Optimal and fast throughput evaluation of CSDF

Optimal and fast throughput evaluation of CSDF

Bruno Bodin
School of Informatics,
University of Edinburgh
Edinburgh, United Kingdom
bbodin@inf.ed.ac.uk

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

Benoît Dupont de Dinechin
KALRAY SA,
445 rue Lavoisier,
Montbonnot, France.
benoit.dinechin@kalray.eu

ABSTRACT

The Synchronous Dataflow Graph (SDFG) and Cyclo-Static Dataflow Graph (CSDFG) are two well-known models, used practically by industry for many years, and for which there is a large number of analysis techniques. Yet, basic problems such as the throughput computation or the liveness evaluation are not well solved, and their complexity is still unknown. In this paper, we propose K-Iter, an iterative algorithm based on K-periodic scheduling to compute the throughput of a CSDFG. By using this technique, we are able to compute in less than a minute the throughput of industry applications for which no result was available before.

Keywords

Cyclo-Static Dataflow Graph, Static analysis, Throughput

1. INTRODUCTION

The execution of streaming applications such as multimedia streaming or signal processing by an embedded system must respect strict constraints of throughput, latency, memory usage, and power consumption. Several dataflow programming languages have been developed to express this class of applications in order to handle such requirements.

Dataflow modeling involves designing an application as a set of tasks which communicate only through channels. Synchronous Dataflow Graph (SDFG) and more generally Cyclo-Static Dataflow Graph (CSDFG) are two models to statically specify an application behavior. They are commonly used to evaluate applications in term of throughput or memory consumption using dataflow static analysis techniques.

The exact determination of the throughput requires to compute an optimal schedule. Most of the authors considered *as soon as possible* schedules [14, 8]. Their computation has an exponential complexity with respect to the dataflow size in the worst case, and is usually too long for real-life applications. Approximative methods were also developed to get polynomial-time evaluations. They usually limit their space of solutions to particular categories of schedules such as the *periodic* ones [1] (or equivalently *strictly periodic*). A *periodic* schedule is a cyclic schedule whose definition is only composed of a first execution time per task and a period of execution per task. Polynomial methods have been proposed to compute periodic schedules of SDFG [1] and CSDFG [4], and these methods can be applied to throughput estimation or buffer sizing.

Meanwhile, existing static dataflow languages are revealing new cases which are not well supported by these techniques [4]. These cases are too complex to be used with exact schedules: as an example, there is no optimal throughput result for the H264 application proposed by [4]. Furthermore, *periodic* solutions remain an over-approximation which might be insufficient to be applied for embedded systems.

K-periodic scheduling [3] was developed as an alternative method when both *as soon as possible* and *periodic* methods are not satisfactory. A K-periodic schedule is built by periodically repeating a schedule of its K_t first executions. The estimated value of the throughput directly depends on the *periodicity vector* K. A *periodic* schedule can be seen as a particular case of K-periodic schedule for which $K_t = 1$ for every task. Setting K equal to what is called the repetition vector provides the optimal value of the throughput, but space and time complexities are not scalable.

An exponential number of pertinent values K may be considered between these two extreme solutions to exactly evaluate the throughput. This paper aims to optimally compute the throughput by iteratively increasing a periodicity vector K until an optimal solution is reached. From a theoretical point of view, the computation of a K-periodic schedule of minimum period is presented, followed by an original optimality test of a periodicity vector. Our algorithm is successfully compared for both SDFG and CSDFG with classical approaches and solves several classical benchmarks.

The contributions of this paper are:

- an extension of K-periodic scheduling to CSDFG,
- an optimality test of a K-periodic schedule,
- and K-Iter, an heuristic to efficiently explore the possible K-periodic schedules for a CSDFG.

Section 2 introduces the model, notations and some definitions on K-schedules. Our algorithm is presented in Section 3. Section 4 is devoted to the experimentations. Related works are presented in Section 5. Section 6 is our conclusion.

DAC '16, June 05 - 09, 2016, Austin, TX, USA

ISBN 978-1-4503-4236-0/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2897937.2898056

2. CYCLO-STATIC DATAFLOW GRAPHS

This section is devoted to the presentation of the model and notations of Cyclo-Static Dataflow Graphs, followed by the definition of the throughput. K-periodic schedules are lastly introduced.

2.1 Model definition

A Cyclo-Static Dataflow Graph (CSDFG) is a directed graph in which nodes model taks, and arcs correspond to buffers. It is denoted by $\mathcal{G} = (\mathcal{T}, \mathcal{B})$ where $\mathcal{T}$ (*resp.* $\mathcal{B}$) is the set of nodes (*resp.* arcs).

Every task $t \in \mathcal{T}$ is decomposed into $\varphi(t)$ phases; for every value $p \in \{1, \cdots, \varphi(t)\}$, the p^{th} phase of t is denoted by t_p and has a constant duration $d(t_p) \in \mathbb{N}$. One iteration of the task $t \in \mathcal{T}$ corresponds to the ordered executions of the phases $t_1, \cdots, t_{\varphi(t)}$.

Furthermore, every task $t \in \mathcal{T}$ is executed several iterations: for every integer n and for every phase p, $\langle t_p, n\rangle$ denotes the n^{th} execution of the p^{th} phase of t.

