Efficient Retiming of Unfolded Synchronous Dataflow Graphs

Xue-Yang Zhu
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China
zxy@ios.ac.cn

***Abstract*—Synchronous dataflow graphs (SDFGs) are widely used to model data driven programs, for which throughput is an important real-time requirement. Retiming and unfolding are important graph transformation techniques for performance optimization of SDFGs. Retiming improves the throughput of an SDFG by redistributing its initial tokens, while unfolding by scheduling several iterations of the graph. In this paper, we present an efficient and exact method to find feasible retiming and optimal retiming of an unfolded SDFG on its original SDFG, without converting it to its equivalent homogeneous SDFG (HSDFG) and therefore without further unfolding the HSDFG. The conversion procedures are usually time and space-consuming. We also extend two state-of-the-art retiming methods for SDFGs to deal with unfolding. We implement all the three methods and perform experiments on graphs with various structures and sizes to evaluate them thoroughly. The results show that the proposal method generally outperforms the extensions of existing retiming methods, especially for the large graphs and the graphs with complex structures.**

I. Introduction and Related Work

Data-driven programs such as digital signal processing (DSP) algorithms and streaming applications are usually non-terminating and repetitive. They often operate on embedded platforms and under real-time requirements. The throughput is an important property of such systems. It decides how fast an application can be. In this paper, we are concerned with the throughput optimization of data-driven applications.

The *synchronous dataflow graphs* (SDFGs) [1] are widely used to represent DSP algorithms and streaming applications [2], [3]. Each node (also called actor) in an SDFG represents a computation and each edge models a FIFO channel. The sample rates of actors of an SDFG may differ. A simple SDFG is shown in Fig. 1 (a). *Homogeneous synchronous dataflow graphs* (HSDFGs) are a special type of SDFGs. All sample rates of actors of an HSDFG are one.

Execution of all the computations of an SDFG for the required number of times is referred to as an *iteration*. An iteration of an SDFG may include more than one execution, or *firing*, of an actor, because of its multi-rate nature. An HSDFG includes exactly one firing of an actor in an iteration. How fast an SDFG could be scheduled is limited by its *iteration period* (IP) – the minimal achievable computation time per iteration of the SDFG. The IP is the reciprocal of the *throughput*.

This work was supported in part by the National Natural Science Foundation of China (No. 61572478).

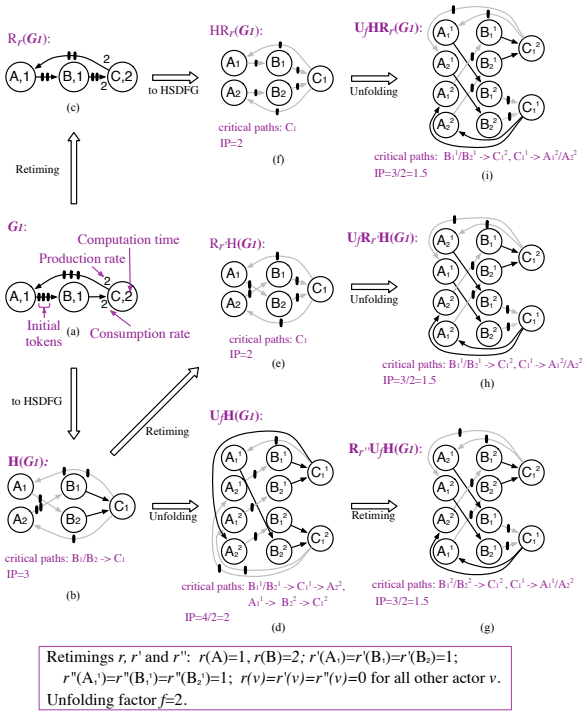

Fig. 1. (a) The SDFG G_1; (b) The equivalent HSDFG of G_1, $H(G_1)$; (c) A retimed graph of G_1, $R_r(G_1)$; (d) An unfolded graph of HSDFG $H(G_1)$; (e) A retimed graph of $H(G_1)$; (f) The equivalent HSDFG of $R_r(G_1)$; (g),(h) and (i) Graphs after combinations of the transformations of G_1. The production/consumption rates are omitted when they are 1. For clarity, edges with tokens are drawn in grey lines.

Reducing the IP of an SDFG improves the throughput of the corresponding application.

A valid SDFG can always be converted to an equivalent HSDFG [4], which captures the data dependencies among firings of actors in the SDFG in an iteration. For example, the HSDFG shown in Fig. 1 (b) is an equivalent HSDFG of SDFG G_1 (Fig. 1 (a)). The IP of an SDFG equals the IP of its equivalent HSDFG, which is the length of its critical path [5]. For example, the IP of G_1 is 3, the length of path $B_1 \rightarrow C_1$ in its equivalent HSDFG (Fig. 1 (b)).

Retiming [5] and unfolding [6] are important techniques for performance optimization of dataflow graphs. Both of them are originally applied to reduce the IP of HSDFGs. Retiming technique reduces the IP of a graph by redistributing its initial tokens. Retiming an actor *once* means firing the actor *once*. The firing moves tokens on the incoming edges of the actor to

its outgoing edges. For example, the HSDFG in Fig. 1 (e) is transformed from the HSDFG in Fig. 1 (b) with retiming A_1, B_1 and B_2 *once*, respectively. The transformation reduces the IP from 3 to 2.

Unfolding technique optimizes a graph by scheduling several iterations of the graph. A graph unfolded with an *unfolding factor* f, called f-unfolded graph, describes f consecutive iterations of its original graph. For example, the HSDFG in Fig. 1 (d) is an unfolded graph of the HSDFG in Fig. 1 (b) with unfolding factor $f = 2$. The transformation reduces the IP from 3 to 2.

Retiming explores intra-iteration parallelism, while unfolding explores inter-iteration parallelism, of an SDFG. Combining these two techniques, the IP can be further reduced. See Fig. 1 (g), (h) and (i) for example. After unfolding and retiming, the IP is further reduced to 1.5. In this paper, we present solutions for both *feasible retiming*, which retimes a graph to meet a given IP constraint, and *optimal retiming*, which retimes a graph to achieve the smallest IP, problems of f-unfolded SDFGs.

Since a valid SDFG can always be converted to an equivalent HSDFG, it can be analyzed or optimized with existing techniques for HSDFGs [7]. However, the conversion procedures are usually time and space-consuming [8]. The size of the equivalent HSDFG can be exponentially larger than that of the original SDFG in some extreme cases. For example, a satellite receiver that modeled in an SDFG with 22 actors has 4515 actors in its equivalent HSDFG [3]. Unfolding further multiplies the problem space. Methods for retiming SDFGs without converting them to their equivalent HSDFGs are presented in [9], [10], [11] and [12]. Only feasible retiming is considered in [9] and [10]. Only optimal retiming is considered in [11]. Both feasible retiming and optimal retiming are considered in [12]. Algorithms in [11] and [12] are exact. Feasible retiming method in [12] is sufficient and necessary, while both methods in [9] and in [10] are sufficient but not necessary. However, how to retime an unfolded SDFG on its original SDFG is still left unsolved.

In this paper, we present an efficient and exact method to find feasible retiming and optimal retiming of an unfolded SDFG on its original SDFG, without explicitly converting it to its equivalent HSDFG and therefore without further unfolding the HSDFG. Our contributions are as follows.

1) We show that some results about the relationship between an SDFG and its equivalent HSDFG in [12] can be extended to the relationship between an SDFG and its unfolded graph. And therefore the IP of the unfolded SDFG can be computed directly on its original SDFG.
2) Combined them with the results about the equivalence of transformations of SDFGs, which is presented as work-in-progress in [13], we prove that retiming of an unfolded SDFG can be safely computed on its original SDFG.
3) With these guarantees the retiming algorithms in [11] and [12] are extended to deal with unfolded SDFGs.
4) We present a new algorithm for the IP computation of unfolded SDFGs, based on which a feasible retiming algorithm, which is sufficient and necessary, and an optimal retiming algorithm are proposed.
5) We implement all the three methods and perform experiments on graphs with various structures and sizes to evaluate them thoroughly.

