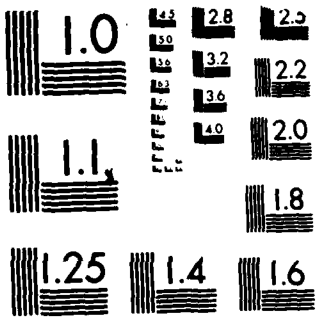

1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 84-0554
6a. NAME OF PERFORMING ORGANIZATION Georgia Institute of Technology 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research
6c. ADDRESS *(City, State and ZIP Code)* School of Electrical Engineering Atlanta GA · 30332	7b. ADDRESS *(City, State and ZIP Code)* Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION AFOSR 8b. OFFICE SYMBOL *(If applicable)* NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAG29-81-K-0024
8c. ADDRESS *(City, State and ZIP Code)* Bolling AFB DC 20332	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 61102F · PROJECT NO. 2304 · TASK NO. A9 · WORK UNIT NO.

11. TITLE *(Include Security Classification)* A GRAPH TECHIQUE FOR THE GENERATION OF SYSTOLIC IMPLEMENTATIONS FOR SHIFT-INVARIANT FLOW GRAPHS

12. PERSONAL AUTHOR(S)
D.A. Schwartz and T.P. Barnwell III

13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT *(Yr., Mo., Day)* 1984	15. PAGE COUNT 4

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	pp 1-4 1984

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This paper presents a general method for the transformation of algorithms described by shift-invariant fully-specified flow graphs into equivalent systolic realizations. The method consists of a set of rules for the systematic manipulation of the flow graphs into systolic form utilizing a set of theorems from graph theory. It is shown that many of the previously published systolic algorithms and many new algorithms can be generated using this single procedure.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Joseph Bram	22b. TELEPHONE NUMBER *(Include Area Code)* (202) 767- 4939 22c. OFFICE SYMBOL NM

1/4

To be presented at the International Conference on Acoustics, Speech, and Signal Processing.

A GRAPH THEORETIC TECHNIQUE FOR THE GENERATION OF SYSTOLIC IMPLEMENTATIONS FOR SHIFT-INVARIANT FLOW GRAPHS

D. A. Schwartz and T. P. Barnwell III

School of Electrical Engineering
Georgia Institute of Technology
Atlanta, GA 30332

A-1

Abstract

This paper presents a general method for the transformation of algorithms described by shift-invariant fully-specified flow graphs into equivalent systolic realizations. The method consists of a set of rules for the systematic manipulation of the flow graphs into systolic form utilizing a set of theorems from graph theory. It is shown that many of the previously published systolic algorithms and many new algorithms can be generated using this single procedure.

Introduction

The fundamental goal of this research is to develop methods for the automatic and optimal *realization of a large class of Digital Signal Processing* (DSP) algorithms on synchronous multiprocessors composed of multiple, identical programmable processors. This research seeks to find the most efficient possible solutions, in which the intrinsic synchrony of the system maintains the data precedence relations, and in which no cycles of any of the processors are used for system control (1). DSP algorithms, as a class, are uniquely well suited to this approach both because of their computational intensity and because of their high level of internal structure.

One of the popular methods which might well be used to accomplish this goal is the application of systolic arrays to DSP implementations. Over the past several years, there has been considerable interest in systolic implementations, and a number of systolic algorithms have been published. For the most part, these algorithms have not been derived by any formal method, but rather have been developed and presented separately, sometimes without extensive verification. The purpose of this paper is to present a simply applied set of techniques which allow for the systematic derivations of systolic solutions for a large and interesting class of algorithms.

