Retiming Multi-Rate DSP Algorithms to Meet Real-Time Requirement

Xue-Yang Zhu
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
China, Beijing 100190
Email: zxy@ios.ac.cn

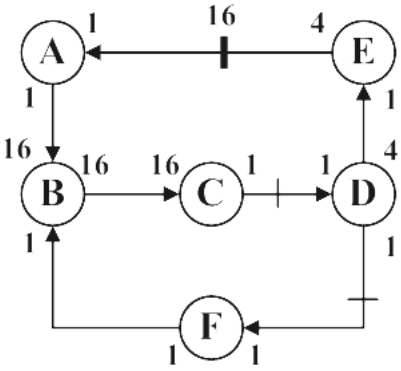
***Abstract*—Multi-rate digital signal processing(DSP) algorithms are usually modeled by synchronous dataflow graphs(SDFGs). Performing with high enough throughput is a key real-time requirement of a DSP algorithm. Therefore how to decrease the iteration period of an SDFG to meet the real-time requirement of the system under consideration is a very important problem. Retiming is a prominent graph transformation technique for performance optimizing. In this paper, by proving some useful properties about the relationship between an SDFG and its equivalent homogeneous SDFG(HSDFG), we present an efficient retiming algorithm, which needn't convert the SDFG to HSDFG, for finding a feasible retiming to reduce the iteration period of an SDFG as required.**

I. Introduction

The data driven property and the real-time requirement are two important features of digital signal processing(DSP) systems. The dataflow models of computation are widely used to represent architecture of DSP applications for their data driven nature. One of the most useful dataflow models used to design multi-rate DSP algorithms is the *synchronous dataflow graph* (SDFG)[1], also called *multi-rate dataflow graph*. Fig. 1, for example, is an SDFG modeling a simplified spectrum analyzer[2].

DSP algorithms are often iterative. Execution of all the computations as required times is referred to as an iteration. The *iteration period* is the time required for execution of one iteration of the algorithm[3]. The less the iteration period of an algorithm is, the higher its throughput is; and performing with high enough throughput is usually a key real-time requirement of a DSP algorithm. It is therefore an important problem to decrease the iteration period to meet the real-time requirement of DSP applications.

For dataflow graph the iteration period is limited by its topology and the computation time of nodes. Many graph transformation techniques have been used for solving this problem. Among them, *retiming* is notable for its simplicity and efficiency. Retiming is introduced originally applied to the *homogeneous synchronous dataflow graph* (HSDFG), which is a special case of the SDFG, to optimize application's performance by redistributing delays without changing its functionality[4].



Node	Time
A	1
B	1
C	2
D	1
E	1
F	2

Fig. 1. A simplified spectrum analyzer

Retiming has also been extended to SDFGs. Different from the HSDFG, however, in an iteration of an SDFG nodes may be executed in different times. In the SDFG in Fig.1, for example, node A running 16 times per iteration but node B once. This disables many useful results derived for HSDFGs, and complicates the analysis of retiming properties of SDFGs. Nevertheless, there have some efforts been made. Some important properties of retiming on SDFGs, such as functional equivalence and reachability, were proven in [5] and [2]. When retiming is used to reduce the iteration period, the method, traditionally, is to transform an SDFG to its equivalent HSDFG first and then retime the HSDFG. However, converting SDFG to HSDFG increase the problem space hugely. O'Neil et al. have proposed a method to reduce the iteration period by directly retiming on SDFGs[6], but the computation for iteration period is still on HSDFGs, which is space and time consuming. N. Liveris et al. have presented algorithms for finding an optimal retiming from the schedule point of view[7].

In this paper, we propose a novel method to check if there exists a retiming that can reduce the iteration period of a multi-rate DSP algorithm modeled by an SDFG to a desired value. We explore the relationship between an SDFG and its equivalent HSDFG, and conclude some properties, by which iteration period can be computed directly on SDFGs. And then an efficient retiming algorithm, without converting the SDFG

Supported by the National Natural Science Foundation of China under Grant Nos. 60721061, 60833001,60673051; the National High-Tech Research and Development Plan of China under Grant No.2009AA01Z148.

to HSDFG, is provided.

