Journal of VLSI Signal Processing, 11, 229–244 (1995)

Determining the Minimum Iteration Period of an Algorithm*

KAZUHITO ITO
Department of Electrical and Electronic System Engineering, Saitama University, Saitama 338, Japan
KESHAB K. PARHI
Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455

Received June 10, 1994; Revised April 24, 1995

Abstract. Digital signal processing algorithms are repetitive in nature. These algorithms are described by iterative data-flow graphs where nodes represent computations and edges represent communications. For all data-flow graphs, there exists a fundamental lower bound on the iteration period referred to as the *iteration bound.* Determining the iteration bound for signal processing algorithms described by iterative data-flow graphs is an important problem. In this paper we review two existing algorithms for determination of the iteration bound. Then we propose another novel method based on the *minimum cycle mean algorithm* to determine the iteration bound with a lower polynomial time complexity than the two existing techniques. It is convenient to represent many multi-rate signal processing algorithms by multi-rate data-flow graphs. The iteration bound of a multi-rate data-flow graph (MRDFG) can be determined by considering the single-rate data-flow graph (SRDFG) equivalent of the MRDFG. However, the equivalent single-rate data-flow graph contains many redundant nodes and edges. The iteration bound of the MRDFG can be determined faster if these redundancies in the equivalent SRDFG are first removed. A previous approach has considered elimination of edge redundancy. In this paper we present an approach to eliminate *node redundancy* in the MRDFG. We combine elimination of node and edge redundancies to propose a novel algorithm for faster determination of the iteration bound of the MRDFG.

1 Introduction

Digital signal processing algorithms are repetitive in nature. These algorithms are described by iterative data-flow graphs (DFGs) where nodes represent tasks and edges represent communication [1], [2]. Execution of all nodes of the DFG once completes an *iteration.* Successive iterations of any node are executed with a time displacement referred to as the *iteration period.* For all recursive signal processing algorithms, there exists an inherent fundamental lower bound on the iteration period referred to as the *iteration period bound* or simply the iteration bound [3]–[5]. This bound is fundamental to an algorithm and is independent of the implementation architecture. In other words, it is impossible to achieve an iteration period less than the bound even when infinite processors are available to execute the recursive algorithm.

Determination of the iteration bound of the data-flow graph is an important problem. First it discourages the designer to attempt to design an architecture with an iteration period less than the iteration bound. Second, the iteration bound needs to be determined in rate-optimal scheduling of iterative data-flow graphs. A schedule is said to be rate-optimal if the iteration period is same as the iteration bound, i.e., the schedule achieves the highest possible rate of operation of the algorithm.

The iteration bound determination may have to be performed repeatedly in the scheduling phase of high-level synthesis. In resource-constrained scheduling, a given processing algorithm is scheduled to achieve the minimum iteration period using the given hardware resources. In order to execute operations of the processing algorithm in parallel, the required number of processors or functional units required to execute the operations in parallel may be larger than the number of available resources. In that case, we must give the order of executions, or the precedence, to these operations to reduce the parallelism. Generally the precedence to be assigned is not unique. Hence the iteration bound should be determined for every possible precedence to check which precedence leads to the final schedule with the minimum iteration period. Consequently,

*This research was supported by the Advanced Research Projects Agency and monitored by Wright—Patterson AFB under contract number F33615-93-C-1309.

the iteration bound may have to be computed many times and hence it is important to determine the iteration bound in minimum possible time.

Two algorithms have been recently proposed to determine the iteration bound. A method based on the negative cycle detection was reported in [6] to determine the iteration bound with polynomial time complexity with respect to the number of nodes in the processing algorithm. Another method based on the first-order longest path matrix was proposed in [7] to determine the lower bound with polynomial time complexity with respect to the number of delays in the processing algorithm. In this paper, based on [8], we propose yet another method based on the *minimum cycle mean algorithm* [9] to determine the iteration bound with lower polynomial time complexity than in [6] and [7].

Several multi-rate signal processing algorithms can be conveniently represented by multi-rate data-flow graphs [2]. To balance production and consumption of data on communicating edges between computations, the nodes of the multi-rate data-flow graph need to be executed different number of times in any iteration of the MRDFG. Sufficient theory to determine how many times a node in the MRDFG needs to be executed in an iteration has been developed in [2]. All MRDFGs can be expanded to equivalent uniform or single-rate DFGs (SRDFGs). Thus, the iteration bound of the MRDFG can be determined by considering the equivalent SRDFG. However, the equivalent SRDFG contains many redundancies with respect to nodes and edges. The iteration bound of the MRDFG can be determined faster if these redundancies are first eliminated. An approach to determine the iteration bound by eliminating the edge degeneracy has been proposed in [10]. In this paper we propose an approach to eliminate node redundancies. Then we present an algorithm to determine the iteration bound in the MRDFGs in a faster manner by eliminating both edge and node redundancies.

This paper is organized as follows. In Section 2, the iteration bound is defined. The two existing algorithms to determine the iteration bound are reviewed in Section 3. Section 4 presents our new algorithm to determine the iteration bound based on the minimum cycle-mean algorithm. The multi-rate DFG model is reviewed in Section 5 and an algorithm to determine the iteration bound in MRDFGs is introduced in Section 6. The execution times of existing and proposed algorithms to determine the iteration bound are compared in Section 7 for a few benchmark algorithms.

2 Data-Flow Graph and the Iteration Bound

A data-flow graph (DFG) is denoted as $G = (N, E, q, d)$ where N is the set of nodes, E is the set of directed edges, q is the set of execution times of the nodes, and d is the set of delay counts associated with the edges. A *directed path* in a DFG contains a series of connected edges. If the start node and the end node of a directed path are identical and no node except the start node appears more than once in the path, the directed path is called a *cycle* (also often referred to as a *loop*). Let C denote the set of all cycles of the DFG and N_c be the set of nodes which belong to the cycle c. The computation time of a cycle is defined as the sum of computation times of the nodes in the cycle. The delay count of a cycle is defined as the sum of delay counts of the edges of the cycle. Let M_c and D_c denote the computation time and the delay count of cycle c, respectively. These are calculated as

$$M_c = \sum_{i \in N_c} q(i), \tag{1}$$

$$D_c = \sum_{e \in c} d(e). \tag{2}$$

Let T denote the iteration period of the DFG. The DFG receives new data samples every T units of time and the computation of each node is executed repetitively and periodically every T units of time. Smaller iteration period implies faster processing. However, there exists a lower bound on the iteration period. It depends on the computation time of nodes which belong to cycles and the data dependencies among them. For each cycle c, M_c must be smaller than or equal to $D_c T$ since all the computations of the nodes in c must be completed within $D_c T$ units of time. Therefore, there exists a lower bound on the iteration period T due to cycle c. It is called the *cycle bound* and the cycle bound T_c is calculated as

$$T_c = \frac{M_c}{D_c}. \tag{3}$$

The lower bound on the iteration period is the maximum of the cycle bounds. This is referred to as the *iteration bound*, T_i, and is calculated as

$$T_i = \max_{c \in C} T_c = \max_{c \in C} \frac{M_c}{D_c}. \tag{4}$$

A *critical cycle* is defined as the cycle c whose cycle bound equals the iteration bound.

We assume that the DFG is a strongly connected graph. A strongly connected graph is a graph such

that there exists a directed path between any pair of nodes. If the DFG is not strongly connected, we can derive strongly connected components with the computational complexity $\mathcal{O}(|N| + |E|)$ [11], where $|N|$ and $|E|$ are the number of nodes and the number of edges, respectively, and determine the iteration bound for each component. Then, the maximum of these iteration bounds is the iteration bound of the DFG.

3 Previous Work on Iteration Bound Determination

In this section, two previously proposed methods of determination of the iteration bound for a given DFG are reviewed.

3.1 *The Negative Cycle Detection Method*

An algorithm to determine the iteration bound based on the negative cycle detection method [12] has been recently proposed in [6], [10]. A negative cycle is a cycle where the sum of weights of edges on the cycle is negative.

Given the DFG, $G = (N, E, q, d)$, and the guess value of the iteration bound T_0, an edge-weighted digraph $\tilde{G} = (N, E, w)$ is constructed. The node set and the edge set of $\tilde{G}$ are exactly the same as N and E, respectively, and the weight $w(e)$ of edge $e = (u, v)(\in E)$ is given by $w(e) = d(e)T_0 - q(u)$. A shortest path algorithm can be executed on this graph to check presence of any negative cycle. We can use the well-known shortest path algorithm such as Floyd method which requires the time complexity $\mathcal{O}(|N|^3)$ or Bellman-Ford method which requires the time complexity $\mathcal{O}(|N||E|)$ in the case when $|E|$ is smaller than $|N|^2$.