Let $|V|$ and $|E|$ be the numbers of actors and edges in an SDFG, respectively, and $|V'|$ and $|E'|$ be the numbers of actors and edges of its equivalent HSDFG, respectively. As the experimental results in [12] show, its optimal retiming method is faster than that in [11] on most cases, but is also much slower on some cases with complex structure. The reason is that the optimal retiming algorithms in [12] has an exponential worst case complexity, while the worst case complexity of the algorithm in [11] is $O(|V||E||V'|^2)$. Both retiming methods have their own advantages. Detailed analysis is shown in [12].

In the proposed method, the advantage of the methods in [12] is kept and the worst case complexity is improved significantly. The worst case complexities of the proposed feasible retiming and optimal retiming algorithms are $O(f^2|V'||E'|)$ and $O(IP \cdot f^3|V'||E'|)$, respectively. Furthermore, on most cases, our algorithms are much faster than the worst case. The efficiency of our method is also shown in the experimental results. Our experimental results also reveal that retiming usually has a better effects than unfolding on throughput optimization of SDFGs.

The remainder of this paper is organized as follows. We first introduce the main concepts used in this paper and formulate the problems we address in Section II. The basis for the correctness of our method is presented in Section III and the details of the proposed retiming method are illustrated in Section IV. Section V provides an experimental evaluation. Finally, Section VI concludes.

II. Preliminaries and Problem Formulation

An SDFG is a finite directed graph $G = (V, E)$, in which V is the set of actors and E is the set of directed edges. Actor v is weighted with its computation time $t(v)$, a nonnegative integer. The source actor and sink actor of edge e are denoted by $src(e)$ and $snk(e)$, respectively. Edge e is weighted with three properties, $d(e)$, $p(e)$ and $c(e)$. The *delay* $d(e)$ is the number of initial tokens on e, the *production rate* $p(e)$ is the number of tokens produced onto e by each firing of $src(e)$, and the *consumption rate* $c(e)$ is the number of tokens consumed from e by each firing of $snk(e)$. The set of incoming edges to actor v is denoted by $InE(v)$, and the set of outgoing edges from v by $OutE(v)$. If $p(e) = c(e) = 1$ for each $e \in E$, G is a *homogeneous SDFG* (HSDFG).

A simple SDFG G_1 is depicted in Fig. 1 (a). Actors A and B need one unit of time to finish and C needs two units. The production rate and consumption rate of edge $\langle C, A\rangle$ are 2 and 1, respectively, and there are three initial tokens on $\langle C, A\rangle$. That is, $p(\langle C, A\rangle) = 2$, $c(\langle C, A\rangle) = 1$ and $d(\langle C, A\rangle) = 3$. A firing of actor C will produce 2 tokens on $\langle C, A\rangle$ and actor A can fire only when there is at least one token on $\langle C, A\rangle$.

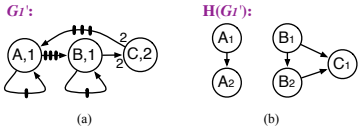

Fig. 2. An example for SDFG with self-loops. (a) The SDFG G_1'; (b) The equivalent HSDFG of G_1', $H(G_1')$. Edges with initial tokens in HSDFGs are omitted.

Actor A can fire three times before the end of the first firing of C, because there are three initial tokens on edge $\langle C, A\rangle$. Actors without any incoming edge are free to fire at any time.

SDFG $G = (V, E)$ is *sample rate consistent* [1] if and only if there exists a positive integer vector $q(V)$ satisfying *balance equations*, $q(src(e)) \times p(e) = q(snk(e)) \times c(e)$ for all $e \in E$. The smallest such q is called the *repetition vector*. We use q to represent the repetition vector directly. The repetition vector of G_1 is $q = [2, 2, 1]$, corresponding to actors A, B and C, for example.

An *iteration* of an SDFG is a firing sequence in which each actor v occurs exactly $q(v)$ times. The average computation time per iteration is called the *iteration period* (IP). The IP of SDFG G is denoted by $G.ip$.

It is always possible to convert a sample rate consistent SDFG to an equivalent HSDFG [4], which captures the dependencies among firings in an iteration of the SDFG. We denote the transformation by *H-transformation*. For each actor v in SDFG G, there are $q(v)$ copies of it in the HSDFG, denoted by $(v, i), i \in [1, q(v)]$. In examples and figures, however, for the simplicity, we use the form like A_1 to represent $(A, 1)$. The H-transformation is also recapped as a mapping H from SDFG G to its equivalent HSDFG $H(G)$ in [12].

A sample rate consistent SDFG is *deadlock-free* if there is at least one initial token on any cycle in its equivalent HSDFG. Only sample rate consistent and deadlock-free SDFGs are meaningful in practice. Therefore we consider only such SDFGs.

The IP of an SDFG equals the IP of its equivalent HSDFG, which is the length of critical paths of the HSDFG [5]. For example, the IP of G_1 (Fig. 1 (a)) is 3, the length of critical path $B_1 \rightarrow C_1$ in $H(G_1)$ (Fig. 1 (b)). That is, we always have $G.ip = H(G).ip$.

A firing of an actor with internal state depends on the result of its previous firings, therefore different firings of such an actor are not allowed to fire concurrently. This limitation can be explicitly represented as a self-loop on the actor. See Fig. 2 for example. The self-loops on actors A and B of SDFG G_1' (Fig. 2 (a)) put the data dependencies between the firings of the same actor. The dependencies are explicitly shown on $H(G_1')$ (Fig. 2 (b)), in which the edge $\langle A_1, A_2\rangle$ indicates that A_2 cannot start before A_1 has finished and $\langle B_1, B_2\rangle$ the same. On the contrary, in $H(G_1)$ (Fig. 1 (b)), there is no edge between A_1 and A_2 and no edge between B_1 and B_2, because there are no self-loops on A and B of G_1. This means that in G_1, A_1 and A_2 can fire concurrently, and so do B_1 and B_2.

Retiming is a graph transformation that redistributes the graph's initial tokens while remains its functionality unchanged. It can be defined either in a forward fashion [9] or in a backward fashion [5], [11], [12]. They are equivalent. We use a forward retiming in this paper. That is, retiming an actor once means firing this actor once. The firing moves tokens on the incoming edges of the actor to its outgoing edges according to the data rates of these edges. We denote the transformation with retiming function r by R_r*-transformation*.

Definition 1. *(R_r-transformation) The* R_r-transformation *of an SDFG $G = (V, E)$ with a retiming function $r : V \rightarrow \mathbb{Z}$, where $\mathbb{Z}$ is the set of integers, is a mapping R_r that maps G to a new SDFG $R_r(G) = (V, E')$ as follows:*

For each edge $e = \langle u, v\rangle \in E$, there is an edge $e' = \langle u, v\rangle \in E'$ with $p(e') = p(e)$, $c(e') = c(e)$, and $d(e') = d(e) + p(e)r(u) - c(e)r(v)$.

The SDFG in Fig. 1 (c) is transformed by G_1 with retiming r: $r(A) = 1$, $r(B) = 2$ and $r(C) = 0$, for example. The retiming transformation moves one token from edge $\langle C, A\rangle$ to edge $\langle A, B\rangle$ and moves two tokens from $\langle A, B\rangle$ to $\langle B, C\rangle$. For a given IP constraint dip, a *feasible retiming* for SDFG G is a retiming r, such that $R_r(G).ip \leq dip$; and an *optimal retiming* is a retiming r, such that $R_r(G).ip$ as small as posssible.

A graph unfolded with an *unfolding factor* f describes f consecutive iterations of its original graph. We denote the transformation with unfolding factor f by U_f*-transformation*. The procedure that constructs an unfolded graph for an HSDFG is presented in [7]. The HSDFG $H(G_1)$ unfolded with unfolding factor $f = 2$ is shown in Fig. 1 (d), in which nodes like A_1^2 denotes the 1st firing of actor A in the 2nd iteration of $H(G_1)$. The unfolding of an SDFG is defined by the unfolding of its equivalent HSDFG.