Every buffer $b = (t, t') \in \mathcal{B}$ represents a buffer of unbounded size from the task t to t' with an initial number of stored data, $M_0(b) \in \mathbb{N}$. $\forall p \in \{1, \cdots, \varphi(t)\}$, $in_b(p)$ data are written in b at the end of an execution of t_p. Similarly, $\forall p' \in \{1, \cdots, \varphi(t')\}$, $out_b(p')$ data are read from b before the execution of ${t'}_{p'}$. In addition, we set

$$i_b = \sum_{p=1}^{\varphi(t)} in_b(p) \text{ and } o_b = \sum_{p=1}^{\varphi(t')} out_b(p').$$

A Synchronous Dataflow Graph (SDFG) can be seen as a special case of CSDFG where each task has only one phase: $\forall t \in \mathcal{T}$, $\varphi(t) = 1$.

Figure 1 shows a buffer b between the two tasks t and t'. The respective numbers of phases of the two tasks are $\varphi(t) = 3$ and $\varphi(t') = 2$. The two associated vectors of b are $in_b = [2, 3, 1]$ and $out_b = [2, 5]$, thus $i_b = 6$ and $o_b = 7$. The initial number of data is $M_0(b) = 0$.

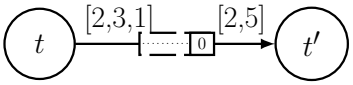

Figure 1: An simple buffer b between two tasks t and t'.

2.2 Schedules and consistency

A feasible (or valid) schedule associated with a CSDFG is a function $\mathcal{S}$ that associates, for every triple (t, p, n) with $t \in \mathcal{T}$, $p \in \{1, \cdots, \varphi(t)\}$ and $n \in \mathbb{N} - \{0\}$, $\mathcal{S}\langle t_p, n\rangle \in \mathbb{R}$, the starting time of $\langle t_p, n\rangle$, such that the number of data in every buffer $b \in \mathcal{B}$ remains non-negative, *i.e.* no data are read before they are produced.

Consistency is a necessary (but non-sufficient) condition for the existence of a valid schedule within bounded memory that was first established for SDFG [10]. It has been extended to CSDFG [2] by considering the cumulative number of data produced/consumed by one iteration of its tasks. A CSDFG is consistent if there exists a repetition vector $q \in (\mathbb{N} - \{0\})^{|\mathcal{T}|}$ such that

$$\forall b = (t, t') \in \mathcal{B}, \quad q_t \times i_b = q_{t'} \times o_b.$$

The repetition vector defines the number of task executions in a sequence that preserves data quantities in each buffer.

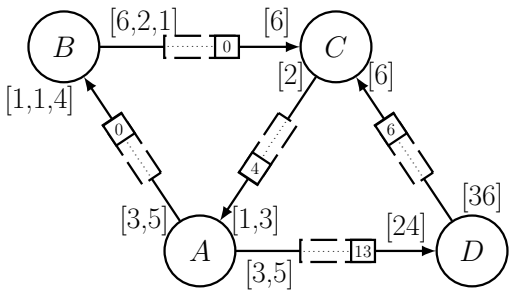

Figure 2: A consistent CSDFG with $d(A) = [1, 1]$, $d(B) = [1, 1, 1]$, $d(C) = [1]$ and $d(D) = [1]$.

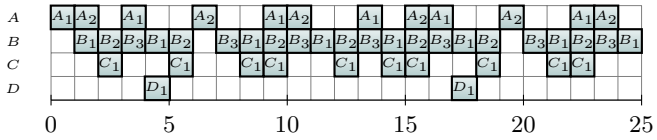

Figure 3: An *as soon as possible* feasible schedule for the CSDFG pictured in Figure 2.

For instance, the repetition vector of the CSDFG presented in Figure 2 is $q = [6, 12, 6, 1]$, the graph is thus consistent.

So, as consistency is a necessary condition for the existence of a valid schedule, our study focuses on consistent CSDFG.

2.3 Throughput of a CSDFG

The throughput of a task $t \in \mathcal{T}$ associated with a schedule $\mathcal{S}$ is usually defined as

$$Th_t^{\mathcal{S}} = \lim_{n \to \infty} \frac{n}{\mathcal{S}\langle t_1, n\rangle}.$$

Theorem 1 proved by Stuijk *et al.* [16] characterizes the relations between the throughput of different tasks using the repetition vector.

THEOREM 1 ([16]). *Let $\mathcal{G} = (\mathcal{T}, \mathcal{B})$ be a consistent CSDFG and $\mathcal{S}$ a valid schedule. For any pair of tasks (t, t'), $\frac{Th_t^{\mathcal{S}}}{q_t} = \frac{Th_{t'}^{\mathcal{S}}}{q_{t'}}$ where q is the repetition vector of $\mathcal{G}$.*

The throughput of a valid schedule $\mathcal{S}$ is then equal to $Th_{\mathcal{G}}^{\mathcal{S}} = \frac{Th_t^{\mathcal{S}}}{q_t}$ for any task $t \in \mathcal{T}$. The period is $\Omega_{\mathcal{G}}^{\mathcal{S}} = \frac{1}{Th_{\mathcal{G}}^{\mathcal{S}}}$.

