International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:10, 2007

Computing the Loop Bound in Iterative Data Flow Graphs Using Natural Token Flow

Ali Shatnawi

Abstract**—** Signal processing applications which are iterative in nature are best represented by data flow graphs (DFG). In these applications, the maximum sampling frequency is dependent on the topology of the DFG, the cyclic dependencies in particular. The determination of the iteration bound, which is the reciprocal of the maximum sampling frequency, is critical in the process of hardware implementation of signal processing applications. In this paper, a novel technique to compute the iteration bound is proposed. This technique is different from all previously proposed techniques, in the sense that it is based on the natural flow of tokens into the DFG rather than the topology of the graph. The proposed algorithm has lower run-time complexity than all known algorithms. The performance of the proposed algorithm is illustrated through analytical analysis of the time complexity, as well as through simulation of some benchmark problems.

Keywords**—** Data flow graph, Iteration period bound, Rate-optimal scheduling, Recursive DSP algorithms.

I. Introduction

The data flow graph (DFG) has proven to be a successful model for exhibiting the parallelism of algorithms. The dataflow graph model is represented by nodes and edges. Digital signal processing algorithms which are repetitive in nature are best represented by iterative data-flow graphs, where nodes represent computations and edges represent communication paths. Iterative applications have inherent parallelism among operations from consecutive iterations. The iteration is defined as the execution of the entire program to consume one input from each input line and produce one output on each output line. Operations from successive iterations can obviously be overlapped. A schedule that achieves the minimum iteration period is called rate-*optimal.* That is, it minimizes the average time between successive outputs, and thus achieves the highest possible throughput.

For all recursive DSP algorithms, there exists a fundamental lower bound on the iteration period, referred to as the *iteration bound* [1]. Recursive DSP algorithms are typically represented by cyclic DFGs. Determining the iteration bound for signal processing algorithms (described by iterative data-flow graphs) is a critical problem. This bound is fundamental to an algorithm and is independent of its implementation architecture. In other words, it is impossible to achieve an iteration period less than the bound, regardless of the number or power of the used processing elements. Thus, for DSP applications, it is a crucial design requirement to compute the iteration bound if a rate optimal implementation is sought. In this case, the maximum sampling rate of an algorithm running on any implementation is upper-bounded by the reciprocal of the iteration bound. In other words, to design a compile-time multiprocessor system, implementing an iterative digital signal processing algorithm, one has to compute this bound.

Several researchers have tackled the problem of finding the iteration bound [2]-[7]. A thorough analysis of these techniques and some others is found in [8]. Most researchers, who tackled the problem of finding the iteration bound, have focused on the graph structure or topology characteristics in their algorithms. Some of them, however, applied combinatorial techniques such as the path length analysis in their algorithms. In this paper, a completely different approach is being used in the proposed algorithm. The approach is based on the natural flow of data into the communication links of the data flow graph.

II. The DFG Model

The DFG is a directed graph *G=(V,E)*, which is uniquely represented by its node set *V(G)* and its edge set *E(G).* An edge $e = (v_I, v_T)$ is said to be incident out of its source node v_I and incident into its target node v_T. The source and target nodes of an edge are said to be its end nodes. The set of all edges incident into a node are said to be its incoming edge set (IE); whereas the set of all edges incident out of a node are its outgoing edge set (OE). The number of edges incident into a node are referred to as its indegree, and the number of outgoing edges is its outdegree [9].

The DFG considered in this paper is assumed to be a proper graph, that is, in the graph there is a path to every node from some input node, and a path from every node to some output node. A node that does not achieve this criterion is redundant, and has no contribution to the input/out behavior of the underlying algorithm.