Flow Graph Representation

In this research, the algorithms to be implemented are all described using fully-specified flow graph representations. As is illustrated in Fig. 1, a fully-specified flow graph is a directed graph in which all operations occur at the nodes, and in which the branches are used exclusively as signal paths. In addition, the graph is constrained so that its node operations are each fundamental operations of the constituent processors which are to be used in the in the implementation. More precisely, the node operations represent the granularity with which the parallelism of the flow graph can be manipulated, and they should be chosen accordingly. The fully-specified flow graph is a very powerful representation which, if properly applied, is not only capable of describing such traditional signal flow graph structures as digital filters and fast transforms, but also such nonlinear structures as those involving decimation, interpolation, homomorphic processing, and a large class of matrix operations. In addition, by allowing the nodes to be low level logic operations, these flow graphs can also describe bit-serial, byte-serial, and many other distributed arithmetic structures.

Flow Graph Bounds

Given that only one processor type is to be used in the eventual multiprocessor implementation and given that the characteristics of this constituent processor are known, then it is possible to compute bounds on the synchronous multiprocessor realization of a fully specified flow graph. Two bounds are of particular interest. The first bound, called the sample period bound, involves the minimum sampling period at which a particular algorithm can be implemented using a particular constituent processor. The sample period bound is best understood in the context of a recursive single-time-index flow graph (such as an IIR digital filter), although the concept is also meaningful in systems which have no explicit sampling period. For such systems, the sample period bound is given by

$$T_x = \underset{p}{\mathrm{MAX}}[\, d_p/n_p \,]$$

where p varies over the set of all loops in the flow graph, d_p is the arithmetic delay in the loop p and n_p is the number of unit delays nodes in loop p. This result is a generalization of a result published by Renfors and Nuevo (2).

The second bound, called the static-pipeline sample period bound, is of particular interest in the derivation of systolic implementations. This bound is the minimum sampling period which can be achieved if the entire graph is implemented using a static pipeline. A static pipeline is an implementation in which the node operations of the graph are explicitly assigned to individual processors, and in which every applications of any particular node operation is always performed by the same processor. A static pipeline realization for a flow graph can be considered to be a complete partitioning of the flow graph in space, in which each node in the flow graph is assigned to a particular processor. This is to be contrasted with cyclo-static implementations such as SSIMD (1) in which different time-index (or space-index) applications of a particular node operation may be realized by different processors at different times. In general, cyclo-static implementations may achieve the sample-period bound while static-pipeline implementations can only achieve the static pipeline sample period bound.

Like the sampling period bound, a static pipeline sample period bound is first computed for each loop, and then the overall bound for the graph is computed as the maximun of the individual loop bounds. Each loop individually can be thought of as consisting of a set of operation nodes and a set of delay nodes. The static pipeline bound for a loop is computed by assuming that the delays elements may be distributed throughout the loop in any desired configuration and by finding that distribution of delays which minimizes the maximum operation time between two consecutive delay elements. This mini-max operation time is the static-pipeline sample period bound for the loop and the maximun value of all such loops bounds is the static-pipeline sample period bound for the flow graph, and can be written as

$$\tau_p = \underset{\ell \in \text{loops}}{\text{Max}} \{D_\ell\}.$$

$$D_\ell = \underset{\substack{\text{all } n_\ell \\ \text{partitions}}}{\text{Min}} \Big[\underset{\substack{\text{partitions} \\ i \in \{0,\ldots,n_\ell\}}}{\text{Max}} \big(\sum_{j \in i} d_i\big)\Big]$$

Optimality

This work makes use of two separate definitions of optimality. An implementation is said to be rate optimal if it achieves the sampling period bound and is said to be processor optimal if it exhibits perfect processor efficiency so that every cycle of every processor is used directly on the fundamental operations of the algorithm and no cycles are used for synchronization or system control. Clearly, these three definitions of optimality are non-exclusive, and any particular implementation may satisfy any combination of these optimality criteria.

Rigorous Systolic Derivations

Two single-time-index systems are said to be essentially equivalent if given the same input sequences, they always give the same output sequences. The systolic derivation procedure is based on two theorems concerning the essential equivalence of systems described by flow graphs.

