Cyclic scheduling of loop-intensive applications on heterogeneous multiprocessor architectures

Philippe Glanon, Selma Azaiez, Chokri Mraidha

▶ **To cite this version:**

Philippe Glanon, Selma Azaiez, Chokri Mraidha. Cyclic scheduling of loop-intensive applications on heterogeneous multiprocessor architectures. RTCSA 2020 - IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications, Aug 2020, On line event, South Korea. pp.1-10, 10.1109/RTCSA50079.2020.9203667 . cea-04485112

HAL Id: cea-04485112
https://cea.hal.science/cea-04485112v1

Submitted on 1 Mar 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cyclic Scheduling of Loop-Intensive Applications on Heterogeneous Multiprocessor Architectures

Philippe Glanon
CEA, List
91191 Gif-sur-yvette, France
philippe.glanon@cea.fr

Selma Azaiez
CEA, List
91191 Gif-sur-yvette, France
selma.azaiez@cea.fr

Chokri Mraidha
CEA, List
91191 Gif-sur-yvette, France
chokri.mraidha@cea.fr

***Abstract*—This paper tackles the scheduling of loop-intensive applications modeled by synchronous dataflow graphs (SDFGs) on heterogeneous multiprocessor architectures under resource and communication constraints. Scheduling an application graph on multiprocessor architectures under resource constraints is a well-known NP-hard problem widely addressed in the previous decades with the goal of optimizing different performance metrics such as latency, memory allocations, energy consumption, throughput, etc. In this paper, we focus on the study of cyclic scheduling strategies and specifically the software pipelined schedules of SDFGs under the resource and communication constraints of heterogeneous multiprocessor architectures and we made two major contributions. The first contribution is an integer linear programming (ILP) model for the exact resolution of the scheduling problem and the second contribution is a time-efficient heuristic that generates scheduling solutions close to the optimal solutions generated with our ILP model.**

***Index Terms*—Cyclic scheduling, software pipelining, synchronous dataflow graphs, heterogeneous multiprocessor architectures, throughput, cyber-physical systems.**

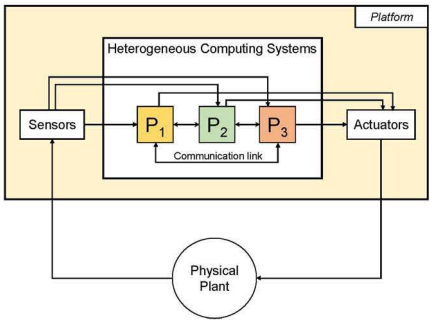
I. Introduction

Cyber-physical systems are increasingly used nowadays in a wide range of application fields to address many technical challenges. A cyber-physical system is a networked system that integrates multiple processing elements, each interacting with the other ones through digital networks to sense and control physical processes [6]. Many cyber-physical systems can be described as a feedback control structure like that sketched in Fig. 1a. This sketch consists of sensors, actuators and a computing system. The sensors acquire data from a physical plant, which can include human operators, mechanical parts, biological or chemical processes. The data acquired by the sensors are sent to the computing system through a logic network. The computing system is actually a heterogeneous multiprocessor system that consists of different types of processing units (PUs) —which can be central processing units, graphical processing units, field-Programmable gate arrays or even a device combining these PUs—, each offering a specific performance to process the data acquired by the sensors and to run software control applications that command the actuators.

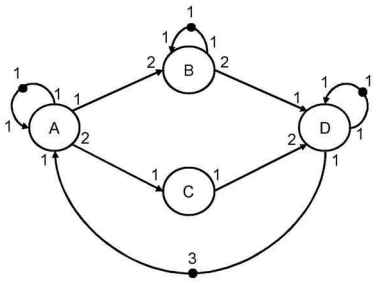
A key aspect in the design of a cyber-physical system (CPS) is the software deployment through which the scheduling and mapping of software programs on heterogeneous multiprocessor architectures (HMAs) are created. In this context, many parameters can be considered to find a deployment that achieves maximal performance for CPS. These parameters include for instance, the number of PUs available on the HMAs, the worst case execution times of software programs on each PU and the inter-PUs communication costs. In addition to these parameters, loops also need to be considered. Actually, loops being the most time-critical parts of many software applications, the performance achievable by a CPS application depends on the optimal execution of loop structures embedded in the software programs. Hence, to provide performance guarantee for CPS applications, there is a need of developing a scheduling framework that can be used to explore and to exploit the parallelism embedded in the repetitive pattern of loops structures under the resource and communications constraints of HMAs. For this purpose, *synchronous dataflow graphs* (SDFGs) can be used. SDFG is a class of dataflow model of computations [7] widely used to describe and analyze the behaviour and performance of loop-intensive applications such as streaming applications, automated control applications, etc. A SDFG is a directed graph that consists of a finite set of nodes (called actors) and a finite set of arcs (called channels or first-in first-out buffers). Actors are used to model computations in a loop-intensive program while channels are used to model the exchange of data between the computations. In order to execute and analyze the performance achievable by a SDFG, static scheduling strategies are usually applied. Static schedules for dataflow graphs can be classified into self-timed schedules (also called as soon as possible schedules) and periodic schedules. In a self-timed schedule, the instances of actors are executed as soon as possible the required data are available while in a periodic schedule, the instances of actors are executed cyclically according to a fixed time period. Self-timed schedules are known as scheduling strategies that achieve optimal performance for SDFGs. However, they are more difficult to implement than periodic schedules. A common way to get around the implementation complexity of self-timed schedules is the implementation of software pipelined (SWP) schedules [1], [11]. Actually, SWP schedules are a subclass of periodic scheduling strategies that also achieve optimal performance for dataflow graphs. These schedules are easier to implement than self-timed schedules although complex to determine under resource constraints and/or communication constraints. Actually, resource-constrained SWP scheduling is

978-1-7281-4403-0/20/$31.00 ©2020 IEEE

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.



(a) A cyber-physical system architecture



(b) A SDFG model

Computation costs matrix (Δ)

	A	B	C	D
P_1	1	3	2	3
P_2	5	2	1	2
P_3	3	1	6	1

Communication costs matrix (Γ)

	P_1	P_2	P_3
P_1	0	1	1
P_2	1	0	2
P_3	1	2	0

(c) Costs Matrices

Fig. 1. Running Example

a difficult problem whose complexity is NP-hard [11]. When communication constraints are further considered, the problem is NP-hard in the strong even for an unlimited number of resources. This paper tackles the SWP scheduling problem for SDFGs under the resource and communication constraints of HMAs with the goal of optimizing throughput. The paper makes two major contributions. The first contribution is an integer linear program (ILP) to characterize and compute the SWP schedules that give maximum throughput for SDFGs under the resource and communication constraints of HMAs and our second contribution is a time-efficient scheduling heuristic that generates SWP scheduling solutions close to the optimal ones generated with our ILP model.

The paper is organized as follows. Section II, presents background and basic definitions. Section III describes our ILP model for the scheduling problem. Section IV presents our scheduling heuristic. Experiments and discussion are described in section V. Related works are presented in section V and conclusions are made in section VI.

II. Background

In this section, we present some basic definitions and notations that will be used throughout the paper. Let us denote by $\mathbb{N}$ the set of positive integers, by $\mathbb{N}^*$ the set of strictly positive integers, by $\mathbb{Q}^+$ the set of rationals, by $\mathbb{Q}^{+*}$ the set of strictly positive rationals, by $[0, q)$ the set of positive integers between 0 and $q-1$ where $q \in \mathbb{N}^*$ and by $[m, n]$ the set of positive integers between m and n where $m, n \in \mathbb{N}$.