Definition 2. *(U_f-transformation) Let $G = \langle V, E\rangle$ be a sample-rate consistent SDFG, q its repetition vector, and f an unfolding factor. Let $Q = f \cdot q$. The* U_f-transformation *is a mapping that maps G to an HSDFG $U_f H(G) = \langle V', E'\rangle$ as follows:*

1) *For each $v \in V$ and $i \in [1, Q(v)]$, there is an actor $(v, i) \in V'$ with $t(v, i) = t(v)$;*
2) *For each $e = \langle u, v\rangle \in E$, $i \in [1, Q(u)]$, $k \in [1, p(e)]$, and*

$$j = \left\lfloor \frac{((i-1)p(e) + (k-1) + d(e)) \bmod c(e)Q(v)}{c(e)} \right\rfloor + 1,$$

there is an edge $e' = \langle (u, i), (v, j), k\rangle \in E'$ with

$$d(e') = \left\lfloor \frac{(i-1)p(e) + (k-1) + d(e)}{c(e)Q(v)} \right\rfloor.$$

We call $U_f H(G)$ a f-unfolded graph of SDFG G. Obviously, $H(G)$ is 1-unfolded graph of G. An SDFG is in fact a directed multigraph, in which between two actors may have multiple edges. Since we are only concerned with the data dependencies among actors, we can just take an SDFG as a directed graph. Similarly, we consider $U_f H(G)$ as a directed graph by using edge $\langle (u, i), (v, j)\rangle$ to represent each set of edges $\langle (u, i), (v, j), k\rangle$, with $d(\langle (u, i), (v, j)\rangle) = \min_k d(\langle (u, i), (v, j), k\rangle)$.

An U_f-transformation takes f iterations of a graph as a *cycle*. There are $Q(u)$ firings in an cycle of $U_fH(G)$ for each actor $u \in G$. We use (u,k) to represent the k^{th} firing of actor u in the unfolded graph, where $k = (j-1)\cdot q(u) + i$ is the i^{th} firing in j^{th} iteration (of G). In figures and examples, the form such as A_1 is used. Only when the discussion is focus on the order of iterations, the form u^j is used.

The *cycle period* (CP) of the unfolded graph $U_fH(G)$, denoted by $U_fH(G).cp$, is the length of its critical path. Since $U_fH(G)$ includes f iterations of G, we have

$$U_fH(G).ip = \frac{U_fH(G).cp}{f}.$$

The combinations of the transformations are also transformations. For simplicity, we use $U_fHR_r(G)$ to express the transformed graph $U_f(H(R_r(G)))$, and so on. All the three transformations of SDFGs preserve their functionalities [4], [5], [6], and so do their combinations.

Given an SDFG, an unfolding factor f and a desired CP dcp, the problems we address are: how to find a feasible retiming r, such that $U_fHR_r(G).cp \le dcp$? how to find an optimal retiming r such that $U_fHR_r(G).cp$ as small as possible (and therefore $U_fHR_r(G).ip$ as small as possible)?

III. The Basis for The Correctness of Our Method

Our method works directly on SDFGs, without explicitly converting an SDFG to its equivalent HSDFG and unfolded HSDFG. This raises a question: why can we compute the retiming of an unfolded SDFG on the original SDFG when it is supposed to be done on an unfolded HSDFG? We answer this question in this section. We show in Section III-A that retiming on an SDFG is equivalent to the retiming on its unfolded HSDFG and in Section III-B that the CP of an unfolded SDFG can be computed directly on the original SDFG. These two results guarantee that the retiming of an unfolded SDFG can be safely computed on its original SDFG.

A. The Equivalence of Transformations of SDFGs

Definition 3. *(IP equivalence) Let $\mathbf{T_1}$ and $\mathbf{T_2}$ be two transformations. If for any SDFG G, $\mathbf{T_1}(G).ip = \mathbf{T_2}(G).ip$, then $\mathbf{T_1}$ is IP equivalent with $\mathbf{T_2}$, denoted by $\mathbf{T_1} \equiv_{ip} \mathbf{T_2}$.*

Our method is correct only when for any retiming r, there exists a retiming r'' such that U_fHR_r and $R_{r''}U_fH$ are IP equivalent. The property is guaranteed by the results about the equivalence of the transformations of SDFGs, which is presented as work-in-progress in [13]. We illustrate the results in this section.

It is proven in [7] that the order of retiming and unfolding is immaterial for the IP of an HSDFG.

Lemma 1. *[7] On HSDFGs, for the given retiming function r' and unfolding factor f, $U_fR_{r'} \equiv_{ip} R_{r''}U_f$, where r'' is a retiming function on the corresponding f-unfolded HSDFG and for each v in the HSDFG and $j \in [1,f]$,*

$$r''(v^j) = \begin{cases} 1 + \lfloor \frac{r'(v)}{f} \rfloor, & \text{if } j \le r'(v) \bmod f; \\ \lfloor \frac{r'(v)}{f} \rfloor, & \text{otherwise.} \end{cases} \quad (1)$$

This property is exemplified in Fig. 1. The HSDFG $U_fR_{r'}H(G_1)$ (Fig. 1 (h)) is transformed from $H(G_1)$ by first retiming with $R_{r'}$ ($r'(A_1) = r'(B_1) = r'(B_2) = 1$) and then unfolding with U_f ($f = 2$); the HSDFG $R_{r''}U_fH(G_1)$ (Fig. 1 (g)) is transformed from $H(G_1)$ by first unfolding with U_f ($f = 2$) and then retiming with $R_{r''}$ ($r''(A_1^1) = r''(B_1^1) = r''(B_2^1) = 1$). The retiming functions r'' is defined by r' according to Eqn. (1). The IPs of $U_fR_{r'}H(G_1)$ and $R_{r''}U_fH(G_1)$ are both 1.5. The two optimization paths for HSDFG $H(G_1)$: (b) $\rightarrow$ (e) $\rightarrow$ (h) and (b) $\rightarrow$ (d) $\rightarrow$ (g), are equivalent. This result guarantees that retiming can be performed on the HSDFG instead of on the unfolded HSDFG, whose size is f times of the original HSDFG.

The equivalence of retiming on and SDFG and on its equivalent HSDFG is proven in [12].

Lemma 2. *[12] Given SDFG G and a retiming function r on G, $HR_r(G)$ and $R_{r'}H(G)$ are equivalent under isomorphism, where r' is a retiming function on $H(G)$ and for $v \in G$ and $i \in [1, q(v)]$,*

$$r'(v,i) = \begin{cases} 1 + \lfloor \frac{r(v)}{q(v)} \rfloor, & \text{if } i \le r(v) \bmod q(v); \\ \lfloor \frac{r(v)}{q(v)} \rfloor, & \text{otherwise.} \end{cases} \quad (2)$$

For example, $R_{r'}H(G_1)$ in Fig. 1 (e) is equivalent to $HR_r(G_1)$ in Fig. 1 (f) under the isomorphic function $\phi : R_{r'}H(G_1) \rightarrow HR_r(G_1)$ that is defined by $\phi(A_1) = A_2$, $\phi(A_2) = A_1$ and $\phi(v) = v$ for other actor v. $HR_r(G_1)$ is transformed from G_1 by first retiming with R_r ($r(A) = 1, r(B) = 2$) and then converting to the equivalent HSDFG; $R_{r'}H(G_1)$ is transformed from G_1 by first converting to its equivalent HSDFG and then retiming with $R_{r'}$ ($r'(A_1) = r'(B_1) = r'(B_2) = 1$). The retiming functions r' is defined by r according to Eqn. (2). The two optimization paths for SDFG G_1: (a) $\rightarrow$ (c) $\rightarrow$ (f) and (a) $\rightarrow$ (b) $\rightarrow$ (e), are equivalent. The conclusion that $HR_r \equiv_{ip} R_{r'}H$ on SDFGs can be drawn directly by Lemma 2. This result guarantees that retiming can be performed on the original SDFG instead of on the equivalent HSDFG, whose size may be much larger than that of the original SDFG.

