A Polynomial-Time Algorithm for the Computation of the Iteration-Period Bound in Recursive Data-Flow Graphs

Sabih H. Gerez, Sonia M. Heemstra de Groot, and Otto E. Herrmann

Abstract—**Rate-optimal scheduling of iterative data-flow graphs requires the computation of the *iteration period bound*. According to the formal definition, the total computational delay in each directed loop in the graph has to be calculated in order to determine that bound. As the number of loops cannot be expressed as a polynomial function of the number of nodes in the graph, this definition cannot be the basis of an efficient algorithm. This paper presents a polynomial-time algorithm for the computation of the iteration period bound based on *longest path matrices* and their multiplications.**

I. Introduction

Parallel processing is unavoidable for the implementation of fast computation-intensive DSP algorithms. As a consequence tools to detect parallelism and appropriate scheduling techniques have to be developed. Especially, techniques for nonpreemptive deterministic scheduling of DSP algorithms without data-dependent conditionals have received some attention (see, for example, [1]–[6]).

The scheduling problem is often modeled by mapping the algorithm specifications onto a *data-flow graph* (DFG). DFG's model the way computational operations depend on data produced by other operations, where data may be delayed by a multiple of the iteration period before being transferred (see, e.g., [3], [5], [7], or [8]). A DFG can be recursive or nonrecursive.

Nonrecursive DFG's can be executed arbitrarily fast by supplying the required computing power, combined with the transformation of the original network into a suitable essentially equivalent form [9] where the original parallelism has been increased, e.g., by introducing pipelining.

However, this is not the case for recursive DFG's, where there exists a limit imposed by the topology of the graph and the speed of the hardware units. Once this limit is reached, the addition of more computing power results only in a decrease of the processor utilization.

That limit is given by the maximum speed of execution of the so-called *critical loop* [1], [10] which determines the minimum sampling period, or *iteration period bound* [3], $T_{0_{\min}}$. A critical loop is that directed loop for which the summation of the processing delays of each operation in the loop divided by the number of delay elements in the loop, n_l, is maximal. This quotient is $T_{0_{\min}}$.

Expressed in another way:

$$T_{0_{\min}} = \max_{\text{all directed loops } l} \left\{ \frac{\sum_{c_i \in l} c_i.\text{duration}}{n_l} \right\} \qquad (1)$$

where c_i.duration is the duration of operation c_i.

A number of rate-optimal (maximum speed) scheduling algorithms has been proposed (see, e.g., [1], [3], and [6]). All of them require the computation of the iteration period bound, $T_{0_{\min}}$.

The computation of $T_{0_{\min}}$ by inspection of all directed loops, as given by (1), does not lead to an efficient algorithm, as will be shown later. This paper proposes a polynomial-time algorithm for the computation of the iteration period bound $T_{0_{\min}}$.

Manuscript received April 23, 1990; revised June 30, 1991. This paper was recommended by Associate Editor K. Thulasiraman.

The authors are with the Faculty of Electrical Engineering, University of Twente, 7500 AE Enschede, The Netherlands.

IEEE Log Number 9104302.

II. The DFG

For the purpose of this paper a DFG is defined as a directed graph: DFG(V, E). $V = C \cup D \cup I \cup O$ constitutes the set of vertices (C, D, I, and O are pairwise disjoint). C is the set of vertices associated with computational nodes (or operations), D is the set of delay elements, and I and O are the inputs and outputs, respectively. The delay associated to a delay element is equal to T_0, the iteration period. A delay of multiple iteration periods is assumed to be modeled here with single delay elements connected in series. An edge $e_k \in E$ can connect any pair of vertices in V with the constraint that no edge is incident *to* an input and no edge is incident *from* an output.

A *directed path* in the graph starts at a certain vertex, goes through zero or more vertices and finishes in another vertex following the edges in the graph according to their orientations. When the first and the last vertex of a directed path coincide, the path is called a *directed loop*. In the rest of the text a directed path will be called a *path* and a directed loop will be called a *loop*. A path or loop is *simple* when the path or loop passes at most once through each vertex in the DFG [11]. When a path or loop is not simple, it has to contain internal loops.