A. Application Model

Definition II.1 (SDFG). *A SDFG is a multi-rate dependency graph* $G_{sdf} = (V, E, P, C, M_0)$ *where* V *is a finite set of nodes called actors.* $E \subseteq V^2$ *is a finite set of arcs called channels.* $P = \{p_{ij} \in \mathbb{N}^* | \ e = (i,j) \in E\}$ *is the set of production rates determined by the function* $p : E \to \mathbb{N}^*$ *that associates a production rate* p_{ij} *with each channel* $e = (i,j) \in E$. $C = \{c_{ij} \in \mathbb{N}^* | \ e = (i,j) \in E\}$ *is the set of consumption rates determined by the function* $c : E \to \mathbb{N}^*$ *that associates a consumption rate* c_{ij} *with each channel* $e = (i,j) \in E$. $M_0 = \{m_{ij} \in \mathbb{N} | \ e = (i,j) \in E\}$ *is the set of initial markings determined by the function* $m : E \to \mathbb{N}$ *that associates an initial marking (i.e an initial number of data tokens)* m_{ij} *with each channel* $e = (i,j) \in E$.

Execution semantic. Let $G_{sdf} = (V, E, P, C, M_0)$ be a SDFG and let us denote the execution instances of actors in terms of firings. An actor $j \in V$ can be fired if and only if the initial marking of each of its incoming channels is greater or equal to the consumption rate. When an actor fires, it consumes a fixed number of tokens —which is predetermined in design time —from each of its incoming channels, and it produces a fixed number of tokens on each of its outgoing channels. Actors of a SDFG can be specified as *stateful* or *stateless*. A stateful actor is an actor whose firings are executed in a sequential order while a stateless actor is an actor whose firings could execute in parallel across different processing units (PUs). These types of actors respectively enable to specify pipeline and the data parallelisms in many loop-intensive applications. A stateful actor is often described as a node with a self-loop channel, where the channel consists of a fixed number of tokens that represents the distance separating the successive firings of the stateful actor. Fig. 1b depicts the SDFG of a loop-intensive program intended to be executed on the PUs of the architecture shown in Fig. 1a. This model consists of four actors (A, B, C, D), each describing a control instruction. Actors A, B and D are stateful actors and C is a stateless actor. All of these actors are connected by a set of channels, each describing the flows of data exchanged between the control instructions, and some containing an initial number of tokens describing the distance separating the successive firings of connected actors. When deploying this model on a multiprocessor architecture, data parallelism can be exploited by mapping and scheduling the firings of the actor C on different PUs. At the same time, pipeline parallelism can be exploited by scheduling the firings of a stateful actor on different PUs in such a way that the successive executions of these firings can overlap. Alongside with data and pipeline parallelisms, task parallelism can also be exploited by executing the firings of actors B and C on different PUs. In order to generalize the scheduling approaches proposed in this paper, we consider SDFG structures that consist of stateful and stateless actors.

Consistency and Liveness. In order to guarantee the existence of a static schedule for a SDFG, consistency [7] and liveness [2] are two properties that are commonly studied. A SDFG $G_{sdf} = (V, E, P, C, M_0)$ is said consistent if there exists a function $q : V \to \mathbb{N}^*$ such that for every channel

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

$e = (i,j) \in E,\ p_{ij} \times q_i = c_{ij} \times q_j$. The solutions of these balance equations determine the granularity $q_i \in \mathbb{N}^*$ of each actor $i \in V$, where the granularity of an actor corresponds to the minimal number of firings (i.e activations) required for this actor to achieve a single stable iteration[1] of G_{sdf}. Moreover, G_{sdf} is said live if and only if each actor in the SDFG can be activated infinitely often with a bounded number of tokens without deadlocks. Liveness checking is a complex problem widely addressed in the literature of SDFG and for which there exist many algorithms. Since this problem is out of the scope of this paper, we assume that every SDFG we consider is live. The SDFG of Fig. 1b is consistent and live and the granularity of each actor is given by q_A=2, q_B=1, q_C=4, q_D=2.

B. Architecture Model

Let us consider an loop-intensive application described as consistent and live SDFG $G_{sdf} = (V, E, P, C, M_0)$ and let G_{hma} be the HMA intended to run this application. G_{hma} is defined as a tuple (R, Δ, Γ) where:

- R is a finite set of heterogeneous PUs, each connecting with the other ones through logic communication links, which enable parallel data transmission.
- Δ is a matrix of size $|R| \times |V|$ that specifies the computation costs Δ_{xi}, where Δ_{xi} is the worst-case execution time of a single firing of an actor $i \in V$ on a PU $x \in R$.
- Γ is a matrix of size $|R| \times |R|$ that specifies the communication costs Γ_{xy}, where Γ_{xy} is the worst-case delay to transmit a single token of data from x to y. Note that if $x = y$ then $\Gamma_{xy} = 0$ otherwise $\Gamma_{xy} \neq 0$.

Fig. 1c illustrates the computation and communication cost matrices associated with the SDFG and the HMA of our running example.

C. Scheduling and Throughput

Let $G_{sdf} = (V, E, P, C, M_0)$ be a consistent and live SDFG and let us assume that G_{sdf} consists of both stateless and stateful actors. Let us denote by q_i and δ_i respectively the granularity and the execution times of any actor $i \in V$, by i_k the k^{th} firing of actor i and by p_{ij}, m_{ij}, c_{ij} respectively the production rate, the initial marking and the consumption rate of a channel $e = (i,j) \in E$.

Definition II.2. *A schedule of G_{sdf} is a function $\sigma : \mathbb{N} \times V \rightarrow \mathbb{Q}^+$ that associates a starting time $\sigma(k,i)$ with every firing i_k.*

Definition II.3 (SWP schedules [1]). *Let $\lambda \in \mathbb{Q}^{+*}$. A schedule σ of G_{sdf} is said software pipelined with period λ if for every actor $i \in V$ the following set of equations hold:*

$$\sigma(k + n \times q_i, i) = \sigma(k,i) + n \cdot \lambda \quad \forall k \in [0, q_i), \forall n \in \mathbb{N} \tag{1}$$

where $\sigma(k + n \times q_i, i)$ is the starting time of the k^{th} firing of actor i in the n^{th} iteration of G_{sdf} and $\sigma(k,i)$ is the time at which this firing must be scheduled to start in the schedule σ. To simplify mathematical notations in the rest of the paper, let us replace the notation $\sigma(k + n \times q_i, i)$ by $\sigma(n,k,i)$.

[1]An *iteration* of a SDFG is an execution sequence that brings back the graph to its initial state.

Proposition II.1 (Admissible SWP Schedules [1]). *Let σ be a SWP schedule. If σ is admissible for G_{sdf}, then the following set of precedence constraints must be hold:*

$$\sigma(n',k',j) \geq \sigma\Big(\Big\lceil \frac{(n' \times q_j + k') \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \Big\rceil, i\Big) + \delta_i, \tag{2}$$
$$\forall e = (i,j) \in E,\ \forall n' \in \mathbb{N},\ \forall k' \in [0, q_j), \forall l \in [1, c_{ij}].$$

Since G_{sdf} is consistent, then for every channel $e = (i,j) \in E$, the following equality is fulfilled: $p_{ij} \times q_i = c_{ij} \times q_j$. Using this equality, Eq. (2) can be rewritten as:

$$\sigma(n',k',j) \geq \sigma\Big(\Big\lceil n' \times q_i + \frac{k' \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \Big\rceil, i\Big) + \delta_i, \tag{3}$$
$$\forall e = (i,j) \in E,\ \forall n' \in \mathbb{N},\ \forall k' \in [0, q_j), \forall l \in [1, c_{ij}].$$

Simplifying further, Eq. (3) can be rewritten in the form:

$$\sigma(n',k',j) \geq \sigma(n,k,i) + \delta_i,\ \forall e = (i,j) \in E, \forall n' \in \mathbb{N}, \forall k' \in [0, q_j). \tag{4}$$

where:

$$n = n' + \Big\lfloor \frac{1}{q_i} \times \Big\lceil \frac{k' \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \Big\rceil \Big\rfloor,\ \forall l \in [1, c_{ij}]$$

$$k = \Big\lceil \frac{k' \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \Big\rceil \bmod q_i,\ \forall l \in [1, c_{ij}]$$

Actually Eq. (4) is a set of precedence constraints, each characterizing the dependency relation induced by any channel $e = (i,j) \in E$ between a pair of dependent firings $(i_k, j_{k'})$ in any admissible SWP scheduling of G_{sdf}.