If the shortest path algorithm terminates without finding a solution, then there exists a negative cycle c where

$$\sum_{e \in c} w(e) = D_c T_0 - M_c < 0. \tag{5}$$

It means that the guess iteration bound T_0 is smaller than the cycle bound $T_c = M_c/D_c$ and therefore than the true iteration bound T_i. On the other hand, if the shortest path algorithm terminates with a solution, then T_0 is greater than or equal to T_i.

Thus, we increase the guess iteration period T_0 if a negative cycle is detected or decrease it otherwise and re-evaluate the weight of edges of $\tilde{G}$ and repeat the shortest path algorithm. By using a binary-search technique to determine the amount by which T_0 is increased or decreased, the above procedure terminates with the result $T_0 = T_i$ after $\mathcal{O}(\log |N|)$ repetitions [12].

Therefore, the overall time complexity to determine the iteration bound is $\mathcal{O}(|N|^3 \log |N|)$ [6] or $\mathcal{O}(|N||E| \log |N|)$. (However, it is claimed in [10] that $\mathcal{O}(|E|) = \mathcal{O}(|N|)$ in ordinary DFGs and therefore the time complexity is $\mathcal{O}(|N|^2 \log |N|)$.) The memory requirement is $\mathcal{O}(|N|^2)$ when Floyd method is used, or $\mathcal{O}(|N| + |E|)$ when Bellman-Ford method is used as the shortest path algorithm.

3.2 *The Longest Path Matrix Method*

Another technique to determine the iteration bound of the given DFG by using the *first-order longest path matrix* was proposed in [7].

From the DFG $G = (N, E, q, d)$, an edge-weighted digraph $G_d = (D, E_d, w)$ is constructed. Figure 1 illustrates an original DFG and the constructed graph corresponding to the DFG.

D is the set of nodes which is a delay in the original graph G. Therefore, a delay in the graph G corresponds to a node in the graph G_d. If a directed path in G consists of edges with no delay except the first and last edges, an edge spanning a pair of nodes in D corresponding to these two delays is included in the edge set E_d. There is such a directed path $(k, h), (h, i), (i, l)$ in Fig. 1(a) and an edge (δ, β) is included in E_d, since δ and β are delays on the first edge (k, h) and the last edge (i, l), respectively.

We define *path length* in this paper as the sum of computation time of the intermediate nodes on the path. The length of the path stated above is the sum of computation time of the nodes h and i. The weight of the edge $e \in E_d$, $w(e)$, is equal to the length of the path corresponding to the edge e. In Fig. 1(a), there is a path from the delay labeled as α to itself, with the intermediate nodes j and h. The weight $w(\alpha, \alpha)$ is the sum of computation times of j and h, i.e., 3. In this case, the edge forms a self cycle as shown in Fig. 1(b). In general, multiple paths may exist between two delays. In that case, the weight of the edge is set to the longest path length among these paths. For example, between delays labeled as α and δ in Fig. 1(a), there are two directed paths which go through nodes j, m, k and j, l, m, k. The lengths of these paths are 5 and 6, respectively. Thus, the weight $w(\alpha, \delta)$ is set to 6.

If more than one delay elements exist on an edge in G, then more than one node is included in the node

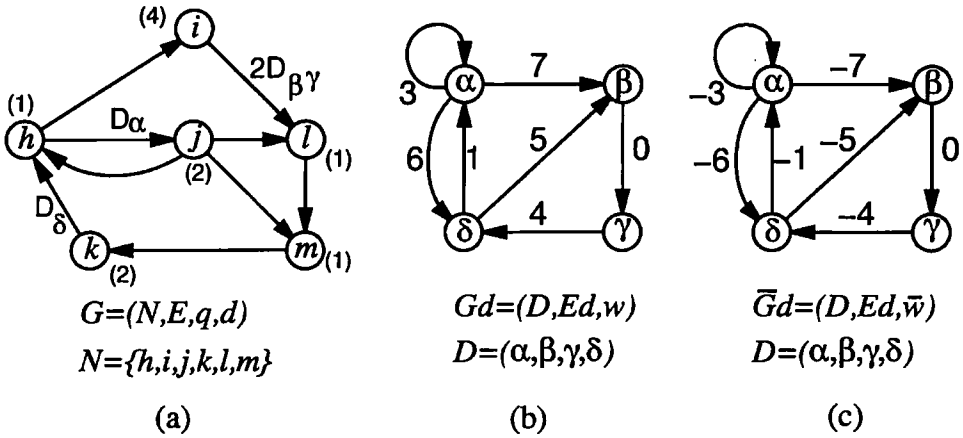

Fig. 1. The DFG G and the corresponding edge-weighted digraph G_d. In parenthesis in G are the computation times of nodes.

set D. The weight of the edge between the nodes corresponding to these delays is 0 since there exists no computation node between these delays. In Fig. 1, there are two delays on the edge (i, j) in G and these correspond to the two nodes β and γ in G_d. The edge (β, γ) is included in E_d and the weight $w(\beta, \gamma)$ is 0.

The first-order longest path matrix L^1 is a $|D| \times |D|$ matrix and its element l_{ij}^1 is equal to the weight $w(i, j)$ of the edge (i, j) of G_d or $-\infty$ if such an edge does not exist. The element l_{ij}^{k+1} of the longest path matrix of the order $k(> 0)$ is recursively defined as

$$l_{ij}^{k+1} = \max_{n \in D} \{l_{in}^1 + l_{nj}^k\}. \tag{6}$$

The diagonal element l_{ii}^k of L^k is the longest computation time among cycles which contains exactly k nodes including the node i. Since a node in G_d represents a delay in G, the maximum of l_{ii}^k for all i corresponds to the longest cycle in G containing exactly k delays. Therefore,

$$T = \max_{1 \le k \le |D|} \frac{\max_i l_{ii}^k}{k} \tag{7}$$

is the largest cycle bound of all the cycles and is the iteration bound of the DFG, G.

From the DFG $G = (N, E, q, d)$, the weighted digraph $G_d = (D, E_d, w)$ and L^1 are constructed in $\mathcal{O}(|D||E|)$ computational time. The time complexities to compute $\max_i l_{ii}^k$ and L^{k+1} from L^k are $\mathcal{O}(|D|)$ and $\mathcal{O}(|D|^3)$, respectively. These computations are repeated $|D|$ times for $1 \le k \le |D|$. Hence, the total computational complexity is $\mathcal{O}((|D| + |D|^3)|D| + |D||E|) = \mathcal{O}(|D|^4 + |D||E|)$ [7].

If we construct an edge-weighted digraph $\tilde{G}_d = (D, E_d, \tilde{w})$ where the weight of an edge $e \in E_d$ is given by $\tilde{w}(e) = T_0 - w(e)$ where T_0 is a guess iteration bound, then the technique mentioned in the previous section can be used to determine the iteration bound. From the DFG $G = (N, E, q, d)$, constructing $G_d = (D, E_d, w)$ requires the computation time of $\mathcal{O}(|D||E|)$ complexity. The computation time to determine the iteration bound of G_d by using $\tilde{G}_d$ is the complexity of $\mathcal{O}(|D|^3 \log |D|)$. Hence, the total time complexity is $\mathcal{O}(|D|^3 \log |D| + |D||E|)$ [7]. The memory requirement is $\mathcal{O}(|N| + |D|^2)$ for calculating L^1 and storing L^k.

The methods described in this section would be faster than the method based on the negative cycle detection where the number of delays $|D|$ is relatively smaller than the number of nodes $|N|$ and the number of edges $|E|$.

4 A New Algorithm to Determine the Iteration Bound

In this section, we propose an algorithm to determine the iteration bound by using the minimum cycle mean algorithm.

The *cycle mean* of a cycle c, $m(c)$, is defined as

$$m(c) = \frac{\sum_{e \in c} w(e)}{p_c} \tag{8}$$

where $w(e)$ is the weight of the edge e and p_c is the number of edges in cycle c. In other words, the cycle mean of a cycle c is the average weight of the edges included in c.

The *minimum cycle mean problem* involves the determination of the minimum cycle mean, λ, of all the cycles in the given digraph where

$$\lambda = \min_{c} m(c). \tag{9}$$

An efficient algorithm was proposed in [9] to determine the minimum cycle mean for a given graph with time complexity $\mathcal{O}(|N||E|)$, where N and E are the set of nodes and the set of edges of the graph, respectively.

