique [10] is not suited to compute the throughput of relaxed executions. And so, only basic methods [8] and [9] can compute the maximum throughput. In the next section we introduce a new method named Evaluate-Schedule-Replace (ESR) to evaluate the throughput of IBSDF graphs under a relaxed execution.



(a) Execute the IBSDF subgraph until it stops.

(b) Move the transmitted data tokens to the upper graph.

Fig. 5: Synchronize the execution of the IBSDF subgraph of figure. 1

VII. Evaluate-Schedule-Replace (ESR) technique

A. methods algorithm

As an alternative to basic methods [8] and [9], we developed a new method named Evaluate-Schedule-Replace (ESR) to compute the IBSDF grah throughput under a relaxed execution. The new method is based on the Schedule-Replace (SR) technique [10] and consists of the following steps:

- **Phase 1** Synchronize the IBSDF subgraphs execution
- **Phase 2** Starting from the bottom level of the hierarchy up to the topgraph, for each level:
  1) Construct a replacement graph for each hierarchical actor in the current graph
  2) Convert the current graph to a srSDF graph and Replace each instance of hierarchical actors by their replacement graph
- **Phase 3** Compute the topgraph throughput with [9]

Similarly to SR technique, the ESR method analyse the IBSDF graph level by level to compute its throughput. However, the new method replaces a hierarchical actor by a small graph modelling the different start time of its subgraph interfaces.

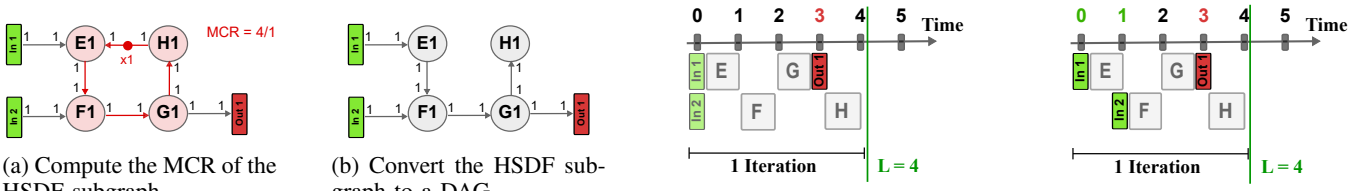
Algorithm 1 shows an implementation of the ESR method. The first phase is mandatory to analyse independently the subgraphs. It consists of moving up the data tokens that are ready to be transmitted from the subgraphs to their upper graphs. Starting from the bottom level of the hierarchy up to the topgraph, each subgraph is executed until it stops. Each time an output interface is executed, the data tokens to transmit are moved from the subgraph to its upper graph. A temporary empty self-loop edge is added to actors without input edges and to input interfaces for preventing an infinite execution of subgraphs during this phase. At the end of phase 1, the current state of the subgraphs is saved as the new initial marking. In the IBSDF graph, each instance of a hierarchical actor is described by an instance of its subgraph. Therefor each data token moved from a subgraph to an upper level is multiplied by the repetition factor of the hierarchical actor parent of the subgraph. Figure 5 shows the synchronization of the IBSDF subgraph of figure 1. Figure 5b represent the final state of the IBSDF graph which will be used for the throughput evaluation.

Algorithm 1 Evaluate-Schedule-Replace (ESR) Pseudocode

```
ListReplacementGraphs = new LIST(Actor, Graph)
                                        ▷ Evaluate an IBSDF graph
function EVALUATE(IBSDF)
    SYNCHRONIZESUBGRAPHS(IBSDF)
    for all actors a ∈ IBSDF.TopGraph do
        if a is hierarchical actor then
            PROCEED(a)
        end if
    end for
    srSDF = CONVERTTOSRSDF(IBSDF.TopGraph)
    for all actors a ∈ srSDF do
        if a is hierarchical actor then
            REPLACE(a, ListReplacementGraphs)
        end if
    end for
    throughput = COMPUTETHROUGHPUT(srSDF)
    return throughput
end function
                                        ▷ Proceed a hierarchical actor
procedure PROCEED(H)
    for all actors a ∈ H.SubGraph do
        if a is hierarchical actor then
            PROCEED(a)
        end if
    end for
    srSDF = CONVERTTOSRSDF(H.SubGraph)
    for all actors a ∈ srSDF do
        if a is hierarchical actor then
            REPLACE(a, ListReplacementGraphs)
        end if
    end for
    HSDF = CONVERTTOHSDF(srSDF)
    K = COMPUTEMCR(HSDF)
    DAG = CONVERTTODAG(HSDF)
    schedule = ASAP&ALAPSCHEDULE(DAG)
    replacementGraph = CONSTRUCTGRAPH(schedule,K)
    ListReplacementGraphs.add(H,replacementGraph)
end procedure
```

