Published in: ProRISC and IEEE Benelux Workshop on Circuits, Systems and Signal Processing, Mierlo, The Netherlands, pp. 107-116 (March 1995).

Special-Case Techniques for the Efficient Computation of the Iteration-Period Bound in Multirate Data-Flow Graphs

Sabih H. Gerez, Michel L.M. de Jong and Sonia M. Heemstra de Groot
University of Twente, Department of Electrical Engineering
P.O. Box 217, 7500 AE Enschede, The Netherlands
e-mail: s.h.gerez@el.utwente.nl

Abstract

A multirate synchronous data-flow graph, in which nodes operate at distinct speeds, has a single-rate equivalent in which all nodes operate at the same speed. In order to find the fastest implementation of the graph, one needs to know the graph's *iteration-period bound*. It is well-known how to compute it after expanding the graph to its single-rate equivalent. However, the single-rate equivalent can be considerably larger than the original graph. In this paper, a method is presented for the construction of a *reduced graph* that has all relevant properties of the single-rate equivalent, but is much smaller in general. It can be proven that the size of the reduced graph does not depend on the relative speeds of the nodes in the original graph and grows as a polynomial function of the original graph's size. The presented method is limited to graphs without selfloops.

1 Introduction

Synchronous data-flow graphs [12] are widely accepted for the representation of digital signal processing algorithms. A data-flow graph (DFG) can be understood as a set of nodes that consume tokens from their incoming edges and produce tokens on their outgoing edges. Synchronous data flow is characterized by the fact that the number of tokens consumed and produced is constant, independent of the data values carried by the tokens. The number of tokens produced on one end of an edge can, however, be different from the number of tokens consumed at the other end. This means that the nodes on both ends operate at different speeds. For example, if each execution of the node at the producing end of an edge puts 3 tokens on the edge and the node at the consuming end takes 1 token at each execution, the consuming node should be executed 3 times faster than the producing node. Otherwise, there will either be an overflow of tokens on the edge or a deadlock situation. This is the consequence of the fact that a DFG is supposed to be executed an indefinite number of times: the computation of the whole DFG has to be repeated every T_0 time units, where T_0 is called the *iteration period*. The synchronous DFG model will be presented more formally in Section 2.

Thus, synchronous data flow allows for the specification of *multirate* signal processing algorithms. *Homogeneous* or *single-rate* data flow is a special case of synchronous data flow, in which each node in each execution consumes a single token from its incoming edges and produces a single token on its outgoing edges. It is obvious that all nodes operate at the same speed in this case. Any multirate DFG has an equivalent single rate DFG [12]. This allows the application of theory known for single-rate DFGs to multirate DFGs after transforming the multirate DFG. The transformation procedure is explained in Section 3.

An interesting issue is the maximal speed at which a DFG can be implemented given a certain *library* of hardware resources. In the case of an *acyclic* DFG (a graph without directed loops), the T_0 of the implementation can be decreased arbitrarily by the use of *pipelining* and the multiplication of the hardware, given a certain library. This does not require any modification of the DFG. However, the same is not possible in case of a *cyclic* DFG. There, the minimum T_0, called the *iteration-period bound* and denoted by $T_{0_{min}}$, is bounded by the topology of the DFG due to feedback loops: a node n whose output propagates through a number of other nodes and then reaches an input of n, has to wait through this propagation time. Of course, a feedback loop always needs to contain an edge with at least one delay element in order for the DFG to be computable.

Research in the past has led to many contributions for the computation of $T_{0_{min}}$ in single-rate DFGs, as will be reviewed in Section 4. Section 5 deals with the computation of $T_{0_{min}}$ in multirate graphs. After transforming a multirate DFG to a single rate one, one could apply the already known algorithms for the computation of $T_{0_{min}}$ of the multirate DFG. The disadvantage of this approach is that the equivalent single-rate DFG can become very large. A method for the construction of a reduced size graph on which the algorithms for the computation of $T_{0_{min}}$ in single-rate DFGs can operate, is presented in [5]. However, no quantitative statements about the size of the reduced graph are made. This paper presents a similar technique for the construction of a reduced graph: while it is only applicable to a subset of all multirate DFGs, it turns out that the upper bound of the reduced-graph size is independent of the number of tokens produced and consumed by individual nodes and grows as a polynomial function of the size of the multirate DFG. This issue is explained in detail in Section 6.