The number of nodes in a cycle is equal to the number of edges of the cycle. According to the definition of the graph $G_d = (D, E_d, w)$, each node in G_d corresponds to a delay in the DFG, G, and the edge weight $w(d_1, d_2)$ of the edge $(d_1, d_2) \in E_d$ is the largest path length among all the paths from the delay d_1 to the delay d_2. Therefore, the cycle mean of the cycle c_d, containing k nodes, $d_1, d_2, \ldots, d_k$, is the maximum cycle bound of the cycles of G, which contain the delays labeled $d_1, d_2, \ldots, d_k$. For example, in the graph shown in Fig. 2(a), there are two delays labeled α and β, respectively. There exist two cycles $\{(l, k), (k, i), (i, l)\}$ and $\{(l, k), (k, j), (j, i), (i, l)\}$, both of which go through delays α and β. Their cycle bounds are $4/2 = 2$ and $6/2 = 3$, respectively, and the maximum of them is 3. Figure 2(b) shows the graph $G_d = (D, E_d, w)$ corresponding to the graph shown in Fig. 2(a). In Fig. 2(b), $D = \{\alpha, \beta\}$, $w(\alpha, \beta) = 1$, and $w(\beta, \alpha) = 5$. There exists one cycle $\{(\alpha, \beta), (\beta, \alpha)\}$ and its cycle mean is 3. It equals the maximum cycle bound of the cycles in the graph shown in Fig. 2(a), which contain the delays α and β.

Since the cycle mean of a cycle c in the graph G_d equals the maximum cycle bound of the cycles in G which contain the delays in cycle c, the maximum cycle mean of the graph G_d equals the maximum cycle bound of all the cycles in the graph G. Therefore, the iteration bound of the graph G can be obtained as the maximum cycle mean of the graph G_d.

Let C_d denote the set of cycles in graph G_d. Then, the maximum cycle mean of the graph G_d is

$$\begin{aligned}\max_{c \in C_d} m(c) &= \max_{c \in C_d} \frac{\sum_{e \in c} w(e)}{p_c} \\ &= \max_{c \in C_d} \frac{-\sum_{e \in c}(-w(e))}{p_c} \\ &= -\min_{c \in C_d} \frac{\sum_{e \in c}(-w(e))}{p_c}.\end{aligned} \tag{10}$$

It is the negative of the minimum cycle mean of the graph $\bar{G}_d = (D, E_d, \bar{w})$, where $\bar{w}(e) = -w(e)$ for every edge $e \in E_d$. Consequently, the maximum cycle mean of the graph G_d, i.e., the iteration bound of the graph G, can be obtained as the negative of the minimum cycle mean of the graph $\bar{G}_d$.

The algorithm to determine the iteration bound of the given graph by means of the minimum cycle mean is summarized in Fig. 3.

From the DFG $G = (N, E, q, d)$, constructing $G_d = (D, E_d, w)$ and $\bar{G}_d = (D, E_d, \bar{w})$ requires the computation time of $\mathcal{O}(|D||E|)$ complexity. The time complexity to calculate the minimum cycle mean for the graph $\bar{G}_d = (D, E_d, \bar{w})$ is $\mathcal{O}(|D||E_d|)$. Hence, the total time complexity to determine the iteration bound is $\mathcal{O}(|D||E_d| + |D||E|)$. This time complexity is better than the $\mathcal{O}(|D|^3 \log |D| + |D||E|)$ complexity of the method described in Section 3.2 since $|E_d| \leq |D|^2$ and therefore $|E_d| \leq |D|^2 \log |D|$ always holds. The memory requirement for calculating the edge weight w and determining the minimum cycle mean for the graph G_d are $\mathcal{O}(|N|)$ and $\mathcal{O}(|D|^2)$, respectively. The total memory requirement is $\mathcal{O}(|N| + |D|^2)$.

EXAMPLE. From the given DFG G illustrated in Fig. 1(a), the edge-weighted digraph G_d and $\bar{G}_d$ are constructed as shown in Fig. 1(b) and (c), respectively. If we choose α as s used in the minimum cycle mean algorithm, $F_k(v)$, the minimum weight of paths consisting of exactly k edges in E_d, and $\max_{0 \leq k \leq |D|-1} \frac{F_{|D|}(v) - F_k(v)}{|D| - k}$ are calculated as follows:

		k						
	$F_k(v)$	0	1	2	3	4	$\max_{0 \leq k \leq 3} \frac{F_4(v) - F_k(v)}{4 - k}$
	α	0	−3	−7	−10	−14	−3.5
v	β	∞	−7	−11	−14	−18	−3.5
	γ	∞	∞	−7	−11	−14	−3
	δ	∞	−6	−9	−13	−16	−3

(11)

Then, $\lambda = \min_{v \in D} \max_{0 \leq k \leq |D|-1} \frac{F_{|D|}(v) - F_k(v)}{|D| - k} = -3.5$ and the iteration bound of the DFG G is 3.5.

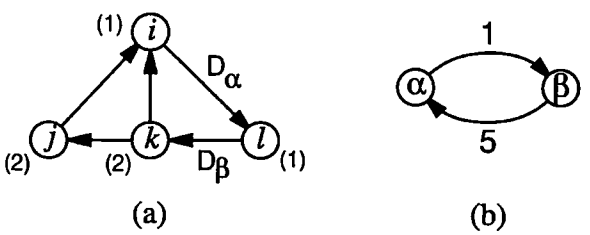

Fig. 2. The cycle mean and the cycle bound.

Input: DFG $G = (N, E, q, d)$.
Output: The iteration bound T_i.
1. Construct the graph $\bar{G}_d = (D, E_d, \bar{w})$ from the given DFG $G = (N, E, q, d)$.
2. Run the minimum cycle mean algorithm on $\bar{G}_d$.
   Minimum cycle mean algorithm
   2.0 Choose one node $s \in D$ arbitrarily.
   2.1 Calculate the minimum weight $F_k(v)$ of an edge progression of length k from s to v as
   $$F_k(v) = \min_{(u,v)\in E_d} \{F_{k-1}(u) + \bar{w}(u, v)\} \quad \text{for } k = 1, 2, \ldots, |D|$$
   with the initial conditions $F_0(s) = 0$; $F_0(v) = \infty$, $v \neq s$.
   2.2 Calculate the minimum cycle mean λ of $\bar{G}_d$.
   $$\lambda = \min_{v\in D} \max_{0\le k\le |D|-1} \frac{F_{|D|}(v) - F_k(v)}{|D| - k}$$
3. Now, $T_i = -\lambda$ is the iteration bound of the DFG G.

Fig. 3. The algorithm to determine the iteration bound.

The reader may confirm that the critical cycle is $\{(h, j), (j, l), (l, m), (m, k), (k, h)\}$ and its cycle bound, that is the iteration bound of the DFG, is 3.5 since the sum of computation times of nodes h, j, l, m, k is 7, the critical cycle contains 2 delays labeled as α and δ, and $7/2 = 3.5$.

5 Multi-Rate Data-Flow Graph

While every node in a single rate data-flow graph (SRDFG) is invoked once in an iteration of the execution, nodes in a multi-rate data-flow graph (MRDFG) are invoked more than once in an iteration. Moreover, different nodes may be invoked for a different number of times in an iteration. In other words, one node is invoked at a different rate from another node in MRDFGs. The definition of the edge in MRDFGs also differs from that in SRDFGs.

An MRDFG can be expanded into the equivalent SRDFG [1]. The 'equivalence' means that the MRDFG and its expanded SRDFG express identical signal processing algorithm. In this section we describe a method to expand an MRDFG into its equivalent SRDFG which is similar to unfolding an SRDFG [5].

5.1 The Number of Invocations of Node

In SRDFGs, it is assumed that one invocation of a node consumes one data from every incoming edge and produces one data on every outgoing edge. Therefore, the number of invocations of each node is one in an iteration. In MRDFGs, on the other hand, one invocation of a node can consume one or more data from each incoming edge and produce one or more data onto each outgoing edge. Let O_{uv} and I_{uv} denote the number of data produced onto the edge (u, v) by an invocation of the node u and the number of data consumed from the edge (u, v) by an invocation of the node v, respectively. They are written near the head and tail of the edge of the MRDFG as shown in Fig. 4(a). For example, 4 data samples are produced onto the edge (a, b) in every invocation of the node a and 3 data samples are consumed from the edge (a, b) in every invocation of the node b. In the case where O_{uv} and I_{uv} are different for the directed edge (u, v), the node u must be invoked different number of times than the node v so that the total number of data produced by the invocations of u and the total number of data consumed by the invocations of v balance within each iteration. Let k_u denote the number of invocations of the node u in an iteration. Then, k_u and k_v must satisfy that $O_{uv} * k_u = I_{uv} * k_v$. These relations must be satisfied for every edge in the MRDFG. For example, in the MRDFG shown in Fig. 4(a), there are four edges. The number of invocations k_a, k_b, and k_c must satisfy the following indeterminate equations:

$$4k_a = 3k_b \tag{12}$$
$$k_b = 2k_c \tag{13}$$
$$3k_c = 2k_a \tag{14}$$
$$k_c = k_c. \tag{15}$$

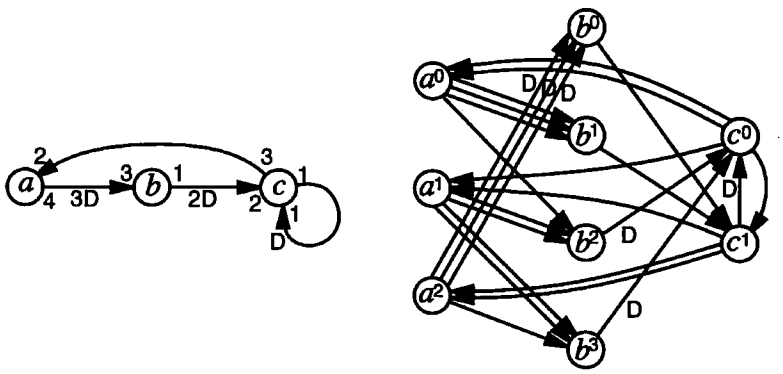

Fig. 4. An MRDFG and its equivalent SRDFG.