We first describe the definitions and properties of SDFGs in section II. Our main results are illustrated in section III and IV. The complexity of our algorithms are discussed in section V. Finally in section VI the conclusions are presented.

II. Preliminary and Notation

A. Synchronous Dataflow Graph

Definition 1. A *synchronous dataflow graph* (SDFG) is a finite directed multigraph $G = \langle V, E, t, d, prd, cns\rangle$, in which

- V is the set of nodes, modeling the functional elements of the system. Each node $v \in V$ is weighted with its computation time $t(v)$, a nonnegative integer;
- E is the directed edge set, modeling interconnections between functional elements. Each edge $e \in E$ is weighted with three properties: $d(e)$, gives the number of initial tokens associated with e, also called **delay**; $prd(e)$, a positive integer that represents the number of tokens produced onto e by each execution of the source node of e ; $cns(e)$, a positive integer that represents the number of tokens consumed from e by each execution of the sink node of e.

We represent the source node and sink node of $e \in E$ as $src(e)$ and $snk(e)$, respectively; the edge e with source node u and sink node v by $e = \langle u, v\rangle$; the set of incoming edges to $v \in V$ by $InE(v)$; and the set of outgoing edges from $v \in V$ by $OutE(v)$. We use $v \in G$ to represent that v is a node of G and use $e \in G$ to represent that e is a edge of G when their meanings are clear in the context. Given an SDFG $G = \langle V, E, t, d, prd, cns\rangle$, if $prd(e) = cns(e) = 1$ for each $e \in E$, then we say that G is a *homogeneous synchronous dataflow graph* (HSDFG), also called *single-rate dataflow graph*. We represent HSDFG as $G_H = \langle V, E, t, d\rangle$.

Applications for signal processing are usually nonterminating and therefore their memories must be bounded no matter how much times they are executed. In order that an SDFG G has well-defined meaning, we place restrictions on it:

VS1. $d(e) \geq 0$ for each $e \in G$.

VS2. G is live. *Liveness* means that there is no execution sequence leading to a deadlock.

VS3. G is bounded. *Boundedness* means that if there are infinite execution sequences then there exist some for which the number of tokens on every arc is finite.

We define a *valid SDFG* as an SDFG that satisfies conditions VS1, VS2, and VS3.

The boundedness of an SDFG can be checked directly on its topology, while its liveness depends on its boundedness and the distribution of delays associated with its edges. We review the sufficient and necessary condition of boundedness as the following property[1], and review that of liveness after the equivalent HSDFG of an SDFG is introduced.

Property 2. An SDFG $G = \langle V, E, t, d, prd, cns\rangle$ is bounded *if and only if* there is a positive integer vector $q(V)$ such that for each $e \in E$,

$$q(src(e)) \times prd(e) = q(snk(e)) \times cns(e). \tag{1}$$

Where (1) is called *balance equation*, and the smallest q is called *repetition vector*. We will refer q to represent repetition vector directly. To guarantee that the SDFG could be executed infinite times with bounded memory, each node v need to be executed $q(v)$ times in an iteration.[1]

Take the SDFG in Fig. 1 for example, its $prd(e)$, $cns(e)$ and $d(e)$ for each edge e are labeled by e; there are 16 delays on edge $\langle E, A\rangle$, one delay on edge $\langle C, D\rangle$ and $\langle D, F\rangle$, respectively; its computation time vector $t = [1, 1, 2, 1, 1, 2]^\top$. A balance equation can be constructed for each edge, e.g. $q(A) \times prd(\langle A, B\rangle) = q(B) \times cns(\langle A, B\rangle)$ for the edge $\langle A, B\rangle$. By solving these balance equations, we have its repetition vector $q = [16, 1, 1, 1, 4, 1]^\top$.

B. Equivalent HSDFG

A bounded SDFG can always be converted to an equivalent HSDFG, in which the data dependency remains identical to that in the original SDFG[8]. Algorithms for transforming an SDFG to its equivalent HSDFG appear in many literatures[9],[10]. For developing our idea clearly, we now formalize the transformation as a multi-valued map.