2 Synchronous Data Flow

A synchronous DFG is denoted by $G_m(V_m, E_m)$, with $V_m = \{v_1, v_2, \ldots, v_{n_m}\}$ the set of *nodes* and $E_m = \{e_1, e_2, \ldots, e_{a_m}\} \subset V_m \times V_m$ the set of *edges* in the graph. Nodes represent computations (or operations) and edges precedence relations: tokens produced by a node v_i are passed to another node v_j along an edge (v_i, v_j). The DFG should receive its inputs from special *input nodes* and send its output to special *output nodes.* However, these nodes are not relevant to the topic of this paper and are further disregarded. Given a library of hardware resources, $\delta(v_i)$ gives the fastest possible execution time of computation v_i by some element of the library. An edge can carry zero or more *delay elements.* The number of delay elements on an edge $e_k = (v_i, v_j)$ is given by $\Delta(e_k)$. The number of tokens produced on an edge $e_k = (v_i, v_j)$ by node v_i in a single execution of itself is given by $\pi(e_k)$ and the number of tokens consumed from the edge in a single execution by node v_j is indicated by $\kappa(e_k)$. An example of a synchronous DFG is given in Figure 1(a). In the figure, $\pi(e_k)$ and $\kappa(e_k)$ are shown at the back and front ends of the arrow representing edge e_k respectively. Delay elements are depicted by nD along an edge e_k, where $n = \Delta(e_k)$. n is left out when $n = 1$.

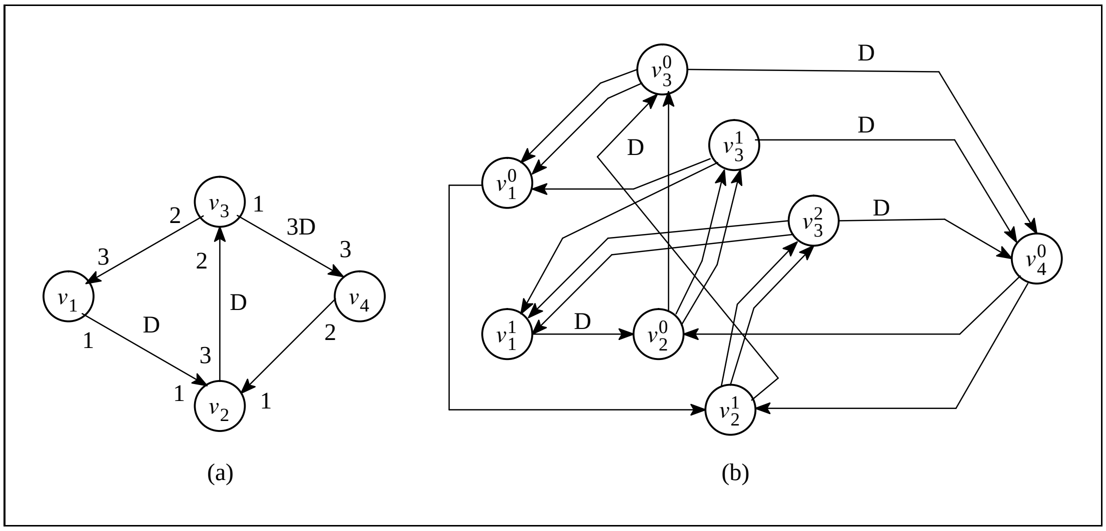

Figure 1: *A multirate DFG (a) and its equivalent single-rate DFG (b).*

In the data-flow model of computation a node v_i can start its execution when all its incoming edges $e_k = (v_j, v_i)$ carry at least $\kappa(e_k)$ tokens. $\kappa(e_k)$ tokens are then consumed from these edges and $\pi(e_l)$ tokens are produced on the outgoing edges $e_l = (v_i, v_m)$. An edge behaves as a FIFO (first-in first-out) buffer, which means that the ordering of tokens on an edge is always preserved. The number of delay elements on an edge should then be understood as the number of tokens initially present on an edge before the first execution of the DFG. Clearly, in a graph with cycles, there should be a sufficient number of delay elements to be able to start the computation. Besides, a consistent graph should keep the computation running such that neither any deadlock occurs due to a shortage of tokens on an edge nor the number of tokens on an edge grows indefinitely leading to buffer overflow.