Then the IP equivalence of U_fHR_r, $U_fR_{r'}H$ and $R_{r''}U_fH$ can be derived from Lemma 1 and Lemma 2.

Theorem 1. *On SDFGs, for a given retiming function r and unfolding factor f, $R_{r''}U_fH \equiv_{ip} U_fR_{r'}H \equiv_{ip} U_fHR_r$, where r' and r'' are defined according to Eqn. (2) and Eqn. (1), respectively.*

Proof. For any SDFG G, $H(G)$ is an HSDFG. By Lemma 1, for any legal retiming r' and unfolding factor f on $H(G)$, the IP of $U_fR_{r'}(H(G))$ equals the IP of $R_{r''}U_f(H(G))$. Therefore, we have that $R_{r''}U_fH \equiv_{ip} U_fR_{r'}H$.

For any legal retiming r on G, by Lemma 2, $HR_r(G)$ and $R_{r'}H(G)$ are equivalent under isomorphism. Suppose the isomorphic function $\phi : R_{r'}H(G) \rightarrow HR_r(G)$ is defined by $\phi(u) = v$ for each $u \in R_{r'}H(G)$. $U_fHR_r(G)$ and $U_fR_{r'}H(G)$ are equivalent under the isomorphic function $\phi' : U_fR_{r'}H(G) \rightarrow U_fHR_r(G)$ defined by $\phi'(u^j) = v^j$ for each $u^j \in U_fR_{r'}H(G)$ and $1 \le j \le f$. Hence, $U_fR_{r'}H$

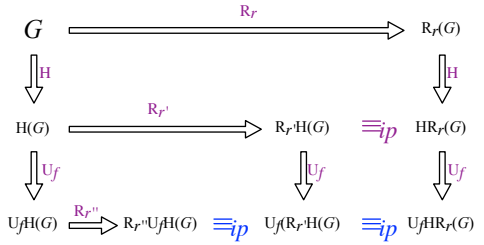

Fig. 3. The relationship of the transformations of SDFGs.

and $U_f H R_r$ are equivalent under isomorphism, and therefore $U_f R_{r'} H \equiv_{ip} U_f H R_r$. □

The relationship of the transformations of SDFGs are summarized in Fig. 3. Examples of the combinations of transformations of G_1, $R_{r''} U_f H(G_1)$, $U_f R_{r'} H(G_1)$ and $U_f H R_r(G_1)$, are shown in Fig. 1 (g), (h) and (i), respectively. The three optimization paths for SDFG G_1: (a) → (b) → (d) → (g), (a) → (b) → (e) → (h) and (a) → (c) → (f) → (i), are equivalent.

The first path is the most straightforward solution. Along this path, one needs to explicitly carry out the H, U_f and $R_{r''}$- transformations and compute the IP of the unfolded HSDFG. The second path improve it by carrying out $R_{r'}$-transformations on $H(G)$ rather than $U_f H(G)$. A method to compute the CP of an unfolded HSDFG directly on the original HSDFG is presented in [14]. Therefore along this path, we do not need to carry out U_f-transformations. Along the last path, it's possible to carry out R_r-transformations on G, the smallest graph, comparing with $H(G)$ and $U_f H(G)$. If the CP of the unfolded SDFG can be computed on the original SDFG, no explicitly H and U_f-transformations are needed.

B. The Basis for CP Computation on SDFGs

In this section we illustrate how the CP of an unfolded SDFG can be computed directly on the original SDFG. We show that certain walks in an SDFG in fact cover all the critical paths in its unfolded graph, and therefore the CP of the unfolded SDFG is the length of the longest one of these walks. Unlike a path, there may be repeated actors or edges in a walk.

The properties of the relationship between an SDFG and its equivalent HSDFG is illustrated in [12]. Recall that in an iteration of an SDFG, each actor v fires $q(v)$ times; and according to the definition, in a cycle of an f-unfolded SDFG, each actor v fires $Q(v)$ ($Q(v) = f \cdot q(v)$) times. Since the $H(G)$ captures the dependencies among firings in an iteration of SDFG G and $U_f H(G)$ capture the dependencies in f iterations, the results in [12] can be extended to the relationship between an SDFG and its f-unfolded graph by replacing all $q(v)$ with $Q(v)$. We review the extension in this section.

A *zero-delay path* in an HSDFG is a longest path each of whose edge has no initial tokens. Path p_{11} ($B_1 \to C_1 \to A_1$) in Fig. 4 (b) for example is a zero-delay path of $H(G_2)$, while path $C_1 \to A_1$ is not because it is a part of p_{11}. A *critical path* of an HSDFG is a longest zero-delay path. For example, p_{11} and p_{12} are critical paths of $H(G_2)$ and p_{21} and p_{22} are critical paths of $U_f H(G_2)$.

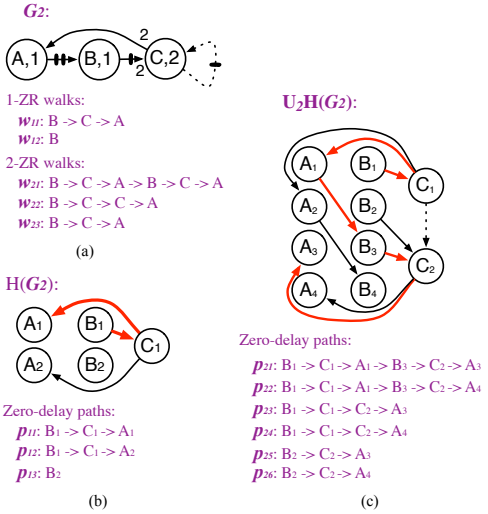

Fig. 4. The f-ZR walks of SDFGs and their corresponding zero-delay paths in the unfolded graphs. (a) The SDFG G_2; (B) the equivalent HSDFG of G_2, i.e. 1-unfolded graph of G_2; (c) the 2-unfolded graph of G_2. Edges with initial tokens in HSDFGs are omitted.

Below, we always suppose that G is an SDFG and f an unfolding factor.

Definition 4. *(f-ZR walk) For actors $u, v \in G$, if there exists a zero-delay path*

$$p \in U_f H(G) : (v_0, l_0) \to (v_1, l_1) \to \cdots \to (v_n, l_n),$$

then the walk $w \in G : v_0 \to v_1 \to \cdots \to v_n$ is called a f zero-delay reachable *(f-ZR) walk of G.*

See Fig. 4 for example. The walk w_{11} ($B \to C \to A$) is a 1-ZR walk of G_2, corresponding to p_{11} ($B_1 \to C_1 \to A_1$), a zero-delay path of $H(G_2)$; it also corresponds to p_{12}. The walk w_{21} ($B \to C \to A \to B \to C \to A$) is a 2-ZR walk of G_2, corresponding to p_{21} ($B_1 \to C_1 \to A_1 \to B_3 \to C_2 \to A_3$), a zero-delay path of $U_2 H(G_2)$; it also corresponds to p_{22}.

By Definition 4, f-ZR walks of SDFG G cover all zero-delay paths in $U_f H(G)$. Since the CP of G's $f-$unfolded graph is the length of the critical paths in $U_f H(G)$, it is also the length of the longest f-ZR walks in G.

Now the problem is how to find a longest f-ZR walk in G. Searching for the f-ZR walks in an SDFG is in fact searching (implicitly) for the zero-delayed paths in its f-unfolded graph. We can see that a f-ZR walk of an SDFG corresponds to a set of zero-delay paths in its f-unfolded graph. Below we show that each f-ZR walk can be found by searching a particular zero-delay path or part of it.