Definition 3. Let $G = \langle V, E, t, d, prd, cns\rangle$ be a bounded SDFG and q its repetition vector. Transformation H maps G to its equivalent HSDFG $H(G) = \langle V', E', t, d\rangle$ as follows:

3-1. $(\forall v \in V, i \in [1, q(v)] : (v, i) \in V', t(v, i) = t(v))$
$\wedge(\forall v' \in V' : \exists v \in V, i \in [1, q(v)] : v' = (v, i))$

3-2. $(\forall e = \langle u, v\rangle \in E, i \in [1, q(u)], k \in [1, prd(e)],$
$j = \lfloor \frac{((i-1)prd(e)+(k-1)+d(e)) \bmod cns(e)q(v)}{cns(e)} \rfloor + 1 :$
$e' = \langle (u, i), (v, j)\rangle \in E',$
$d(e') = \lfloor \frac{(i-1)prd(e) + (k-1) + d(e)}{cns(e)q(v)} \rfloor)$
$\wedge(\forall e' \in E' : \exists e = \langle u, v\rangle \in E, i \in [1, q(u)],$
$j \in [1, q(v)] : e' = \langle (u, i), (v, j)\rangle)$

Intuitively, each node v in G has $q(v)$ copies in $H(G)$; each edge e in G has $q(snk(e))cns(e)$ instances in $H(G)$; and edges in $H(G)$ preserve the data dependency among executions of nodes in G. There are some examples to illustrate the conversion from SDFGs to HSDFGs in Appendix 1 of [10].

We extend the delay count function d of HSDFG from single edges to arbitrary paths. For any path

$$p' = v'_0 \xrightarrow{e'_1} v'_1 \xrightarrow{e'_2} \ldots \xrightarrow{e'_n} v'_n$$

in the HSDFG, we define the *path delay* as the sum of the delays of the edges in the path:

$$d(p') = \sum_{i=1}^{n} d(e'_i).$$

If $d(p') = 0$ then we say p' is a *zero-delay path*. Similarly, we define the *path computation time* of p' as the sum of the computation time of the nodes in the path p':

$$t(p') = \sum_{i=0}^{n} t(v'_i).$$

To check if a bounded SDFG is live, two methods exist. One is to construct a single-processor schedule for one iteration of the SDFG; if such a schedule exists, then it is live[1]. Another is to check if a zero-delay cycle in its equivalent HSDFG exists, if not, then it is live[11]. We will use the later as a premise of our proofs in the upcoming sections.

Property 4. A bounded SDFG G is live *if and only if* there is no zero-delay cycle in $H(G)$.

By inserting precedence constraints between source and sink nodes with a finite number of delays, every SDFG can be converted to a strongly connected graph[4],[2]. We will therefore consider only strongly connected graphs in the proposed algorithms.

III. SDFG and its equivalent HSDFG

In this section, we explore the relationship between the SDFG and its equivalent HSDFG, and prove some important properties which will guarantee the correctness of algorithms in the next section.

Lemma 5. Let $G = \langle V, E, t, d, prd, cns\rangle$ be a bounded SDFG, $H(G) = \langle V', E', t, d\rangle$, and $e = \langle u, v\rangle \in E$. If $d(e) < cns(e)$ then $e' = \langle (u, 1), (v, 1)\rangle \in E'$ and $d(e') = 0$.

Proof: By Definition 3-2, let $i = 1$ and $k = 1$, since $d(e) < cns(e)$, we have

$$j = \lfloor \frac{d(e)}{cns(e)} \rfloor + 1 = 1.$$

Then $e' = \langle (u, 1), (v, 1)\rangle \in E'$ and

$$d(e') = \lfloor \frac{d(e)}{cns(e)q(v)} \rfloor = 0.$$

■

We use $u \twoheadrightarrow v$ to represent a path from vertex u to v. And an important definition is formulated as below.

Definition 6. Let G be a valid SDFG and the nodes $u, v \in G$. If there exists a zero-delay path $p' \in H(G) : (u, i) \twoheadrightarrow (v, j)$ for some $i \in [1, q(u)]$, $j \in [1, q(v)]$, then v is *zero-delay reachable* from u.