The second phase represents the core algorithm of the ESR method. It consists of evaluating the maximum execution behaviour of a subgraph, based on its structure. Modelling it by a relaxed subgraph execution model that abstracts the subgraph structure and shows only the execution time difference between its input and output interfaces. Finally, replacing the hierarchical parent actor of the subgraph by by its relaxed subgraph execution model. This process is repeated for each level starting from the bottom level up to the topgraph. At the end of $Phase\ 2$ the topgraph abstracts the complete hierarchy of the IBSDF graph. $Phase\ 3$ computes the throughput of the topgraph to determine the IBSDF graph throughput.

In the following, we presents how to construct the equivalent relaxed subgraph execution model of a subgraph.



(a) Compute the MCR of the HSDF subgraph

(b) Convert the HSDF subgraph to a DAG

(c) ASAP Schedule

(d) ALAP Schedule

Fig. 6: Schedule the subgraph using ASAP + ALAP schedule to identify the execution start time of its interfaces.

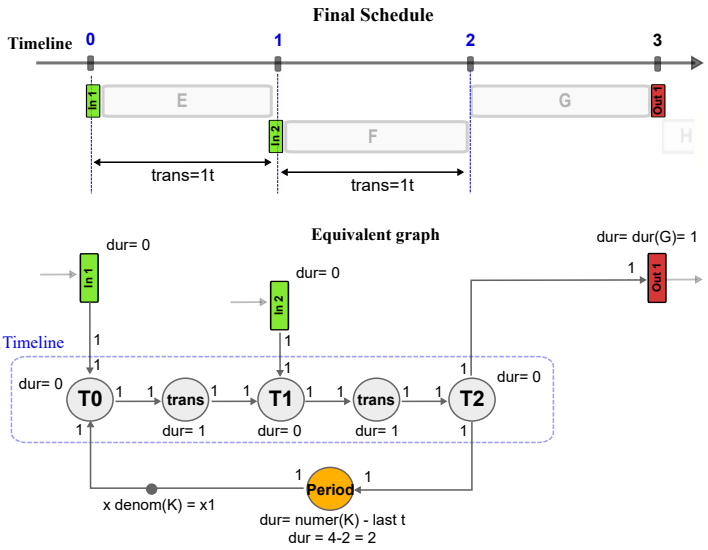


Fig. 7: Construct the time line, connect the interfaces and add the execution period constrain to models the subgraph execution.

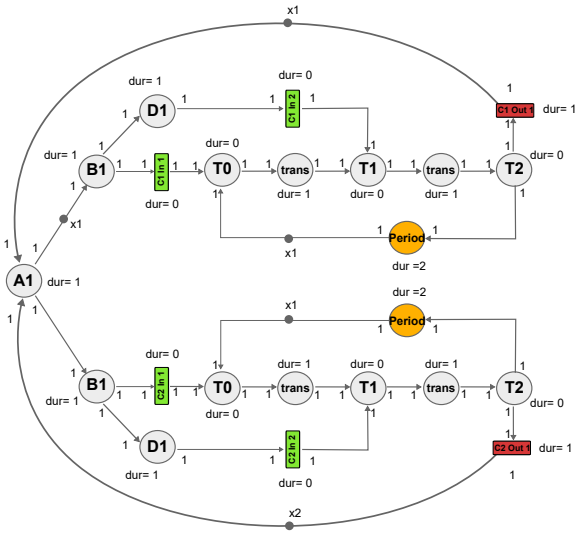


Fig. 8: Replace each instance of the hierarchical actor C by its subgraph execution model.