Let's consider a consistent CSDFG $\mathcal{G} = (\mathcal{T}, \mathcal{B})$ initially marked by $M_0(b)$, $b \in \mathcal{B}$. The problem addressed by the present paper is to evaluate the maximum reachable throughput of $\mathcal{G}$ which is denoted by $Th_{\mathcal{G}}^{\star}$.

The most common scheduling policy consists of executing the tasks *as soon as possible*. Figure 3 presents first executions of the *as soon as possible* schedule for the CSDFG pictured in Figure 2.

The *as soon as possible* schedule maximizes the throughput. Nevertheless, its description can be of exponential size, as it depends on the repetition vector rather than the problem size. So other scheduling policies must be considered to reduce the computation time of the maximum throughput.

2.4 K-periodic scheduling

Let $K = [K_1, \ldots, K_{|\mathcal{T}|}] \in (\mathbb{N} - \{0\})^{|\mathcal{T}|}$. A schedule $\mathcal{S}$ is K-periodic with a fixed vector K if, for any task $t \in \mathcal{T}$

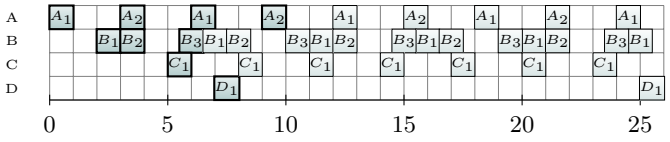

Figure 4: A K-periodic schedule for the CSDFG pictured in Figure 2 of periodicity factor $K = [2, 1, 1, 1]$. Highlighted executions correspond to initial starting times, others are derived from the period $\mu_t^{\mathcal{S}}$.

and for any integers $p \in \{1, \cdots, \varphi(t)\}$ and $\beta \in \{1, \cdots, K_t\}$, the period $\mu_t^{\mathcal{S}}$ and values $\mathcal{S}\langle t_p, \beta\rangle$ are fixed. Then for any integer $n \in \mathbb{N} - \{0\}$ such that $n = \alpha \times K_t + \beta$ with $\alpha \in \mathbb{N}$ and $\beta \in \{1, \cdots, K_t\}$, we get

$$\forall p \in \{1, \cdots, \varphi(t)\},\ \ \mathcal{S}\langle t_p, n\rangle = \mathcal{S}\langle t_p, \beta\rangle + \alpha\mu_t^{\mathcal{S}}.$$

If the periodicity vector K is unitary (*i.e.* $K_t = 1, \forall t \in \mathcal{T}$), the shedule is said to be periodic, or 1-periodic.

The throughput of any task $t \in \mathcal{T}$ for a valid K-periodic schedule $\mathcal{S}$ verifies $Th_t^{\mathcal{S}} = \frac{K_t}{\mu_t^{\mathcal{S}}}$. The throughput of $\mathcal{S}$ is then

$$Th_{\mathcal{G}}^{\mathcal{S}} = \frac{K_t}{q_t \times \mu_t^{\mathcal{S}}}.$$

Alternatively, the period of $\mathcal{S}$ is denoted by

$$\Omega_{\mathcal{G}}^{\mathcal{S}} = \frac{1}{Th_{\mathcal{G}}^{\mathcal{S}}} = \frac{q_t \times \mu_t^{\mathcal{S}}}{K_t}\ \ \forall t \in \mathcal{T}.$$

As example, the periodicity factor of task A in Figure 4 equals 2, thus the $K_A \times \varphi(A) = 4$ first executions of A are fixed; starting times of all the successive ones are implicitly defined using the period $\mu_A^{\mathcal{S}} = 12$. The period of $\mathcal{S}$ is thus $\Omega_{\mathcal{G}}^{\mathcal{S}} = \frac{12\times 6}{2} = 36$. It is important to note that for a 1-periodic schedule the maximal reachable throughput of $\mathcal{G}$ was only $\Omega_{\mathcal{G}}^{\mathcal{S}} = 108$.

All these notations will be used in the Section 3 to present our contributions.

3. THROUGHPUT EVALUATION METHOD

This section presents our algorithm. Subsection 3.1 recalls the characterization of periodic schedules of a CSDFG, which is extended to K-periodic schedules in subsection 3.2 using a simple transformation. Next subsection treats the computation of the minimum period of a K-periodic schedule. Subsection 3.4 is devoted to an original optimality test of a fixed K. Subsection 3.5 is devoted to our algorithm K-Iter which heuristically explores the possible K-periodic schedules for a CSDFG in order to optimally provide its maximal throughput.

3.1 Periodic scheduling of a CSDFG

The following Theorem 2 defines a feasible periodic schedule as a set of linear constraints. This set of constraints composes a linear program which solve the maximal throughput of a periodic schedule. In order to define the Theorem 2, several definition are required.

First, the total number of data produced by t in the buffer b at the completion of $\langle t_p, n\rangle$ is defined as

$$I_a\langle t_p, n\rangle = \sum_{\alpha=1}^{p} in_b(\alpha) + (n-1) \times i_b.$$

Similarly, the number of data consumed by t' in the buffer b at the completion of $\langle {t'}_{p'}, n'\rangle$ is defined by $O_a\langle {t'}_{p'}, n'\rangle = \sum_{\alpha=1}^{p} out_b(\alpha) + (n'-1) \times o_b$.