Throughput and Iteration Period. Let σ be a SWP schedule of period λ for G_{sdf}. The throughput β achievable by a schedule σ of G_{sdf} is defined as the average number of stable iterations of G_{sdf} initiated per time unit in this schedule. More formally, β is given by:

$$\beta = \frac{1}{\lambda} \tag{5}$$

III. ILP Formulation

In this section, we present our ILP model to perform the SWP scheduling of SDFGs on HMAs. Our model consists of different constraints separated into cyclicity, resource, communication and precedence constraints. We consider as inputs a SDFG $G_{sdf} = (V, E, P, C, M_0)$ and a HMA $G_{hma} = (R, \Delta, \Gamma)$ and as objective function, the period λ achievable by a SWP schedule σ of G_{sdf}. By Eq. 5, the throughput β achievable by G_{sdf} in the schedule σ is proportional to the period λ of this schedule. Hence, a schedule that minimizes λ, maximizes implicitly the throughput. The scheduling entities considered are the firings of actors within G_{sdf}. For the rest of the section, let q_i be the natural granularity of an actor $i \in V$.

Cyclicity constraints. In order to ensure that each firing of each actor is executed periodically with a period λ our ILP model incorporates the set of cyclicity constraints that must be fulfilled by any SWP schedules σ. These constraints are expressed by Eq. (1). Since $\sigma(n,k,i) = \sigma(k + n \times q_i, i)$,

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

we reformulate Eq. (1) and we get the following cyclicity constraints:

$$\sigma(n,k,i) = \sigma(k,i) + n \cdot \lambda, \quad \forall i \in V, \forall k \in [0, q_i), \forall n \in \mathbb{N} \tag{6}$$

Resource constraints. When scheduling G_{sdf} on G_{hma}, we need to ensure that each firing of each actor is assigned exactly to one PU. In order to formulate these constraints, we define a $0-1$ integer variable $w_{x,k,i}$ such that, $w_{x,k,i} = 1$ implies that the k^{th} firing of actor i is assigned to the PU x and $w_{x,k,i} = 0$ otherwise. Using this variable, we formulate the following set of resource constraints which ensure that each firing of each actor is assigned to a single PU:

$$\sum_{\forall x \in R} w_{x,k,i} = 1 \qquad \forall i \in V, \ \forall k \in [0, q_i) \tag{7}$$

Since G_{sdf} consists of stateless actors whose firings may be executed in parallel and/or stateful actors whose firings may be pipelined, we need to ensure that the execution of independent firings of actors cannot overlap on a same PU. For this purpose, we formulate the following set of inequalities:

$$\begin{cases} \sigma(n,k,i) + \Delta_{xi} - \sigma(n',k',j) \leq M(1 - w_{x,k,i} \cdot w_{x,k',j}) \\ \qquad\qquad or \\ \sigma(n',k',j) + \Delta_{xj} - \sigma(n,k,i) \leq M(1 - w_{x,k,i} \cdot w_{x,k',j}) \\ \forall n, n' \in \mathbb{N}, \forall i, j \in V, \forall k \in [0, q_i), \forall k' \in [0, q_j), \forall x \in R \end{cases} \tag{8}$$

Actually, Eq. (8) is a set of non-linear disjunctive constraints which assert that two firings assigned to the same PU cannot be executed at the same time on this PU. In these constraints we use M, a big integer value such that the constraints hold only for the firings assigned to the same computing unit, i.e $w_{x,k,i} = w_{x,k,j} = 1$. The disjunctive constraints described by Eq. (8) could be linearized in two steps. First, we replace $M(1 - w_{x,k,i} \cdot w_{x,k',j})$ by $M(2 - w_{x,k,i} - w_{x,k',j})$ and then Eq. (8) becomes:

$$\begin{cases} \sigma(n,k,i) + \Delta_{xi} - \sigma(n',k',j) \leq M(2 - w_{x,k,i} - w_{x,k',j}) \\ \qquad\qquad or \\ \sigma(n',k',j) + \Delta_{xj} - \sigma(n,k,i) \leq M(2 - w_{x,k,i} - w_{x,k',j}) \\ \forall n, n' \in \mathbb{N}, \forall i, j \in V, \forall k \in [0, q_i), \forall k' \in [0, q_j), \forall x \in R \end{cases} \tag{9}$$

Second, we introduce another $0-1$ integer variable $d_{k,i,k',j}$ such that $d_{k,i,k',j} = 1$ implies that the k^{th} firing of actor i is scheduled before the k'^{th} firing of actor j and $d_{k,i,k',j} = 0$ otherwise. Now, using the variable $d_{k,i,k',j}$ the set of disjunctive constraints described by Eq. (9) could be linearized and rewritten as:

$$\begin{cases} \sigma(n,k,i) + \Delta_{xi} - \sigma(n',k',j) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j}) + \\ \qquad\qquad M(1 - d_{k,i,k',j}) \\ \sigma(n,k,i) + \Delta_{xi} - \sigma(n',k',j) \leq M(2 - w_{x,k_i,i} - w_{x,k_j,j} + \\ \qquad\qquad M d_{k,i,k',j} \\ \forall n, n' \in \mathbb{N}, \forall i, j \in V, \forall k \in [0, q_i), \forall k' \in [0, q_j), \forall x \in R \end{cases} \tag{10}$$

Simplifying further, equation (10) becomes:

$$\begin{cases} \sigma(n,k,i) + \Delta_{xi} - \sigma(n',k',j) \leq M(3 - w_{x,k,i} - w_{x,k',j} - \\ \qquad\qquad d_{k,i,k',j}) \\ \sigma(n',k',j) + \Delta_{xj} - \sigma(n,k,i) \leq M(2 - w_{x,k,i} - w_{x,k',j} - \\ \qquad\qquad d_{k,i,k',j}) \\ \forall n, n' \in \mathbb{N}, \forall i, j \in V, \forall k \in [0, q_i), \forall k' \in [0, q_j), \forall x \in R \end{cases} \tag{11}$$

Eq. (11) is actually a set of linear constraints that enforces a ILP solver to schedule the k^{th} firing of actor i before the k'^{th} firing of actor j when $d_{k,i,k',j} = 1$ or to schedule k'^{th} firing of actor j before the k^{th} firing of actor i when $d_{k,i,k',j} = 1$.