Theorem 7. Let G be a valid SDFG, and $u, v \in G$. Then v is zero-delay reachable from u *if and only if* there exists a path

$$p' = (v_0, l_0) \xrightarrow{e'_1} (v_1, l_1) \xrightarrow{e'_2} \cdots \xrightarrow{e'_n} (v_n, l_n)$$

in $H(G)$, for each $i \in [1, n]$:

$$\lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{cns(e_i)} \rfloor + 1 \le q(v_i),$$

where $v_0 = u, v_n = v$ and $e_i = \langle v_{i-1}, v_i\rangle \in G$.

Proof: For each $i \in [1, n]$,

$$\lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{cns(e_i)} \rfloor + 1 \le q(v_i)$$
$$\Leftrightarrow \frac{(l_{i-1} - 1)prd(e_i) + d(e_i) + 1}{cns(e_i)} \le q(v_i)$$
$$\Leftrightarrow (l_{i-1} - 1)prd(e_i) + d(e_i) + 1 \le q(v_i)cns(e_i)$$
$$\Leftrightarrow (l_{i-1} - 1)prd(e_i) + d(e_i) < q(v_i)cns(e_i)$$
$$\Leftrightarrow \lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{q(v_i)cns(e_i)} \rfloor = 0$$

By Definition 3, select $k = 1$ for each i, we have

$$l_i = \lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{cns(e_i)} \rfloor + 1 \le q(v_i), \text{ and}$$
$$d(e'_i) = \lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{q(v_i)cns(e_i)} \rfloor = 0$$

$\Leftrightarrow p'$ is a zero-delay path in $H(G)$

$\Leftrightarrow v$ is zero-delay reachable from u in G

■

This theorem indicates that for each i, only by selecting

$$l_i = \lfloor \frac{(l_{i-1} - 1)prd(e_i) + d(e_i)}{cns(e_i)} \rfloor + 1$$

and check if $l_i \le q(v_i)$, we can find a zero-delay reachable path in the SDFG.

Theorem 8. Let $G = \langle V, E, t, d, prd, cns\rangle$ be a valid SDFG, and $V_0 = \{v \in V : \forall e \in InE(v), d(e) \ge cns(e))\}$. Then we have

8-1. $V_0 \neq \emptyset$;

8-2. for each $v \in V$, there exists $u \in V_0$, such that v is zero-delay reachable from u.

Proof: **8-1.** (By Contradiction.) Assume to the contrary that $V_0 = \emptyset$. Then select $v_n \in V$, there exists $e_n \in InE(v_n)$, such that $d(e_n) < cns(e_n)$. Let $v_{n-1} = src(e_n)$, by Lemma 5, there exists $e'_n = \langle (v_{n-1}, 1), (v_n, 1)\rangle \in H(G), d(e'_n) = 0$. And so forth, we can find $v_{n-2}, v_{n-3}, \ldots$. Since G is finite and strongly connected, we can find eventually a node v_i, which is identical with some node v_j, $i + 1 \le j \le n$. This means that there is a zero-delay cycle in $H(G)$, therefore G is not live (by property 4), contradicting the condition that G is valid.

8-2. If $v \in V_0$ then obviously it holds. We are going to prove the case when $v \in V - V_0$ by contradiction. Suppose v is not zero-delay reachable from any $u \in V_0$. Since $v \notin V_0$, there exists $e \in InE(v)$, $d(e) < cns(e)$. Let $v' = src(e)$, then by Lemma 5, there exists $e' = \langle (v, 1), (v', 1)\rangle \in H(G)$, $d(e') = 0$; and by assumption, $v' \notin V_0$. Using the strategy like the proof of **8-1**, to trace backward, one zero-delay cycle in $H(G)$ will be found eventually. ■

IV. Retiming of SDFG

Retiming can be used for improving performance of a system in many aspects. In this paper, we focus on using it to reduce the iteration period of an application to meet the real-time requirement. In the improving process, the iteration period has to be computed many times to check if a retiming is satisfied. So, the complexity of computation of iteration period is a key factor to that of retiming algorithm.