The total number of data contained in a buffer must remain non-negative. That is, any execution $\langle {t'}_{p'}, n'\rangle$ can be done at the completion of $\langle t_p, n\rangle$ if and only if $M_0(a) + I_a\langle t_p, n\rangle - O_a\langle {t'}_{p'}, n'\rangle \geq 0$.

For example, considering the CSDFG pictured in Figure 1, the execution $\langle {t'}_2, 1\rangle$ can be done at the completion of $\langle t_1, 2\rangle$ since $M_0(a) + I_a\langle t_1, 2\rangle - O_a\langle t'_2, 1\rangle = 0 + 8 - 7 \geq 0$.

For any pair of values $(\alpha, \gamma) \in \mathbb{Z} \times \mathbb{N} - \{0\}$, we set $\lfloor\alpha\rfloor^\gamma$ and $\lceil\alpha\rceil^\gamma$ as:

$$\lfloor\alpha\rfloor^\gamma = \left\lfloor \frac{\alpha}{\gamma} \right\rfloor \times \gamma \ \text{ and } \ \lceil\alpha\rceil^\gamma = \left\lceil \frac{\alpha}{\gamma} \right\rceil \times \gamma.$$

Let us consider a buffer $b = (t, t') \in \mathcal{B}$. For any pair $(p, p') \in \{1, \cdots, \varphi(t)\} \times \{1, \cdots, \varphi(t')\}$, let us define

$$Q_a^{\mathcal{G}}(p, p') = O_a\langle {t'}_{p'}, 1\rangle - I_a\langle t_p, 1\rangle - M_0(b) + in_b(p).$$

We also note $\gcd_a = \gcd(i_b, o_b)$,

$$\alpha_a^{\mathcal{G}}(p, p') = \left\lceil Q^{\mathcal{G}}(p, p') - \min\{in_b(p), out_b(p')\} \right\rceil^{\gcd_a}$$

and

$$\beta_a^{\mathcal{G}}(p, p') = \left\lfloor Q^{\mathcal{G}}(p, p') - 1 \right\rfloor^{\gcd_a}.$$

We now recall Theorem 2 which characterizes any feasible periodic schedule.

THEOREM 2 ([3]). *Let $\mathcal{G}$ be a consistent CSDFG. Any periodic schedule $\mathcal{S}$ of period $\Omega_{\mathcal{G}}^{\mathcal{S}}$ is feasible if and only if, for any buffer $b = (t, t')$ and for every pair $(p, p') \in \{1, \cdots, \varphi(t)\} \times \{1, \cdots, \varphi(t')\}$ with $\alpha_a^{\mathcal{G}}(p, p') \leq \beta_a^{\mathcal{G}}(p, p')$,*

$$\mathcal{S}\langle {t'}_{p'}, 1\rangle - \mathcal{S}\langle t_p, 1\rangle \geq d(t_p) + \Omega_{\mathcal{G}}^{\mathcal{S}} \times \frac{\beta_a^{\mathcal{G}}(p, p')}{q_t \times i_b}.$$

3.2 Extension to K-Periodic scheduling

The extension of Theorem 2 to K-periodic schedules with a fixed periodicity vector K comes from a transformation of the initial CSDFG $\mathcal{G} = (\mathcal{T}, \mathcal{B})$ to another equivalent one $\widetilde{\mathcal{G}} = (\mathcal{T}, \widetilde{\mathcal{B}})$ of the same structure for which the adjacent vectors of any task t are duplicated K_t times.

For any vector v of size s, and any integer $P > 0$, $[v]^P$ denotes the vector of size $s \times P$ obtained by duplicating v exactly P times, *i.e.* $\forall k \in \{1, \cdots, s\}$,

$$[v]^P(k) = [v]^P(k+s) = \cdots = [v]^P(k + (P-1) \times s) = v(k).$$

For any task $t \in \mathcal{T}$, we set $\widetilde{\varphi}(t) = K_t \times \varphi(t)$ and for any $p \in \{1, \cdots, \widetilde{\varphi}(t)\}$, $\widetilde{d}(t) = [d(t)]^{K_t}$. For any buffer $b = (t, t') \in \mathcal{B}$, we set $\widetilde{in}_b = [in_b]^{K_t}$, $\widetilde{out}_b = [out_b]^{K_{t'}}$ and $\widetilde{M_0}(b) = M_0(b)$. A consequence of this transformation is $\widetilde{i}_b = K_t \times i_b$ and $\widetilde{o}_b = K_{t'} \times o_b$.

$\widetilde{\mathcal{G}}$ is a consistent graph. Indeed, by definition of q, for any buffer $b = (t, t') \in \mathcal{B}$, $q_t \times i_b = q_{t'} \times o_b$, and thus

$$q_t \times \frac{\mathrm{lcm}(K)}{K_t} \times \widetilde{i}_b = q_{t'} \times \frac{\mathrm{lcm}(K)}{K_{t'}} \times \tilde{o}_b,$$

where $\mathrm{lcm}(K)$ is the least common multiple of values K_t, $t \in \mathcal{T}$. Let $\tilde{q}_t = q_t \times \frac{\mathrm{lcm}(K)}{K_t}$ for $t \in \mathcal{T}$ be a repetition vector of $\widetilde{\mathcal{G}}$.