Theorem 2. *Walk $w : v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} v_n$ is a f-ZR walk in G if and only if there is a path $p : (v_0, l_0) \to (v_1, l_1) \to \cdots \to (v_n, l_n)$ in $U_f H(G)$, where $v_0 = u, v_n = v, l_0 = 1$, and for each $i \in [1, n]$:*

$$l_i = \left\lfloor \frac{(l_{i-1} - 1)p(e_i) + d(e_i)}{c(e_i)} \right\rfloor + 1 \text{ and } l_i \le Q(v_i). \tag{3}$$

The detailed proof of Theorem 2 in the case of 1-unfolded graph is referred to [12]. Path p defined in Theorem 2 is one or part of one zero-delay path beginning with some actor $(u, 1)$. We can always find some walks longer than those walks that correspond to a part of a zero-delay path. So we neglect those walks, considering only the walks corresponding a complete zero-delay path. This means that we can always find the longest f-ZR walk that begins with actor u in G by searching zero-delay paths beginning with $(u, 1)$ in $U_f H(G)$. For example, by (implicitly) searching path p_{11} in $H(G_2)$ (Fig. 4 (b)), we find 1-ZR walk w_{11}, which is the longest 1-ZR walk of G_2. It is the only search we need for the computation of the CP of the 1-unfolded graph of G_2. Similarly, walks w_{21} and w_{22}, which are found by searching paths p_{21} and p_{23} of $U_2 H(G_2)$ (Fig. 4 (c)), respectively, are the only walks we need for the computation of the CP of the 2-unfolded graph of G_2.

A *critical f-ZR walk* of G is a f-ZR walk that corresponds to a zero-delay path in $U_f H(G)$ as defined in Theorem 2. Notice that, a critical f-ZR walk is not necessarily corresponding to a critical path in $U_f H(G)$, but the f-ZR walk corresponding to a critical path in $U_f H(G)$ is definitely a critical f-ZR walk.

Theorem 3. *The CP of an f-unfolded graph of SDFG G is the length of its longest critical f-ZR walk.*

IV. Retiming of Unfolded SDFGs

Retiming algorithms generally work by relaxation [5], [9], [10], [11], [12]. A feasible retiming is a repeated procedure that checks the IP of the graph and then shortens the zero-delay paths/ZR walks that are longer than the given IP constraint until finds a retiming function that meets the IP constraint or concludes no such function existing. An optimal retiming is a procedure that repeatedly reduces the IP constraint and checks its feasibility until a feasible retiming leading to the smallest possible IP (optimal IP) is found.

Generally, the speed of a retiming algorithm is affected by the execution time for an IP/CP check and the number of the checks, that is, the *retiming steps*.

In this section, we first present a brief description about how to extend the state-of-the-art retiming methods in [11] and [12] to deal with unfolding. A new CP computation algorithm is introduced in Section IV-B, followed with feasible retiming algorithm and optimal retiming algorithm for unfolded SDFGs in Sections IV-C and IV-D, respectively.

A. Unfolding Extension to the methods in [11] and [12]

Based on discussions in Section III, the feasible retiming algorithm and optimal retiming algorithm in [12] and the optimal retiming algorithm in [11] can be extended to deal with unfolded SDFGs by replacing all $q(v)$ in the IP computation and retiming procedures with $Q(v)$. The extensions of those in [12] still have an exponential worst case complexity, because its CP computation is exponentially complex. The extension of the algorithm in [11] has an $O(f^2|V||E||V'|^2)$ worst case complexity.

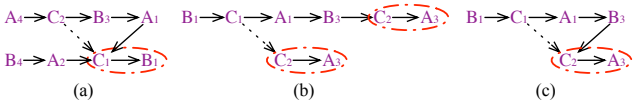

Fig. 5. Search paths in $U_2 H(G_2)$ of the CP computation in different methods. (a) in [11]; (b) in [12]; (c) in the proposed method. Shared subpaths are circled in red dot lines.

B. CP Computation

The CP computation of an unfolded SDFG G is in fact finding the longest zero-delay paths in the HSDFG $U_f H(G)$. The procedure is to find the length of the longest path to each actor (u, i), and then take the largest one as the CP. The lengths of the longest paths can be computed forward or backward. Lengths of longest forward paths (T_f) and lengths of reverse paths (T_r) to actors of $U_2 H(G_2)$ are shown in Table I. From any of them, we can get that $U_2 H(G_2).cp = 8$. The CP computation procedures in [11] and [12] find the CP of an SDFG by computing part of T_f, as shown in the first two parts of Table II. The proposed method finds the CP by computing part of T_r as we illustrate below.

TABLE I
Lengths of longest forward paths (T_f) and lengths of reverse paths to actors (T_r) of $U_2 H(G_2)$

	T_f				T_r			
	1	2	3	4	1	2	3	4
A	4	2	8	8	5	2	1	1
B	1	1	5	5	8	4	4	1
C	3	7	N	N	7	3	N	N

Obviously, we have

$$\forall u \in G : T_r(u, i) \geq T_r(u, j) \text{ for } i < j. \tag{4}$$

Fig. 5 shows the search paths in $U_2 H(G_2)$ of the CP computations in different methods. When SDFG G_2 has no self-loop on actor C (dot line in Fig. 4), the CP computation of [12] outperforms that of [11] on this case because it needs only to search one path, $B_1 \to C_1 \to A_1 \to B_3 \to C_2 \to A_3$ (Fig. 5 (b)), while that of [11] needs to search two paths, $A_4 \to C_2 \to B_3 \to A_1 \to C_1 \to B_1$ and $B_4 \to A_2 \to C_1 \to B_1$ (Fig. 5 (c)). The shared subpath $C_1 \to B_1$ is computed only once in [11]. But when there is a self-loop, the CP computation of [12] needs to search two paths as shown in Fig. 5 (b) and may computes the shared subpath $C_2 \to A_3$ twice, depending on which path is searched first. When there are many such shared subpaths, the CP computation of [12] goes to be very slow, while that of [11] has no repeated computations of those shared subpaths, as show in Fig. 5 (a).

We can see that these two methods have their own advantages and disadvantages. Combining the advantages of these two approaches, we present a new CP computation procedure of f-unfolded SDFGs, uSdfCP, in Algorithm 1. Procedure uSdfCP searches as fewer paths as that in [12] and computes shared subpaths only once as that in [11]. Again, take $U_2 H(G_2)$ as an example. The search paths of uSdfCP is shown in Fig. 5 (c). In the case that there is a self-loop on actor C, it only searches two paths as that in Fig. 5 (b) and

computes the shared subpath ($C_2 \rightarrow A_3$) only once as that in Fig. 5 (a).

Algorithm 1 uSdfCP(G, f, dcp)

Require: A valid SDFG $G = \langle V, E \rangle$, unfolding factor f and the desired CP dcp
Ensure: $criT$ and CP
1: $\forall u \in G, i \in [1, f \cdot q(u)]$, let $criT(u, i) = -1$
2: **for all** $u \in V$ **do**
3: $j = 1$
4: $criT(u, 1) = getNextT(u, 1)$
5: **while** $criT(u, j) > dcp$ and $j < q(u)$ **do**
6: $j = j + 1$
7: $criT(u, j) = getNextT(u, j)$ {To make sure that all $criT(u, i)$s larger than dcp are computed. }
8: **end while**
9: **end for**
10: $CP = \max_{u \in V, i \in [1, f \cdot q(u)]} criT(u, i)$
11: **return** CP and $criT$

getNextT(u, l)

12: **if** $l > f \cdot q(u)$ **then**
13: **return** 0
14: **end if**
15: **if** $criT(u, l) \neq -1$ **then**
16: **return** $criT(u, l)$ {This step guarantees that the shared subpaths are only searched once.}
17: **end if**
18: $maxL = -1$
19: **if** $OutE(u) = \emptyset$ **then**
20: $maxL = t(u)$
21: **else**
22: **for all** $e \in OutE(u)$ **do**
23: $l' = \lfloor \frac{(l-1)p(e)+d(e)}{c(e)} \rfloor + 1$ {by Eqn. (3)}
24: $v = snk(e)$
25: $maxL = \max(maxL, getNextT(v, l') + t(u))$
26: **end for**
27: **end if**
28: $criT(u, l) = maxL$
29: **return** $criT(u, l)$

To make the difference clearer, Algorithm 1 is structured the same as the IP computation (sdfIP) in [12] and the same names are used for key variables. The most important variable is vector $criT$. Rather than used to contain the lengths of ZR walks [12] corresponding to T_f, $criT$ here is used to contain the lengths of reverse f-ZR walks, corresponding to T_r. According to Theorem 3, it is not necessary to compute the lengths of all f-ZR walks, rather, searching those critical f-ZR walks, which begin with some $(u, 1)$ (Line 4) is sufficient. A recursive procedure getNextT is used as a subroutine to explore f-ZR walks in a depth-first search strategy, and to compute the $criT$. For each actor u, its successor in a f-ZR walk is found according to Eqn. (3) (Line 23). The recursive process terminates when (u, l) is the last firing of a zero-delay path in the unfolded graph (Lines 12-14), or when $criT(u, l)$ has been computed (Lines 15-17), that is, there are shared subpaths from (u, l) to the end of a zero-delay path and those subpaths have been explored. Because the SDFGs we consider are deadlock-free, a zero-delay path of its unfolded graph is definitely ends at some (u, l) with $l > f \cdot q(u)$; otherwise a loop without initial token exists and a deadlock occurs. Therefore, the recursive process is guaranteed to terminate.