III. Iteration Bound

For a cyclic DFG, the iteration bound is not only constrained by the hardware resources, but also by the

Manuscript received Feb 9, 2004
Ali Shatnawi is with the Jordan University of Science and Technology, Irbid 22110, Box 3030 JORDAN (phone: 962-79-666-9076; fax: 962-2-702-2440; e-mail: ali@just.edu.jo).

topology of the graph. If the hardware resources are unlimited (in fact, higher than a certain lower bound), the iteration period bound as constrained by the topology of the given graph is given by

$$T_0 = \max_{C_i \in C(G)} \left[\frac{D_C}{N_C} \right]$$

where D_C is the total computational time of all the nodes in the circuit *C*, N_C is the total number of ideal delays in the circuit *C*, and *C(G)* is the set of all circuits (loops) in *G*. D_C and N_C are, respectively, given by

$$D_C = \sum_{v_j \in V(C)} d^c_{v_j}$$

and

$$N_C = \sum_{e_j \in E(C)} n_{e_j}$$

$d^c_{v_j}$ denotes the computational delay of node v_i and n_{e_j} the ideal delay of edge e_i.

IV. The Algorithm

All previous iteration-finding algorithms focused on the DFG topology and the characteristics of the graph loops to compute the iteration bound. In contrast, the idea of the proposed algorithm depends on the natural flow of tokens to compute the natural limit imposed by the graph structure on the maximum attainable throughput. In other words, we will let the graph handle tokens as they become available at the input edges of the nodes. Then, based on the average time between successive outputs of each node, we will compute the value of the iteration bound.

The algorithm starts by performing a simple topological sorting procedure. In this procedure, which is pseudo coded in Figure 1, the indices of the nodes of the given graphs are sorted such that the index of a source of every edge is less than the index of its target. This is impossible to achieve in cyclic graphs; but the algorithm will find a reasonable topological sorting.

```
Topological-Sorting(G)
For each edge $e \in G$
    If Source(e).index > target(e).index
   Swap(source(e).index, target(e).index)
```

Figure 1: A simple topological sorting algorithm.

The second procedure in the algorithm is depicted in Figure 2. This procedure is responsible for removing, form the graph, all nodes that are not part of any loop. As these nodes are insignificant in the calculation of the iteration bound, removing them form the underlying graph will improve the performance of the algorithm in the average case. This procedure searches for a node whose indegree is zero. If such a node is found, it is removed from the graph by removing all edges outgoing of it. The same process is applied to any node whose outdegree is zero. In this case, all incoming edges to the node are also removed.

```
PureCyclic(G)

1. For each $v \in G$ set cyclic[v]=true
2. Pick a node v such that cyclic[v]=true. If there
   is no such node STOP.
3. If indegree(v)≠0 go to Step 5.
4. Remove from G all edges that are outgoing of
   the node v. That is, set E(G)=E(G)-OE(v). Set
   cyclic[v] =false. Go to Step 7.
5. If outdegree(v)≠0 go to Step 2.
6. Remove from G all edges that are incoming
   into node v. That is, set E(G)=E(G)-IE(v). Set
   cyclic[v] =false.
7. If no node remained with (cyclic[v]=true)
   AND (indegree(v)=0 OR outdegree(v)=0)
   STOP
8. Go to Step 2.
```

Figure 2: The algorithm to reduce the graph to a pure cyclic graph.

After refining the graph by applying topological sorting to it and removing all nodes which are not part of any loop, the main procedure to compute the iteration bound is applied. The pseudo code program representing this procedure is given in Figure 3, which is further detailed in the following paragraphs. However, self explanatory steps are left without discussion.

Step1: Initial tokens which are available at time zero are placed on the queues of the edges which have nonzero ideal delays. The number of tokens queued on each edge is exactly equal to the number of ideal delays of this edge. Destinations of edges whose ideal delay are greater than zero are more likely to be ready for firing as some of their inputs are already available. Thus, they are placed in a special queue called **LikelyReady**.

Step 3: Since iterations may overlap, we define an iteration index for each node separately.

Step 5: Get one of the likely to be ready nodes from the **LikelyReady** queue and remove it from the queue.