B. Relaxed subgraph execution model

Despite the precedence relationship of a hierarchical actor, the subgraph cannot exceed the maximum relaxed execution defined by its own structure. The purpose of the relaxed subgraph execution model is to abstract the subgraph structure by representing only the maximum throughput of the subgraph and the execution time difference between its interfaces. Which is all required for the throughput evaluation.

The construction of an equivalent relaxed subgraph execution model consists of the following steps:

1) Convert the subgraph to a srSDF graph and compute the minimum execution period of its actors using [9]
2) Schedule the srSDF subgraph using ASAP followed by As Late As Possible (ALAP) schedule
3) Construct the equivalent execution model using the schedule and the minimum execution period obtained in the previous steps

Step 1 consists of computing the the minimum execution period K of the srSDF subgraph using the periodic schedule [9]. The K value represents the maximum throughput $1/K$ and the minimum period to respect between two successive executions of subgraph actors. Figure 6a shows the MCR of the HSDF subgraph which defines the value of $k = 4$.

Step 2 consists of scheduling the subgraph by an ASAP followed by ALAP schedule to define the minimum time difference between its interfaces execution. The ASAP schedule allows to define the minimum time that output interfaces take to transmit data tokens when the subgraph starts an iteration. Scheduling the subgraph with ALAP schedule allows to define the maximum delay for input interfaces to execute without affecting the execution of output interfaces. To use existent algorithm of ASAP and ALAP schedules, the HSDF subgraph of step 1 is converted to a Directed Acyclic Graph (DAG). The DAG version of the HSDF subgraph is obtained simply by removing all FIFOs that contain initial marking (see fig. 6b). Figures 6c and 6d show the ASAP and ALAP schedule of the IBSDF subgraph of figure 1. The final schedule highlight only the execution of the interfaces, which is all required in step 3.

Step 3 constructs the subgraph execution model by representing the final schedule with an equivalent srSDF graph. The start time of each interface is represented by an actor $(TimeLineActor)$ with a null duration. The time difference between two consecutive executions of interfaces in the time line is represented by an actor $(TransitionActor)$ with the duration equals to that time difference. Each interface is connected to the associated actor of its execution start time. For time dependency, the execution start time of the source actor of output interfaces is considered instead of their start time. Similarly, output interface take the execution duration of their source actor. A final actor, named $PeriodActor$, is added to model the minimum execution period of the subgraph

actors by creating a cycle with the $TimelineActors$. The duration of the $PeriodActor$ is computed as $numerator(K) - \sum dur(TransitionActor)$. And by adding $denominator(K)$ data tokens to the cycle created by the $PeriodActor$, its MCR value represents the exact maximum throughput of the subgraph. Figure 7 shows the relaxed subgraph execution model of the IBSDF subgraph of figure 1. The three interfaces $in1$, $in2$ and $out1$ are connected respectively to the time line actors $T0$, $T1$ and $T2$. Figure 8 shows the resulted srSDF graph of ESR method. It is obtained by replacing the instances of hierarchical actor C in the srSDF topgraph (fig. 2a) by the relaxed subgraph execution model of its subgraph (fig. 7).

C. Special case of IBSDF graphs

For a particular case of IBSDF graphs, the ESR method may failed to compute the maximum throughput. Specially when the initial marking of the graph removes the data dependence between output and input interfaces in a subgraph iteration. For instance the initial marking of the IBSDF graph in figure 9a, allows to execute 5 times the output interface $out1$ without the need of executing the input interface $in1$. That makes the output interface $out1$ independent from the input interface $in1$ in terms of data tokens for 5 subgraph iterations.

Figure 9b shows the equivalent relaxed srSDF graph of the IBSDF graph of figure 9a. The MCR of the relaxed srSDF graph is the value of the cycle $(A, B, D, in2, F, G, out1)$ and so the throughput is $1/MCR = 5/5 = 1$. Using ESR method, the throughput is $1/3$ computed as 1 over the MCR value of its resulted srSDF graph (fig. 9c) which is the value of the cycle $(A, B, in1, T1, out1)$. The throughput computed with the ESR method represents 33.3% of the maximum throughput.

VIII. Experimental Results

A. Experimental Setup

In this section we compare the performance of the new method ESR with the flat srSDF conversion based methods [8] and [9], and with SR technique. The numerical experiments consists of measuring the running time for each method to compute the throughput including the srSDF conversion when relevant. The maximum throughput value computed by basic methods [8] and [9] is used to compare the quality of SR and ESR techniques. Experimental results are summarized in Table II. The difference of the throughput value between the two techniques ESR and SR shows how much the execution of IBSDF graphs can speed up by ignoring the execution rules.