By solving the set of equations with a constraint that at least two of the numbers of invocations are coprime to each other, the minimum balanced numbers of invocations are 3, 4, and 2 for the nodes a, b, and c, respectively.

5.2 *Edges in the Equivalent Single-Rate DFG*

The equivalent SRDFG contains k_u nodes which are the k_u copies of node u of the MRDFG. Since every node is executed once in an iteration of the SRDFG, the total number of executions of the copies of node u in an iteration of the equivalent SRDFG is the same as the number of executions of node u in an iteration of the MRDFG.

Edges are also copied in the equivalent SRDFG. Each invocation of the node u produces O_{uv} data on the edge (u, v) in the MRDFG. It can be emulated in the SRDFG by appending O_{uv} copies of the edge (u, v) outgoing from each copy of the node u. Since the execution of a copy of the node u produces one data on every outgoing edge, the number of data produced by the execution is O_{uv}. Similarly, I_{uv} copies of the edge (u, v) are incoming to each copy of the node v and total number of data consumed by the execution of a copy of the node v is I_{uv}. There are $k_v I_{uv} (= k_u O_{uv})$ copies of the edge (u, v).

To simplify notation, in the remainder of this paper, let $x \backslash y$ and $x \% y$ denote $\lfloor \frac{x}{y} \rfloor$ and $x - \lfloor \frac{x}{y} \rfloor y$, respectively, where $\lfloor z \rfloor$ is the largest integer less than or equal to z and $x \% y$ is x modulo y.

Let u^k $(k = 0, 1, \ldots, k_u - 1)$ denote the node in the equivalent SRDFG, which corresponds to the kth invocation of the node u of the MRDFG. Since O_{uv} edges are outgoing from the node u^k, u^k can be regarded as having O_{uv} output terminals for the edge (u, v). Similarly, the node v^k can be regarded as having I_{uv} input terminals for the edge (u, v). There are $O_{uv}k_u$ output terminals on k_u nodes $u^0, u^1, \ldots, u^{k_u - 1}$ and $I_{uv}k_v (= O_{uv}k_u)$ input terminals on k_v nodes $v^0, v^1, \ldots, v^{k_v - 1}$. Then, the ith $(i = 0, 1, \ldots, O_{uv}k_u - 1)$ copy of the edge connects the ith output terminal of the copies of the node u (it is the $(i \% O_{uv})$th terminal of the node $u^{i \backslash O_{uv}}$) and the ith input terminal of the copies of the node v (it is the $(i \% I_{uv})$th terminal of the node $v^{i \backslash I_{uv}}$) if there is no delay on the edge (u, v).

The delay on an edge in an SRDFG implies the inter-iteration data dependency, where the computation of the current iteration depends on the data generated in the previous iteration. On the other hand, in the MRDFG, the delay on an edge does not imply the inter-iteration data dependency. It is the offset between the output terminal and the input terminal connected by a copy of the edge in the equivalent SRDFG. If the number of delays on the edge (u, v) is W in the MRDFG, the ith copy of the edge connects the ith output terminal of the copies of u with the $(i + W)$th input terminal of the copies of v. Therefore, the ith copy of the edge (u, v) is outgoing from the node $u^{i \backslash O_{uv}}$ and incoming to the node $v^{(i+W) \backslash I_{uv}}$. In the case that $(i + W) \backslash I_{uv} \geq k_v$, the edge is incoming to the node $v^{((i+W) \backslash I_{uv}) \% k_v}$ and $((i+W) \backslash I_{uv}) \backslash k_v = (i+W) \backslash (I_{uv} k_v)$ delays are associated with the edge. These delays now imply inter-iteration data dependencies since they exist in the SRDFG.

EXAMPLE. Figure 4 shows an MRDFG and its equivalent SRDFG. The numbers of invocations of node a,

b, and c are 3, 4, and 2, respectively. There are 12 copies of the edge (a, b) since $O_{ab} = 4$ and $k_a = 3$. The edge (a, b) is associated with 3 delays. Therefore, the 9th, 10th, and 11th copies of the edge, all incoming to b^0, are associated with one delay.

5.3 *Constructing the Equivalent SRDFG*

The algorithm to construct the equivalent SRDFG for a given MRDFG is summarized in Fig. 5. In Fig. 5, N_m and E_m are the set of nodes and the set of edges, respectively, of the MRDFG. $q_m(u)$ is the computation time of node $u \in N_m$ and $d_m(u, v)$ is the number of delays associated with the edge $(u, v) \in E_m$. Similarly, N_s and E_s are the set of nodes and the set of edges, respectively, of the equivalent SRDFG. $q_s(u)$ is the computation time of node $u \in N_s$ and $d_s(u, v)$ is the number of delays associated with the edge $(u, v) \in E_s$.

Construction of the equivalent SRDFG requires the computations of $\mathcal{O}(|N_m|k)$ complexity for generating N_s and $\mathcal{O}(|E_m|k)$ complexity for generating E_s where k is the average of k_u for all $u \in N_m$. The total computational complexity to construct the equivalent SRDFG is $\mathcal{O}((|N_m| + |E_m|)k)$.

Input: MRDFG $G_m = (N_m, E_m, q_m, d_m)$
Output: SRDFG $G_s = (N_s, E_s, q_s, d_s)$
$N_s = \emptyset$, $E_s = \emptyset$;
Calculate k_u for all $u \in N_m$;
for all $u \in N_m$ **do**
  for $k = 0$ to $k_u - 1$ **do**
    Include u^k into N_s;
    $q_s(u^k) \leftarrow q_m(u)$
  enddo
enddo
for all $(u, v) \in E_m$ **do**
  $W \leftarrow d_m(u, v)$.
  for $i = 0$ to $O_{uv}k_u - 1$ **do**
    Include $(u^{i \backslash O_{uv}}, v^{((i+W) \backslash I_{uv}) \% k_v})$ into E_s;
    $d_s(u^{i \backslash O_{uv}}, v^{((i+W) \backslash I_{uv}) \% k_v}) \leftarrow (i + W) \backslash (I_{uv}k_v)$
  enddo
enddo

Fig. 5. An algorithm to construct the equivalent SRDFG.

6 The Iteration Bound of Multi-Rate DFG

The iteration bound of an MRDFG is identical to the iteration bound of the equivalent SRDFG since these express the identical signal processing algorithm. Hence, the iteration bound of the MRDFG can be determined by determining the iteration bound of the equivalent SRDFG. From the viewpoint of determining the iteration bound, however, the equivalent SRDFG is redundant in the number of edges and the number of nodes in some cases. This kind of redundancy immediately worsens the computation time to determine the iteration bound since it depends on the number of nodes and the number of edges. The computation time to determine the iteration bound of the equivalent SRDFG may be improved by eliminating these redundancies. In this section, the technique to eliminate the edge redundancy of the equivalent SRDFG is discussed. Then, we propose a technique to eliminate the node redundancy of the equivalent SRDFG.

6.1 *Edge Degeneration*

One of the redundancy is the existence of parallel edges in the equivalent SRDFG. If we remove all the parallel edges except one edge with the least delays, a simple DFG is obtained. Let this edge removal procedure be called *edge degeneration*. Although the edge degeneration alters the signal processing algorithm, the iteration bound is not changed before and after the edge degeneration. Figure 6(a) and (b) show a SRDFG equivalent of the MRDFG in Fig. 4(a) and the edge-degenerated SRDFG, respectively. For example, there are three edges from the node a^2 to the node b^0 in the equivalent SRDFG. These edges are degenerated into a single edge from a^2 to b^0 as shown in Fig. 6(b).

O_{uv} edges outgoing from the node u^k are input to the $(O_{uv}u^k + W)$th to $(O_{uk}(u^k + 1) - 1 + W)$th terminals where W is the number of delays associated with the edge (u, v). The edges incoming to the same node v^j are degenerated into a single edge. Therefore, there exist the degenerated edges outgoing from u^k and incoming to $v^{j \% k_v}$, $j = ((O_{uv}u^k + W) \backslash I_{uv}), ((O_{uv}u^k + W) \backslash I_{uv}) + 1, \ldots, (O_{uv}(u^k + 1) - 1 + W) \backslash I_{uv}$, with $j \backslash k_v$ delays for each j. The algorithm to construct the edge-degenerated SRDFG directly from the given MRDFG is summarized in Fig. 7. The time complexity of the algorithm is $\mathcal{O}((|N_m| + |E_m|)k)$ where k is the average of k_u for all $u \in N_m$. The edge degeneration technique was also reported in [10][1].