A. Iteration Period

One *iteration* of a HSDFG G_H is an execution of each node in G_H exactly *once*. The *iteration period* $IP(G_H)$ is the max computation time of the zero-delay paths[12]:

$$IP(G_H) = \max_{p \in G_H} \{t(p) : d(p) = 0\}.$$

An SDFG allows each node to be executed more than once per iteration, and *two* nodes are not required to execute the same number of the times in an iteration. The iteration period of a valid SDFG G is defined on $H(G)$:

$$IP(G) = IP(H(G)).$$

We will show, however, armed with the results in previous section it can be computed directly on SDFGs. The iteration period of an SDFG is the max computation time of its zero-delay reachable paths. Theorems 7 and 8 are basis for the following algorithm, which is a variation of *depth first search* algorithm.

Algorithm IP(G)

Input: An valid SDFG $G = \langle V, E, t, d, prd, cns\rangle$ and q
Output: $T(V)$, $eofZD(V)$, ip

1: $V_0 = \{v \in V : \forall e \in InE(v), d(e) \geq cns(e))\}$;
2: **for all** $v \in V$ **do**
3: $T(v) = T_1(v) = t(v)$;
4: $eofZD(v) = false$;
5: **end for**
6: **for all** $v \in V_0$ **do**
7: $T_1(v) = t(v)$;
8: $getNextT(v, 1)$;
9: **end for**
10: $ip = \max_{v \in V} T(v)$;

Procedure getNextT(u, l)

11: $isEof = true$; $currT = T_1(u)$;
12: **for all** $e \in OutE(u)$ **do**
13: $l_1 = \lfloor \frac{(l-1)prd(e)+d(e)}{cns(e)} \rfloor + 1$;
14: $uNext = snk(e)$;
15: **if** $l_1 \leq q(uNext)$ **then**
16: $isEof = false$;
17: **if** $T(uNext) < T_1(u) + t(uNext)$ **then**
18: $T(uNext) = T_1(u) + t(uNext)$;
19: **end if**
20: $T_1(uNext) = T_1(u) + t(uNext)$;
21: $getNextT(uNext, l_1)$;
22: **end if**
23: $T_1(u) = currT$;
24: **end for**
25: **if** $isEof = true$; **then**
26: $eofZD(u) = true$;
27: **end if**

The vector $T(V)$ records the max computation time of zero-delay paths to nodes in G. If v is not zero-delay reachable from any other nodes in G, then $T(v) = t(v)$. The vector $eofZD(V)$ indicates the end of zero-delay reachable path to nodes in G. If zero-delay paths to v find their end at v, then $eofZD(v) = true$. The iteration period of G, ip, is the max value of $T(V)$.

About Algorithm IP, we may concern that: is V_0 empty? is it enough to search only from V_0 to get correct $T(V)$? is it terminable? The answers all are *yes*! By Theorem 7, the correctness of $T(V)$ is guaranteed; by Theorem 8, we know that V_0 is not empty and all the correct $T(V)$ can be found by searching from V_0; since the SDFG is live and finite, there is no zero-delay cycle in its equivalent HSDFG, then the value of l_1 will exceed its corresponding $q(uNext)$ in finite steps. Therefore the termination of the algorithm are ensured.

We now prove another property of this algorithm to prepare for illustrating the correctness of the next algorithm.

Theorem 9. Let G be a valid SDFG, $u \in G$. If $eofZD(u) = true$ when finish algorithm IP, then for each $e \in OutE(u)$, there has $d(e) \geq prd(e)$.

Proof: For each $e \in OutE(u)$, let $v = snk(e)$. By Algorithm IP, there exists $i \in [1, q(u)]$, and

$$j = \lfloor \frac{(i-1)prd(e) + d(e)}{cns(e)} \rfloor + 1,$$

such that $j > q(v)$.

$\Rightarrow j \geq q(v) + 1$
$\Rightarrow (i-1)prd(e) + d(e) \geq q(v)cns(e)$
$\Rightarrow q(u)prd(e) - prd(e) + d(e) \geq q(v)cns(e)$ (A)
$\Rightarrow d(e) \geq prd(e)$ (B)

(A)by $i \leq q(u)$; (B)by Property 2. ■

B. retiming

Definition 10. Given an SDFG $G = \langle V, E, t, d, prd, cns\rangle$, a *retiming* of G is a function $r : V \to \mathbb{Z}$, specifying a transformation r of G into a new SDFG $r(G) = \langle V, E, t, d_r, prd, cns\rangle$, where the delay-function d_r is defined for each edge $u \xrightarrow{e} v$ by the equation

$$d_r(e) = d(e) + cns(e)r(v) - prd(e)r(u). \quad (2)$$

Other than [2] and [6], we define retiming in a backward fashion, as [4] and [7].