Communication and precedence constraints. In order to ensure that our ILP model can generate admissible SWP schedules, we should incorporate the precedence constraints induced by every channel of G_{sdf}. As shown previously, Eq. 4 characterizes the set of precedence constraints that must be fulfilled by any admissible SWP schedule of G_{sdf}. Actually, the constraints expressed by Eq. 4 are constructed by assuming that the computation cost of every firing of an actor i is equal to δ_i and there is no communication cost between a pair of dependent firings. However, regarding the description of G_{hma}, if a channel $e = (i,j) \in E$ induces a dependency relation between two firings i_k and $j_{k'}$ which are assigned respectively to a PU $x \in R$ and a PU $y \in R$ with $x \neq y$, there exists a non-zero communication cost between these firings. In order to consider these costs in our ILP formulation, we reformulate the precedence constraints described by Eq. 4 and we get the following set of precedence constraints:

$$\begin{aligned} \sigma(n',k',j) \geq \sigma(n,k,i) + \sum_{\forall x \in R} \Delta_{xi} \times w_{x,k,i} + \sum_{\forall x \in R} \sum_{\forall y \in R} \Gamma_{xy} \times \\ w_{x,k,i} \times w_{x,k',j}, \quad \forall e = (i,j) \in E, \forall n' \in \mathbb{N}, \forall k' \in [0, q_j) \end{aligned} \tag{12}$$

where:

$$n = n' + \left\lfloor \frac{1}{q_i} \times \left\lceil \frac{k' \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \right\rceil \right\rfloor, \ \forall l \in [1, c_{ij}]$$

$$k = \left\lceil \frac{k' \times c_{ij} + l - m_{ij} - p_{ij}}{p_{ij}} \right\rceil \bmod q_i, \ \forall l \in [1, c_{ij}]$$

Actually, Eq. 12 is a set of non-linear constraints, each characterizing the dependency relation imposed by each channel $e = (i,j) \in E$ in each stable iteration of G_{sdf} on G_{hma}. These constraints could be linearized and rewritten as follows:

$$\begin{aligned} \sigma(n',k',j) \geq \sigma(n,k,i) + \Delta_{xi} \cdot w_{x,k,i} + \Gamma_{xy} \cdot (w_{x,k,i} + \\ w_{y,k',j} - 1), \ \forall e = (i,j) \in E, \forall n' \in \mathbb{N}, \forall k' \in [0, q_j) \end{aligned} \tag{13}$$

In order to justify the equivalence between the constraints described by equations (12) and (13), let us consider a channel $e = (i,j) \in E$ and let i_k and $j_{k'}$ be two firings such that the execution of i_k precedes the execution of $j_{k'}$. Since each firing is assigned exactly to a single PU (i.e. $\sum_{\forall x \in R} w_{x,k,i} = \sum_{\forall x \in R} w_{x,k',j} = 1$), the sums $u = \sum_{\forall x \in R} \Delta_{xi} \cdot w_{x,k,i}$ and $v = \sum_{\forall x \in R} \sum_{\forall y \in R} \Gamma_{xy} \cdot w_{x,k,i} \cdot w_{y,k',j}$ contain only one term different from zero. In fact, if i_k is assigned to $x^* \in R$ and $j_{k'}$ is assigned to $y^* \in R$, then we can write $w_{x^*,k,i} = w_{y^*,k',j} = 1$, $u = \Delta_{x^*i}$, $v = \Gamma_{x^*y^*} = \Gamma_{x^*y^*}(w_{x^*,k,i} + w_{x^*,k',j} - 1)$ and thus, the constraints described by equations (12) and (13) can be rewritten as follows:

$$\begin{aligned} \sigma(n',k',j) \geq \sigma(n,k,i) + u + v, \quad \forall e = (i,j) \in E, \\ \forall n' \in \mathbb{N}, \forall k' \in [0, q_j) \end{aligned} \tag{14}$$

which shows the equivalence between equations (12) and (13).

To summarize, the constraints of our ILP model are given by equations (6), (7), (11) and (13) and the objective function is λ. For any SDFG and HMA, this ILP model can be instantiated

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

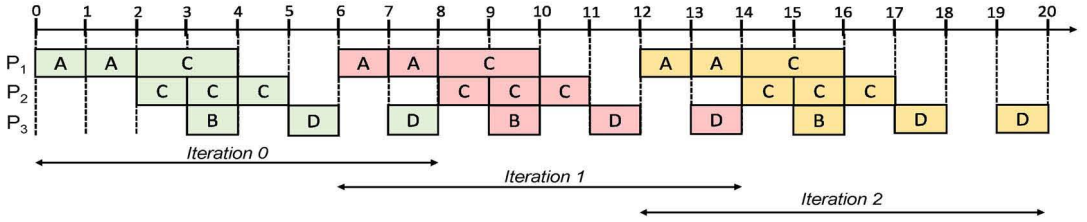


Fig. 2. An optimal scheduling solution obtained by our ILP formulation for the SDFG and the HMA of the running example.

and solved using the ILP solver of CPLEX[2], a well known-industrial solver that are very useful for solving complex optimization problem. Considering the SDFG of our running example (refer to figure 1b) and the architecture presented in Fig. 1a, an optimal scheduling solution returned by the ILP solver of CPLEX is that depicted in figure 2, where $\lambda^* = 6$. In this schedule, one can note that each firing of each actor is executed periodically according to the period λ^*. Moreover, one can note that this schedule exploit data parallelism since some firings of the stateless actor C can be executed in parallel across different PUs.

IV. Scheduling Heuristic

In the general case, when a given SDFG instance induces a large number of cyclicity and dependency constraints, the time to find an optimal scheduling solution with an ILP solver can be exponential because of the resources constraints of the ILP model. Therefore, there is a need of designing a scheduling heuristic (with a reasonable time complexity) that may generate scheduling solutions close to the optimal solutions. In this section, we present our heuristic (HCS) for solving the scheduling and optimization problem formulated in section III. HCS is a decomposed SWP scheduling heuristic inspired from the heuristic of Gasperoni and Schwielgelshohn [4], one of the first decomposed SWP scheduling algorithm for scheduling cyclic dataflow graphs under resource constraints. Actually, decomposed SWP scheduling is an approach that consists in separating the SWP scheduling problem of dataflow graphs under resource constraints into two sub-problems; the first to satisfy the dependency constraints of the graphs and the second to satisfy resource constraints. In this section, we review the heuristic of Gasperoni and Schwielgelshohn that we denote by GS, and then, we present our heuristic.

A. Overview of the heuristic GS

Let $G^t_{hsdf} = (V, E, M_0, \delta)$ be a timed homogeneous SDFG i.e. a timed SDFG where the production and consumption rates of actors are all equal to 1 where V is the set of actors, E is the set of channels, M_0 is the set of initial marking and σ is a function that associates a latency with each actor $i \in V$. Now, let us consider a multiprocessor architecture with p identical PUs for G^t_{hsdf} , where p is a finite number ($p \neq \infty$) and the inter-PU communication costs are negligible. The main idea of GS is the following. Assume that we have an optimal SWP schedule σ_∞ of period λ_∞ for G^t_{hsdf} without considering the resource constraints of the architecture —i.e. a SWP schedule with $p = \infty$—and that we want to deduce a SWP schedule of period λ for G^t_{hsdf} under resource constraints —i.e. a SWP schedule with $p \neq \infty$—. A way of building the schedule σ is to keep the structure of the schedule σ_∞ and to reorganize the execution of actors within this latter schedule in such a way as to find the period λ that meet both the resource constraints of the architecture and the precedence constraints of G^t_{hsdf}. The heuristic GS proceeds in four steps. Each of these steps are described in Algorithm 1. In the first step, the schedule σ_∞ is built for an infinite number of PUs. Since this schedule is not constrained by the number of resources, it can be determined in a polynomial-time [4]. In the second step, some dependency information from the schedule σ_∞ are used to delete some channels within G^t_{hsdf} in such a way as to obtain an acyclic graph G_{adg} which contains only direct precedence arcs (i.e. arcs without tokens). A list scheduler is then used in the third step to determine the schedule σ_a of G_{adg} on the p PUs, where $p \neq \infty$. Finally, in the fourth step, a SWP σ of G^t_{hsdf} under resource constraints is deduced from the schedules σ_a and σ_∞.