As was already mentioned in Section 1, one iteration of $G_m(V_m, E_m)$ possibly requires multiple executions of individual nodes in V_m. The number of executions per iteration of each node v_i will be denoted by $q(v_i)$, which has an integer value. For each edge $e_k = (v_i, v_j)$, the *equilibrium condition*, $q(v_i)\pi(e_k) = q(v_j)\kappa(e_k) = \epsilon(e_k)$ should be satisfied in order to avoid deadlock or overflow. In the rest of this paper, $\epsilon(e_k)$ will be used to refer to the total number of tokens passing through an edge in one iteration. A method for checking the consistency of a DFG is given in [11]. In a consistent DFG, there are an infinite number of functions q that satisfy the equilibrium condition. By definition, that q should be chosen that has the smallest possible integer values $q(v_i)$.

3 Equivalent Single-Rate DFGs for Multirate DFGs

The construction procedure for an equivalent single-rate DFG of a multirate DFG was shown in [10, 12] by means of examples. A more formal presentation of the transformation involved can be found in [5]. Here, such a presentation is repeated following the notation of Section 2 and extending it when necessary. The extensions to the notation will be used in the rest of

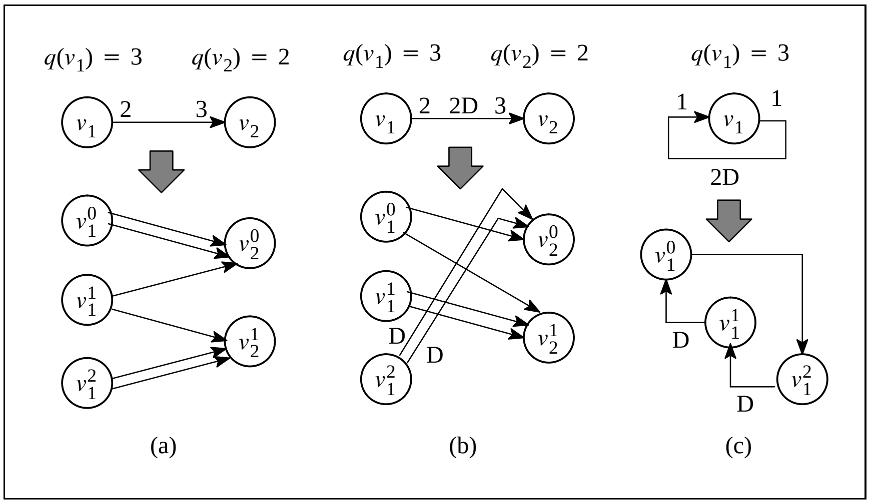

Figure 2: *Three multirate edges (above) and their single-rate representations (below).*

this paper.

The single-rate equivalent of a DFG $G_m(V_m, E_m)$ will be denoted by $G_s(V_s, E_s)$. For each node $v_i \in V_m$, there are nodes $v_i^p \in V_s$, with $0 \leq p \leq q(v_i) - 1$. Each edge $e_k = (v_i, v_j) \in E_m$ is transformed into edges $e_k^l \in E_s$, with $0 \leq l \leq \epsilon(e_k) - 1$. Each v_i^p will have $\pi(e_k)$ outgoing edges and each v_j^r will have $\kappa(e_k)$ incoming edges due to e_k. The sources and destinations of the edges obey:

$$e_k^l = (v_i^{l \div \pi(e_k)}, v_j^{((l+\Delta(e_k)) \div \kappa(e_k)) \bmod q(v_j)}) \tag{1}$$

where $\div$ represents integer division and "mod" is the modulo operator (rest after integer division). Besides, the numbers of delay elements on the new edges follow from:

$$\Delta(e_k^l) = (l + \Delta(e_k)) \div \epsilon(e_k) \tag{2}$$

Figure 2 shows a number of examples of edges in a multirate graph and their corresponding representation in a single-rate graph. The application of the single-rate transformation to multirate graph as a whole is illustrated in Figure 1.