THE DATA INTERLEAVE THEOREM: A set of N identical shift invariant systems operating on N separate data streams is essentially equivalent to a single system for which the N sets of inputs and outputs have been separately interleaved as ordered sets and the order of all the delay nodes in the flow graph has been multiplied by N.

COROLLARY: A shift invariant system is always essentially equivalent to a shift invariant system where the input has been up-sampled by N, the output has been down-sampled by N, and the order of all the delay nodes has been multiplied by N.

A Nodal Cutset is defined as that set of branches which are cut when a closed surface is constructed inside a flow graph in such a way that it passes through no nodes.

THE CUTSET DELAY TRANSFORMATION THEOREM: Any shift invariant flow graph is essentially equivalent to a flow graph which is formed by adding ideal delay (advance) nodes to all the input branches in a nodal cutset and adding ideal advance (delay) nodes to all the output branches in the same nodal cutset.

A fundamental constraint placed on systolic arrays in their definition (3) is that the transfer of data between cells must be simultaneous. This translates into a flow graph constraint that every output branch from a cell must be terminated by a delay node (pipeline register). Hence, the generation of systolic solutions for flow graphs reduces to the distribution of the delay nodes throughout the flow graph so that this condition is met. In this procedure, the static pipeline sample period bounds for the individual loops in the flow graph are used to determine where the delay nodes should be redistributed, the required interleaving factor, and the appropriate nodal cutsets. If the sample period is known, it is always simple to introduce delay nodes into the nonrecursive portions of the graph such that the maximum delay between pipeline registers is less than or equal to the sample period.

The way in which these theorems are used to derive a systolic implementation from a fully-specified flow graph is illustrated in Fig. 1. Other examples of the derivation of systolic implementations from flow graphs are given in (1). In particular, Fig. 1 illustrates the generation of the systolic implementation for the two multiplier Markel-Gray lattice filter. This example has been chosen for four specific reasons. First, and most important, it was chosen because it uses all of the theorems and exhibits all of the properties which are typical of the derivation of systolic implementations from flow graphs. Second, it illustrates clearly the central roll of the Data Interleave Theorem in finding systolic derivations for recursive systems. Third, in spite of being typical in the ways the theorems are applied, this particular derivation itself has some additional interesting and surprising features which are worthy of note. Finally, the systolic implementations presented here for this important digital filter structure have not been presented before, and are interesting in their own right.

From Fig. 1a, it is clear that the sample period bound for this form of the lattice filter is given by $T_s = t_m+2t_a$, where t_m and t_a are the multiply and add times. Also, since this filter is recursive, it is clear that there must be a bi-directional data flow between the systolic cells. The problem is that, in its original form, there are not enough delays nodes inside the loops to meet the systolic requirement for positioning delay elements at all output branches (it should be obvious here that the Cutset Delay Transformation Theorem will always conserve the number of delay elements in a loop, although it will allow them to be easily redistributed). As a result, an interleaving transformation is required to generate the needed delay nodes. Fig. 1b shows the filter after a 2-way interleaving tra nsformation. Computing the static-pipeline bound for this flow graph results in a sampling period of $T_p = t_m+2t_a$. However, for this case, this is not an achievable bound since the lattice filter has coupled, overlapping loops. It is hence necessary to arrange the delays in each loop so they do not conflict with the delay requirements of other loops. With this added requirement, the achievable pipeline sample period bound, T_a , is given by $T_a = 2t_m+2t_a$. With knowledge of the sample period bounds, the cutsets are easily constructed as shown in Fig. 1b. The resulting network is shown in Fig. 1c. This network is now in systolic form. The corresponding systolic cell interconnection details are shown in Fig. 1d, where the last cell (Type II) is a degenerate form of the other cells (Type I). When the 2-way interleaving is considered, this implementation can achieve a sampling period of $T_s = 4t_m+4t_a$. This is well short of the sample period bound, and when no second signal is available to be interleaved, this represents a 50% processor efficiency.