We assume that a DFG is always connected. The topology of a DFG obeys the constraint that any loop passes at least through one delay element.

The symbols c, d, and e will be used to refer to the cardinality of the sets C, D, and E, respectively. The c computational nodes are: $c_1, c_2, \cdots, c_c$. The d delay elements are: $d_1, d_2, \cdots, d_d$.

III. The Computation of the Iteration Period Bound

In the formal definition of $T_{0_{\min}}$ (see (1)), all possible simple loops have to be considered. This does not lead to an efficient algorithm. Consider, e.g., DFG's that have at least one directed simple path consisting of zero or more computational nodes between every pair of delay elements. This type of DFG will be called *path complete* (see Fig. 1 for an example of a path-complete DFG). In path-complete DFG's, the minimal number of simple loops passing through k delay elements is $(d!/k(d-k)!)$. Hence the minimal number of distinct loops is

$$\sum_{k=1}^{d} \frac{d!}{k(d-k)!}.$$

Clearly, as the factorial function grows even faster than an exponential function, the number of loops cannot be expressed as a polynomial function of d. So, an algorithm based on an exhaustive enumeration of all simple loops cannot operate in polynomial time.

This paper presents a polynomial-time algorithm to compute $T_{0_{\min}}$. It consists of two parts.

1) The computation of the *first-order longest path matrix* $\boldsymbol{L}^1$, a modification of the *path matrix* introduced in [12].
2) The computation of $\boldsymbol{L}^k$, the longest path matrix of order k, for finding the loops that go through k delay elements.

3.1. The Longest Path Matrix of Order k

The longest path matrix of order $k (k \geq 1)$, $\boldsymbol{L}^k$, is a $d \times d$ matrix, and each of its elements l_{ij}^k contains the *length of the longest path* from delay element d_i to delay element d_j that passes through exactly $k-1$ delay elements. The length of a path is the

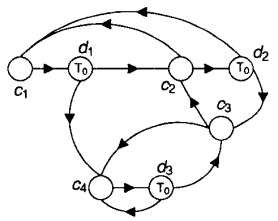

Fig. 1. Example of a path-complete DFG with three delay elements; inputs and outputs have not been included in the picture.

sum of the durations of the operations in the path. When such a directed path from d_i to d_j does not exist, l_{ij}^k is defined to be -1.

The diagonal elements of $\boldsymbol{L}^k$ represent the length of the longest paths starting and finishing at the same delay element and going through $k-1$ other delay elements. This means that they represent loops passing through k delay elements.

From now on, a path going through $k-1$ delay elements will be called a *path of order* k, since such a path is an element of the longest path matrix of order k. If the path is a loop, such a loop will be called a *loop of order* k. Note that for $k > 1$ a longest path of order k is not necessarily simple.

3.2. The Computation of the First-Order Longest Path Matrix

The first-order longest path matrix $\boldsymbol{L}^1$ can be computed by inspecting the DFG. An algorithm that performs this computation has been displayed in Figs. 2 and 3. Any vertex v is supposed to have a field v.max-length where some accounting can be done. The description is in pseudocode; the intention is to transfer the principal ideas instead of being precise.

In the main program, all delay elements are used one by one to compute the longest paths starting at them and finishing in any delay element. A preparatory step is to "inactivate" all edges in the DFG that cannot be reached from the concerned delay element. The longest paths are then found in a way similar to critical-path detection in acyclic graphs.

For each delay element, the edges in the DFG are made inactive only once, so the time complexity of the algorithm to compute $\boldsymbol{L}^1$ is $\boldsymbol{O}(de)$.

3.3. The Computation of Longest Path Matrices of Higher Order

Higher order longest path matrices can be obtained by the following recursive rule:

$$\boldsymbol{L}^{k+1} = \boldsymbol{L}^1\boldsymbol{L}^k,\ k > 0.$$

The rule implies an operation called lp-*multiplication* (longest-path matrix multiplication). This multiplication combines all first-order longest paths with all longest paths of order k, to get longest paths of order $k+1$.