In $G_s(V_s, E_s)$, $|V_s| = \sum_{i=1}^{n_m} q(v_i)$ and $|E_s| = \sum_{k=1}^{a_m} \epsilon(e_k)$. The number of delay elements remains the same in the sense that $\sum_{k=1}^{a_m} \Delta(e_k) = \sum_{k=1}^{a_m} \sum_{l=0}^{\epsilon(e_k)-1} \Delta(e_k^l)$. Note that the size of the graph depends on the values $\pi(e_k)$ and $\kappa(e_k)$ ($q(v_i)$ and $\epsilon(e_k)$ are derived from these values). It is not difficult to define families of multirate graphs whose sizes grow linearly as a function of some parameter n, while their single-rate equivalents grow exponentially as a function of n. An example of such a family is given in Figure 3.

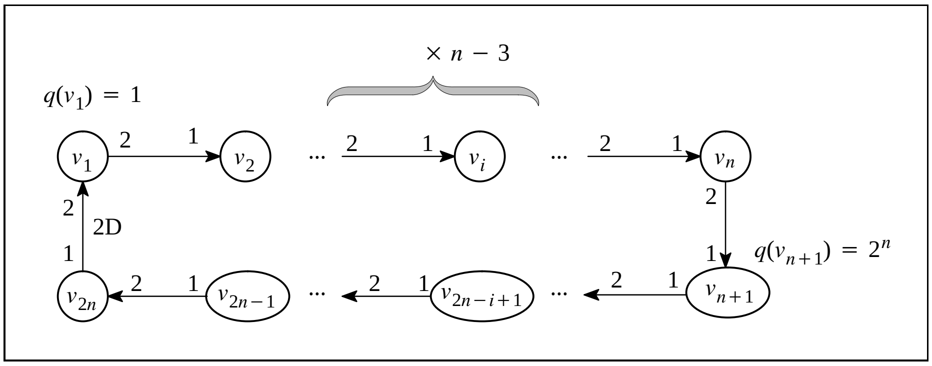

Figure 3: *A family of multirate graphs whose single-rate equivalents grow exponentially as a function of parameter* n.

4 The Iteration-Period Bound for Single-Rate Graphs: Definition and Efficient Computation

In a single-rate graph, the iteration-period bound $T_{0_{min}}$ is defined as follows [14, 15][1]:

$$T_{0_{min}} = \left\lceil \max_{\text{all loops } L} \frac{\sum_{v \in L} \delta(v)}{\sum_{e \in L} \Delta(e)} \right\rceil \tag{3}$$

Theoretically, one should inspect all loops in the graph to compute $T_{0_{min}}$. But the number of loops can grow exponentially with the graph size [4] and more efficient methods are desirable and turn out to exist. Several researchers have proposed algorithms that have a polynomial worst-case time complexity [3, 4, 5, 7, 8, 9, 13].

Of these methods, the one described by Karp [7] has the lowest worst-case time complexity; it also shows the best performance in practical situations as is shown in [5]. The first step in the method is to compute the longest-path distances between all pairs of delay elements in a DFG $G(V, E)$. If $|D|$ is the number of delay elements in G, this step has a worst-case time complexity of $\mathcal{O}(|D||E|)$ [4]. These pairwise distances can be represented in a new graph $G_d(D, E_d)$. Application of Karp's algorithm on this graph gives the iteration-period bound in $\mathcal{O}(|D||E_d|)$.

5 The Iteration-Period Bound for Multirate Graphs

The iteration-period bound of a multirate graph $G_m(V_m, E_m)$, is defined to be equal to the iteration-period bound of its transformed single-rate equivalent $G_s(V_s, E_s)$ [5]. However, because the size of G_s can be considerably larger than the size of G_m, as was shown in Section 3, it is justified to look for methods that compute G_m's $T_{0_{min}}$ without directly operating on G_s.