Another important point is illustrated in Fig 1e. The point is that there exists a systolic form which has a smaller sample period than the system of Fig. 1c. This lower sampling period is achieved by applying a 3-way interleaved transformation, which, after the appropriate cutset transformations have been applied, result in the systolic network of Fig. 1e. In Fig. 1e, the dashed lines partition the network into systolic cells. This new network has a static-pipeline sample period bound of $T_a = T_p = t_m+t_a$. With the interleaving considered, this gives an achievable sampling period of $T_s = 3t_m+3t_a$. While this is 4/3 times faster than the 2-way interleaved form, it still does not achieve the optimal sample period bound, so it is not rate-optimal. For this particular network, the optimal sample period bound cannot be achieved with a systolic implementation, nor will any higher interleaving factor result in a faster processing rate. However, for any given network it is clear that there is an easily determined optimum interleaving factor that achieves the minimum possible static-pipeline sample period bound, and for some networks, this is equal to the sample period bound.

Optimal Pipeline Solutions

This paper has described a general procedure for the derivations of systolic implementations from fully-specified flow graphs. Such derivations are rigorous in the sense that each step is achieved by the application of a theorem from graph theory to the flow graph, and so long as the theorems are properly applied, the resulting implementation is guaranteed to be correct. This method is simple to apply (in fact, it can be largely automated), and should be of great utility in the study of systolic algorithms. It also serves to gain further insight into the fundamental nature of systolic algorithms. For example, the 50% efficiency so often found in systolic implementations can be seen to be a result of the required 2-way interleave transformation for recursive systems (1/N efficiency for N-way interleaved system with one data stream). However, a point which is made clear in this context which is not clear from the original systolic presentations is that 100% efficiency can easily be obtained if two independent data streams are available to be processed simultaneously.

The real problem is that traditional systolic approaches do not necessarily lead to the best synchronous multiprocessor implementations. A fundamental reason for this is that the basic systolic definition requires that all of the data transfers between cells must be done simultaneously. This can lead to both very low processor efficiency and a low achievable sampling rate. It is simple to understand why this is true. The global systolic clock leads to a succession of information wavefronts separated by one global clock period. The global clock period clearly must be greater than or equal to the largest cell processing delay in the system and also, as illustrated above, recursive systems require interleaving transformations which lead to the introduction of extra, "padding" wavefronts. Therefore if all the cell processing delays are not of equal duration or if padding wavefronts are present, then the processors are not always doing useful work. If the global systolic clock constraint is relaxed, then the information wavefronts may be separated by less than the maximum cell processing delay, and much more efficient implementations are possible. SSIMD is a perfect illustration of this effect (1). In SSIMD, the processing delay of each cell is typically quite long (since each cell generally includes all the operations in one iteration of the flow graph), but the information wavefronts are separated by only a small fraction of a cell processing delay. This extra flexibility is one reason why SSIMD and other cyclo-static implementations can achieve processor optimal and rate optimal solutions when systolic implementations for the same flow graph cannot (1).

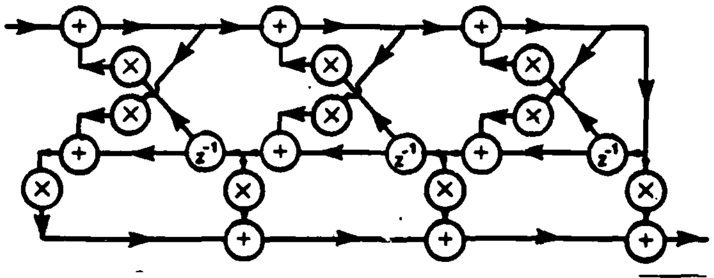
REFERENCES

1. T. P. Barnwell III and D. A. Schwatrz, "Optimal Implementations of Flow Graphs on Synchronous Multiprocessors," Proc. 1983 Asilomar Conf. on Circuits and Sy., Pacific Grove, CA, November 1983.