Step 8: Search for every node whose all incoming edges are loaded with tokens. Such a node is ready for firing. The earliest firing time of the ready node will be the latest of all token times located at the top of the queues of all incoming edges.

```
IterationBound(G)

  1. For each edge $e \in G$ where G is the cyclic
     version of the original:
        a. Define Queue Tokens(e)=Empty
        b. Tokens(e).append(0) as many times as
             $n_{e_j}$
        c. If $n_{e_j} > 0$
             LikelyReady.append(target(e))
  2. Set iteration_no =1
  3. For each $v \in G$ set iteration[v]=0
  4. If iteration_no > MAXITERATIONS go to
     Step 11.
  5. Set v = LikelyReadyQueue.Retrieve;
  6. Set Ready=true
  7. Set Latest= MINNUM
  8. While (Ready=true and there is an unexamined
     edge $e \in INCOMING(v)$)
        If Tokens(e) is not empty, update
             Latest=Max(Latest,
             Tokens(e).earliest_time)
        Else set Ready=false
  9. If (Ready=true)
     a.  dequeue one token from each edge
           $e \in INCOMING(v)$
     b.  set new_token_time= Latest + weight(v)
     c.  For each $e \in OUTGOING(v)$
           Tokens(e).append (new_token_time)
          LikelyReadyQueue.append(target(e))
     d.  Set iteration[v]=iteration[v]+1;
     e.  Set iteration_no=iteration[v];
     f.  If (iteration[v]=skips)
          skiptime[v]=new_token_time
     g.  Current_time(v)=new_token_time
  10.    Go to Step 4
  11.    For each cyclic node v set
  $T(v) = \frac{Current\_Time(v) - Skip\_time(v)}{iteration[v] - skips}$
  skips being the number of skipped iterations
  12. Set T= average of all T(v) computed in Step
      11.
```

Figure 3: The Main Algorithm to compute the iteration bound.

Step 9: Dequeue one token from each incoming edge and queue one token on each outgoing edge of a ready node found in the previous step. The token queued at the outgoing edges of a node has a time stamp that is equal to the sum of the earliest firing time of the node and its computational delay. Advance the iteration index of this node by unity, and adjust the iteration index accordingly.

Step 11: To minimize the transient behavior of the DFG, a pre-specified number of iterations are ignored upon computing the average iteration period. Thus, we compute the iteration period using the formula given in this step of the algorithm.

Step 12: Compute the iteration bound (T) as the average of iteration periods with respect to all participating nodes (nodes existing in loops and contributing to the input/output relationship).

V. Example

Consider the second order IIR filter shown in Figure 4. The * symbol denotes a multiplication operation and the + symbol represents an addition. The ideal delay of and edge is denoted by 1D. It is assumed that the multiplication operation requires 2 cycles and the addition 1 cycle. The IIR filter is redrawn in a data flow graph format as depicted in Figure 5. Applying the PurCyclic procedure to the DFG will result in the graph shown in Figure 6. To make the tracing of the algorithm simpler, the delay of a node is marked as a label. The diamonds on the edges of the DFG represent the data tokens. Each token is labeled with the cycle index at which the token has become available. The number of ideal delays associated with an edge reflects the number of tokens available at cycle 0, before the consumption of any input has taken place. The numbers in squares represent the number of outputs produced by the designated nodes. By iteratively applying the main procedure of the algorithm to the graph in Figure 6, the results of this application are depicted in Figure 8 to Figure 14. Table 1 shows the number of outputs produced by each node versus the cycle index. It is clear that after cycle 7, the transient behavior of the DFG has disappeared and one output is produced for each node exactly every 3 cycles. Thus, the algorithm will exactly compute the iteration bound if enough cycles are skipped. To give more insight about the operation of the algorithm, let us consider the transition, for example, from the DFG in Figure 7 to that in Figure 8. Node 1 can fire twice at the end of cycle 2 resulting in 2 tokens at its output at the end of cycle 3, as the node computational delay is 1 cycle. No other nodes can fire as their inputs are not all loaded with tokens.