Parameter dcp is not necessary for the CP computation but is helpful for the retiming steps as we show later. It is set to be *infinite* when uSdfCP is only used to compute the CP of an f-unfolded graph. Although this procedure has an $O(f|E'|)$ worst case complexity, in most cases, the searched edges are much fewer than $f|E'|$. See uSdfCP($G_2, 2, \infty$) for example. There are 14 edges in $U_f H(G_2)$, i.e. $f|E'| = 14$, of which only 6 edges are on the paths (p_{21} and p_{23} in Fig. 4 (c)) corresponding to critical 2-ZR walks (w_{21} and w_{22} in Fig. 4 (c)). As a result, only some of the elements of $criT$ are computed and the CP is the maximum of them. As shown in Table II, after uSdfCP($G_2, 2, \infty$) is finished, there are still elements in $criT$ uncomputed, remaining as -1, as initialized at Line 1.

TABLE II
$criT$ OF G_2 IN DIFFERENT CP COMPUTATION METHODS

	[11]				[12]				uSdfCP(G_2, 2, ∞)				uSdfCP(G_2, 2, 6)			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
A	4	2	-1	8	4	-1	8	-1	5	-1	1	-1	5	-1	1	-1
B	1	-1	5	5	1	-1	5	-1	8	-1	4	-1	8	4	4	1
C	3	7	N	N	3	7	N	N	7	3	N	N	7	3	N	N

C. Feasible Retiming

As the experimental results shown in [12], the retiming steps in its optimal retiming algorithm are much fewer than that of [11]. This is also the reason that the former is more efficient than the latter on most cases. The fewer retiming steps, the fewer CP computations.

Retiming algorithm in [12] can achieve this is because the retiming function at each loop is computed directly from the results of its IP computation and is exactly equivalent to the retiming function of its equivalent HSDFG at each loop in the retiming algorithm in [5]. By this property, the feasible retiming procedure sdfFEAS in [12] mimics the feasible retiming procedure FEAS in [5], which is sufficient and necessary for feasible retiming of HSDFGs. This results in that sdfFEAS in [12] is also sufficient and necessary for feasible retiming of SDFGs, while other feasible retiming methods [9], [10] are only sufficient but not necessary. Procedure sdfFEAS further reduces the retiming steps by improving the termination condition of FEAS. Below we show that, similar to sdfFEAS, the proposed feasible retiming algorithm for the unfolded SDFGs is also sufficient and necessary, and has sharp termination condition.

For the retiming function computation, our CP computation uSdfCP(G, f, ∞) has not such a good property as the IP computation in [12]. We have to add some extra operations in the procedure to achieve such effect.

Lines 5 to 8, as well as parameter dcp, in Algorithm 1 are not necessary for CP computation itself. They compute all $criT(u, i)$s that are larger than dcp, so that the increment of retiming function at each retiming step ($\triangle r(u)$) can be calculated according to $criT$. By Eqn (4), $\triangle r(u)$ is the largest label of those (u, j)s that have $criT(u, j) > dcp$. That is,

$$\triangle r(u) = j, criT(u, j) \leq dcp \text{ and } \forall i \in [0, j-1], criT(u, i) > dcp. \quad (5)$$

Let dcp for G_2 be six, the reverse paths to B_1 and C_1 are longer than six, as the $criT$ of procedure uSdfCP($G_2, 2, 6$) shown in the rightmost part of Table II. In this case, $\triangle r(A) = 0$ and $\triangle r(B) = \triangle r(C) = 1$.

The feasible retiming algorithm is outlined in Algorithm 2.

Algorithm 2 uSdfFEAS(G, f, dcp)

Require: A valid SDFG $G = \langle V, E\rangle$ and a nonnegative integer dcp
Ensure: A retiming r of G such that $U_f HR_r(G).cp \leq dcp$, if such a retiming exists
1: **if** $dcp < \max_{v\in V} t(v)$ **then**
2: **return false**
3: **end if**
4: $\forall v \in V$, let $r(v) = 0$
5: $G' = G$
6: **repeat**
7: get $criT$ and CP of G' from uSdfCP
8: **if** $CP \leq dcp$ **then**
9: **return** r and CP
10: **end if**
11: **for all** $v \in V$ **do**
12: $r(v) = r(v) + \triangle r(v)$ {by Eqn. (5)}
13: **end for**
14: $G' = R_r(G)$
15: **until** find a repetitive delay distribution of G_r
16: **return false**

Lines 1 to 3 guarantee that the desired CP is larger than the largest computation time of actors. Line 4 initializes the retiming vector r. At each step of the loop (Lines 6 to 15) of Algorithm 2, the retiming function r is increased by $\triangle r$, which is calculated via Eqn. (5). The length of a reverse f-ZR walk to firing (u, j) is always not less than that of (u, i) when $i > j$, so $\triangle r(u)$ is the number of all firings of u to which the lengths of reverse f-ZR walks are larger than dcp. Therefore, a step of Algorithm 2 simulates exactly a step of Algorithm FEAS in [5], which is sufficient and necessary for HSDFGs. A *delay distribution* of a SDFG is a vector containing delays on all edges of the SDFG. Since a retiming only changes the delay distribution, when a repetitive one is checked, the whole procedure can be finished safely (Line 15). This termination condition is the same as that of [12] – no potential feasible retiming functions are missed and no redundant retiming functions are checked. Consequently, Algorithm 2 is sufficient and necessary and with sharp termination condition.

When it has to repeat $f|V'|$ times, the worst case occurs. Therefore the complexity of Algorithm 2 is $O(f^2|V'||E'|)$. The worst cases usually occurs when checking an infeasible dcp, for which no feasible retimings exists. For the cases when a feasible retiming exists, Algorithm 2 converges rapidly and needs much fewer retiming steps. Our experimental results in Section V also confirm this observation.

D. Optimal Retiming

An optimal retiming is a procedure that repeatedly reduces the CP constraint and checks its feasibility until a feasible retiming leading to the smallest possible CP (therefore the optimal IP) is found. We outline optimal retiming algorithm in Algorithm 3.

Algorithm 3 finds the optimal retiming by repeatedly calling uSdfFEAS to check whether a dcp is feasible. As explained in [12], a binary search may check more unfeasible cases, which are bottlenecks of feasible retiming problem as we explain before. Hence we choose $optCP - 1$ as dcp, where $optCP$ is the CP of the previously retimed graph (Line 7). The CP is initialized at Line 3 and updated at each feasible retiming check (Line 8). The times of repeating never exceed the CP of f- graph of G. In fact, the lower bound of the dcp is $\max_{v\in V} t(v)$. Therefore the complexity of Algorithm 3 is $O(CP \cdot f^2|V'||E'|)$ or $O(IP \cdot f^3|V'||E'|)$. Since only one infeasible dcp is checked in Algorithm 3, its average complexity is much better than the worst case.