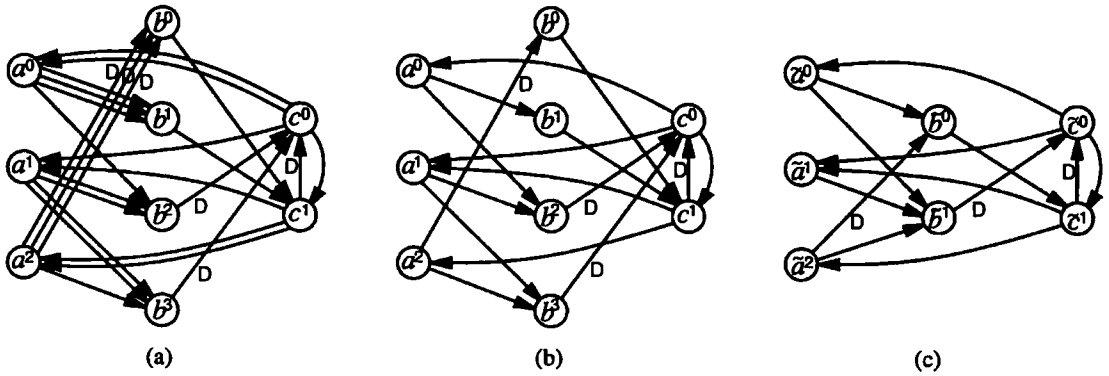

Fig. 6. Degeneration of the equivalent SRDFG. (a) The equivalent SRDFG. (b) The edge-degenerated equivalent SRDFG. (c) The node-degenerated equivalent SRDFG.

Input: MRDFG $G_m = (N_m, E_m, q_m, d_m)$
Output: the edge-degenerated SRDFG $G_s = (N_s, E_s, q_s, d_s)$
$N_s = \emptyset$, $E_s = \emptyset$;
Calculate k_u for all $u \in N_m$;
for all $u \in N_m$ **do**
  for $k = 0$ to $k_u - 1$ **do**
    Include u^k into N_s.
    $q_s(u^k) \leftarrow q_m(u)$.
  enddo
enddo
for all $(u, v) \in E_m$ **do**
  $W \leftarrow d_m(u, v)$.
  for $k = 0$ to $k_u - 1$ **do**
    for $j = (O_{uv}k + W) \backslash I_{uv}$ to $(O_{uv}(k+1) - 1 + W) \backslash I_{uv}$ **do**
      Include $(u^k, v^{j\%k_v})$ into E_s;
      $d_s(u^k, v^{j\%k_v}) \leftarrow j \backslash k_v$
    enddo
  enddo
enddo

Fig. 7. An algorithm to construct the edge-degenerated equivalent SRDFG.

6.2 *Node Degeneration*

Another redundancy in the equivalent SRDFG is the existence of nodes which could be degenerated into one node without altering the iteration bound. For example in edge-degenerated SRDFG illustrated in Fig. 6(b), the edges outgoing from the nodes b^0 and b^1 are incoming to an identical node c^1 and these edges are the only edges outgoing from b^0 and b^1. In this case, as shown in Fig. 6(c), we can merge b^0 and b^1 into a single node $\tilde{b}^0$ with the computation time identical to b^0 (also identical to b^1). Such a node $\tilde{b}^0$ is called the *degenerated node*. The edges connected to either b^0 or b^1 in the original SRDFG are connected to the degenerated node $\tilde{b}^0$ without changing the number of delays on these edges. Furthermore, newly introduced parallel edges in the SRDFG are also degenerated. This procedure is called *node degeneration*.

The *node degeneration* is carried out such that the iteration bound of the node-degenerated SRDFG is identical to that of the original SRDFG. The node degeneration is proper if the iteration bound is unchanged before and after the node degeneration. The node degeneration is proper in the case that either the *convergence condition* or the divergence condition is satisfied. These conditions are defined as follows:

- Convergence condition (Fig. 8(a))
  - — The nodes to be degenerated into one degenerated node are the copies of a node of the MRDFG.
  - — The nodes have outgoing edges which are incoming to an identical node and are the only edges outgoing from these nodes.
  - — The number of delays on the outgoing edges are the same.
- Divergence condition (Fig. 9(a))
  - — The nodes to be degenerated into one degenerated node are the copies of a node of the MRDFG.

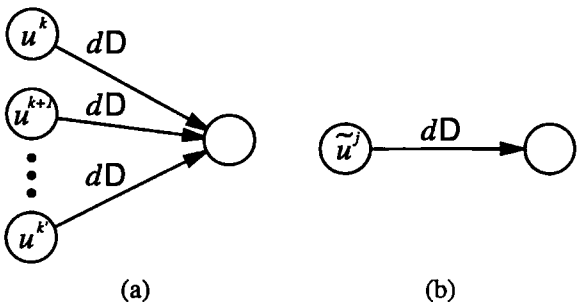

Fig. 8. The node degeneration with the convergence condition. (a) Convergence condition for node u^k. (b) The node degeneration for node u^k.

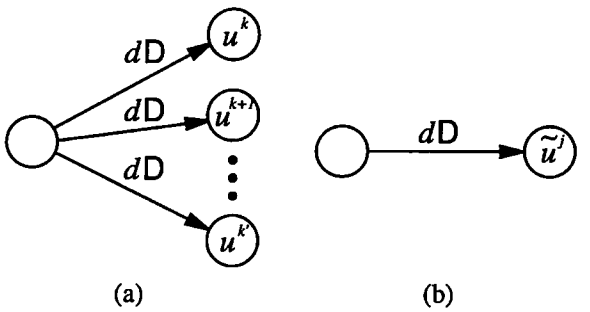

Fig. 9. The node degeneration with the divergence condition. (a) Divergence condition for node u^k. (b) The node degeneration for node u^k.

— The nodes have incoming edges which are outgoing from an identical node and are the only edges incoming to these nodes.

— The number of delays on the incoming edges are the same.

By the node degeneration with the convergence condition, the graph shown in Fig. 8(a) is degenerated as illustrated in Fig. 8(b). Similarly, by the node degeneration with the divergence condition, the graph shown in Fig. 9(a) is degenerated as illustrated in Fig. 9(b). For example in the SRDFG shown in Fig. 6(b), the nodes b^0 and b^1 are the copies of the node b of the MRDFG shown in Fig. 4(a), and all of their outgoing edges are incoming to the identical node c^1. Therefore, the convergence condition is satisfied and degenerating b^0 and b^1 into a degenerated node $\tilde{b}^0$ is a proper node degeneration. The legitimacy of the proper node degeneration is ensured by the following theorem.

THEOREM. *The iteration bound of the properly node-degenerated SRDFG is the same as the original SRDFG.*

PROOF. Suppose a set of nodes is degenerated into a single node. By the condition of the proper node degeneration, all the outgoing edges from these nodes must be incoming to an identical node, or all the incoming edges to these nodes must be outgoing from an identical node. We prove the theorem for the first case only. However, the theorem can be proven for the second case similarly.

Without loss of generality, we consider that two nodes u^0 and u^1 are degenerated into a single degenerated node $\tilde{u}^0$ as illustrated in Fig. 10. Note that no cycle contains the same node more than once. If a path contains both u^0 and u^1 as intermediate nodes, it must contain v twice. Such a path can never be a cycle. Therefore, there exists no cycle which contains both u^0 and u^1 in the original SRDFG and merging u^0 and u^1 into a single node $\tilde{u}^0$ does not eliminate any cycle.

All the edges either incoming to or outgoing from u^0 or u^1 in the original SRDFG are substituted by the edges either incoming to or outgoing from $\tilde{u}^0$ in the degenerated SRDFG. Therefore, the degenerated SRDFG has the same set of cycles as the original SRDFG except that either the edges (x, u^0) and (u^0, v) or the edges (x', u^1) and (u^1, v) for any node x and x' of a cycle are substituted by the edges $(x, \tilde{u}^0)$ and $(\tilde{u}^0, v)$ or

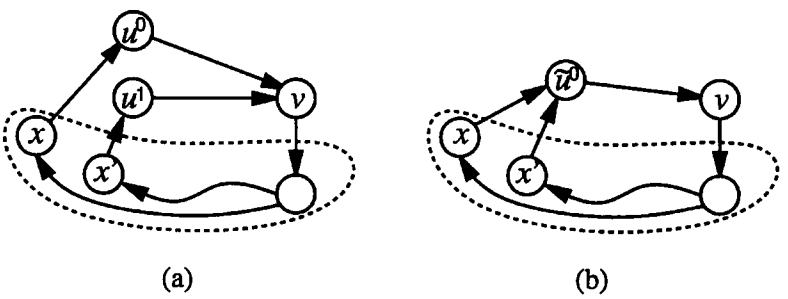

Fig. 10. Node degeneration. (a) A part of the original SRDFG. (b) The reduced SRDFG with u^0 and u^1 merged into $\tilde{u}^0$.