VI. Time Complexity

Let in the given DFG, N be the number of nodes and M the number of edges. The topological sorting procedure visits each edge once and thus has a linear time complexity in terms of the number of edges. That is TC(**Topological-Sorting) = O(M).**

The **PureCyclic** has the following complexities. The first step requires N operations while the second and the third steps require constant time. The fourth step is repeated iteratively but no edge will be removed more than once. Hence, the time complexity of this step in all iteration will not exceed O(M). Step 5 requires constant time. Step 6 is similar to step 4 and thus requires O(M) operations. Steps 7 and 8 require constant Time. Thus the overall time complexity of the PureCyclic procedure is O(M).

The time complexity of IterationBound main algorithm is summarized in Table 2. The time complexity of the steps 4 to 10 combined for one iteration is given by

$$TC_{4-10} = O\left(\sum_{v_i \in V(G)} \left(ID(v_i) + OD(v_i) \right) \right) + O(N)$$

But $\sum_{v_i \in V(G)} ID(v_i) = \sum_{v_i \in V(G)} OD(v_i) = 2M$

Hence TC_{4-10} = O(M) + O(N). In a cyclic proper graph the number of edges is greater than or equal to the number of nodes. Hence, the time complexity of one iteration of the algorithm is *O(M)*. Since, the algorithm iterates for a pre-specified number of iterations, the overall time complexity of the algorithm is *O(NI×M)*, where NI is the number of iterations.

Simulation Results showed that the algorithm is extremely fast and requires only a small number of iterations to converge to the exact value of the iteration bound. Table 1 shows the computed iteration period versus the total number of iterations (NI) and number of skipped iterations (Skips) for each of the three indicated benchmark problems.

The benchmarks are the second order IIR filter with 8 nodes, the fifth order elliptic filter with 34 nodes, and the fourth order all-pole lattice filter with 15 nodes. Note that the exact values of the iteration bound for each or the three benchmarks are 3, 16, and 14, respectively.

Further, simulation results have shown that only a small number of iterations are required to converge to the exact value of the iteration bound.

VII. Conclusion

In this paper, a new approach for the formulation of the problem of finding the iteration bound has been presented. It has been shown that using the natural flow of tokens into the data flow graph (representing a recursive DSP algorithm), the iteration bound can be efficiently computed. The given DFG is initially subjected to topological sorting, followed by conversion to a pure cyclic graph. The individual iteration period for every node is computed, and the average over all cyclic nodes is obtained. It has been shown that each iteration of the algorithm application requires an order of O(M) operations. Simulation results has shown that only a few number of iterations are required to converge to the exact value of the iteration bound. With the very small number of iterations required, the time complexity of the proposed algorithm is superior to all previously developed algorithms.

