Optimal and fast throughput evaluation of CSDF

Citation for published version:

Digital Object Identifier (DOI):
10.1145/2897937.2898056

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
DAC '16: The 53rd Annual Design Automation Conference 2016

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
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 Universities,
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 K1 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 K2 = 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.
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 also introduced.

2.1 Model definition

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

Every task $t \in T$ is decomposed into $\varphi(t)$ phases; for every value $p \in \{1, \ldots, \varphi(t)\}$, 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 $t \in T$ corresponds to the ordered executions of the phases $t_1, \ldots, t_{\varphi(t)}$.

Furthermore, every task $t \in T$ is executed several iterations: for every integer $n$ and for every phase $p$, $(t_p, n)$ denotes the $n$th execution of the $p$th phase of $t$.

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

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

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

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 $M_0(b) = 0$.

2.2 Schedules and consistency

A feasible (or valid) schedule associated with a CSDFG is a function $S$ that associates, for every triple $(t, p, n)$ with $t \in T$, $p \in \{1, \ldots, \varphi(t)\}$ and $n \in \mathbb{N} - \{0\}$, a $S(t_p, n) \in \mathbb{R}$, the starting time of $(t_p, n)$, such that the number of data in every buffer $b \in B$ 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 $q \in (\mathbb{N} - \{0\})^{|T|}$ such that

$$\forall b = (t, t') \in B, \quad q_t \times i_b = q_{t'} \times o_b.$$
and for any integers \( p \in \{1, \ldots, \varphi(t)\} \) and \( \beta \in \{1, \ldots, K_t\} \), the period \( \mu^S_t \) and values \( S(t_p, \beta) \) are fixed. Then for any integer \( n \in \mathbb{N} \) such that \( n = \alpha \times K_t + \beta \) with \( \alpha \in \mathbb{N} \) and \( \beta \in \{1, \ldots, K_t\} \), we get
\[
\forall p \in \{1, \ldots, \varphi(t)\}, \quad S(t_p, n) = S(t_p, \beta) + \alpha \mu^S_t.
\]
If the periodicity vector \( K \) is unitary (i.e., \( K_t = 1 \), \( \forall t \in T \)), the schedule is said to be periodic, or 1-periodic.

The throughput of any task \( t \in T \) for a valid K-periodic schedule \( S \) verifies \( Th^S_t = \frac{K_t}{\mu^S_t} \). The throughput of \( S \) is then
\[
Th^S = \frac{K_t}{q_t \times \mu^S_t}.
\]
Alternatively, the period of \( S \) is denoted by
\[
\Omega^S = \frac{1}{Th^S} = \frac{q_t \times \mu^S_t}{K_t} \quad \forall t \in T.
\]
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 \( \mu^S_A = 12 \). The period of \( S \) is thus \( \Omega^S = \frac{12 \times 2}{2} = 36 \). It is important to note that for a 1-periodic schedule the maximal reachable throughput of \( G \) was only \( \Omega^S = 108 \).

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
\[
I_a(t_p, n) = \sum_{\alpha=1}^p \text{in}_b(\alpha) + (n - 1) \times \bar{o}_b.
\]
Similarly, the number of data consumed by \( t' \) in the buffer \( b \) at the completion of \( \langle t'_p, n' \rangle \) is defined by \( O_a(t'_p, n') = \sum_{\alpha=0}^p \text{outs}_b(\alpha) + (n' - 1) \times \bar{o}_b \).

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 \( M_0(a) + I_a(t_p, n) - O_a(t'_p, n') \geq 0 \).

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 \( M_0(a) + I_a(t_1, 2) - O_a(t'_2, 1) = 0 + 8 - 7 \geq 0 \).

For any pair of values \( (\alpha, \gamma) \in \mathbb{Z} \times \mathbb{N} \) - \{0\}, we set \([\alpha]^\gamma\) and \([\alpha]^{\gamma \%}\) as:
\[
[\alpha]^{\gamma \%} = \left[ \frac{\alpha}{\gamma} \right] \times \gamma \quad \text{and} \quad [\alpha]^\gamma = \left[ \frac{\alpha}{\gamma} \right] \times \gamma.
\]
Let us consider a buffer \( b = (t, t') \in B \). For any pair \( (p, p') \in \{1, \ldots, \varphi(t')\} \times \{1, \ldots, \varphi(t)\} \), let us define
\[
Q^S_a(p, p') = O_a(\langle t'_p, 1 \rangle) - I_a(\langle t_p, 1 \rangle) - M_0(b) + in_b(p).
\]
We also note \( \text{gcd}_a = \text{gcd}(\bar{o}_a, \bar{o}_b) \), \( \alpha^S_a(p, p') = \left[ Q^S_a(p, p') - \min\{in_b(p), outs_b(p')\} \right] \text{gcd}_a \) and
\[
\beta^S_a(p, p') = \left[ Q^S_a(p, p') - 1 \right] \text{gcd}_a.
\]
We now recall Theorem 2 which characterizes any feasible periodic schedule.