Let us set for any buffer $b = (t, t') \in \widetilde{\mathcal{B}}$,

$$\mathcal{Y}(a) = \{(p,p') \in \{1,\cdots,\tilde{\varphi}(t)\} \times \{1,\cdots,\tilde{\varphi}(t')\}, \alpha_a^{\tilde{\mathcal{G}}}(p,p') \leq \beta_a^{\tilde{\mathcal{G}}}(p,p')\},$$

The determination of the minimum period $\Omega_{\widetilde{\mathcal{G}}}^{\star}$ of a periodic schedule can be modeled with the following linear program following Theorem 2:

Minimize $\Omega_{\widetilde{\mathcal{G}}}^{\widetilde{\mathcal{S}}}$ with

$$\begin{cases} \forall a = (t,t') \in \tilde{\mathcal{B}}, \quad \forall (p,p') \in \mathcal{Y}(a), \\ \widetilde{\mathcal{S}}\langle t'_{p'}, 1\rangle - \widetilde{\mathcal{S}}\langle t_p, 1\rangle \geq \tilde{d}(t_p) + \Omega_{\widetilde{\mathcal{G}}}^{\widetilde{\mathcal{S}}} \times \dfrac{\beta_a^{\tilde{\mathcal{G}}}(p,p')}{\widetilde{i}_a \times \widetilde{q}_t} \\ \forall t \in \mathcal{T}, \forall p \in \{1,\cdots,\widetilde{\varphi}(t)\}, \widetilde{\mathcal{S}}\langle t_p, 1\rangle \in \mathbb{R}^+ \\ \Omega_{\widetilde{\mathcal{G}}}^{\widetilde{\mathcal{S}}} \in \mathbb{R}^+ - \{0\} \end{cases}$$

The next theorem highlights the relationship between the periods of $\widetilde{\mathcal{G}}$ and $\mathcal{G}$:

THEOREM 3. *Let $\widetilde{\mathcal{S}}$ be a 1-periodic feasible schedule of $\widetilde{\mathcal{G}}$ of period $\Omega_{\widetilde{\mathcal{G}}}^{\widetilde{\mathcal{S}}}$. Starting times of $\widetilde{\mathcal{S}}$ define a K-periodic feasible schedule $\mathcal{S}$ of $\mathcal{G}$ with normalized period $\Omega_{\mathcal{G}}^{\mathcal{S}} = \Omega_{\widetilde{\mathcal{G}}}^{\widetilde{\mathcal{S}}}/lcm(K)$.*

PROOF. By construction of $\widetilde{\mathcal{G}}$, any periodic feasible schedule $\widetilde{\mathcal{S}}$ is a K-periodic feasible schedule $\mathcal{S}$ of $\mathcal{G}$. Then, for any task $t \in \mathcal{T}$, $\mu_t^{\mathcal{S}} = \mu_t^{\widetilde{\mathcal{S}}}$ and $\Omega_{\mathcal{G}}^{\mathcal{S}} = \frac{q_t \times \mu_t^{\mathcal{S}}}{K_t}$, thus

$$\Omega_{\mathcal{G}}^{\widetilde{\mathcal{S}}} = \widetilde{q}_t \times \mu_t^{\widetilde{\mathcal{S}}} = q_t \times \frac{\mathrm{lcm}(K)}{K_t} \times \mu_t^{\mathcal{S}} = \Omega_{\mathcal{G}}^{\mathcal{S}} \times \mathrm{lcm}(K).$$

□

3.3 Resolution of the linear program

The linear program for the determination of a minimum period can be transformed to a Max Cost-to-time Ratio Problem (MCRP in short), which is a polynomially solved problem [5]. Considering a bi-valued directed graph $G = (N, E)$ where any arc $e \in E$ is bi-valued by $L(e)$ and $H(e)$, the Cost-to-time Ratio of any circuit $c = (e_1, e_2, \cdots, e_p)$ is defined as

$$R(c) = \frac{\sum_{i=1}^{p} L(e_i)}{\sum_{i=1}^{p} H(e_i)}.$$

Let $\mathcal{C}(G)$ be the set of elementary circuits of G. The maximum Cost-to-time Ratio of a graph H is then

$$\lambda_H = \max_{c \in \mathcal{C}(G)} R(c).$$

An elementary circuit $c \in \mathcal{C}(G)$ is critical if $R(c) = \lambda_H$.

The bi-valued directed graph $G = (N, E)$ associated with our linear program is defined as follows:

- $N = \{\langle t_p, 1\rangle, t \in \mathcal{T}, p \in \{1,\cdots,\tilde{\varphi}(t)\}\}$ is the set of nodes;
- $E = \{(\langle t_p, 1\rangle, \langle t'_{p'}, 1\rangle), a = (t,t') \in \tilde{\mathcal{B}}, (p,p') \in \mathcal{Y}(a)\}$ is the set of arcs; any arc $e = (\langle t_p, 1\rangle, \langle t'_{p'}, 1\rangle) \in E$ is bi-valued by

$$(L(e), H(e)) = (\widetilde{d}(t_k), -\frac{\beta_a^{\tilde{\mathcal{G}}}(p,p')}{\widetilde{i}_a \times \widetilde{q}_t}).$$

The determination of the minimum period $\Omega_{\widetilde{\mathcal{G}}}^{\star}$ is then equivalent to the computation of the maximum Cost-to-time Ratio, *i.e.* $\Omega_{\widetilde{\mathcal{G}}}^{\star} = \lambda_H$.