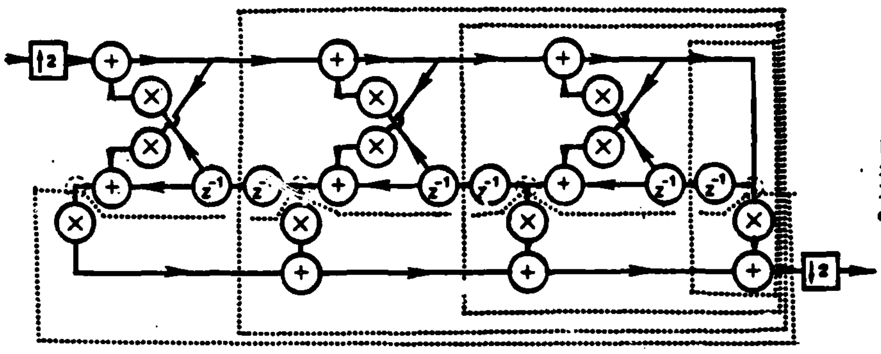
2. M. Renfors and Yrjo Nuevo, "The Maximum Sampling Rate of Digital Filters Under Hardware Constraints," IEEE Transaction on Circuits and Systems, T-CAS, pp. 196-202, March 1981.

3. C. E. Leiserson, Area Efficient VLSI Computation, MIT Press, 1983.

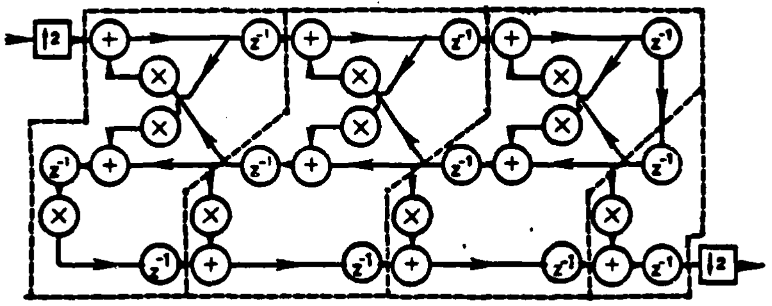
Fig. 1



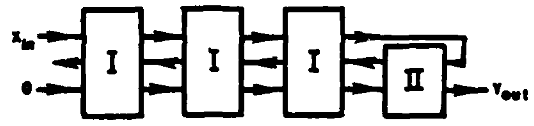
a) Fully specified signal flow graph of a third order Markel-Gray lattice filter.



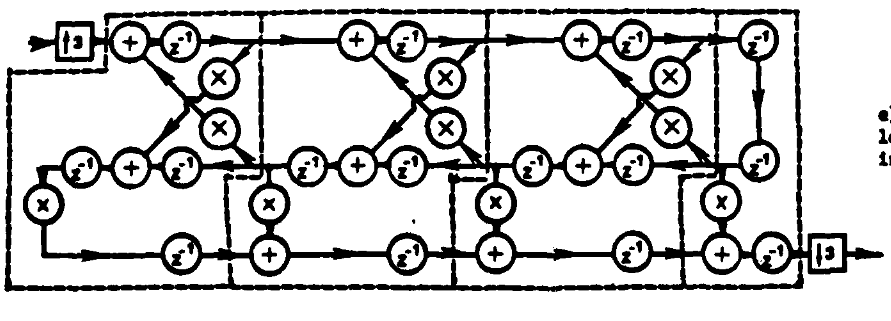
b) Flow graph after a 2-way interleaved transformation. Dotted lines indicate desired cutsets for delay transformation.



c) Systolic form of 2-way interleaved lattice filter, after cutset delay transformations. Dashed lines indicate systolic cell partitions.



d) Detail of systolic cell interconnections.



e) Systolic form of 3-way interleaved lattice filter. Dashed lines indicate systolic cell partions.

DATE
ILME