The longest path of order $k+1$ from d_i to d_j can obviously be found by inspecting all first-order longest paths from d_i to all elements d_h together with all longest paths of order k from d_h to d_j, and selecting the combination through that d_h that gives rise to the longest total path.[1] This principle is presented in a more formal way in the algorithm of Fig. 4.

The diagonal elements l_{ii}^k of $\boldsymbol{L}^k$ give the loops passing through k delay elements. $T_{0_{\min}}^k$, the iteration period implied by loops of order

[1] This idea is essentially the same one as the matrix multiplication method mentioned in [13] for the *all-pairs shortest path* problem.

```
procedure inactive-labeling(v)
        for all e = (v, w)
        do "label e to be inactive";
           if "all edges incident to w are inactive and w ∉ D"
           then inactive-labeling(w)
           fi
        od
end of inactive-labeling;

procedure longest-path(v, start)
        for all e = (v, w)
        do "label e to be inactive";
           if v = start
           then if w.max-length < 0
                   then w.max-length := 0
                fi
           else if w.max-length < v.max-length + v.duration
                   then w.max-length := v.max-length + v.duration
                fi
           fi
           if "all edges incident to w are inactive"
           then if w ∉ D
                   then longest-path(w, start)
                fi
           fi
        od
end of longest-path;
```

Fig. 2. The algorithm to compute the longest path matrix. Part I: The procedures.

```
for i := 1 to d
do for all v ∈ (D \ {d_i}) ∪ I
      do inactive-labeling(v)
      od;
   d_i.max-length := 0;
   for all v ∈ (C ∪ (D \ {d_i})) ∪ O
   do v.max-length := -1
   od;
   longest-path(d_i, d_i);
   for j := 1 to d
   do l_ij := d_j.max-length
   od
   for all e ∈ E
   do "reset e's label to active"
   od
od
```

Fig. 3. The algorithm to compute the longest path matrix. Part II: The main program.

k, is defined as follows:

$$T_{0_{min}}^k = \frac{\max_{i\in\{1,2,\cdots,d\}} l_{ii}^k}{k}.$$

As the critical loop is a simple loop and therefore passes through at most d delay elements, $\boldsymbol{L}^d$ is the highest order longest path matrix that must be computed. Hence, the iteration period bound is the maximal value of all $T_{0_{\min}}^k$:

$$T_{0_{\min}} = \max_{k\in\{1,2,\cdots,d\}} T_{0_{\min}}^k.$$

The loops found by means of the longest path matrix of order k may or may not be simple; however, they are always the longest possible ones of the given order. It will be shown now that the participation of nonsimple loops in the computations does not lead to an incorrect value of $T_{0_{\min}}$.

Suppose that a loop of order k consists of two subloops of orders m and n: $m+n=k$. Let the length of the loop be l_k and the length of the subloops be l_m and l_n: $l_m + l_n = l_k$. The following inequality should be proved to be correct:

$$\frac{l_k}{k} \le \max\left(\frac{l_m}{m}, \frac{l_n}{n}\right).$$

Without loss of generality, it can be stated that $l_m/m \ge l_n/n$, or

$$ml_n \le nl_m. \tag{2}$$

Adding ml_m to both sides and undertaking some simple manipula-

```
for i := 1 to d
do for j := 1 to d
   do H := {1, 2, ..., d};
      for h := 1 to d
      do if l^1_ih = -1 or l^k_hj = -1
         then H := H \ {h}
         fi
      od;
      max-length := 1;
      for all h ∈ H
      do if l^1_ih + l^k_hj > max-length
         then max-length := l^1_ih + l^k_hj
         fi
      od;
      l^{k+1}_ij := max-length
   od
od
```

Fig. 4. The lp-multiplication algorithm.

tions shows the correctness of the inequality to be proved:

$$ml_m + ml_n \le ml_m + nl_m$$
$$m(l_m + l_n) \le (m + n)l_m$$
$$ml_k \le kl_m$$
$$\frac{l_k}{k} \le \frac{l_m}{m}.$$

In a similar way the result can be generalized for loops built from an arbitrary number of simple subloops. So, the algorithm computes $T_{0_{\min}}$ correctly.