[1]In this expression, $\lceil x \rceil$ means the smallest integer $\geq x$. Besides, the set membership predicate $\in$ is assumed to be applicable to nodes and edges in a loop (a loop is formally not a set, but an alternating sequence of nodes and edges).

In [5] a method based on *edge degeneration* and *node degeneration* is presented. Edge degeneration means that parallel edges in G_s are combined to a single edge having a number of delay elements equal to the minimum of the delay elements of the combined edges (note that any loop in G_s going through an edge with a nonminimal number of delays will give a nonmaximal value in the quotients of Equation 3). After the graph has been simplified by means of edge degeneration, node degeneration is applied. This step merges nodes in G_s whose incoming edges originate from the same node or whose outgoing edges have the same destination. It is not difficult to see that any loop present before merging will have a counterpart with the same properties (the quotient in Equation 3) in the graph obtained after merging.

The degeneration method is presented in [5] as a generally applicable heuristic. No statements are made about the size of the graph obtained after degeneration and the order in which nodes are merged affects the size of the final graph obtained. In the next section, a method is presented that allows for the formulation of maximal size bounds of the "degenerated" graph. Unfortunately, the method is only applicable to a subset of all multirate graphs, viz. those without selfloops.

6 The Reduced Graph

In this section a method is presented for the construction of a *reduced graph* $G_r(V_r, E_r)$ derived from the multirate graph $G_m(V_m, E_m)$, which in general is considerably smaller than $G_s(V_s, E_s)$ while preserving all properties with respect to the quotients in Equation 3 (no maximal-quotient loop will be lost). The method is restricted to those G_m that do not have selfloops: $\forall (v_i, v_j) \in E_m : v_i \neq v_j$. The method works as follows: first all edges $(v_i, v_j) \in G_m$ are analyzed one by one, which results in the creation of at most three node groups for v_i and three node groups for v_j; then all node groups for the same original node are reconciled to form new node groups; finally, appropriate edges are generated between the specific node groups.

The principle used here is the same as in [5]: multiple nodes $v_i^p \in V_s$ can be represented in G_r by the same node as long as all loops in G_s have a counterpart in G_r. This is the reason why selfloops cannot be adequately handled by this method. As is illustrated in Figure 2(c), a selfloop in G_m results in edges between the $v_i^p \in V_s$ (i is constant, p varies); contracting the v_i^p into a single node will lead to loss of path and loop information.

Consider a single edge $e_k = (v_i, v_j) \in E_m$. In G_s, v_i is represented by nodes v_i^p with $0 \leq p \leq q(v_i) - 1$ and v_j by nodes v_j^r with $0 \leq r \leq q(v_j) - 1$ while there are $\epsilon(e_k)$ edges interconnecting these nodes. From Equation 2 it can be seen that the number of delay elements on these edges are either all equal (when $\Delta(e_k)$ is an integer multiple of $\epsilon(e_k)$; note that $0 \leq l \leq \epsilon(e_k) - 1$) or have two consecutive integer values, say n and $n+1$, with $n = \Delta(e_k) \div \epsilon(e_k)$. Equation 2 can now be formulated more precisely as:

$$\begin{array}{lll} \Delta(e_k^l) = n & \text{when} & 0 \leq l < \epsilon(e_k) - \Delta(e_k) \bmod \epsilon(e_k) \\ \Delta(e_k^l) = n+1 & \text{when} & \epsilon(e_k) - \Delta(e_k) \bmod \epsilon(e_k) \leq l < \epsilon(e_k) \end{array} \tag{4}$$