A retiming r of a valid SDFG is meaningful only when it's *legal*, that is, $r(G)$ is a valid SDFG. It is interesting that it is enough by checking if $r(G)$ satisfies VS1 to ensure that a retiming is legal.

Theorem 11. Let G be a valid SDF, and let r be a retiming on the vertices of G such that $r(G)$ satisfies condition VS1, then r is a legal retiming.

Proof: Since VS1 is true, it remains only to show that VS2 and VS3 are satisfied by $r(G)$. They are guaranteed by Theorem 1 and 2 of [2], respectively. ■

Combining with this conclusion, theorem 9 will guarantee the retiming in our algorithm is legal.

Given a valid SDFG G and a nonnegative integer dip, a retiming r of G is a *feasible retiming* if $r(G)$ is valid and $IP(r(G)) \leq dip$.

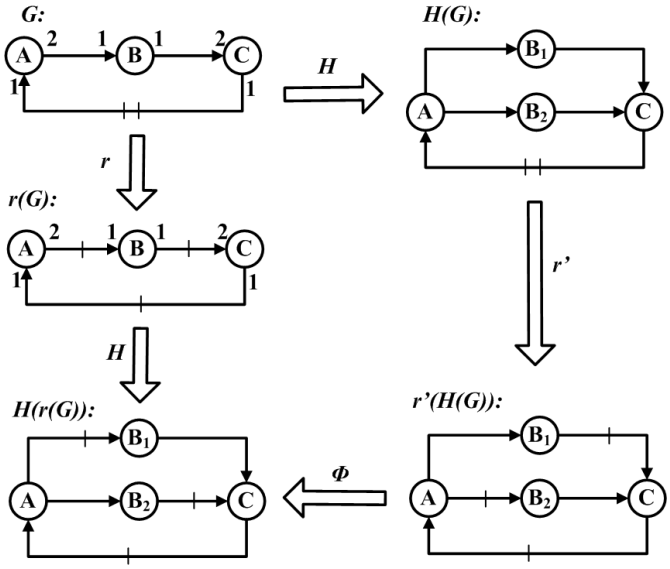


Fig. 2. The equivalence of $H(r(G))$ and $r'(H(G))$

About how to check if a feasible retiming of an SDFG can be found, many literatures have mentioned that first to transform it to its equivalent HSDFG and then to check on the HSDFG. But none of them have discussed this method in detail, as to [6] doubts its correctness and provides an example for disproval. In our opinion, before concluding if it is correct, there are two questions need to be clarified:

- when we have $r(v) = k$ in the SDFG G, how to define its equivalent r' in $H(G)$ since there may not be only one copy of v in $H(G)$?
- The equivalent HSDFG of retimed version of G, $H(r(G))$, may have different topology from $H(G)$; but retiming on $H(G)$ will never change its topology. Then may $H(r(G))$ and $r'(H(G))$ be equivalent?

The answer of the first question is obtained after making a closely examination on the definition of equivalent HSDFG, which must keep the semantics of the original SDFG. For each node v in G, if $r(v) = k$, then for each (v, i) in $H(G)$,

$$r'(v,i) = \lfloor \frac{k}{q(v)} \rfloor + \lfloor \frac{i + (k \bmod q(v)) - 1}{q(v)} \rfloor.$$

Then we can conclude that $H(r(G))$ and $r'(H(G))$ are equivalent under the isomorphic transformation $\Phi : r'(H(G)) \rightarrow H(r(G))$. For each (v, i) in $r'(H(G))$,

$$\Phi(v,i) = (v, (i + r(v) - 1) \mod q(v) + 1).$$

We also use the example in [6] to illustrate our observation by Fig. 2. Retiming r in G is: $r(A) = 0, r(B) = r(C) = 1$; its equivalent retiming r' in $H(G)$ is: $r'(A) = r'(B_1) = 0, r'(B_2) = r'(C) = 1$; and the isomorphic transformation Φ is: $\Phi(B_2) = B_1, \Phi(B_1) = B_2$.