3.4. Time Complexity

The presented method gives a polynomial-time algorithm for the computation of the critical loop. lp-multiplication for getting $\boldsymbol{L}^{k+1}$ from $\boldsymbol{L}^1$ and $\boldsymbol{L}^k$ requires $\boldsymbol{O}(d^3)$ steps; so computing $\boldsymbol{L}^d$ can be done in $\boldsymbol{O}(d^4)$ time. The time complexity for computing $\boldsymbol{L}$ was $\boldsymbol{O}(de)$, so the total time complexity of the critical loop detection algorithm becomes $\boldsymbol{O}(de + d^4)$.

3.5. An Example

In this subsection the algorithm of this paper will be illustrated using the DFG of Fig. 5 as an example. It represents an all-pole lattice filter.

The matrix $\boldsymbol{L}^1$ has the following value:

$$\boldsymbol{L}^1 = \begin{bmatrix} 8 & 12 & 16 & 16 \\ 4 & 8 & 12 & 12 \\ -1 & 4 & 8 & 8 \\ -1 & -1 & 4 & 4 \end{bmatrix}.$$

So, $T^1_{0_{\min}} = 8$ TU. $\boldsymbol{L}^2$ is computed below:

$$\boldsymbol{L}^2 = \begin{bmatrix} 8 & 12 & 16 & 16 \\ 4 & 8 & 12 & 12 \\ -1 & 4 & 8 & 8 \\ -1 & -1 & 4 & 4 \end{bmatrix} \begin{bmatrix} 8 & 12 & 16 & 16 \\ 4 & 8 & 12 & 12 \\ -1 & 4 & 8 & 8 \\ -1 & -1 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 16 & 20 & 24 & 24 \\ 12 & 16 & 20 & 20 \\ 8 & 12 & 16 & 16 \\ -1 & 8 & 12 & 12 \end{bmatrix}.$$

In a similar way, we find:

$$\boldsymbol{L}^3 = \begin{bmatrix} 24 & 28 & 32 & 32 \\ 20 & 24 & 28 & 28 \\ 16 & 20 & 24 & 24 \\ 12 & 16 & 20 & 20 \end{bmatrix} \boldsymbol{L}^4 = \begin{bmatrix} 32 & 36 & 40 & 40 \\ 28 & 32 & 36 & 36 \\ 24 & 28 & 32 & 32 \\ 20 & 24 & 28 & 28 \end{bmatrix}.$$

It follows:

$$T^2_{0_{\min}} = T^3_{0_{\min}} = T^4_{0_{\min}} = 8 \text{ TU}.$$

So, $T_{0_{\min}} = 8$ TU.

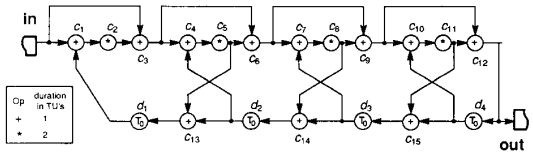

Fig. 5. The DFG of an all-pole lattice filter.

IV. Final Remarks

Formally, all directed loops of a DFG have to be investigated to compute the iteration period bound, $T_{0_{\min}}$. The number of loops cannot be expressed as a polynomial function of the number of nodes. This paper shows that $T_{0_{\min}}$ still can be computed in polynomial time.

The algorithm presented here has a time complexity of $\boldsymbol{O}(d^4 + de)$. A more sophisticated algorithm exists with time complexity $\boldsymbol{O}(d^3 \log d + de)$. This algorithm is obtained by adapting the algorithm for the *minimal cost-to-time ratio cycle* problem presented in [13] to the problem of computing $T_{0_{\min}}$ [12].