Therefore, if all other edges in G_m are neglected temporarily, one can group together the nodes v_i^p that have outgoing edges with n delays, those that have outgoing edges with $n+1$ delays, and at most one node with outgoing edges of both types. Analogously, the nodes v_j^r can be

k	v_i n	$n/n+1$	$n+1$	v_j $n+1$	$n/n+1$	n
1	$w^{[0,4]}$	–	$w^{[5,9]}$	–	$v^{[0,0]}$	–
2	–	$v^{[0,0]}$	–	$w^{[0,4]}$	–	$w^{[5,9]}$
3	$v^{[0,0]}$	–	–	–	–	$x^{[0,1]}$
4	$y^{[0,14]}$	–	$y^{[15,19]}$	$w^{[0,1]}$	$w^{[2,2]}$	$w^{[3,9]}$
5	$y^{[0,19]}$	–	–	–	–	$x^{[0,1]}$
6	$x^{[0,0]}$	–	$x^{[1,1]}$	$y^{[0,9]}$	–	$y^{[10,19]}$
7	$z^{[0,19]}$	–	–	–	–	$x^{[0,1]}$
8	$y^{[0,19]}$	–	–	–	–	$z^{[0,1]}$

Table 1: *The construction of node groups as a first step in the construction of the reduced graph.*

partitioned into three groups according to the delays of their incoming edges. A node in G_r replacing a number of consecutive nodes $v_i^a, v_i^{a+1}, \ldots, v_i^b \in V_s$ will be denoted by an "interval superscript" as $v_i^{[a,b]}$.

First the partitioning of the nodes originating from v_i is considered, using λ for the value of the expression $\epsilon(e_k) - \Delta(e_k) \bmod \epsilon(e_k)$. There exist a node with distinct outgoing edges having n and $n+1$ delays, if the following condition is satisfied:

$$(\lambda - 1) \div \pi(e_k) = \lambda \div \pi(e_k)$$

The condition follows from Equations 1 and 4.

If the condition is satisfied, v_i is partitioned into three node groups, to be called of type "n", "$n/n+1$" (or "mixed"), and "$n+1$" respectively:

$$v_i^{[0,(\lambda-1)\div\pi(e_k)-1]}, v_i^{[(\lambda-1)\div\pi(e_k),(\lambda-1)\div\pi(e_k)]}, v_i^{[(\lambda-1)\div\pi(e_k)+1,q(v_i)-1]}$$

If the condition is not satisfied, v_i is partitioned into two groups of type "n" and "$n+1$":

$$v_i^{[0,(\lambda-1)\div\pi(e_k)]}, v_i^{[\lambda\div\pi(e_k),q(v_i)-1]}$$

In both cases, an empty interval (with an upper bound smaller than the lower bound) means that the corresponding node group does not exist.

The partitioning of v_j into at most three groups due to edge $e_k = (v_i, v_j)$ can be done in a similar way.

So, the first step in the construction of G_r is to iterate over all edges in G_m and construct at most three node groups for both nodes connected to the edge. This step is illustrated in Table 1 for the example multirate graph taken from [1] as shown in Figure 4(a). In the figure, the edges e_k have been labeled by their index k enclosed in a rectangle, following the style of [12]. Besides, following [1], the nodes are called v, w, etc. instead v_1, v_2, etc. In the table, there is a row for each edge $e_k = (v_i, v_j) \in E_m$ and there are columns for the different categories of node groups obtained from v_i and v_j respectively.