We can see that this method is safe. It is not practical just for its complexity. Now our algorithm to find a feasible retiming directly on SDFGs is following.

Algorithm FIPtest(G, dip)

Input: An valid SDFG $G = \langle V, E, t, d, prd, cns \rangle$ and a desired iteration period dip

Output: A retiming r of G such that $r(G)$ is a valid SDFG with iteration period $IP(r(G)) \leq dip$, if such a retiming exists

1: **for all** $v \in V$ **do**
2: $r(v) = 0$;
3: **end for**
4: $G_r = G; i = 1; isTrue = false$;
5: **while** $i \leq \sum_{v \in V} q(v)$ and $isTrue = false$ **do**
6: $isTrue = true$;
7: get $T(V)$ and $eofZD(V)$ of G_r from IP();
8: **for all** $v \in V$ **do**
9: **if** $(T(v) > dip) \wedge eofZD(v)$ **then**
10: $isTrue = false$;
11: $r(v) = r(v) + 1$;
12: $rePrev(v)$;
13: **end if**
14: **end for**
15: **if** $isTrue = false$ **then**
16: $G_r = r(G); i++$;
17: **end if**
18: **end while**
19: **if** $isTrue = true$ **then**
20: **return** r;
21: **else**
22: **return** $NULL$;
23: **end if**

Procedure rePrev(v)

24: **for all** $e \in InE(v)$ **do**
25: $u = src(e)$;
26: **if** $(T(u) > dip) \wedge \neg eofZD(u) \wedge prd(e) \leq cns(e) + d(e)$ **then**
27: **if** $(\forall e_1 \in OutE(u) - \{e\}, d(e_1) \geq prd(e_1))$ **then**
28: $r(u) = r(u) + 1$;
29: $rePrev(u)$;
30: **end if**
31: **end if**
32: **end for**

Algorithm FIPtest works by relaxation. Each iteration of the while loop is equivalent to that of the algorithm FEAS in [4]. After each iteration of the while loop, the tentative retiming is guaranteed to be legal by Theorem 9 and 11. By Theorem 9, when $eofZD(v) = true$, since we increase $r(v)$ by only *one* at a time, then for each $e \in OutE(v)$, there has $d_r(e) = d(e) - prd(e) \geq 0$. For improving the efficiency, we check previous nodes of v by the procedure $rePrev(v)$, and increase their values of r if only the retiming is still legal.

We illustrate our algorithm by applying it to the example modeled by Fig. 1. We want to check if a desired iteration period, $dip = 3$, will be satisfied by any retiming. Three vectors r, T, and $eofZD$ go on as follows.

$$
\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \end{array}
\begin{pmatrix} 0 & 1 & f \\ 0 & 3 & f \\ \mathbf{0} & \mathbf{5} & \mathbf{t} \\ 0 & 1 & f \\ 0 & 2 & t \\ 0 & 2 & f \end{pmatrix}
\mapsto
\begin{pmatrix} 0 & 1 & f \\ 0 & 3 & t \\ 1 & 2 & f \\ 0 & 3 & f \\ \mathbf{0} & \mathbf{4} & \mathbf{t} \\ 0 & 2 & f \end{pmatrix}
\mapsto
\begin{pmatrix} 0 & 2 & f \\ 0 & 3 & t \\ 1 & 2 & f \\ 0 & 3 & f \\ \mathbf{1} & \mathbf{4} & \mathbf{t} \\ 0 & 2 & f \end{pmatrix}
$$

$$
\mapsto
\begin{pmatrix} 0 & 2 & f \\ 0 & 3 & t \\ 1 & 2 & f \\ 0 & 3 & f \\ \mathbf{2} & \mathbf{4} & \mathbf{t} \\ 0 & 2 & f \end{pmatrix}
\mapsto
\begin{pmatrix} 0 & 2 & f \\ 0 & 3 & t \\ 1 & 2 & f \\ 0 & 3 & f \\ \mathbf{3} & \mathbf{4} & \mathbf{t} \\ 0 & 2 & f \end{pmatrix}
\mapsto
\begin{pmatrix} 0 & 2 & f \\ 0 & 3 & t \\ 1 & 2 & f \\ 0 & 3 & t \\ 4 & 3 & f \\ 0 & 2 & f \end{pmatrix}
$$

Where the first, second and third column represents the value of vector r, T, and $eofZD$, respectively. It terminates once the feasible retiming, $r = [0, 0, 1, 0, 4, 0]^\top$, is found, running 6 times of iteration rather than $\sum_{v \in V} q(v) = 24$ times, as many practical algorithms like to be. Fig. 3 is the retimed version of Fig. 1.