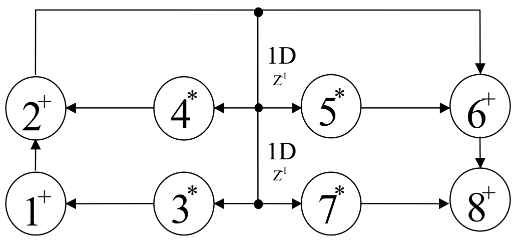

Figure 4: Second Order IIR Filter

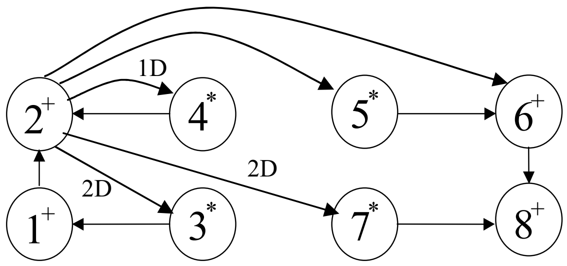

Figure 5: The DFG reprsentation of the 2nd order IIR filter

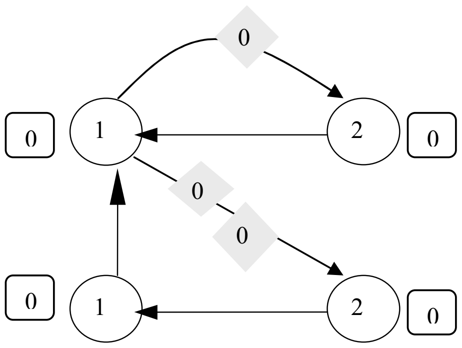

Figure 6: The cyclic version of the IIR filter

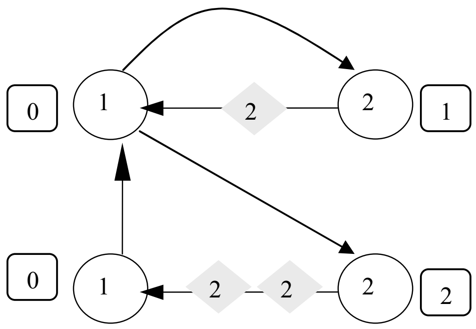

Figure 7: The filter after the end of cycle 2.

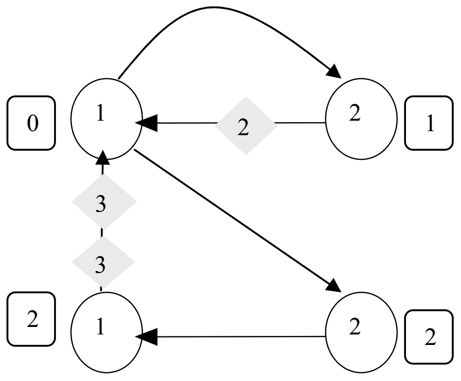

Figure 8: The filter after the end of cycle 3

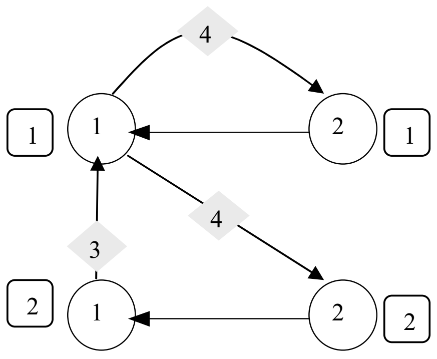

Figure 9: The filter after the end of cycle 4

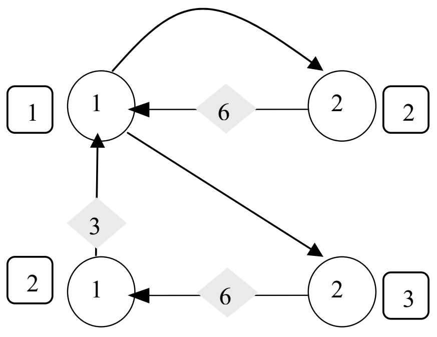

Figure 10: The filter after the end of cycle 6.

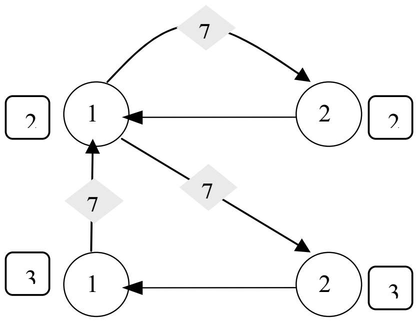

Figure 11: The filter after the end of cycle 7.

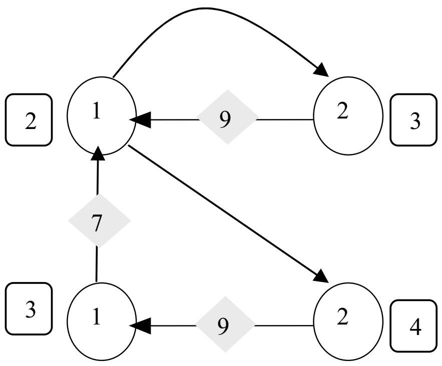

Figure 12: The filter after the end of cycle 9.