**Theorem 2.** ([3]). Let \( G \) be a consistent CSDFG. Any periodic schedule \( S \) of period \( \Omega^S \) is feasible if and only if, for any buffer \( b = (t, t') \in B \) and for every pair \( (p, p') \in \{1, \ldots, \varphi(t')\} \times \{1, \ldots, \varphi(t)\} \) with \( \alpha^S_a(p, p') \leq \beta^S_a(p, p') \), \( S(t'_p, 1) - S(t_p, n) \geq d(t_p) + \Omega^S \times \frac{\beta^S_a(p, p')}{q_t \times \bar{o}_b} \).

3.2 Extension to K-Periodic scheduling

The extension of Theorem 2 to K-periodic schedules with a fixed periodicity factor \( K \) comes from a transformation of the initial CSDFG \( \tilde{G} = (T, \tilde{B}) \) to another equivalent one \( \tilde{G} = (T, \tilde{B}) \) 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. \( \forall k \in \{1, \ldots, s\} \),
\[
[v]^P(k) = [v]^P(k + s) = \cdots = [v]^P(k + (P - 1) \times s) = v(k).
\]
For any task \( t \in T \), we set \( \tilde{\varphi}(t) = K_t \times \varphi(t) \) and for any \( p \in \{1, \ldots, \tilde{\varphi}(t)\} \), \( \tilde{d}(t) = [d(t)]^{K_t} \). For any buffer \( b = (t, t') \in B \), we set \( \tilde{\text{in}}_b = [\text{in}_b]^{K_t} \), \( \tilde{\text{outs}}_b = [\text{outs}_b]^{K_t} \) and \( \tilde{M}_b(b) = M_b(b) \).

A consequence of this transformation is \( \tilde{\text{in}}_b = K_t \times \text{in}_b \) and \( \tilde{\text{in}}_b = K_t \times \bar{o}_b \).

\( \tilde{G} \) is a consistent graph. Indeed, by definition of \( q \), for any buffer \( b = (t, t') \in B \), \( q_t \times \bar{o}_b = q_{t'} \times \bar{o}_b \), and thus
\[
q_t \times \frac{\text{lcm}(K)}{K_t} \tilde{\text{in}}_b = q_{t'} \times \frac{\text{lcm}(K)}{K_t} \tilde{\text{in}}_b = \tilde{o}_b,
\]
where \( \text{lcm}(K) \) is the least common multiple of values \( K_t \), \( t \in T \). Let \( \tilde{q}_t = q_t \times \frac{\text{lcm}(K)}{K_t} \) for \( t \in T \) be a repetition vector of \( \tilde{G} \).
Let us set for any buffer \( b = (t, t') \in \tilde{B} \),
\[ \mathcal{Y}(a) = \{(p, p') \in \{1, \cdots, \bar{\varphi}(t)\} \times \{1, \cdots, \bar{\varphi}(t')\}, \]
\[ \alpha^0_a(p, p') \leq \beta_a^0(p, p') \}, \]
The determination of the minimum period \( \Omega^S_0 \) of a periodic schedule can be modeled with the following linear program following Theorem 2:

Minimize \( \Omega^S_0 \) with
\[ \begin{aligned}
\forall a = (t, t') \in \tilde{B}, \quad \forall (p, p') \in \mathcal{Y}(a),
\end{aligned} \]
\[ \begin{aligned}
\tilde{S}(t', 1) - \tilde{S}(t, 1) & \geq d(t_p) + \Omega^S_0 \times \beta^0_a(p, p') \\
\forall t \in T, \forall p \in \{1, \cdots, \bar{\varphi}(t)\}, \tilde{S}(t, 1) & \in \mathbb{R}^+ \\
\Omega^S_0 & \in \mathbb{R}^+ - \{0\}
\end{aligned} \]
The next theorem highlights the relationship between the periods of \( \tilde{G} \) and \( G \):

**Theorem 3.** Let \( \tilde{S} \) be a 1-periodic feasible schedule of \( \tilde{G} \) of period \( \Omega^S_0 \). Starting times of \( \tilde{S} \) define a \( K \)-periodic feasible schedule \( S \) of \( G \) with normalized period \( \Omega^S_0 = \Omega^S_0 / \text{lcm}(K) \).