Algorithm 1: GS Heuristic

Input: A timed homogeneous dataflow graph $G^t_{hsdf} = (V, E, M_0, \delta)$, a multiprocessor architecture with p identical PUs.
Output: A valid SWP schedule σ of G^t_{hsdf} over T iterations.
// Step 1: Schedule the graph G^t_{hsdf} for unlimited resources.
Compute an optimal schedule σ_∞ of period λ_∞ for G^t_{hsdf} and let s^∞_i be the starting time of the first activation of a node $i \in V$ in the schedule σ_∞;
// Step 2: Construct the acyclic dependency graph G_{adg}
$G_{adg} \leftarrow G^t_{hsdf}$;
foreach *channel* $e = (i, j) \in E$ **do**
  if $(s^\infty_j \bmod \lambda_\infty < s^\infty_i \bmod \lambda_\infty + \delta_i)$ **then**
    delete the channel e from G_{adg};
  end
end
// Step 3: Schedule G_{adg} under resource constraints
Compute a list schedule σ_a of G_{adg} for the p identical PUs and let λ be the length of σ_a and $\sigma_a(i)$ be the starting time of an actor $i \in V$ in this schedule;
// Step 4: Compute a valid SWP schedule of G^t_{hsdf} under resource constraints
foreach *actor* $i \in V$ **do**
  for $n \leq T$ **do**
    $\sigma(n, i) = \sigma_a(i) + \left(n + \left\lfloor \frac{s^\infty_i}{\lambda_\infty} \right\rfloor\right) \cdot \lambda$
  end
end
return σ;

[2]https://www.ibm.com/analytics/cplex-optimizer

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2: HCS-Heterogeneous Cyclic Scheduling

Input: $G_{sdf} = (V, E, M_0, P, C)$, $G_{hsdf} = (V'E', M'_0)$, $G_{hma} = (R, \Delta, \Gamma)$
Output: A SWP schedule σ of G_{hsdf} over T iterations.
// Step 1: Compute an initial schedule
Set G_{hsdf} into a timed homogeneous graph G^t_{hsdf} and calculate an optimal SWP schedule σ_∞ of period λ_∞ for G^t_{hsdf} and let $s^\infty_{i_k}$ be the starting time of the first activation of a node $i_k \in V'$ in the schedule σ_∞ ;
// Step 2: Construct the acyclic dependency graph G_{adg}
$G_{adg} \leftarrow G_{hsdf}$;
foreach *arc* $e = (i_k, j_{k'})$ *in* G_{adg} **do**
  if $(s^\infty_{j_{k'}} \bmod \lambda_\infty < s^\infty_{i_k} \bmod \lambda_\infty + \delta_{i_k})$ **then**
    delete arc e from G_{adg};
  end
end
// Step 3: List scheduling of G_{adg} on G_{hma}
Use Algorithm 3 to calculate a schedule σ_a of G_{adg} under the resource and communication constraints of on G_{hma}.;
// Step 4: SWP scheduling of G_{sdf} on G_{hma}
$\lambda \leftarrow 0$;
foreach *node* $i_k \in V'$ **do**
  Let $succ(i_k)$ be the set of successors of i_k in the graph G_{hsdf}, $AFT(i_k)$ be the actual finishing time of i_k in the schedule σ_a and $proc(i_k)$ be the processing unit on which i_k is mapped to in the schedule σ_a;
  foreach $j_{k'} \in succ(i_k)$ **do**
    if $(\lambda < AFT(i_k) + \Gamma_{proc(i_k)proc(j_{k'})})$ **then**
      $\lambda \leftarrow AFT(i_k) + \Gamma_{proc(i_k)proc(j_{k'})}$;
    end
  end
end
foreach *node* $i_k \in V'$ **do**
  for $n \leq T$ **do**
    $\sigma(n, i_k) = \sigma_a(i_k) + n \cdot \lambda$
  end
end
return σ;

The correctness of GS can be found in [4].

B. Description of the heuristic HCS

Let $G_{sdf} = (V, E, M_0, P, C)$ be a consistent, live and non-timed SDFG and let $G_{hma} = (R, \Delta, \Gamma)$ be the architecture on which G_{sdf} is intended to be deployed.

The heuristic HCS shares the same idea with the heuristic GS. However, the main difference between these heuristics is that HCS accommodates both the resource and communication constraints of heterogeneous multiprocessor architectures to schedule SDFG. Before presenting HCS, we first convert G_{sdf} into an equivalent homogeneous SDFG $G_{hsdf} = (V', E', M'_0)$. The construction of $G_{hsdf} = (V', E', M'_0)$ is performed with the algorithm of de Groote et al. [8], which generates for any consistent SDFG, an equivalent homogeneous representation (also called linear constraint graph). Actually, V' is a set of nodes i_k where $i \in V$ and $k \in [1, q_i]$, q_i being the granularity of actor i. E' is the set of arcs between these firings and M'_0 is a function, that associates with each arc $e' = (i_k, j_{k'}) \in E'$, an initial number of tokens. Fig. 3a depicts the equivalent homogeneous graph obtained for the SDFG of our running example.

Algorithm 3: HAS-Heterogeneous Acyclic Scheduling

Input: an acyclic dependency graph G_{adg}, a HMA $G_{hma} = (R, \Delta, \Gamma)$
Output: A list schedule of σ_a of lenght λ
// Phase 1: prioritizing
Compute the scheduling rank of each node $i_k \in V'$ and generate a scheduling list, where nodes are sorted by increasing order of scheduling ranks;
// Phase 2: mapping
while *the scheduling list is not empty* **do**
  Select the first node i_k from the list;
  foreach $x \in R$ **do**
    Compute $EFT(x, i_k)$ using an insertion-based scheduling policy;
  end
  Get the PU x that minimizes the value of EFT and map the node i_k on x;
end
return the schedule σ_a obtained ;

TABLE I
SCHEDULING LIST FOR THE ACYCLIC DEPENDENCY GRAPH OF FIG. 3C

i_k	A_1	A_2	C_1	C_2	B_1	C_3	C_4	D_1	D_2
$rank(i_k)$	0	7	7	7	14	14	14	19	24

HCS takes as inputs G_{sdf}, G_{hsdf}, G_{hma} and it outputs a SWP schedule of G_{hsdf} under resource and communication constraints. The heuristic consists of four steps (refer to Algorithm 2):

Step 1. In this step, the graph G_{hsdf} is firstly set into a timed homogeneous graph $G^t_{hsdf} = (V', E', M'_0, \delta)$, where δ is a function that associates with every node $i_k \in V'$, a time cost δ_{i_k} such that:

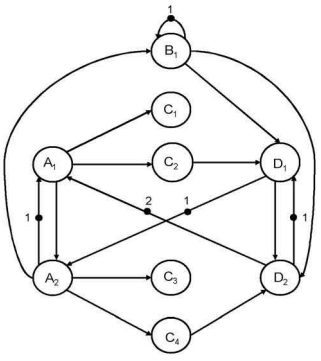
$$\delta_{i_k} = \Delta_i^{max} + \Gamma^{max} \tag{15}$$

where Δ_i^{max} is the worst delay to process a firing actor i on the architecture G_{hma} and Γ^{max} is the maximum inter-PU communication delay. Fig. 3b illustrates the graph G^t_{hsdf} for our running example, where for any value of k the time costs of nodes are given by δ_{A_k}=7, δ_{B_k}=5, δ_{C_k}=8, and δ_{D_k}=5. Secondly, an initial SWP schedule σ_∞ is calculated. Actually, this schedule does not satisfy neither the resource constraints nor the communication of G_{hma}, however it gives some interesting information about the dependency relations between the nodes of G_{hsdf}. Fig. 3d shows the schedule σ_∞ of period $\lambda_\infty = 17$ for the graph G^t_{hsdf} of our running example.