$(x', \tilde{u}^0)$ and $(\tilde{u}^0, v)$, respectively. Since $\tilde{u}^0$ has the same computation time as u^0 (and as u^1) and the number of delays on $(\tilde{u}^0, v)$ is the same as (u^0, v) and (u^1, v), the maximum cycle bound of the degenerated SRDFG is identical to that of the original SRDFG. □

6.3 *Algorithm of Node Degeneration*

By the node degeneration, the set of nodes u^k $(k = 0, 1, \ldots, k_u - 1)$ is degenerated into one or more nodes. Let $\tilde{u}^j$ $(j = 0, 1, \ldots)$ denote such degenerated nodes. The node degeneration is represented by the *node degeneration function*, $f_u(k) = j$, which indicates that the node u^k is degenerated into the node $\tilde{u}^j$.

Suppose that node u^{k_1} is degenerated into the node $\tilde{u}^{j_1}$ and node $u^{k_2}(k_2 > k_1)$ is degenerated into the node $\tilde{u}^{j_2}(j_2 > j_1)$ by the node-degeneration procedure mentioned above. Then, node $u^{k_3}(k_3 > k_2)$ is degenerated either into the node $\tilde{u}^{j_2}$ or into another node $\tilde{u}^{j_3}(j_3 > j_2)$ by the node-degeneration procedure. It is important to note that the node u^{k_3} is never degenerated into the node $\tilde{u}^{j_1}$ if the node u^{k_2} is not degenerated into $\tilde{u}^{j_1}$. Consequently, the node degeneration function is monotonically increasing function, i.e., $f_u(k) \leq f_u(k+1)$ for any node u and $0 \leq k \leq k_u - 1$.

Deriving the node degeneration functions which lead to a properly node-degenerated SRDFG with the least number of nodes is the main task in the node degeneration.

Determining Node Degeneration Function. The node degeneration function for the case of the convergence condition is derived as follows. Deriving the node degeneration function for the case of the divergence condition can be described in a similar way.

Suppose that the edge (u, v) with W delays is the only outgoing edge from the node u in the MRDFG. Furthermore, the nodes u and v are invoked k_u times and k_v times, respectively. Hence, $O_{uv}k_u = I_{uv}k_v$. Assume that $(O_{uv}k^0 + W)\backslash I_{uv} = (O_{uv}(k^0 + 1) - 1 + W)\backslash I_{uv}$ for a node u^{k^0}. In other words, all the edges outgoing from u^{k^0} are incoming to a single node $v^{((O_{uv}k^0+W)\backslash I_{uv})\%k_v}$. Let $j = (O_{uv}k^0 + W)\backslash I_{uv}$. Thus, all the edges outgoing from u^{k^0} are incoming to the node $v^{j\%k_v}$.

If $(O_{uv}(k+1) - 1 + W)\backslash I_{uv} = j$ for a $k(\geq k^0 + 1)$, that is, all the edges outgoing from the node u^k are also incoming to the node $v^{j\%k_v}$, then the nodes u^{k^0} and u^k can be degenerated into the same node. Hence, $f_u(k)$ is set as $f_u(k^0)$ to indicate that the nodes u^{k^0} and u^k are degenerated into an identical degenerated node $\tilde{u}^{f_u(k^0)}$.

On the other hand, if $(O_{uv}(k+1) - 1 + W)\backslash I_{uv} \neq j$, then u^k has an outgoing edge incoming to the node $v^{j'}$ other then v^j. In this case, $f_u(k)$ is set as $f_u(k^0) + 1$ to indicate that the nodes u^k cannot be degenerated into the same degenerated node as u^{k^0}.

More generally, the set of nodes v^k may have been degenerated. In that case, it is checked whether outgoing edges are incoming to an identical degenerated node. First, k^0 is chosen as the smallest number to satisfy $f_v((O_{uv}k^0 + W)\backslash I_{uv}) = f_v((O_{uv}(k^0 + 1) - 1 + W)\backslash I_{uv})$, i.e., all the outgoing edges from u^{k^0} are incoming to the degenerated node $\tilde{v}^{f_v(((O_{uv}k_u+W)\backslash I_{uv})\%k_v)}$. Let $f_v(((O_{uv}k_u + W)\backslash I_{uv})\%k_v) = j$. If $f_v((O_{uv}(k + 1) - 1 + W)\backslash I_{uv} = j$ for a $k \geq k^0 + 1$, then let $f_u(k) = f_u(k^0)$ since all the edges outgoing from u^k are incoming to $\tilde{v}^j$ and therefore u^k can be degenerated into the same degenerated node as u^{k^0}. Otherwise, let $f_u(k) = f_u(k^0) + 1$.

It must be noted that there can be the case where the convergence condition and/or the divergence condition is not satisfied for some node u. In that case, node u^k is degenerated into the node $\tilde{u}^k$ for $k = 0, 1, \ldots, k_u - 1$ by the node-degeneration procedure and no node degeneration is performed.

The Order of Deriving Node Degeneration Functions. Although the node degeneration function can be derived for each node independent of the processing order of the nodes, one order of the nodes to be processed may result in a node-degenerated SRDFG with less nodes than another. If the set of nodes u^k is degenerated into the degenerated node $\tilde{u}^k$, more copies of the edge (u, v) may be outgoing from an identical degenerated node $\tilde{u}^k$. In that case, the possibility of degenerating the set of nodes v^k would be increased and they could be degenerated into less number of degenerated nodes.

Figure 11 shows an example. Let Fig. 11(a) be the equivalent SRDFG to be node-degenerated by the divergence condition. The nodes b^0 and b^1 satisfy the divergence condition since all the incoming edges to nodes b^0 and b^1 are outgoing from an identical node a^0 and there are no other edges incoming to nodes b^0 and b^1. By applying the node degeneration procedure on the node b, nodes b^0 and b^1 are degenerated into a single degenerated node $\tilde{b}^0$ as shown in Fig. 11(b). Then, nodes c^0 and c^1 can be degenerated into a single degenerated node $\tilde{c}^0$ since all the incoming edges to nodes c^0 and c^1 are outgoing from an identical node $\tilde{b}^0$ and there are no other edges incoming to nodes c^0 and c^1. Hence, the equivalent SRDFG shown in Fig. 11(a) is node-degenerated into the SRDFG shown in Fig. 11(c)

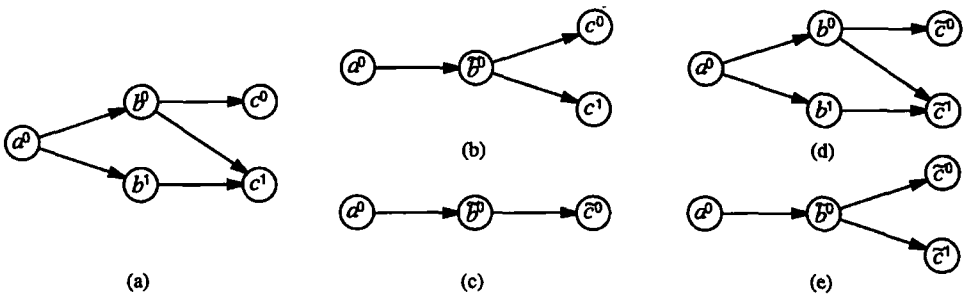

Fig. 11. The order of degenerating nodes. (a) An example SRDFG. Degenerating the node b first (b) and then the node c (c) results in 3-node degenerated SRDFG. Another order of the node c first (d) and then the node b results in the degenerated SRDFG with more nodes.

by applying the node degeneration procedure first on the node b and then on the node c.

If the node generation procedure is applied first on the node c, on the other hand, no actual node-degeneration is performed on the nodes c^0 and c^1 since the convergence condition is not satisfied for the nodes c^0 and c^1. The resultant SRDFG is illustrated in Fig. 11(d) and is identical to the original equivalent SRDFG. Then, applying the node degeneration procedure on the node b results in the node-degenerated SRDFG shown in Fig. 11(e).

Consequently, applying the node degeneration procedure on the node b first and then the node c results in the node-degenerated SRDFG with 3 nodes as shown in Fig. 11(c) and is better than applying the node degeneration procedure on the node c first and then the node b which leads to the node-degenerated SRDFG with 4 nodes as shown in Fig. 11(e).