For the sake of comparison, we have used the same sets of IBSDF graphs that were used in [10] as a benchmark to compare the performance of ESR technique with [8] and [9] methods. The graph set is composed of two categories of IBSDF graphs. The first category is a set of real DSP applications modeled as IBSDF graphs, available in [14]. The second category is a set of synthetics IBSDF graphs generated randomly using a generator of IBSDF graphs based on Turbine tool [15]. Table I shows the characteristics of the graphs.

The Algorithm 1 implementation of ESR method is available in the open-source Preesm framework [1] as well as the SR method. The ASAP based method [8] used is the open source implementation of SDF^3 [16]. For the Periodic Schedule [9], we used a mathematical model to compute the optimal period solved with the mathematical programming solver Gurobi [17]. All methods were tested on one core of an Intel i5-6300 processor clocked at 2.40 GHz, and with 8GB of RAM.

B. Results

As table I shows, the size of the resulted srSDF graph of SR and ESR methods remains small comparing to both flat and relaxed srSDF graph. In fact, it was not possible to convert the last two synthetics graphs with the available RAM. Since [8] and [9] methods rely on the flattening, they failed to compute the throughput of the last two graphs (see table II). Besides, both methods SR and ESR computes successfully the throughput of all graphs as shown in table II.

For small graphs, [8] and [9] are faster than SR and ESR techniques. But, the execution time of both methods [8] and [9] increases exponentially as the number of levels and the size of the srSDF graph grow. The results confirm that SR and ESR techniques are suitable for larger graphs than [8] and [9] methods. Furthermore, the ESR method is more likely to compute the maximum throughput than SR technique.

For the last two graphs we are not able to evaluate the optimallity of the throughput since [8] and [9] methods have failed. But, the throughput difference between the two methods ESR and SR shows how much the execution of an IBSDF graph can speed up by relaxing the execution rules. In fact, the throughput of the last graph, computed by SR method, represents less than 1% of the one computed by ESR method.

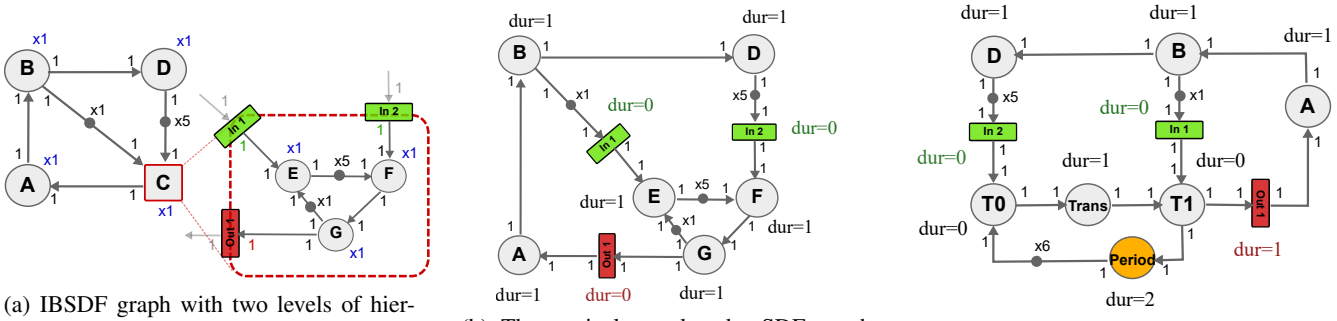
IX. Conclusion

We have introduced a relaxed execution for the IBSDF graph by ignoring its execution rules. The relaxed execution allows applications to reach their maximum throughput. We have presented a new method named Evaluate-Schedule-Replace (ESR) to evaluate the throughput of IBSDF graph under relaxed execution. The new method is based on the Schedule-Replace (SR) technique and takes advantage of the interface-based hierarchy to compute the IBSDF graph throughput without flattening its hierarchy. Experiments show that, the new method is suitable for large IBSDF graphs comparing to basic methods that require the relaxed srSDF graph. Additionally, the ESR method allows to compute the maximum throughput of IBSDF graphs in less than 2 seconds. A future work is to study special cases of IBSDF graphs for new specific methods.