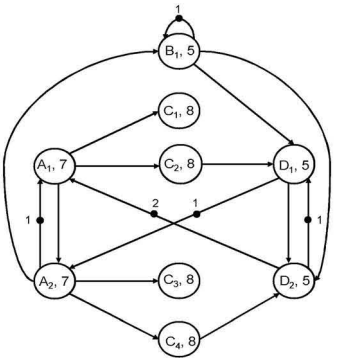
Step 2. The execution pattern of σ_∞ is used to delete some arcs in G_{hsdf} in such a way as to obtain an acyclic dependency graph G_{adg}. The arc deletion strategy used in HCS is the same than that of the heuristic GS (see Algorithm 1). Using this strategy, every arc with tokens in G_{hsdf} will not be considered in G_{adg}. Fig. 3c illustrates the acyclic dependency graph obtained for our running example.

Step 3. The acyclic graph G_{adg} is scheduled under the resource and communication constraints of G_{hma}. In order to achieve this, we have designed a list scheduling algorithm denoted by HAS (algorithm 3), where HAS stands for heterogeneous acyclic scheduling. HAS takes as inputs the graph G_{adg}, the architecture model G_{hma} and it outputs a schedule

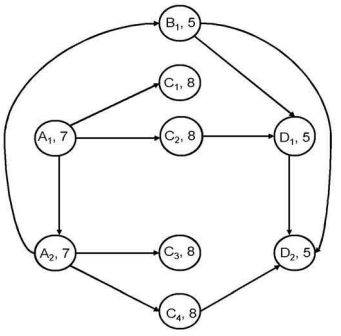
Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.



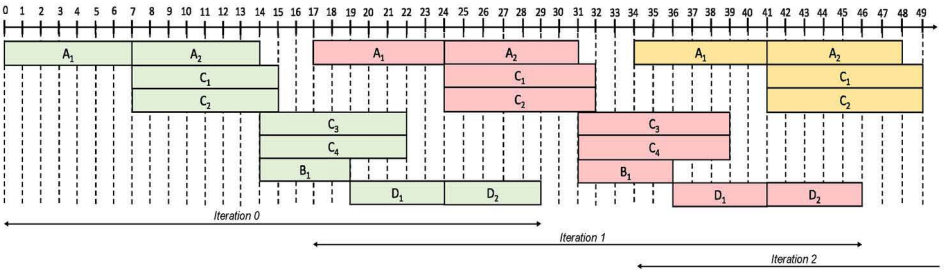
(a) Homogeneous representation G_{hsdf} of the SDFG depicted in Fig.1b.



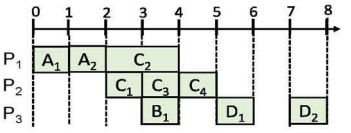
(b) Timed homogeneous graph G^t_{hsdf}.



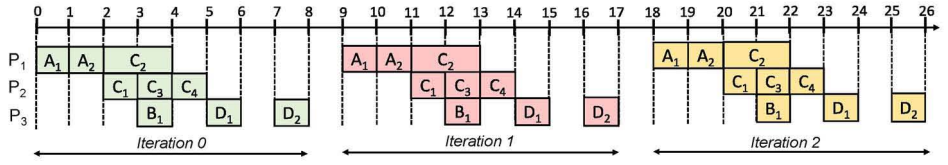
(c) Acyclic dependency graph G_{adg}.



(d) An optimal SWP schedule σ_∞ of G^t_{hsdf} without resource and communication constraints.



(e) List Schedule of G_{adg} under resource and communication constraints of G_{hma}.



(f) SWP schedule σ of G_{hsdf} under the resource and communication constraints of G_{hma}.

Fig. 3. Illustration of HCS Heuristic for the running example.

σ_a of G_{adg} on G_{hma}. This scheduling algorithm consists of a prioritizing phase and a mapping phase.

- **Prioritizing Phase.** This phase requires the scheduling rank (i.e. the priority) of each node of G_{adg} to be calculated firstly. The scheduling rank of a node $i_k \in V'$ is calculated by a recursive function *rank* given by:

$$rank(i_k) = \max_{j_{k'} \in\, pred(i_k)} \{rank(j_{k'}) + \delta_{j_{k'}}\} \quad (16)$$

where $pred(i_k)$ is the set of immediate predecessors of i_k. For every node without predecessors, $rank(i_k)$ is set to zero. Secondly, a scheduling list is generated by sorting the nodes by increasing order of scheduling ranks. Tie-breaking is performed randomly to sort the nodes with equal ranks. It can easily be shown that the increasing order of scheduling ranks provides a topological order of the nodes, which is a linear order that preserves the dependency relations. Table I gives the scheduling list of the acyclic graph (G_{adg}) depicted in Fig. 3c on the HMA of the running example. As it can be noted, nodes within this list are sorted by increasing order of ranks and this order preserves the dependency constraints of G_{adg}.

- **Mapping Phase.** Nodes are selected from the scheduling list by increasing order of the ranking and each node is mapped on the PU of G_{hma} that minimizes its earliest finishing time (EFT). To map a selected node on a selected PU, we use an insertion-based scheduling policy that tries to insert if possible the node in an earliest idle time slot of the PU (i.e an idle time interval between two already scheduled nodes on this PU) while ensuring the preservation of precedence relations. Let $EFT(x, i_k)$ be the earliest finishing time of the node i_k on the PU x:

$$EFT(x, i_k) = \max\{avail(x), ready(x, i_k)\} + \Delta_{xi} \quad (17)$$

where $avail(x)$ is the earliest time at which the PU x is available to execute a new node and $ready(x, i_k)$ is the instant at which the node i_k can be processed on the resource x. The value of $ready(x, i_k)$ is given by:

$$ready(x, i_k) = \max_{j_{k'} \in\, pred(i_k)} \left\{AFT(j_{k'}) + \Gamma_{proc(j'_k)x}\right\} \quad (18)$$

where $pred(i_k)$ is the set of direct predecessors of the node $i_k \in V'$, $AFT(j_{k'})$ is the actual finishing time of the node $j_{k'}$ and $proc(j_{k'})$ gives the PU on which the

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

TABLE II
BENCHMARKS CHARACTERISTICS

Benchmarks	Description	Number of Actors	Stateful actors
bitonicSort	Recursive implementation of the bitonic sorting network	61	7
fft	Fast Fourier Transform kernel	17	2
filterBank	A filter bank to perform multi-rate signal processing	53	6
radar	Radar Array Front-End	54	6
tde	Time delay equalization	42	5

TABLE III
RESULTS OF AVERAGE SOLVING TIMES (SEC) OF HCS VERSUS ILP SOLVER

Benchmarks	NP=2		NP=4		NP=8		NP=16	
–	HCS	ILP	HCS	ILP	HCS	ILP	HCS	ILP
bitonicSort	5.81	776.72	15.97	3391.98	34.13	9017.86	114.05	∞
fft	3.44	323.29	10.98	1798.13	21.13	6123.16	64.61	18765.57
filterBank	5.62	747.08	15.18	3065.29	32.66	8656.55	99.08	∞
radar	5.76	756.65	15.11	3168.76	33.25	8659.23	102.23	∞
tde	5.12	685.64	14.01	2891.86	28.75	7745.71	82.24	∞

node $j_{k'}$ is mapped to. For nodes without predecessors, $ready(x, i_k)$ is set to zero.

Fig. 3e depicts the schedule obtained with HAS by considering the acyclic graph shown in Fig. 3c and the HMA of our running example. The length of this schedule is equal to 8.

Step 4. A valid SWP schedule σ of period λ for G_{hsdf} is derived under the resource and communication constraints of G_{hma}. In order to derive this schedule, we first calculate the period λ with the information provided by the σ_a and the communication matrix Γ. Actually, λ is the minimum time required to process both the computations and communications of every actor in a single iteration of G_{hsdf}. Using this period and the equation $\sigma(n, i_k) = \sigma_a(i_k) + n \cdot \lambda$, we derive the schedule σ which respect both resource, precedence, communication and cyclicity constraints. Fig. 3f depicts a valid SWP schedule returned by HCS for the SDFG presented in Fig. 1b considering the costs matrices presented in Fig. 1c. In this schedule, a new iteration of the SDFG occurs every 9 time units (i.e $\lambda = 9$).