Acknowledgment

The authors would like to thank Prof. U. Faigle of the Faculty of Applied Mathematics of Twente University for pointing out to them that the minimal cost-to-time ratio cycle problem described in [13] is essentially the same problem as the problem of computing $T_{0_{\min}}$. The authors are also grateful to Dr. H. Alblas of the Faculty of Computer Science for proofreading the text and to R. Peer for discovering minor errors after implementing parts of the presented algorithm.

References

[1] 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.

[2] T. P. Barnwell and C. J. M. Hodges, "Optimal implementation of DSP algorithms on synchronous multiprocessors," in L. Sneyder, L. H. Jamieson, D. B. Gannon, and H. J. Siegel, Eds., *Algorithmically Specialized Parallel Computers*. New York: Academic, 1985, pp. 119-128.

[3] D. A. Schwartz and T. P. Barnwell, "Cyclo-static multiprocessor scheduling on the optimal realization on shift-invariant flow graphs," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 1384-1387, 1985.

[4] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, pp. 24-35, Jan. 1987.

[5] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, pp. 178-195, Feb. 1991.

[6] S. M. Heemstra de Groot and O. E. Herrmann, "Rate-optimal scheduling of recursive DSP algorithms based on the scheduling range chart," in *Proc. IEEE Int. Symp. Circuits and Systems*, pp. 1805-1808, May 1990.

[7] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE*, vol. 77, pp. 1879-1895, Dec. 1989.

[8] S. M. Heemstra de Groot and O. E. Herrmann, "Evaluation of some multiprocessor scheduling techniques of atomic operations for recursive DSP graphs," in *Proc. European Conf. Circuit Theory and Design*, pp. 400-404, Sept. 1989.

[9] A. Fettweis, "Realizability of digital filter networks," *Archiv für Elektronik und Übertragungstechnik*, vol. 30, pp. 90-96, 1976.

[10] M. Renfors and Y. Neuvo, "Fast multiprocessor realizations of digital filters," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, pp. 916-919, 1980.

[11] A. Gibbons, *Algorithmic Graph Theory*. Cambridge, UK: Cambridge University Press, 1985.
[12] S. M. Heemstra de Groot, "Scheduling techniques for iterative data-flow graphs, an approach based on the range chart," Ph.D. dissertation, Univ. Twente, Faculty of Electrical Engineering, Dec. 1990.
[13] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart and Winston, 1976.

A Simple Configuration for Realizing Voltage-Controlled Impedances

Raj Senani and D. R. Bhaskar

Abstract—**In earlier literature, op-amp JFET-based configurations have been advanced for the realization of voltage-controlled resistances (VCR's) having wide dynamic range and low distortion with linear range considerably extended than other FFT VCR's. In this paper an extremely simple configuration is proposed that achieves the same objectives with only two op-amps and four resistors along with a JFET. Moreover, the new proposition is more versatile than previously known configurations in that it facilitates realization of voltage-controlled inductance and voltage-controlled capacitance from the same circuit. Possible variations of the schemes suitable for CMOS implementation have also been suggested.**

I. Introduction

Voltage-controlled filters, voltage-controlled oscillators, and voltage-to-time period/frequency converters find applications in many instrumentation and measurement situations. A simple way to realize such circuits is to start from some known circuits and then to replace some resistor(s) by voltage-controlled resistor(s). Some schemes to realize such voltage-controlled-resistors using FET's are known in literature. Nay and Budak [1], [2] have presented a technique of obtaining a linear positive/negative voltage-controlled resistance (VCR) having wide dynamic range and low distortion with linear range considerably more extended than other FET VCR's. The configuration of [1] and [2] employs two op-amps (one connected as a noninverting amplifier, the other connected as a unity gain buffer (UGB)) and six resistors, along with a JFET.

Here, we present an alternative configuration, with similar properties, which is simpler than [1] and [2] and yet is more versatile than an NB-circuit in that it not only provides a VCR but can provide voltage-controlled inductance (VCL) and voltage-controlled capacitance (VCC) elements, too, which cannot be realized in the NB-circuits.