Figure 5 presents the bi-valued graph H that corresponds to the CSDFG pictured in Figure 2 with $K = [1,1,1,1]$. The maximum Cost-to-time Ratio equals 108 and is reached by the circuit $c = \{A_1, D_1, C_1\}$ with $H(c) = \frac{1}{36}$ and $L(c) = 3$. This therefore implies that the minimum period of a feasible periodic schedule for the CSDFG is $\Omega_{\mathcal{G}}^{\star} = 108$.

3.4 K-periodic Schedule optimality test

A method based on the MCRP returns critical circuits, *i.e.* circuits c for which the value $R(c)$ is maximum. We take advantage of them to testify the optimality of a K-periodic schedule. The next theorem will allow us to test if the maximum throughput associated with a periodicity vector K is the maximal reachable throughput of the graph $\mathcal{G}$.

THEOREM 4 (OPTIMALITY TEST). *Let $\mathcal{G} = (\mathcal{T}, \mathcal{B})$ be a consistent CSDFG, a periodicity vector K and the associated bi-valued graph $G = (N, E)$. Let us suppose that $c \in \mathcal{C}(G)$ is a critical circuit such that for every execution $\langle t_p, 1\rangle$ of c, K_t is a multiple of $\bar{q}_t = \dfrac{q_t}{\gcd\{q_{t'}, t' \in c\}}$. Then, the maximum reachable throughput of $\mathcal{G}$ equals $\dfrac{lcm(K)}{R(c)}$.*

PROOF. By using Theorem 3, the minimum period of the CSDFG $\widetilde{\mathcal{G}}$ associated with $\mathcal{G}$ and K verifies $\Omega_{\widetilde{\mathcal{G}}}^{\star} = R(c)$. Let C be a sub-graph of $\mathcal{G}$ composed of the task from the circuit c. The minimum period of C is obtained for a K-periodic schedule with K following the assumption of the theorem. □

For instance, let us consider the bi-valued graph from Figure 5. With the critical circuit A, D, C, we observe $\bar{q}_B = 2$ and $K_B = 1$, thus K_B is no a multiple of $\bar{q}_B$, the optimality test is not checked.

3.5 The K-iter algorithm

Algorithm 1 computes iteratively a sequence of critical circuits by increasing the periodicity factor until the optimality test from Theorem 4 is fulfilled. The update of the periodicity factor ensures that the circuit c will realize the optimality test if it remains critical at the next step.

The values of the periodicity factor necessarily increase at every time the loop test is false. The convergence of the algorithm is guaranteed since the number of elementary circuits of a graph $\mathcal{G}$ is bounded and each circuit c is modified at most once: indeed, any modified circuit tested subsequently in the algorithm will fulfill the optimality test.

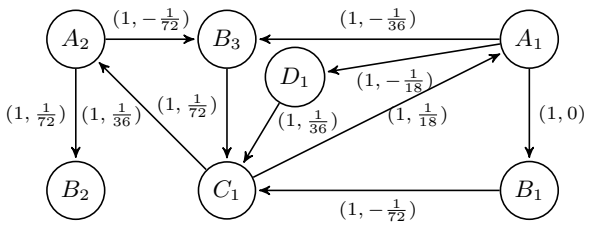

Figure 5: A bi-valued graph $H = (N, E)$ that corrresponds to the CSDFG pictured in Figure 2 when the periodicity vector is [1,1,1,1]. The maximum Cost-to-time Ratio is reached by the circuit $c = \{A_1, D_1, C_1\}$ and is equal to $\Omega^{\mathcal{S}} = 108$.

noend 1 Compute the CSDFG maximal throughput

Require: A CSDFG $\mathcal{G} = (\mathcal{T}, \mathcal{B})$
Ensure: A vector K and a circuit c of $\mathcal{G}$ such that the maximal reachable throughput of $\mathcal{G}$ verifies $Th^{\star}_{\mathcal{G}} = \frac{\text{lcm}(K)}{R(c)}$.
Set $K_t = 1$, $\forall t \in \mathcal{T}$;
repeat
Compute a critical circuit c associated with $\widetilde{\mathcal{G}}$;
OptCircuitFound:= $\forall t \in c$, K_t is a multiple of $\bar{q}_t$;
if not OptCircuitFound **then**
$\forall t \in c$, set $K_t = \text{lcm}(K_t, \frac{q_t}{\text{gcd}\{q_{t'}, t' \in c\}})$;
until OptCircuitFound

As an example, with the previous bi-valued graph from Figure 5, because the optimality test is not checked, the algorithm would continue to run with a new K-periodic schedule of periodicity vector $K = [1, 2, 1, 1]$.