V. EXPERIMENTS AND DISCUSSION

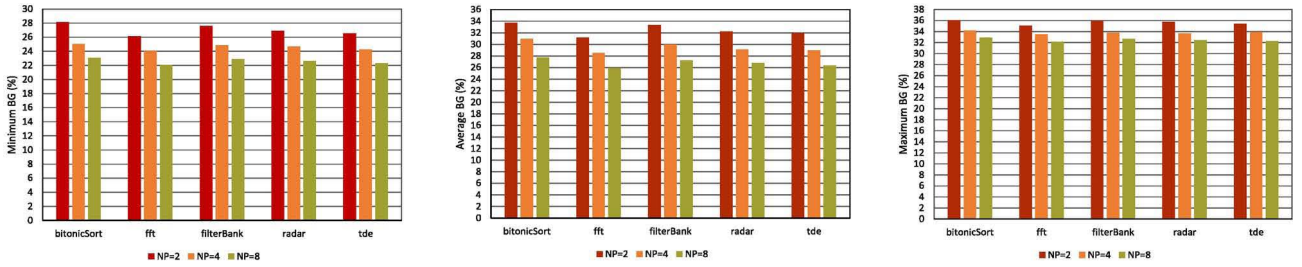
In order to evaluate the performance of HCS heuristic, we performed a broad range of experiments. Experiments were performed with the application benchmarks of StreamIt [9]. We chose StreamIt benchmarks because they provide a good representation of many loop-intensive applications. A brief description of the chosen benchmarks is given in Table II. These benchmarks are streaming applications that embed data, task and pipeline parallelisms. A detailed description of the benchmarks can be found in [9]. We set the number of stateful actors for each benchmark to approximately 10% of the total number of actors. In order to generate heterogeneous multi-processor architecture for the benchmarks, we have adapted the StreamIt compiler with a function that takes as inputs five parameters (NP, HFS, HFC, MCC, MIPCC) and outputs asymmetric computation and communication cost matrices as described in Fig. 1c. The parameter NP stands for the number of PUs on a given architecture. HFS and HFC stand respectively for the heterogeneity factor for PUs speed and the heterogeneity factor for PUs communication. A high percentage of HFS implies high difference in computation costs for the PUs and a high percentage of HFC implies high difference in communication costs. MCC and MIPCC stand respectively for the mean computation cost of the input SDFG instance and the mean inter-PUs communication cost. In order to generate the computation cost matrix, the generation function selects randomly a mean computation cost $\overline{\Delta}_i$ of every actor i from an uniform distribution in the range of 0 to $0.2 \times MCC$ and then, the computation cost of every actor on every PU is randomly selected from an uniform distribution of range $\overline{\Delta}_i \times \left(1 - \frac{HFS}{2}\right) \leq \Delta_{xi} \leq \overline{\Delta}_i \times \left(1 + \frac{HFS}{2}\right)$. Replacing MCC, HFS, Δ_{xi}, $\overline{\Delta}_i$ respectively by MIPC, HFC, Γ_{xy}, $\overline{\Gamma}_{xy}$ in this distribution, we generate the communication cost matrix. By definition, we set $\Gamma_{xy} = \Gamma_{yx}$ for each pair (x, y) of PUs and whether x is equal to y, we set the value of Γ_{xy} to 0. The following sets were considered for the experiments: NP={2, 4, 8, 16}, HFS={0.5, 1, 1.5, 2}, HFC={0.1, 0.75, 1.25, 2}, MCC={15, 30, 60} and MIPCC={10, 25, 40}. Experiments were performed on a PC Intel(R) core TM i7-7600U running at 2.80GHz with 16GB of RAM. In order to calculate an exact scheduling solution for a given benchmark, we have used the ILP solver of CPLEX 12.5.0 and OPL script language. In order to generate the equivalent homogeneous representation of each SDFG model, the transformation technique of de Groote et al. [8] was implemented.

A. Evaluation Metrics

Our experiments are based on the following metrics:

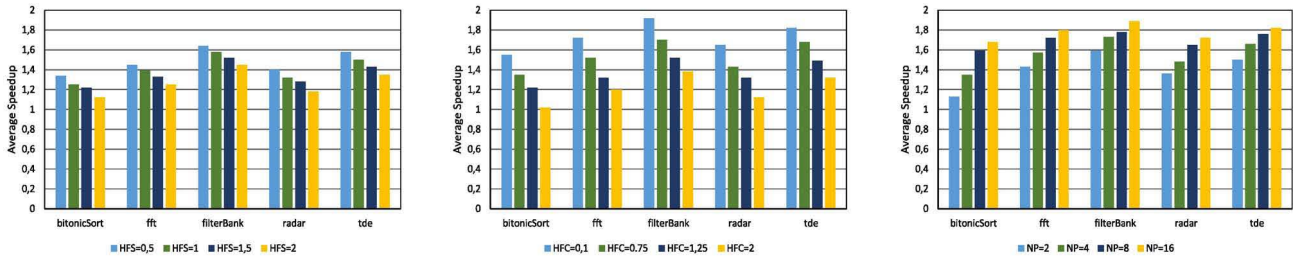
- **Solving Time:** the time to find a scheduling solution.
- **Bound Gap (BG):** this metric is the average ratio between the scheduling solutions obtained with HCS and the ILP solver of CPLEX. It enables to evaluate how far the throughput of schedules returned by the heuristic HCS is from the throughput of schedules generated with the ILP solver. The value of BG is given by:

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.



(a) Minimum values of BG (b) Average values of BG (c) Maximum values of BG

Fig. 4. Results of minimum, average and maximum BG for different values of NP



(a) Speedup for different values of HFS (b) Speedup for different values of HFC (c) Speedup for different values of NP

Fig. 5. Results of average speedup for different values of HFS, HFC, NP

$$BG = \frac{\lambda_{hcs} - \lambda_{cplex}}{\lambda_{cplex}} \times 100 \tag{19}$$

where λ_{hcs} and λ_{cplex} are respectively the periods of scheduling solutions obtained with HCS and CPLEX. A low percentage of BG means that the scheduling solution obtained with HCS is very close to the solution obtained with the ILP solver. Conversely, a high percentage of BG implies that the solution obtained with HCS is suboptimal compared to that obtained with the ILP solver.

- **Speedup.** Speedup is defined as the sequential execution time of a SDFG divided by the latency ($\mathcal{T}$) of this graph, where $\mathcal{T}$ is the amount of time required to execute all the firings of every actor in each stable iteration of the graph. To calculate the sequential execution time of a SDFG, we assign the firings of every actor to the single PU that minimizes the cumulative computation costs and we characterize the speedup by the following equation:

$$Speedup = \frac{\min\limits_{x \in R} \left[\sum\limits_{i \in V} q_i \times \Delta_{xi} \right]}{\mathcal{T}} \tag{20}$$

B. Performance Results

In order to evaluate the timing performance of the heuristic HCS, we compared the solving times of the ILP solver with the solving times of the heuristic HCS. Since We have limited the running time of CPLEX to 8 hours. Table. III plots the results of the average solving times for our benchmarks. The ILP solver was able to find a scheduling solution for all multiprocessor architectures except for the 16-PUs architectures, where ∞ means that the solver fails to find a scheduling solution within 8 hours. Conversely, the scalability of HCS is easily visible and in some cases HCS is approximately $265 \times$ *faster* than the ILP solver.