II. Proposed Configuration

The proposed configuration is shown in Fig. 1. A routine analysis of the circuit, when FET is confined to operate in nonsaturated region, under the same constraints as in [1] and [2], yields

$$R_{eq} = R_1\left[1 + \left\{V_p^2/I_{DSS}\left(V_C - 2V_p\right)R_2\right\}\right] \tag{1}$$

which represents a VCR whose value is controllable through V_c. Note that since we have made $V_{GS} = (V_c + V_{DS})/2$, substitution in

$$i_D = 2I_{DSS}V_{DS}\left\{V_{GS} - V_p - V_{DS}/2\right\}/V_p^2 \tag{2}$$

Manuscript received February 19, 1990; revised August 23, 1991. This paper was recommended by Associate Editor R. K. Hester.

R. Senani is with the Electronics and Communication Engineering Department, Delhi Institute of Technology, Kashmere Gate, Delhi-110006, India.
D. R. Bhaskar is with the Electrical Engineering Department, Delhi College of Engineering, Kashmere Gate, Delhi-110006, India.

IEEE Log Number 9104662.

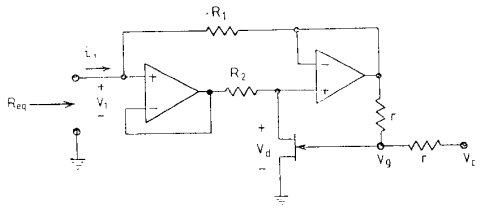

Fig. 1. The proposed VCR configuration.

results in the cancellation of the nonlinear term in the $V_{DS} - i_D$ characteristic of the FET, represented by (2). This, together with the potential divider action constituted by R_2 and resistance of the FET (due to which the FET has to handle a small fraction of the input voltage V_1) is responsible for the extension of the linear range for the input resistance realized by the proposed circuit.

III. Generalization

Fig. 2 shows the generalization of the proposed configuration from which one can realize VCL and VCC elements by appropriate choice (resistive/capacitive) of impedances Z_1 and Z_2. For this circuit

$$Z_{eq} = Z_1\left[1 + \left\{V_p^2/I_{DSS}Z_2\left(V_c - 2V_p\right)\right\}\right]. \tag{3}$$

Therefore, with $Z_1 = 1/sc_1$ and $Z_2 = R_2$, the circuit realizes a VCC with

$$C_{eq} = C_1/\left[1 + \left\{V_p^2/I_{DSS}\left(V_c - 2V_p\right)R_2\right\}\right]. \tag{4}$$

On the other hand, with $Z_1 = R_1$ and $Z_2 = 1/sC_2$, the circuit realizes[1] a VCL with associated L_{eq} and r_{eq} given by

$$L_{eq} = C_2R_1V_p^2/I_{DSS}\left(V_c - 2V_p\right) \tag{5a}$$

$$r_{eq} = R_1. \tag{5b}$$

The implementation of the voltage-tunable resistor Fig. 1 can be simplified by eliminating the input buffer in which case R_{eq} would be

$$R_{eq} = \frac{R_2 + r_{DS}}{\left(1 + \dfrac{R_2}{R_1}\right)} \cong R_1 + \frac{R_1r_{DS}}{R_2}, \quad \text{provided } \frac{R_2}{R_1} \gg 1 \tag{6a}$$

where

$$r_{DS} = V_p^2/I_{DSS}\left(V_c - 2V_p\right). \tag{6b}$$

An alternative simplification is to eliminate the input buffer together with resistor R_1 (or R_1 and R_2 both) in which case the resulting simplified voltage-tunable resistor reduces to the circuit of fig. 3 of [1].

A detailed nonideal analysis of the proposed configuration is presented in Appendix A, whereas a distortion analysis based upon a more general model equation for the FET is carried out in Appendix B.

IV. Experimental Results

The practically obtained *v-i* characteristics of the VCR of Fig. 1 using 741 type op-amps (biased with ±15 V), n-channel JFET

[1] It may be pointed out that with the FET replaced by a resistor and the two resistors connected at the gate of the FET eliminated, the circuit becomes similar to the lossy grounded inductance circuit of [3].