One good order is to derive the node degeneration function for the ancestor node first in the case of the divergence condition. Such an order is achieved by the depth first search (DFS) with an appropriate root node. Similarly, processing the successor node first may be used in the case of the convergence condition. This order is also achieved by DFS by assuming the directions of edges are reversed. According to the discussion above, a node which could be degenerated should not be the root node of the DFS so as to derive the node-degenerated graph with the least number of nodes. Therefore, we choose a node with incoming edges outgoing from more than one ancestor nodes for the case of the divergence condition and a node with outgoing edges incoming to more than one successor nodes for the case of the convergence condition. In the case there is no such node, we may choose a node with the smallest number of invocations for the maximal node degeneration.

Node Degeneration During Expanding the MRDFG. Instead of degenerating the equivalent SRDFG, we can expand the MRDFG directly into the node-degenerated SRDFG by determining and utilizing the node degeneration function during the expansion of the MRDFG. The overall algorithm to construct the node-degenerated and the edge-degenerated equivalent SRDFG is summarized in Fig. 12.

At first, $f_u(k) = k$ for every node u as an initial proper node degeneration function. It means no node degeneration is performed at the beginning. DFSconv(u) updates the node degeneration functions for the node v if the convergence condition is satisfied for an edge (u, v). Similarly, DFSdiv(v) updates the node degeneration functions for the node u if the divergence condition is satisfied for an edge (u, v). After deriving the proper node degeneration functions by these DFSs, degenerated nodes and edges between them are included into the node set N_s and the edge set E_s, respectively.

The DFS procedures require time complexity $\mathcal{O}(|E_m|k)$ where k is the average of k_u for all $u \in N_m$. Including the node $\tilde{u}^k$ requires time complexity $\mathcal{O}(|N_m|k)$. Including edges between these nodes requires time complexity of $\mathcal{O}(|E_m|k)$ since $\mathcal{O}(k)$ copies are included for each edge in E_m. Therefore, the total time complexity of the algorithm is $\mathcal{O}(|N_m|k + |E_m|k)$. The memory requirement is $\mathcal{O}(|N_m|k)$ for storing the node degeneration function $f_u(k)$.

EXAMPLES. By the node degeneration, we can reduce the number of nodes and the number of edges. From the SRDFG shown in Fig. 6(b), we can degenerate the node b^0 and b^1 into one node and the node b^2 and b^3 into another node. Figure 6(c) shows the maximally node-degenerated SRDFG. In this example, the number of nodes is reduced from 9 to 7, and the num-

```
Input:  MRDFG G_m = (N_m, E_m, q_m, d_m)
Output: the node- and edge-degenerated SRDFG
        G_s = (N_s, E_s, q_s, d_s)
N_s = ∅, E_s = ∅;
Calculate k_u for all u ∈ N_m;
for all u ∈ N_m do
  for k = 0 to k_u − 1 do
    f_u(k) ← k
  enddo
enddo
/* Node degeneration for the convergence condition */
for all u ∈ N_m do
  Label u as 'not processed'
enddo
Choose the root node s for the convergence condition;
call DFSconv(s);
/* Node degeneration for the divergence condition */
for all u ∈ N_m do
  Label u as 'not processed'
enddo
Choose the root node s for the divergence condition;
call DFSdiv(s);
for all u ∈ N_m do
  for k = 0 to f_u(k_u − 1) do
    Include ū^k into N_s;
    q_s(ū^k) ← q_m(u)
  enddo
enddo
for all (u, v) ∈ E_m do
  W ← d_m(u, v);
  f^0 ← −1;
  for k = 0 to k_u − 1 do
    if f_v(((O_uv(k + 1) − 1 + W)\I_uv)%k_v) ≠ f^0 then
      f^0 ← f_v(((O_uv(k + 1) − 1 + W)\I_uv)%k_v);
      j ← f_v(((O_uv k + W)\I_uv)%k_v);
      W^0 ← (O_uv k + W)\(I_uv k_v);
      if f^0 ≥ j then f^1 ← f^0
                 else f^1 ← f^0 + f_v(k_v − 1) + 1;
      for j^1 = j to f^1 do
        Include (ū^{f_u(k)}, v̄^j) into E_s;
        d_s(ū^{f_u(k)}, v̄^j) ← W^0;
        j ← j + 1;
        if j > f_v(k_v − 1) then j ← 0, W^0 ← W^0 + 1
      enddo
    endif
  enddo
enddo.
```

```
procedure DFSconv(u)
for all (u, v) ∈ E_m do
  if v is labeled as 'processed' Skip to the enddo;
  if (u, v) is not the only incoming edge to v
    then goto descent;
  W ← d_m(u, v), k^0 ← 0;
  while f_u(((I_uv k^0 − W)\O_uv)%k_u) ≠
          f_u(((I_uv(k^0 + 1) − 1 − W)\O_uv)%k_u) do
    k^0 ← k^0 + 1
  enddo
  j^0 ← f_u(((I_uv(k^0 + 1) − 1 − W)\O_uv)%k_u);
  k ← k^0 + 1;
  while k ≤ k_v do
    if f_u(((I_uv(k + 1) − 1 − W)\O_uv)%k_u) = j^0 then
      f_v(k) ← f_v(k^0)
    else
      f_v(k) ← f_v(k^0) + 1, k^0 ← k;
      j^0 ← f_u(((I_uv(k + 1) − 1 − W)\O_uv)%k_u)
    endif
    k ← k + 1
  enddo
descent:
  Label v as 'processed';
  call DFSconv(v)
enddo.

procedure DFSdiv(v)
for all (u, v) ∈ E_m do
  if v is labeled as 'processed' then Skip to the enddo;
  if (u, v) is not the only outgoing edge from u then
    then goto descent;
  W ← d_m(u, v), k^0 ← 0;
  while f_v(((O_uv k^0 + W)\I_uv)%k_v) ≠
          f_v(((O_uv(k^0 + 1) − 1 + W)\I_uv)%k_v) do
    k^0 ← k^0 + 1
  enddo
  j^0 ← f_v(((O_uv(k^0 + 1) − 1 + W)\I_uv)%k_v);
  k ← k^0 + 1;
  while k ≤ k_u do
    if f_v(((O_uv(k + 1) − 1 + W)\I_uv)%k_v) = j^0 then
      f_u(k) ← f_u(k^0)
    else
      f_u(k) ← f_u(k^0) + 1, k^0 ← k;
      j^0 ← f_v(((O_uv(k + 1) − 1 + W)\I_uv)%k_v)
    endif
    k ← k + 1
  enddo
descent:
  Label u as 'processed';
  call DFSdiv(u)
enddo.
```

Fig. 12. An algorithm to construct the node-degenerated equivalent SRDFG.

ber of edges is reduced from 16 to 13. The number of delays is also reduced from 4 to 3.

The number of invocations of nodes V, W, X, Y, and Z in the MRDFG illustrated in Fig. 13(a) are 1, 10, 2, 20, and 20, respectively [13]. Therefore, the equivalent SRDFG consists of 53 nodes, 122 edges, and 65 delays. The edge degeneration procedure cannot reduce any edges from this equivalent SRDFG since there

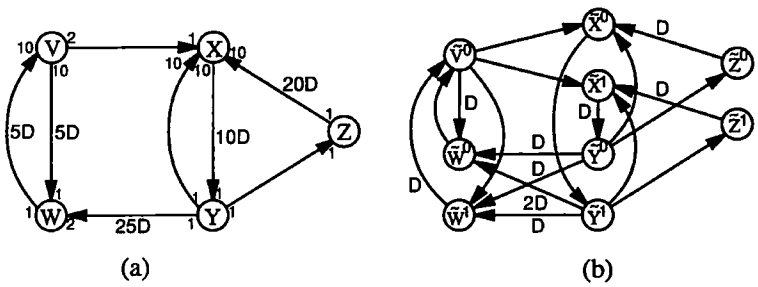

Fig. 13. An example of the node degeneration. (a) The given MRDFG. (b) The node-degenerated equivalent SRDFG.

exist no parallel edges. However, the node-degenerated SRDFG as shown in Fig. 13(b) is constructed by the node degeneration procedure. This graph consists of only 9 nodes, 18 edges, and 10 delays.

7 Experimental Results

The CPU time to determine the iteration bound for practical SRDFGs are compared. We chose the 5th order elliptic wave filter (EWF) [14] and the recursive part of the 4-level pipelined lattice filter (PLF) [15] as benchmarks. EWF which consists of 34 nodes, 56 edges, and 7 delays is an SRDFG where the number of delays, $|D|$, is relatively smaller than the number of nodes, $|N|$, and the number of edges, $|E|$. On the other hand, PLF which consists of 8 nodes, 10 edges, and 8 delays is an SRDFG where $|D|$ is comparable to $|N|$ and $|E|$.

Table 1 shows the comparison of time complexity, memory requirement, and CPU time to determine each iteration bound of EWF and PLF. In this table, NCD is the negative cycle detection method by using Bellman-Ford shortest path algorithm to detect negative cycles, LPM is the longest path matrix method, LPM′ is the mixture of LPM and NCD methods by using Floyd shortest path algorithm to detect negative cycles, and MCM is the minimum cycle mean based method. The computation time of node i, $q(i)$, is assumed 1 if node i is an addition or 2 if it is a multiplication. All the CPU times are measured on a SparcStation 2 and do not include the time consumed in reading the DFG from a file.