4. EXPERIMENTAL RESULTS

The K-Iter algorithm is implemented as a C++ application and is available online[1]. We compared K-Iter with the state-of-the-art throughput evaluation methods for SDFG [7, 6] and CSDFG [16, 4]. SDF^3 benchmarks [8] are considered for SDFG. Our experimentations are summarized in Table 1 and is composed of four categories of graphs, including an actual DSP category. For CSDFG, we considered IB+AG5CSDF [4]; our results are presented in Table 2, which is also composed of actual and synthetic applications. All these experiments were performed on an Intel i5-4570 computer with 16GB RAM.

4.1 Evaluation of SDFG

We compare K-Iter with two optimal SDFG techniques. First, the symbolic execution based method [8], which consists of executing an application until it reaches a previously known state. This ensures a cyclic execution pattern, and then the application throughput can be computed. Second, we consider the cycle-induced sub-graph method [6], which consists of producing a dependency graph, similar to expansion techniques [10], and solving its maximal cycle ratio problem. These experiments are summarized in Table 1.

We observe that for the two category MimicDSP and LgTransient, the overall performance of K-Iter is between one and two orders of magnitude better than [6] and [8]. For the LgHSDF category, the performance of [6] and K-Iter are similar when [8] is two orders of magnitude slower. For the ActualDSP category, K-Iter is slower in average. When we look at the detail of these experiments, K-Iter is only slower for a particular graph. For this graph (namely, H263 Decoder) K-Iter computation time is 148 ms when [6] is 4ms and [8] is 36ms. This is the longest computation time observed for K-Iter in the whole SDF3 benchmark. In comparison, the longest duration for [6] was 3 sec and 22 sec for [7].

4.2 Evaluation of CSDFG

For the CSDFG evaluation, we compared the K-Iter algorithm with two existing techniques: an approximative method [4] based on periodic scheduling, and an exact technique based on symbolic execution [16]. The symbolic execution technique we used was the publicly available implementation of SDF^3 [15], including a correction of the repetition vector computation method to avoid integer overflow. For this reason, our results differ from [4]. These experiments are summarized in Table 2.

The industry cases are considered with and without buffer size constraints. In both cases K-Iter performs well. The K-Iter algorithm is several order of magnitude faster than the symbolic execution. Futhermore, K-Iter provides optimal results for applications which hasn't be done before (namely, the JPEG200 and the H264 with buffer size constraints). Indeed, if [4] provided an optimal solution for one of them, because this was an approximative method, there was no way to prove optimality until now.

For the synthetic graphs, if the periodic method always provides a solution, these solutions are not necessarily optimal. In contrast the K-Iter algorithm does provide optimal solutions for three examples and is always faster than the symbolic execution. For the two most complex examples ($\sum_t q_t$ greater than a billion) K-Iter doesn't provide solution, nor does the symbolic execution method.

5. RELATED WORK

A first throughput evaluation method has been proposed [10] which consists of the transformation of an SDFG to a particular HSDFG (a case of SDFG for which every production and consumption rate is equal to 1) where each node corresponds to a task execution and where edges are precedence relationships. This is the *expansion*. However, this transformation is not polynomial, its complexity is related to the repetition vector of an SDFG. Later, it was proved that this transformation was considering more arcs and nodes than required and two solutions were proposed to reduce the HSDFG's size [12, 6]. More recently, a max-plus albegra solution proposed to progressively build an expansion until it reachs optimality [9]. This solution uses pessimistic and optimistic throughput evaluation methods to test optimality.

In contrast, a throughput evaluation technique based on symbolic execution has been proposed for SDFG [8] and extended to CSDFG [16]. These methods rely on the fact that the state-space of a consistent (C)SDFG is a finite set. By executing every tasks *as soon as possible*, a previously known state has to be met again. Then, when a cyclic execution pattern is revealed, the throughput can easily be computed. Yet the minimal distance between two identical states is not polynomially related to the instance size. In consequence the complexity of this method is exponential.

When the throughput evaluation is used as a decision function (such as in design space exploration), accurate solutions are no longer required and approximative methods can be used. Several solutions were proposed to reduce the complexity of the problem by ignoring cycles [13] or restricting the considered schedules to periodic schedules [1, 4, 11].

6. CONCLUSION

This article presents K-Iter, an optimal algorithm to fastly evaluate CSDFG throughput and which is based on a K-periodic scheduling technique. If its worst case complexity is comparable to other optimal methods, it has been observed to be more efficient. However several cases exist for which the K-Iter algorithm is as slow as or even slower than other

[1]https://github.com/bbodin/kiter

Table 1: Average computation time of three optimal throughput evaluation methods using the SDF^3[8] benchmark.

Category name	Total number of graphs	Task count min/avg/max	Channel count min/avg/max	$\sum_t(q_t)$ min/avg/max	Evaluated algorithms K-Iter	[6]	[8]
ActualDSP	5	4/12/22	6/26/52	13/1988/4754	29.82 ms	2.42 ms	38.32 ms
MimicDSP	100	3/20/25	3/24/35	3/1008/10213	0.24 ms	2.99 ms	5.30 ms
LgHSDF	100	6/13/15	6/22/31	47/8166/208751	0.69 ms	0.40 ms	1110.31 ms
LgTransient	100	181/284/300	216/359/394	181/284/300	0.03 ms	70.13 ms	320.00 ms

Table 2: Performance comparison between, K-Iter, a periodic method [4] and an optimal method [16] using IB+AG5CSDF [4]. Percentages correspond to the result optimality and are followed by computation times. N/S means no periodic solution and ?? is unknown optimality.