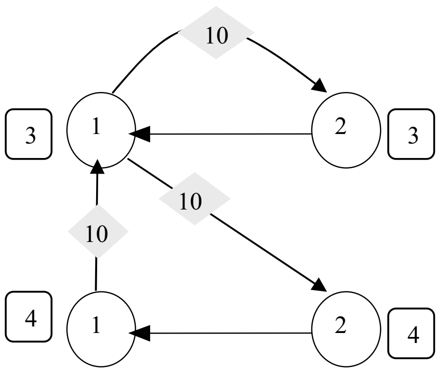

Figure 13: The filter after the end of cycle 10.

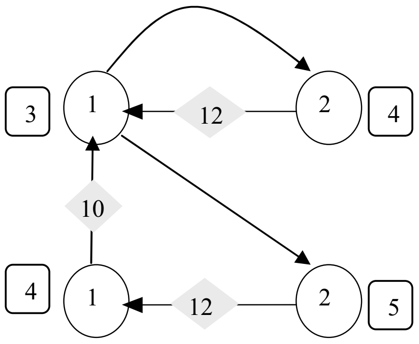

Figure 14: The filter at the end of cycle 12.

Table 1: The number of outputs produced by each cyclic node.

Cycle	Node1	Node 2	Node3	Node4
0	0	0	0	0
2			2	1
3	2			
4		1		
6			3	2
7	3	2		
9			4	3
10	4	3		
12			5	4

Table 2: The time complexity for each step in the IterationBound Algorithm.

Step	Time Complexity	Comments
1	O(M)	
2	Constant	
3	O(N)	
4	Constant	
5	Constant	
6	Constant	
7	Constant	
8	ID(vi)	In-Degree of the node vi
9	OD(vi)	Out-Degree of the node vi
10	Constant	
11	O(N)	
12	O(N)	

Table 3: The computed iteraton bound for different benchmarks versus the number of algoirthm iterations.

	2nd order IIR Filter	5th order elliptic filter	4th order all-pole lattice filter
NI=5 Skips=2	3.2	16	14
NI=10 Skips =2	3.06	16	14
NI=100 Skips =2	3.005	16	14
NI=1000 Skips =2	3.0005	16	14
NI=10 Skips=5	3	16	14

REFERENCES

[1] M. Renfors, and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," *IEEE Transactions on Circuits and Systems*, vol. CAS-28, no. 3, pp. 196-202, Mar. 1981

[2] 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.

[3] 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

[4] 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 System*s, Taipei, Taiwan, pp. 6A.1.1–6A.1.6, Dec. 1994.

[5] R. M. Karp, "A Characterization of the Minimum Cycle Mean in a Digraph," *Discrete Mathematic*s, vol. 23, pp. 309–311, 1978.

[6] R. Govindarajan and G. R. Gao, "A Novel Framework for Multi-Rate Scheduling in DSP Applications," in *Proc. 1993 Int. Conf. Application-Specific Array processor*s, pp. 77–88, IEEE Computer Society Press, 1993.

[7] K. Ito and K. K. Parhi," determining the minimum iteration period of an algorithm" *Journal of VLSI Signal Processing*, 11, (Dec.1995) 229–244.

[8] A. Dasdan, S. S. Irani and R. K. Gupta, "An Experimental Study of Minimum Mean Cycle Algorithms", UCI-ICS TR #98-32, UIUC, Urbana, USA.

[9] M. N. S. Swamy, and K. Thulasiraman, "Graphs, networks, and algorithms," John Wiley & Sons, Inc., New York, 1981.

Ali Shatnawi received the B.Sc and M.Sc in electrical and computer engineering from the Jordan University of Science and Technology in 1989 and 1992, respectively; and the Ph.D. degree in electrical and computer engineering from Concordia University, Canada, in 1996. He has been on the faculty of the Jordan University of science and Technology since 1996. He is presently on a leave and working as the dean of Information Technology as well as a president assistant at the Hashemite University, Jordan. His present research covers hardware design, high level synthesis of DSP applications, algorithms, and wireless networks.