In NCD and LPM′ methods, the calculation of the iteration bound is terminated when the difference between successive guess iteration bounds becomes smaller than $1/|N|^2\gamma^2$ where $|N|$ is the number of nodes in the DFG and γ is the longest computation time of nodes [6]. While LPM and MCM derive the exact iteration bound, NCD and LPM′ derive only an approximate iteration bound. Some post-calculations may be necessary to identify the exact iteration bound from the approximate.

Table 2 shows CPU times to determine the iteration bounds of the equivalent and node-degenerated SRD-

Table 2. Iteration bound determination of DFG in Fig. 13.

Method	CPU [mS]	
	Equivalent	Node-degen.
NCD	52.7[a]	1.45[c]
LPM	4350[b]	3.02[b]
LPM′	442[a]	4.10[c]
MCM	40.7[b]	0.767[b]

[a]The obtained iteration bound = 4.0003128.
[b]The obtained iteration bound = 4.0000000.
[c]The obtained iteration bound = 3.9902344.

Table 1. Comparison of iteration bound determination algorithms.

Method	Time complexity	Memory requirement	CPU [mS]	
			EWF	PLF
NCD	$\mathcal{O}(\|N\|\|E\|\log\|N\|)$	$\mathcal{O}(\|N\|+\|E\|)$	25.2[a]	1.00[c]
LPM	$\mathcal{O}(\|D\|\|E\|+\|D\|^4)$	$\mathcal{O}(\|N\|+\|D\|^2)$	1.92[b]	2.97[d]
LPM′	$\mathcal{O}(\|D\|\|E\|+\|D\|^3\log\|D\|)$	$\mathcal{O}(\|N\|+\|D\|^2)$	3.58[a]	6.38[c]
MCM	$\mathcal{O}(\|D\|\|E\|+\|D\|\|E_d\|)$	$\mathcal{O}(\|N\|+\|D\|^2)$	0.717[b]	0.650[d]

[a]The obtained iteration bound = 16.0002594.
[b]The obtained iteration bound = 16.0000000.
[c]The obtained iteration bound = 1.50439453.
[d]The obtained iteration bound = 1.50000000.

FGs of the MRDFG in Fig. 13(a). All the computation time of nodes are assumed to be 1. We can see that the node degeneration greatly improves CPU time in any iteration bound methods and that our proposed iteration bound determination method is the fastest.

8 Conclusions

In this paper we proposed a new method to determine the iteration bound of SRDFGs. Its time complexity is better than the previously reported methods in the case where the number of delays is relatively smaller than the number of nodes in the SRDFG. If the number of delays is much larger than the number of nodes, then the NCD method would be the fastest; however, most digital signal processing algorithms do not fall into this category.

The node degeneration technique to reduce the number of nodes and the number of edges of the equivalent SRDFG of an MRDFG is also proposed. In some cases, node degeneration may not be applicable. However, if the node-degeneration is applicable, then it is shown that the iteration bound of the node-degenerated SRDFG can be computed faster than the approach where either only edge degeneration or no degeneration is applied.

In rate-optimal scheduling, the iteration bound is computed many times. Therefore, the node degeneration technique would play an important role in speeding up the scheduling by minimizing the iteration bound determination time. In the case of further combining the node degeneration technique into scheduling, careful attention should be paid since some nodes may have been removed from the node-degenerated SRDFG and operations of those nodes have to be scheduled as well as the nodes in the node-degenerated SRSFG.

Note

1. The technique described in [10] can also eliminate some transitive edges and hence derives a reduced SRDFG with the less number of edges than the edge-degenerated SRDFG.

References

1. K.K. Parhi, "Algorithm Transformation Techniques for Concurrent Processors," *Proc. of the IEEE*, Vol. 77, pp. 1879–1895, Dec. 1989.
2. E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Program for Digital Signal Processing," *IEEE Trans. Computers*, Vol. C-36, pp. 24–35, Jan. 1987.
3. M. Renfors and Y. Neuvo, "The Maximum Sampling Rate of Digital Filters under Speed Constraints," *IEEE Trans. Circuits Syst.*, Vol. CAS-28, pp. 196–202, Mar. 1981.
4. D.A. Schwartz and T.P. Barnwell, III, "A Graph Theoretic Technique for the Generation of Systolic Implementations for Shift Invariant Flow Graphs," in *Proc. of the 1984 IEEE ICASSP*, San Diego, CA, Mar. 1984.
5. K.K. Parhi and D.G. Messerschmitt, "Static Rate-Optimal Scheduling of Iterative Data-Flow Programs via Optimum Unfolding," *IEEE Trans. Computers*, Vol. C-40, pp. 178–195, Feb. 1991.
6. D.Y. Chao and D.Y. Wang, "Iteration Bounds of Single-Rate Data Flow Graphs for Concurrent Processing," *IEEE Trans. Circuits Syst.-I*, Vol. CAS-40, pp. 629–634, Sept. 1993.
7. S.H. Gerez, S.M. Heemstra de Groot, and O.E. Herrmann, "A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data-Flow Graphs," *IEEE Trans. Circuits Syst.-I*, Vol. CAS-39, pp. 49–52, Jan. 1992.
8. K. Ito and K.K. Parhi, "Determining the Iteration Bounds of Single-Rate and Multi-Rate Data-Flow Graphs," in *Proc. of 1994 IEEE Asia-Pacific Conf. on Circuits and Systems*, Taipei, Taiwan, Dec. 1994, pp. 6A.1.1–6A.1.6.
9. R.M. Karp, "A Characterization of the Minimum Cycle Mean in a Digraph," *Discrete Mathematics*, Vol. 23, pp. 309–311, 1978.
10. R. Govindarajan and G.R. Gao, "A Novel Framework for Multi-Rate Scheduling in DSP Applications," in *Proc. 1993 Int. Conf. Application-Specific Array Processors*, pp. 77–88, IEEE Computer Society Press, 1993.
11. E.M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Englewood Cliffs, NJ: Prentice Hall, 1977.
12. E.L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.
13. S.S. Bhattacharyya, J.T. Buck, S. Ha, and E.A. Lee, "A Scheduling Framework for Minimizing Memory Requirements of Multirate DSP Systems Represented as Dataflow Graphs," in *VLSI Signal Processing, VI*, pp. 188–196, 1993.
14. S.Y. Kung, H.J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1985.
15. J.-G. Chung and K.K. Parhi, "Pipelining of Lattice IIR Digital Filters," *IEEE Trans. Signal Processing*, Vol. SP-42, pp. 751–761, Apr. 1994.

Kazuhito Ito received the BS, the MS, and the Ph.D. degrees in Electrical Engineering from Tokyo Institute of Technology, Tokyo

(Japan), in 1987, 1989, and 1992, respectively. He was with the Tokyo Institute of Technology from 1992 and visited the University of Minnesota from April 1993 to July 1994. He is now an associate professor of the Department of Electrical and Electronic System Engineering, Saitama University, Urawa (Japan). His research interests include high-level synthesis in digital signal processing, VLSI signal processing, and asynchronous systems.
kazuhito@elc.ees.saitama-u.ac.jp

Keshab K. Parhi received the B. Tech. (Honors) degree from the Indian Institute of Technology, Kharagpur (India), in 1982, the M.S.E.E. degree from the University of Pennsylvania, Philadelphia, in 1984, and the Ph.D. degree from the University of California, Berkeley in 1988.

He has been with the University of Minnesota since 1988 where he is currently a Professor of Electrical Engineering. He has held short term positions in several industries. His research interests include concurrent algorithm and architecture designs for communications, signal and image processing systems, digital integrated circuits, VLSI digital filters, computer arithmetic, high-level DSP synthesis, and multiprocessor prototyping and task scheduling for programmable software systems.

Dr. Parhi received the 1994 Darlington and 1993 Guillemin-Cauer best paper awards from the IEEE Circuits and Systems society, the 1992 Young Investigator Award of the National Science Foundation, the 1992–1994 McKnight-Land Grant professorship of the University of Minnesota, the 1991 best paper award from the IEEE signal processing society, the 1991 Browder Thompson prize paper award of the IEEE, the 1989 research initiation award of the National Science Foundation, and the 1987 U.C. Berkeley Eliahu Jury award for excellence in systems research. He is a former associate editor of the IEEE Transactions on Circuits and Systems, an associate editor of the IEEE Transactions on Signal Processing, an editor of the *Journal of VLSI Signal Processing*, an associate editor of the IEEE Transactions on Circuits and Systems—Part II, and is a senior member of IEEE and a member of the Eta Kappu Nu.
parhi@ee.umn.edu