In order to measure the throughput performance of scheduling solutions returned by the heuristic HCS, we studied the variations of BG for the benchmarks according to different values of NP. Fig. 4 plots the results of minimum, average and maximum values of BG. As it can be observed, the average values of BG decrease as the values of NP increase. This means that HCS performs better for a large number of PUs and more the number of PUs is greater, more the scheduling solutions returned by HCS are getting closer to those returned by CPLEX. Moreover, for all the benchmarks, it can be observed that the minimum and maximum values of BG respectively vary approximately in the range of 22% to 28% and the range of 31% to 36% as when as the value of NP increases. This means that the scheduling solutions returned by HCS are getting closer to the solutions returned by CPLEX in the range of 72% to 78% in the best case and 64% to 69% in the worst case. Actually, these results characterize the performance bounds of HCS in terms of throughput achievement for the different benchmarks considered.

Now, if we want to characterize the performance of the heuristic HCS with respect to hardware features, we should study the variations of speedup for different types of heterogeneous multiprocessor architectures. For this purpose, we set the parameter NP to 4 and we evaluate the speedup ofthe heuristic with respect to different values of HFS and HFC and, we set the parameters HFS and HFC respectively to 1.5 and 1.25 and we evaluate the speedup of HCS for different values of NP. The results of speedup are shown in Fig. 5. It can be observed in Fig. 5a and Fig. 5b respectively that, the speedup of HCS gradually decreases when the values of HFS and HFC are increased. The interpretation of these results is that, whether there is a high variability in the computation and inter-

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.

PUs communication costs, the latency of scheduling solutions returned by HCS is higher than the sequential execution time of the application graphs and parallelism is less exploited by the heuristic. At the same time, it can be noted for the highest values of HFS and HFC that, the speedup of the heuristic is greater than or equal to 1. This means that, if we take a higher risk to increase the value of HFS and HFC, we will certainly loose parallelism but, there is a guarantee that the latency of scheduling solutions returned by the heuristic can not be worse than the sequential execution time of the application graphs. Moreover, Fig. 5c, it can be observed that the average speedup of all the benchmarks increases as the values of NP is increased. Based on these results, we can clearly state that the number of PUs on a given architecture, has a great impact on the performance achievable by the heuristic HCS and the more greater is the number of PUs, the better is the performance of the heuristic.

VI. Related Works

Resource-constrained SWP scheduling of loop-intensive program is a problem that has been extensively studied during the past decades. A variety of techniques have been proposed in the literature to tackle this scheduling problem. Feautrier [3] and Govindarajan et al. [5] have proposed discrete-time ILP formulations for solving the resource-constrained SWP scheduling of loop-intensive programs. These ILP formulations hold both for architectures with homogeneous resources and those with heterogeneous resources. However, none of the proposed ILP models consider communication constraints, which are inherent to the SWP scheduling problem tackled by this paper. Moreover, in their ILP formulations, Feautrier [3] and Govindarajan et al. [5] assumed that the loop-intensive programs are described with homogeneous SDFGs, which are particular cases of SDFG where the production and consumption rate of actors are equal to 1. Recently, Udupa et al. [1] have proposed an ILP formulation to tackle the resource-constrained SWP scheduling problem of SDFGs on multiprocessor architectures with graphical processing units (GPUs). Although this ILP formulation operates directly on SDFGs and hold both the architectures with homogeneous or heterogeneous GPUs, it does not consider communication constraints. In this paper, we have proposed a new ILP formulation for the SWP scheduling problem of loop-intensive programs. Our ILP formulation directly on SDFGs and it accommodates both resource and communication constraints while optimizing throughput. To the best of our knowledge, there is no work in the current literature that proposes such an ILP formulation. Lam [11], Gasperoni and Schwielgelshohn [4] and Robert et al. [10] have introduced SWP scheduling heuristics, to derive resource-constrained SWP schedules for homogeneous SDFGs that may contain cyclic dependencies. Although these heuristics are guaranteed, none of them consider neither architectures with heterogeneous resources, nor with communication constraints. Our heuristic (HCS) shares the same idea than the heuristic of Gasperoni and Schwielgelshohn. However, contrary to this heuristic, our heuristic deals with any type of SDFGs and it accommodates both the resource and communication constraints of heterogeneous multiprocessor architectures to derive SWP schedules.

VII. Conclusion and Future works

In this paper, we have proposed an ILP formulation and a cyclic scheduling heuristic (HCS) to tackle the SWP scheduling problem of SDFGs on heterogeneous multiprocessor architectures under resource and communication constraints, with the goal of optimizing throughput. Our ILP formulation operates on SDFGs instance and it accommodates task, data and pipeline parallelisms. Experiments made with StreamIt application benchmarks show interesting performance results for the heuristic HCS. Indeed, the heuristic is $\approx 265 \times faster$ than CPLEX solver and it was able to generate scheduling solutions close to the scheduling solutions returned by CPLEX within a range of $\approx 72\%$ to 78% in the best case and $\approx 64\%$ to 69% in the worst case. As future work, we would like to minimize the ratio between the scheduling solutions returned by the heuristic HCS and the scheduling solutions returned by CPLEX (i.e the values of BG). For this purpose, there is a need of minimizing the period of the scheduling solutions returned by the heuristic HCS. To achieve this goal, a possible research direction would be to investigate new prioritizing strategies for the list scheduling algorithm (i.e. Algorithm 3), that could minimize the length of schedules returned by this algorithm.

References

[1] A. Udupa, R. Govindarajan, and M. J. Thazhuthaveetil. Software Pipelined Execution of Stream Programs on GPUs. In CGO, pages 200-209, 2009, IEEE computer Society.

[2] A. H. Ghamarian, M. C. W. Geilen, T. Basten, B. D. Theelen, M. R. Mousavi and S. Stuijk. Liveness and Boundedness of Synchronous Data Flow Graphs, 2006 Formal Methods in Computer Aided Design, San Jose, CA, 2006, pp. 68-75.

[3] Paul Feautrier. Fine-grain scheduling under resource constraints. In Languages and Compilers or Parallel Computing , number 892 in Lectures Notes in Computer Science, pages 1-15.Springer Verlag, 1994.

[4] F. Gasperoni and U. Schwielgelshohn, Generating Close to Optimum Loop-Schedules on Parallel Processors, Parallel Processing Letters, 4(4), 1994, pp. 391-403.

[5] R. Govindarajan, E. R. Altman, and G. R. Gao, Minimizing Register Requirements Under Resource-constrained Rate-optimal Software Pipelining, in MICRO 27: Proc. of the 27th annual Intl. Symp. on Microarchitecture, 1994, pp. 8594

[6] E. A. Lee, "Cyber Physical Systems: Design Challenges," 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), Orlando, FL, 2008, pp. 363-369.

[7] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing," in IEEE Transactions on Computers, vol. C-36, no. 1, pp. 24-35, Jan. 1987.

[8] R. de Groote, J. Kuper, H. Broersma and G. J. M. Smit, "Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs," 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir, 2012, pp. 29-38, doi: 10.1109/SEAA.2012.20.

[9] W. Thies, M. Karczmarek, and S. P. Amarasinghe. 2002. StreamIt: A language for streaming applications. In Proceedings of the 11th International Conference on Compiler Construction (CC'02). Springer, London, UK, 179-196.

[10] Y. Robert, A. Darte and P. Calland, "Circuit Retiming Applied to Decomposed Software Pipelining" in IEEE Transactions on Parallel & Distributed Systems, vol. 9, no. 01, pp. 24-35, 1998. doi: 10.1109/71.655240

[11] M. Lam, "Software pipelining: An effective scheduling technique for VLIW machines", In Proceedings of the ACM SIGPLAN 88 Conference on Programming Language Design and Implementation (PLDI 88), July 1988 pages 318-328.

Authorized licensed use limited to: CEA. Downloaded on March 01,2024 at 12:21:57 UTC from IEEE Xplore. Restrictions apply.