V. Discussion

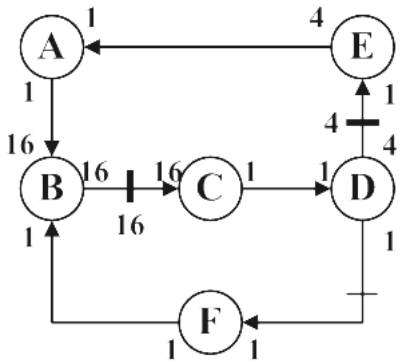
Complexity of algorithm FIPtest is not only affected by the number of nodes and edges of G, but the token rate functions prd and cns, on which q determines. In algorithm IP, we search zero-delay reachable paths in an SDFG G in some degree as that in $H(G)$, but we concern only the zero-delay edges. For each edge e in G, the number of zero-delay edge in $H(G)$ searched is

$$q(snk(e)) - \min\{\lfloor \frac{d(e)}{cns(e)} \rfloor, q(snk(e))\}.$$

Therefore, in the worst case, IP will take $\sum_{e \in E} q(snk(e))$ time, the same as that of [7]. In the worst case, FIPtest will take $\sum_{v \in V} q(v)$ iterations. Then the time consumed by FIPtest is

$$\sum_{e \in E} q(snk(e)) \times \sum_{v \in V} q(v).$$

While in [6], at each iteration, $H(G_r)$ is constructed and the iteration period of G_r is computed on $H(G_r)$. Their retiming algorithm takes $O(|V||V'| + |V||E'|)$ time to execution. Where V' and E' are vertex set and edge set of $H(G)$, respectively. $|V'| = \sum_{v \in V} q(v)$ and $|E'| = \sum_{e \in E} q(snk(e))cns(e)$. Especially when the token rates are quite different in an SDFGs, e.g. co-prime, our method is much more efficient. In some situations, such as, when every $d(e)$ is large enough as to each cycle in G correspond to a path in $H(G)$ without multiple appearances of the same node of G, IP only takes $|E|$ time to compute, no matter how many the value of q is. Although we have shown the efficiency of our algorithms by complexity analysis, we will perform more experiments to confirm it in the near future.



Node	Time
A	1
B	1
C	2
D	1
E	1
F	2

Fig. 3. Retimed version of Figure 1

VI. Conclusion

In this paper, we have examined closely the relationship between an SDFG and its equivalent HSDFG, and proved some useful properties, by which iteration period can be computed directly on SDFGs. Based on the efficient algorithm computing iteration period, a retiming algorithm for reducing iteration period of SDFGs has been provided. Because our method is completely working on SDFGs, it turns out to have much lower complexity than related works.

Acknowledgment

The author would like to thank the anonymous referees for their helpful comments and suggestions; thank Sander Stuijk for correcting a mistake in Fig.3.

References

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

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

[3] K. Parhi, *VLSI digital signal processing systems: design and implementation*. Wiley India Pvt. Ltd., 2007.

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

[5] V. Zivojnovic and R. Schoenen, "On retiming of multirate DSP algorithms," in *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference-Volume 06*. IEEE Computer Society, 1996, pp. 3310–3313.

[6] 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.

[7] 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.

[8] E. Lee, "A coupled hardware and software architecture for programmable digital signal processors (synchronous data flow)," 1986.

[9] S. Sriram and S. Bhattacharyya, *Embedded multiprocessors: scheduling and synchronization*. CRC, 2000.

[10] G. Sih, "Multiprocessor Scheduling to Account for Interprocessor Communication, Ph. D," Ph.D. dissertation, Thesis, 1999.

[11] R. Karp and R. Miller, "Properties of a model for parallel computations: Determinancy, termination, queueing," *SIAM Journal on Applied Mathematics*, pp. 1390–1411, 1966.

[12] 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.