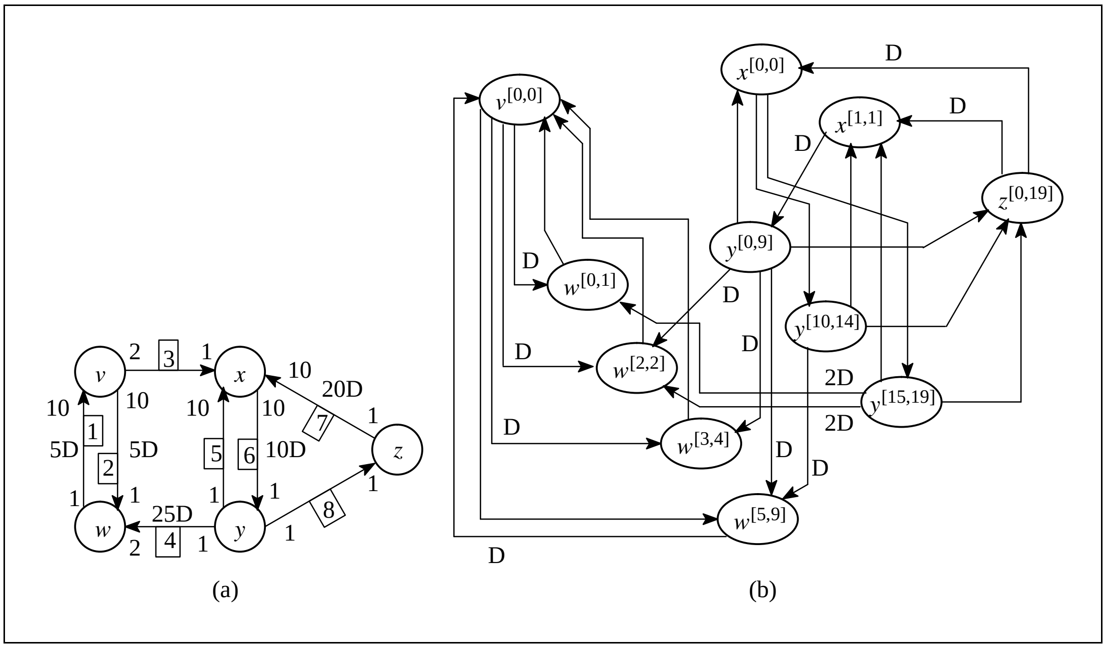

Figure 4: *An example multirate graph (a) and its reduced graph (b).*

Nodes can get partitioned in different ways due to different edges. In the second stage of the construction of G_r, for each node a new partitioning is constructed such that the number of node groups is minimal, while any of the node groups found in the first stage can be constructed as the union of one or more new node groups. These node groups form the set of nodes V_r of G_r. For the example, this means that the following node groups are now obtained: $v^{[0,0]}$, $w^{[0,1]}$, $w^{[2,2]}$, $w^{[3,4]}$, $w^{[5,9]}$, $x^{[0,0]}$, $x^{[1,1]}$, $y^{[0,9]}$, $y^{[10,14]}$, $y^{[15,19]}$, and $z^{[0,19]}$.

The final step is the construction of the edge set E_r. This is done by considering once more all edges $e_k = (v_i, v_j) \in E_m$ one by one. For each edge the node groups $v_i^{[a,b]}$ are processed separately. As a consequence of its construction, the node group has either only outgoing edges with n delays, only outgoing edges with $n+1$ delays, or it consists of a single node with outgoing edges of both types. In the first two cases, first the edges with smallest and largest index l are computed. So, the values $l_{min} = a\pi(v_i)$ and $l_{max} = (b+1)\pi(v_i) - 1$ for l are considered, leading to two values r, say r_{min} and r_{max}, by computing v_j^r for l_{min} and l_{max} using Equation 1. Now, edges $(v_i^{[a,b]}, v_j^{[c,d]})$ are created for those node groups of v_j obeying $c \geq r_{min}$ and $d \leq r_{max}$. The number of delays on the edges is either n or $n+1$ depending on the type of the node group $v_i^{[a,b]}$. The case of the nodes of mixed type is slightly more complex: values r are not only computed for l_{min} and l_{max}, but also for $\lambda - 1$ and λ. By computing the corresponding values r, node groups $v_j^{[c,d]}$ are identified to which an edge with n respectively $n+1$ will be connected.

The reduced graph of the DFG in Figure 4(a), obtained by the method described in this section is shown in Figure 4(b). The reduced graph has 11 nodes, 25 edges and 14 delay elements compared to the 53 nodes, 122 edges and 65 delays of the single-rate equivalent. The degenerated graph for this example obtained by the methods of [5] is still smaller. However, node degeneration applied to the graph in Figure 4(b) will combine the nodes $w^{[0,1]}$, $w^{[2,2]}$ and

$w^{[3,4]}$, as well as the nodes $y^{[10,14]}$ and $y^{[15,19]}$, leading to a smaller graph than in [5].