Algorithm 3 uSdfOPT(G, f)

Require: A valid SDFG $G = \langle V, E\rangle$
Ensure: A retiming r of G such that $U_f HR_r(G).cp$ is as small as possible and the optimal IP
1: $\forall v \in V$, let $r(v) = 0, r'(v) = 0$
2: $G' = G$
3: get CP of G' from uSdfCP
4: **repeat**
5: $G' = R_{r'}(G')$
6: $r+ = r'$
7: $optCP = CP$
8: run uSdfFEAS($G', f, optCP - 1$) to determine if a feasible retiming r' exists; if r' exists, get its CP
9: **until** no feasible retiming found for G'
10: **return** r and $\frac{optCP}{f}$

V. Experimental Evaluation

A. Experimental Setup

We have implemented the proposed method (Alg. 2 and Alg. 3) and the extensions of optimal retiming algorithm in [11] and both feasible and optimal retiming algorithms in [12] in the open source tool SDF3 [15]. The algorithm in [11] deals with only strongly connected SDFGs. Others have no such limitation. The tested SDFGs are generated with SDF3. They are divided into four sets according to their structures and each set is further divided into groups according to the size of the graphs. We test three unfolding factors ($f = 1, 2, 3$) for each graph. The experiments were carried out on a 2.67GHz CPU with 12MB cache. The experimental results are shown in Tables III to VIII. All execution times are measured in milliseconds (ms) unless otherwise specified.

The SDFGs are generated according to four types of structure: acyclic graphs (Acyc), connected (but not strongly connected) graphs (Con), strongly connected graphs (SC) and strongly connected graphs with self-loops (SCsl). A graph in SCsl is generated by adding a self-loop on each actor of a graph in SC. The last set is used to model graphs with complex structure. Graphs with different structure are grouped into different set. There are four groups in each set: $A20Q100$, $A50Q100$, $A50Q1000$ and $A100Q1000$, where $AxQy$ means there are x actors in an SDFG and y actors in its equivalent HSDFG, and $f \cdot y$ actors in its f-unfolded graphs. Each group includes 30 SDFGs.

B. Experimental Results

All the three methods considered are exact, and they have no significant difference on the use of the memory because they all work directly on the original SDFGs. Therefore, we compare only the execution times of them. The general improvements of our feasible retiming algorithm (Alg. 2) and optimal retiming algorithm (Alg. 3) are shown in Tables III and IV, respectively. Each point in the tables is an average of execution times of the algorithms on 360 unfolded graphs in each set. The improvement is measured by the execution time of the compared method divided by that of our method.

In Table III, Alg. 2 is compared with the extension of feasible retiming algorithm in [12]. The optimal CP (optCP) is used as feasible dcp and optCP minus one as infeasible dcp. On the feasible tests, Alg. 2 is over ten percent better than that in [12]. The significant improvement, however, is on the infeasible tests, especially on those graphs in set SCsl, on which Alg. 2 outperforms the algorithm in [12] more than 500 times faster. We explain later why the infeasible case for set Acyc and the feasible case for set SCsl are not shown here.

TABLE III
PERFORMANCE IMPROVEMENT OF FEASIBLE RETIMING (MS)

Type	[12]	Alg. 2	imp	Type	[12]	Alg. 2	imp
Acyc-feas	0.99	0.91	109%	SC-feas	0.60	0.44	137%
Con-feas	0.52	0.36	143%	SC-infeas	4.64	3.17	147%
Con-infeas	46.54	24.93	187%	SCsl-infeas	266s	512.36	51879%

In Table IV, Alg. 3 is compared with the extensions of optimal retiming algorithms in [11] (imp1) and [12] (imp2). The algorithm in [11] is only tested on sets SC and SCsl, which are strongly connected. Alg. 3 is more than twice as fast as the algorithm in [11]. Not surprisingly, Alg. 3 outstandingly improves the algorithm in [12] on the SCsl set, since the feasible retiming used by it has the worst execution time when deals with infeasible dcp.

TABLE IV
PERFORMANCE IMPROVEMENT OF OPTIMAL RETIMING (MS)

Type	[12]	Alg. 3	imp	Type	[11]	[12]	Alg. 3	imp1[a]	imp2[b]
Acyc.	9.8	6.5	150%	SC	10.0	7.1	3.9	259%	183%
Con	51.3	26.3	195%	SCsl	1s	267s	515.2	203%	51769%

[a] the improvement to [11] .
[b] the improvement to [12].

The detailed experimental results are shown in Tables V, VI, VII and VIII. Each table shows results for one set of graphs. Each point in the tables is an average of the results of 30 graphs in the same group. The first part of each table shows IPs and optimal IPs (optIPs) of f-unfolded graphs, and the last part shows the execution time and retiming steps of different optimal retiming algorithms. The results for feasible retiming algorithms are shown between them. We only show the results of cases when $f = 1$ and 2 because of space limitation.

The results of acyclic graphs are shown in Table V. The optimal CP of an acyclic graph is the maximal execution time of its actors. In the infeasible tests, since dcp is set to be $optCP - 1$, Alg. 2 always returns at Line 2 and do nothing. Therefore, the results for infeasible cases of feasible retiming algorithms are not shown here. Tables VI and VII shows the results of graphs in sets Con and SC, respectively. Both feasible cases and infeasible cases of feasible retiming algorithms are included. The results of strongly connected graphs with self-loops are shown in Table V. With a self-loop on each actor, graphs in set SCsl have very limited space for throughput optimization. For most graphs, the $optCP$ used as dcp for feasible tests is the initial CP. This causes Alg. 2 returns at Line 9 at the first loop and do nothing more. Therefore, the results for feasible cases of feasible retiming algorithms are not shown here.

TABLE V
RESULTS OF ACYCLIC GRAPHS

	f	A20Q100	A50Q100	A50Q1000	A100Q1000
IP/optIP					
	1	58.0/6.5	88.4/6.8	99.7/6.8	129.1/7.0
	2	29.0/3.3	44.2/3.4	49.9/3.4	64.6/3.5
Feasible retiming for dcp=optCP (dcp is feasible)					
Execution time (ms) / Retiming steps					
[12]	1	0.3/10.9	0.7/16.8	0.9/19.1	2.1/24.7
	2	0.3/10.9	0.7/16.8	0.8/19.1	2.2/24.7
Alg. 2	1	0.2/10.9	0.5/16.8	0.6/19.1	1.8/24.7
	2	0.3/10.9	0.6/16.8	0.7/19.1	1.9/24.7
Optimal retiming					
Execution time (ms) / Retiming steps					
[12]	1	1.3/60.2	6.7/137.1	8.3/161.4	20.5/256.9
	2	1.4/60.2	8.0/137.1	9.3/161.4	21.0/256.9
Alg. 3	1	0.9/54.8	3.4/122.5	4.1/142.9	10.8/215.7
	2	1.5/76.8	5.3/165.1	5.8/189.8	15.4/282.6

TABLE VI
RESULTS OF CONNECTED GRAPHS

	f	A20Q100	A50Q100	A50Q1000	A100Q1000
IP/optIP					
	1	63.2/47.5	107.0/80.4	128.8/77.7	182.7/106.9
	2	55.1/47.4	93.5/80.4	102.8/77.6	143.0/106.9
Feasible retiming for dcp=optCP (dcp is feasible)					
Execution time (ms) / Retiming steps					
[12]	1	0.1/1.2	0.3/1.8	0.3/2.2	0.8/2.5
	2	0.2/1.0	0.5/1.6	0.3/1.4	0.9/1.4
Alg. 2	1	0.1/1.2	0.3/2.2	0.3/2.0	0.8/2.4
	2	0.0/1.0	0.2/1.8	0.4/1.4	0.7/1.4
Feasible retiming for dcp =optCP-1 (dcp is infeasible)					
Execution time (ms) / Retiming steps					
[12]	1	1.3/65.3	3.0/72.1	24.1/641.7	60.4/714.7
	2	1.6/65.2	5.3/72.0	43.6/641.6	116.9/715.2
Alg. 2	1	1.3/73.6	1.9/70.3	25.1/742.7	42.1/689.2
	2	1.6/73.9	2.7/70.2	34.5/742.6	61.5/689.3
Optimal retiming					
Execution time (ms) / Retiming steps					
[12]	1	1.8/70.0	4.9/82.7	27.7/660.9	66.4/751.5
	2	2.7/69.7	9.3/82.0	48.9/659.4	125.3/747.9
Alg. 3	1	1.5/80.2	2.7/81.6	26.4/762.4	43.8/720.4
	2	2.2/80.2	3.9/80.4	35.9/760.9	64.1/718.6