	Application	Tasks	Buffers	$\sum_t(q_t)$	periodic [4]		K-Iter		symbolic execution [16]	
no buffer size	BlackScholes	41	40	11895	**100%**	0.28ms	**100%**	0.28ms	**100%**	22.43ms
	Echo	240	703	802971540	**100%**	0.12ms	**100%**	0.26ms	**100%**	37.57ms
	JPEG2000	38	82	336024	**100%**	1.07ms	**100%**	1.02ms	**100%**	3960.73ms
	Pdetect	58	76	3883200	**100%**	6.15ms	**100%**	5.96ms	**100%**	117.60ms
	H264 Encoder	665	3128	24094980	**100%**	3.83ms	**100%**	7.34ms	-	> 1 d
fixed buffer size	BlackScholes	41	80	11895	98%	0.36ms	**100%**	0.51ms	**100%**	4sec
	Echo	240	1406	802971540	33%	0.14ms	**100%**	27770.91ms	**100%**	188sec
	JPEG2000	38	164	336024	N/S	2.37ms	**100%**	531.13ms	-	> 1d
	Pdetect	58	152	3883200	**100%**	10.98ms	**100%**	10.77ms	**100%**	4928sec
	H264 Encoder	665	6256	24094980	**100%**	6.76ms	**100%**	9.60ms	-	> 1d
	graph1	90	617	752976	0.1%	3 ms	**100%**	312sec	**100%**	646sec
	graph2	70	473	2479863720	??%	4 ms	**-**	> 1d	-	> 1d
	graph3	154	671	3705826224	??%	9 ms	**-**	> 1d	-	> 1d
	graph4	2426	2900	615612	96%	218 ms	**100%**	300ms	**100%**	2320sec
	graph5	2767	4894	1874910	2%	600 ms	**100%**	18sec	**100%**	3874sec

optimal solutions. We believe such cases are key to study the complexity of the throughput evaluation problem. This is an opportunity for future direction.

7. ACKNOWLEDGEMENTS

This work is supported by the EPSRC grant PAMELA EP/K008730/1. We also thank the reviewers for their constructive feedback.

8. REFERENCES

[1] A. Benabid, C. Hanen, O. Marchetti, and A. Munier-Kordon. Periodic Schedules for Bounded Timed Weighted Event Graphs. *IEEE Transactions on Automatic Control*, 57(5):1222 – 1232, 2012.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-static data flow. *IEEE Transactions on Signal Processing*, pages 3255–3258, 1995.

[3] B. Bodin, A. Munier-Kordon, and B. Dupont de Dinechin. K-Periodic Schedules for Evaluating the Maximum Throughput of a Synchronous Dataflow Graph. In *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII*, pages 152–159, 2012.

[4] B. Bodin, A. Munier-kordon, and B. Dupont de Dinechin. Periodic Schedules for Cyclo-Static Dataflow. In *Embedded Systems for Real-Time Multimedia (ESTIMedia '13)*, pages 105–114, 2013.

[5] A. Dasdan, S. S. Irani, and R. K. Gupta. Efficient algorithms for optimum cycle mean and optimum cost to time ratio problems. *Design Automation Conference (DAC'99)*, pages 37–42, 1999.

[6] R. de Groote, J. Kuper, H. Broersma, and G. J. Smit. Max-Plus Algebraic Throughput Analysis of Synchronous Dataflow Graphs. *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pages 29–38, Sept. 2012.

[7] A. Ghamarian and M. Geilen. Liveness and boundedness of synchronous data flow graphs. *Formal Methods in Computer Aided Design*, 2006.

[8] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. J. Bekooij, B. D. Theelen, and M. Mousavi. Throughput Analysis of Synchronous Data Flow Graphs. In *International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 25–36, 2006.

[9] R. D. Groote, P. K. F. Hölzenspies, J. Kuper, and G. J. M. Smit. Incremental analysis of cyclo-static synchronous dataflow graphs. *ACM Trans. Embed. Comput. Syst.*, 14(4):68:1–68:26, Dec. 2015.

[10] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[11] D. Liu, J. Spasic, J. T. Zhai, T. Stefanov, and G. Chen. Resource optimization for csdf-modeled streaming applications with latency constraints. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 188:1–188:6, 2014.

[12] O. Marchetti and A. Munier-Kordon. Cyclic Scheduling for the Synthesis of Embedded Systems. In Y. Vivien and R. Frederic, editors, *Introduction to scheduling*, chapter 6, pages 135–164. Chapman and Hall/CRC Press, 2009.

[13] S. Meijer, H. Nikolov, and T. Stefanov. Throughput modeling to evaluate process merging transformations in polyhedral process networks. *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 747–752, Mar. 2010.

[14] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(2):175–187, 1993.

[15] S. Stuijk, M. Geilen, and T. Basten. SDFˆ3: SDF For Free. In *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 276–278. Ieee, 2006.

[16] S. Stuijk, M. Geilen, and T. Basten. Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs. *IEEE Transactions on Computers*, 57(10):1331–1345, 2008.