Actually, discussions about precise graph sizes are irrelevant to the main idea of this paper: the size of G_r is bounded by a polynomial of the size parameters of G_m, while the size of G_s can grow exponentially as function of the size of G_m (see Section 3). Suppose that the total number of edges incident to or from any node in V_m is bounded by d_{max}. Then, assuming that each edge in $e_k = (v_i, v_j) \in E_m$ leads to distinct node groups for v_i and v_j, there will be $2d_{max} + 1$ node groups for each node in the worst case, while each edge in G_m will lead to the creation of $(2d_{max} + 1)^2$ edges in G_r. This means that $|V_r| = \mathcal{O}(d_{max}|V_m|)$ and $|E_r| = \mathcal{O}(d_{max}^2|E_m|)$. The size of G_r grows polynomially even in the extreme case that $d_{max} = |E_m|$. Note as well that G_r's size does not depend on the values $\pi(v_i)$ and $\kappa(v_i)$ in G_m.

Another issue is the time complexity of the procedure to construct G_r: the first step can be performed in $\mathcal{O}(|E_m|)$ (a constant number of computations for each edge), the second step requires $\mathcal{O}(|V_m| d_{max} \log d_{max})$ (sorting the intervals and splitting them for each node) and the final step requires $\mathcal{O}(d_{max}^2|E_m|)$ (constant time for each newly created edge). The final step is dominant, resulting in $\mathcal{O}(d_{max}^2|E_m|)$ for the worst-case time complexity of the construction procedure.

7 Conclusions and Discussion

In this paper, a method has been presented for the efficient computation of the iteration-period bound in a multirate synchronous data-flow graph. The idea is to construct a reduced single-rate graph whose size is polynomially bounded, instead of constructing the actual single-rate equivalent, whose size can grow exponentially. Well-known methods for the computation of the iteration-period bound can then be applied to the reduced graph.

Unfortunately, the method presented is limited to graphs without selfloops. However, graphs with selfloops occur in practice, e.g. in hierarchical graphs, where a selfloop represents a state update in a node [10]. Also the requirement that all invocations of the same multirate node in one iteration should be executed on the same hardware unit, which is desirable from a scheduling point of view [2], gives rise to additional precedence edges in the single-rate graph, comparable to those originating from selfloops. So, efficient computation methods for the iteration-period bound in the general case remain desirable. Some first ideas on this can be found in [6].

References

[1] 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 L.D.J. Eggermont, P. Dewilde, E. Deprettere, and J. van Meerbergen, editors, *VLSI Signal Processing VI*. IEEE, New York, 1993.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J.A. Peperstraete. Static scheduling of multirate and cyclo-static DSP-applications. In *IEEE Workshop on VLSI Signal Processing*, La Jolla, CA, October 1994.

[3] D.Y. Chao and D.T. Wang. Iteration bounds of single-rate data flow graphs for concurrent processing. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(9):629–634, September 1993.

[4] 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 Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(1):49–52, January 1992.

[5] K. Ito and K.K. Parhi. Determining the iteration bounds of single-rate and multi-rate data-flow graphs. In *IEEE Asia-Pacific Conference on Circuits and Systems*, pages 163–168, Taipei, December 1994.

[6] M.L.M. de Jong. Iteration period bound of multirate synchronous data flow in digital signal processing. Technical Report EL-BSC-94N041, University of Twente, Department of Electrical Engineering, August 1994.

[7] R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[8] J.Y. Kim and H.S. Lee. Lower bound of sample word length in bit/digit serial architectures. *Electronics Letters*, 28(1):60–62, January 1992.

[9] E.L. Lawler. Optimal cycles in doubly weighted directed linear graphs. In *International Symposium on the Theory of Graphs*, pages 209–213, Rome, 1966.

[10] E.A. Lee. *A Coupled Hardware and Software Architecture for Programmable Digital Signal Processors*. PhD thesis, University of California, Berkeley, 1986.

[11] E.A. Lee and D.G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, C-36(1):24–35, January 1987.

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

[13] M. Potkonjak and J. Rabaey. Optimizing throughput and resource utilization using pipelining: Transformation based approach. *Journal of VLSI Signal Processing*, 8:117–130, 1994.

[14] R. Reiter. Scheduling parallel computations. *Journal of the ACM*, 15(4):590–599, October 1968.

[15] M. Renfors and Y. Neuvo. The maximum sampling rate of digital filters under hardware speed constraints. *IEEE Transactions on Circuits and Systems*, CAS-28(3):196–202, March 1981.