**Proof.** By construction of \( \tilde{G} \), any periodic feasible schedule \( \tilde{S} \) is a \( K \)-periodic feasible schedule \( S \) of \( G \). Then, for any task \( t \in T \), \( \mu^S_t = \mu^S_t \tilde{S} \) and \( \Omega^S_0 = \frac{2 \times \text{lcm}(K)}{K} \), thus

\[ \Omega^S_0 = \bar{q}_c \times \mu^S_t = q_t \times \frac{\text{lcm}(K)}{K_t} \times \mu^S_t = \Omega^S_0 \times \text{lcm}(K). \]

\[ \square \]

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

\[ \lambda_H = \max_{c \in C(G)} R(c). \]

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

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

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

\[ (L(e), H(e)) = (d(t_k), -\frac{\beta^0_a(p, p')}{t_a \times \bar{q}_t}). \]

The determination of the minimum period \( \Omega^S_0 \) is then equivalent to the computation of the maximum Cost-to-time Ratio, i.e. \( \Omega^S_0 = \lambda_H \).

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 \( H(c) = \frac{1}{2} \) and \( L(c) = 3 \). This therefore implies that the minimum period of a feasible periodic schedule for the CSDFG is \( \Omega^S_0 = 108 \).

### 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 \( G \).

**Theorem 4 (Optimality test).** Let \( G = (T, B) \) be a consistent CSDFG, a periodicity vector \( K \) and the associated bi-valued graph \( G = (N, E) \). Let us suppose that \( c \in C(G) \) is a critical circuit such that for every execution \( (t_p, 1) \) of \( c \), \( K_t \) is a multiple of \( \bar{q}_t = \frac{q_t}{\text{lcm}(q_t, t' \in c)} \). Then, the maximum reachable throughput of \( G \) equals \( \frac{\text{lcm}(K)}{T(c)}. \)

**Proof.** By using Theorem 3, the minimum period of the CSDFG \( \tilde{G} \) associated with \( G \) and \( K \) verifies \( \Omega^S_0 = R(c) \). Let \( C \) be a sub-graph of \( G \) 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. \( \square \)

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 \( kB \) 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 \( G \) 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 that corresponds 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^S = 108.](image-url)
noem 1 Compute the CSDFG maximal throughput

Require: A CSDFG $G = (\mathcal{T}, \mathcal{B})$
Ensure: A vector $K$ and a circuit $c$ of $G$ such that the
maximal reachable throughput of $G$ verifies $\text{Th}_G = \frac{\text{gcd}(K, R(c))}{H(c)}$.

Set $K_1 = 1, \forall t \in \mathcal{T}$;
repeat
Compute a critical circuit $c$ associated with $\tilde{G}$;
OptCircuitFound := $\forall t \in c, K_1$ is a multiple of $\tilde{q}_t$;
if not OptCircuitFound then
$\forall t \in c, \text{set } K_1 = \text{gcd}(K_1, \tilde{q}_t R(c, t'))$;
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 sched-
ulation technique we used was the publicly available implement-
tation 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.

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 con-
sidered for SDFG. Our experiments are summarized in
Table 1 and is composed of four categories of graphs,
including an actual DSP category. For CSDFG, we consid-
ered 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 algorithm 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 ex-
pansion 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 com-
parison, 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 tech-
nique based on symbolic execution [16]. The symbolic execu-
in the considered schedules to periodic schedules [1, 4, 11].

5. RELATED WORK

A first throughput evaluation method has been proposed
[10] which consists of the transformation of an SDFG to a
particular HSDF (a case of SDFG for which every produc-
tion and consumption rate is equal to 1) where each node
corresponds to a task execution and where edges are prece-
dence 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 reaches optimality [9]. This solution uses pessimistic
and optimistic throughput evaluation methods to test optimal-
ity.