For the feasible retiming algorithms, no matter on which sets, they are much slower in the infeasible cases. This also

TABLE VII
RESULTS OF STRONGLY CONNECTED GRAPHS

	f	A20Q100	A50Q100	A50Q1000	A100Q1000
IP/optIP					
	1	62.2/45.4	104.9/86.7	126.9/86.3	185.5/118.5
	2	53.4/45.3	96.0/86.7	104.9/86.2	150.8/118.5
Feasible retiming for dcp=optCP (dcp is feasible)					
Execution time (ms) / Retiming steps					
[12]	1	0.1/1.8	0.3/1.8	0.4/3.1	0.8/2.1
	2	0.1/1.5	0.5/1.1	0.5/1.6	1.0/1.5
Alg. 2	1	0.1/1.5	0.2/1.7	0.2/3.5	1.0/2.6
	2	0.1/1.5	0.2/0.9	0.4/2.1	0.9/1.3
Feasible retiming for dcp =optCP-1 (dcp is infeasible)					
Execution time (ms) / Retiming steps					
[12]	1	0.2/13.9	1.4/28.2	1.6/30.6	5.0/50.1
	2	0.4/14.0	2.4/27.5	3.3/33.4	10.5/51.2
Alg. 2	1	0.4/18.1	1.0/35.5	2.0/43.7	5.6/76.8
	2	0.5/18.2	1.7/35.2	2.6/44.6	7.8/74.7
Optimal retiming					
Execution time (ms) / Retiming steps					
[11]	1	0.6/38.8	1.1/43.7	7.7/311.4	14.2/257.9
	2	0.9/48.7	2.4/59.4	12.5/325.9	22.1/264.7
[12]	1	0.4/19.8	1.6/35.6	2.3/47.5	7.8/80.3
	2	0.7/19.2	3.2/34.4	4.4/45.7	16.3/79.4
Alg. 3	1	0.1/24.3	1.4/42.5	2.3/57.7	6.7/99.4
	2	0.4/24.1	1.8/41.7	3.3/56.4	9.8/96.6

TABLE VIII
RESULTS OF STRONGLY CONNECTED GRAPHS WITH SELF-LOOPS

	f	A20Q100	A50Q100	A50Q1000	A100Q1000
IP/optIP					
	1	323.1/319.3	241.1/232	3951.9/3938.4	2981.7/2958.1
	2	321.2/319.3	236.5/232	3945.2/3938.4	2969.9/2958.1
Feasible retiming for dcp=optCP-1 (dcp is infeasible)					
Execution time (ms) / Retiming steps					
[12]	1	70/74.2	44/54.4	82,596/870.6	53,155/699.7
	2	388/74.2	234/53.4	525,977/870.6	310,456/699.7
Alg. 2	1	6/98.9	7/88.0	497/999.1	554/994.7
	2	11/98.9	12/86.6	986/999.1	1,056/994.7
Optimal retiming					
Execution time (ms) / Retiming steps					
[11]	1	5/76.3	5/62.9	505/898.4	477/774.2
	2	17/147.3	15/110.0	1,953/1750.7	1,659/1426.8
[12]	1	71/75.2	47/59.0	82,862/873.6	53,456/704.9
	2	391/75.2	243/56.4	527,358/873.6	312,246/704.9
Alg. 3	1	6/99.8	7/90.3	500/1003.5	558/1001.6
	2	11/99.8	12/88.8	991/1003.5	1,060/1001.6

confirms the observation in [16] that it is the bottleneck of a feasible retiming algorithm when a desired IP/CP is infeasible. The performance of the three methods varies from case by case, but in most cases, the proposed method outperforms the others. Algorithm in [11] often has a fastest procedure for CP computation, but it also has much more retiming steps than the other two in most cases. Algorithms in [12] is very slow for the fourth set because for those graphs its CP computation procedure has to repeatedly compute many shared subpaths.

Our experimental results also reveal that retiming usually has a better effects than unfolding on throughput optimization of SDFGs, and that the more complex the graphs are, the less room left for optimization, no matter retiming or unfolding.

VI. CONCLUSIONS

In this paper, we have presented efficient and exact method for finding the feasible retiming and the optimal retiming of an unfolded SDFG, without converting it to its equivalent HSDFG and therefore without further unfolding the HSDFG. The correctness of our method is guaranteed by the results about the equivalence of transformations of SDFGs [13]. We have also extended two state-of-the-art retiming methods for SDFGs [11], [12] to deal with unfolding. Our method has a better worst case complexity than them. The experimental results on graphs with various structures and sizes show that, in most cases, the proposed method outperforms the extensions of [11], [12]; for the graphs with extremely complex structure, it can be over 500 times faster. Our experimental results also reveal that, on throughput optimization of SDFGs, retiming usually has a better effects than unfolding, and the structure of the graphs has an impact.

REFERENCES

[1] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. on Comput*, vol. 36, no. 1, pp. 24–35, 1987.

[2] V. Zivojnovic, S. Ritz, and H. Meyr, "Optimizing DSP programs using the multirate retiming transformation," *Proc. EUSIPCO Signal Process. VII, Theories Applicat*, vol. 3, pp. 1579–1600, 1994.

[3] S. Ritz, M. Willems, and H. Meyr, "Scheduling for optimum data memory compaction in block diagram oriented software synthesis," in *Proc. of the 1995 Acoustics, Speech, and Signal Processing Conf.*, 1995, pp. 2651–2654.

[4] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: scheduling and synchronization*. CRC Press, 2009.

[5] C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[6] K. Parhi and D. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput on Computers*, vol. 40, no. 2, pp. 178–195, 1991.

[7] L. Chao and E. Sha, "Scheduling Data-Flow Graphs via Retiming and Unfolding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 12, pp. 1259–1267, 1997.

[8] V. Zivojnovic and R. Schoenen, "On retiming of multirate DSP algorithms," in *Proc. of the Acoustics, Speech, and Signal Processing*, 1996, pp. 3310–3313.

[9] T. O'Neil and E. Sha, "Retiming synchronous data-flow graphs to reduce execution time," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2397–2407, 2001.

[10] X.-Y. Zhu, "Retiming multi-rate DSP algorithms to meet real-time requirement," in *Proc. of the 13th Design, Automation and Test in Europe (DATE)*. IEEE, 2010, pp. 1785–1790.

[11] N. Liveris, C. Lin, J. Wang, H. Zhou, and P. Banerjee, "Retiming for Synchronous Data Flow Graphs," in *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*. IEEE Computer Society, 2007, pp. 480–485.

[12] X.-Y. Zhu, T. Basten, M. Geilen, and S. Stuijk, "Efficient retiming of multirate DSP algorithms," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 6, pp. 831–844, 2012.

[13] X.-Y. Zhu, "Equivalence of transformations of synchronous data flow graphs: Work-in-progress," in *Proc. of the International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES '18. Piscataway, NJ, USA: IEEE Press, 2018, pp. 11:1–11:2.

[14] L. Chao, "Scheduling and behavioural transformations for parallel systems," Ph.D. dissertation, Princeton University, 1993.

[15] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF For Free," in *Proc. of the 6th Int. Conf. on Application of Concurrency to System Design*. IEEE, 2006. http://www.es.ele.tue.nl/sdf3/, pp. 276–278.

[16] N. Shenoy and R. Rudell, "Efficient implementation of retiming," in *Proc. of the 1994 IEEE/ACM int. conf. on Computer-Aided Design*. IEEE, 1994, p. 233.