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.



(a) IBSDF graph with two levels of hierarchy. Each actor is executed once.

(b) The equivalent relaxed srSDF graph without the extra-edges.

(c) The resulting srSDF graph of the ESR method.

Fig. 9: A Particular case of IBSDF graph for which the ESR method fails to compute the maximum throughput

TABLE I: Graphs description

IBSDF graph			Conversion of IBSDF to srSDF graph				Resulted srSDF graph of the methods			
			Flat srSDF graph		Relaxed srSDF graph		Schedule-Replace [10]		ESR	
Name	Levels	Actors	Actors	FIFOs	Actors	FIFOs	Actors	FIFOs	Actors	FIFOs
Crypto	2	10	34	85	34	49	10	17	30	37
Large FFT	2	10	267	1300	267	777	4	5	8	8
LTE	4	18	250	641	272	337	3	4	7	7
Stereo Matching	2	41	1604	5829	1606	3143	39	63	56	78
Graph 1	3	15	503	1654	513	816	15	26	23	32
Graph 2	5	20	17727	80976	17730	56703	12	30	24	42
Graph 3	6	24	84440	338391	87440	196019	12	35	22	45
Graph 4	5	150	653289	3253811	654566	2034980	90	196	95	201
Graph 5	8	240	39 E10	-	39 E10	-	150	441	174	465
Graph 6	10	100	31 E15	-	31 E15	-	50	124	68	142

TABLE II: Performance comparison between basic methods [8] and [9], SR technique [10], and the Evaluate-Schedule-Replace method.

IBSDF graph	ASAP [8] (Relaxed srSDF)			Periodic [9] (Relaxed srSDF)			Schedule-Replace [10]			Evaluate-Schedule-Replace		
Name	% of Opt.	Throughput	Exec.Time	% of Opt.	Throughput	Exec.Time	% of Opt.	Throughput	Exec.Time	% of Opt.	Throughput	Exec.Time
Crypto	100%	1.25 E-03	1 ms	100%	1.25 E-03	6 ms	100%	1.25 E-03	38 ms	100%	1.25 E-03	45 ms
Large FFT	100%	1.25 E-03	44 ms	100%	1.25 E-03	33 ms	100%	1.25 E-03	36 ms	100%	1.25 E-03	74 ms
LTE	100%	7.69 E-04	152 ms	100%	7.69 E-04	20 ms	100%	7.69 E-04	32 ms	100%	7.69 E-04	58 ms
Stereo Matching	100%	2.76 E-05	4320 ms	100%	2.76 E-05	80 ms	99.18%	2.74 E-05	37 ms	100%	2.76 E-05	130 ms
Graph 1	100%	2.50 E-02	11984 ms	100%	2.50 E-02	30 ms	34.48%	8.62 E-03	34 ms	100%	2.50 E-02	59 ms
Graph 2	-		>1h	100%	1.73 E-03	1190 ms	55.05%	9.52 E-04	34 ms	100%	1.73 E-03	70 ms
Graph 3	-		>1h	100%	1.98 E-03	2319 ms	20.70%	4.10 E-04	34 ms	100%	1.98 E-03	90 ms
Graph 4	-		>1h	100%	9.80 E-04	55407 ms	12.49%	1.22 E-04	61 ms	100%	9.80 E-04	306 ms
Graph 5	-		-	-		-	??	6.76 E-05	61 ms	??	7.35 E-04	560 ms
Graph 6	-		-	-		-	??	4.58 E-13	72 ms	??	3.42 E-06	1930 ms

[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] H. Deroui, K. Desnos, J.-F. Nezan, and A. Munier-Kordon, "Throughput evaluation of dsp applications based on hierarchical dataflow models," in *Proceedings of the 50th International Symposium on Circuits and Systems. ISCAS*, 2017.

[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] B. Bodin, A. Munier-Kordon, and B. de Dinechin, "K-periodic schedules for evaluating the maximum throughput of a synchronous dataflow graph," in *2012 International Conference on Embedded Computer Systems (SAMOS)*, pp. 152–159.

[13] 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.

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

[15] 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.

[16] 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.

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