In contrast, a throughput evaluation technique based on
symbolic execution has been proposed for SDFG [8] and ex-
tended 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 solu-
tions are no longer required and approximative methods can
be used. Several solutions were proposed to reduce the com-
plexity 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 SDF³[8] benchmark.

<table>
<thead>
<tr>
<th>Category name</th>
<th>Total number of graphs</th>
<th>Task count min/avg/max</th>
<th>Channel count min/avg/max</th>
<th>( \sum_t(q_t) ) min/avg/max</th>
<th>Evaluated algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActualDSP</td>
<td>5</td>
<td>4/12/22</td>
<td>6/26/32</td>
<td>13/1988/4754</td>
<td>K-Iter</td>
</tr>
<tr>
<td>MimicDSP</td>
<td>100</td>
<td>3/20/25</td>
<td>3/24/35</td>
<td>3/1008/10213</td>
<td>29.82 ms</td>
</tr>
<tr>
<td>LgHSDF</td>
<td>100</td>
<td>6/13/15</td>
<td>6/22/31</td>
<td>47/8166/208751</td>
<td>0.24 ms, 29.95 ms</td>
</tr>
<tr>
<td>LgTransient</td>
<td>100</td>
<td>181/284/300</td>
<td>216/359/394</td>
<td>181/284/300</td>
<td>0.03 ms, 70.13 ms, 320.00 ms</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison between, K-Iter, a periodic method [4] and an optimal method [16] using IB+AG5CDF[4]. Percentages correspond to the result optimality and are followed by computation times. N/S means no periodic solution and ?? is unknown optimality.

<table>
<thead>
<tr>
<th>Application</th>
<th>Tasks</th>
<th>Buffers</th>
<th>( \sum_t(q_t) )</th>
<th>periodic [4]</th>
<th>K-Iter</th>
<th>symbolic execution [16]</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackScholes</td>
<td>41</td>
<td>40</td>
<td>11895</td>
<td>100% 0.28ms</td>
<td>100%</td>
<td>0.28ms</td>
</tr>
<tr>
<td>JPEG2000</td>
<td>240</td>
<td>703</td>
<td>802971540</td>
<td>100% 0.12ms</td>
<td>100%</td>
<td>0.26ms</td>
</tr>
<tr>
<td>Pdetect</td>
<td>38</td>
<td>82</td>
<td>336024</td>
<td>100% 1.07ms</td>
<td>100%</td>
<td>1.02ms</td>
</tr>
<tr>
<td>H264 Encoder</td>
<td>58</td>
<td>76</td>
<td>3883200</td>
<td>100% 6.15ms</td>
<td>100%</td>
<td>5.96ms</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>41</td>
<td>80</td>
<td>11895</td>
<td>98% 0.36ms</td>
<td>100%</td>
<td>0.51ms</td>
</tr>
<tr>
<td>JPEG2000</td>
<td>240</td>
<td>1406</td>
<td>802971540</td>
<td>33% 0.14ms</td>
<td>100%</td>
<td>27770.91ms</td>
</tr>
<tr>
<td>Pdetect</td>
<td>38</td>
<td>164</td>
<td>336024</td>
<td>N/S 2.37ms</td>
<td>100%</td>
<td>531.13ms</td>
</tr>
<tr>
<td>H264 Encoder</td>
<td>58</td>
<td>152</td>
<td>3883200</td>
<td>100% 10.98ms</td>
<td>100%</td>
<td>10.77ms</td>
</tr>
<tr>
<td>graph1</td>
<td>90</td>
<td>617</td>
<td>752976</td>
<td>0.1% 3ms</td>
<td>100%</td>
<td>312sec</td>
</tr>
<tr>
<td>graph2</td>
<td>70</td>
<td>473</td>
<td>2479863720</td>
<td>4% 4ms</td>
<td>-</td>
<td>&gt; 1d</td>
</tr>
<tr>
<td>graph3</td>
<td>154</td>
<td>671</td>
<td>3705826224</td>
<td>9% 9ms</td>
<td>-</td>
<td>&gt; 1d</td>
</tr>
<tr>
<td>graph4</td>
<td>2426</td>
<td>2900</td>
<td>615612</td>
<td>96% 218ms</td>
<td>100%</td>
<td>300ms</td>
</tr>
<tr>
<td>graph5</td>
<td>2767</td>
<td>4894</td>
<td>1874910</td>
<td>2% 600ms</td>
<td>100%</td>
<td>18sec</td>
</tr>
</tbody>
</table>

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


