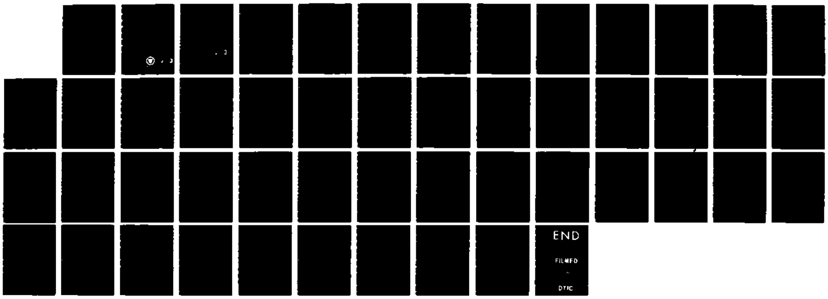
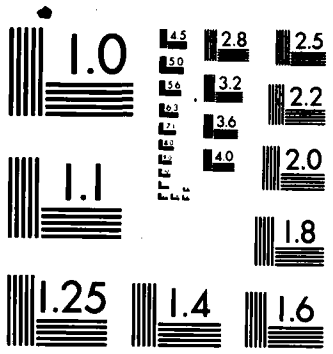
AD-A161 972 THE APPLICATION OF A SEQUENCE NOTATION TO THE DESIGN OF SYSTOLIC COMPUTATIONS(U) PITTSBURGH UNIV PA INST FOR COMPUTATIONAL MATHEMATICS AND APP R MELHEM OCT 85 1/1

UNCLASSIFIED ICMA-85-87 N00014-85-K-0339 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(14)

AD-A161 972

INSTITUTE FOR COMPUTATIONAL MATHEMATICS AND APPLICATIONS

Technical Report ICMA-85-87 October 1985

THE APPLICATION OF A SEQUENCE NOTATION TO THE DESIGN OF SYSTOLIC COMPUTATIONS*)

by

Rami Melhem**)

Department of Mathematics and Statistics

University of Pittsburgh

Technical Report ICMA-85-87 October 1985

THE APPLICATION OF A SEQUENCE NOTATION TO THE DESIGN OF SYSTOLIC COMPUTATIONS*)

by

Rami Melhem**)

Department of Mathematics and Statistics
University of Pittsburgh
Pittsburgh, PA 15260

*) This work was in part supported under ONR contract N00014-85-K-0339.

**) On leave from the Department of Computer Science, Purdue University, West Lafayette, IN 47907.

A

ABSTRACT

The sequence notation suggested in ~~[14]~~ an earlier work provides a tool for the clear and precise specification of systolic computations. Namely, it separates the static and dynamic levels of the specification. At the static level, the topology of the network and the function of each cell are described by a system of causal equations on sequences, and at the dynamic level, the data flow is described by the elements of the individual sequences.

~~In~~ this paper, ~~we~~ describes a method for the transformation of a given algorithm into a system of causal sequence equations/input-output description which specifies a systolic computation. The basic idea of the method is to pack arrays of variables along one or more dimensions into sequences. Doing this, however, may result in a system of equations that is not causal, and hence, a transformation of indices in the original algorithm may be essential in order to guarantee causality (the positive increment of time).

The derivation of index transformations from the data dependence vectors of an algorithm was discussed in the literature. However, data dependence vectors do not carry any information about absolute values of the indices, and hence, allow only the derivation of linear transformations. In order to overcome this problem, we suggest a method for the derivation of the index transformation from <used,defined> pairs. These pairs retain information about the absolute values of the indices, and thus allow for non linear transformations.

Although the model of [14] allows arbitrary interconnections in systolic networks, our design technique is restricted to the class of networks in which the interconnection pattern may be non-linear only along specific directions. Ring-like networks are elements of this class.

1. INTRODUCTION

In the past few years, many formal techniques have been suggested for the design of VLSI computations, in general, and of systolic computations, in particular. These techniques include the systematic mapping of wavefront-like computations into hardware (e.g. [5,7]), the derivation of alternative systolic networks from a given, provably correct, design (e.g. [8,9,10]), and the reindexing of the variables in a given algorithm such that the dependence between the variables suit VLSI implementation. This latter technique was first suggested by Kuhn [6], and later studied carefully by Moldovan et al. [16], Miranker et al. [15], and Quinton et al [17]. Cappello et al. [1] also conceived this reindexing from a geometric point of view and Ipsen et al. [3] extended the idea to include the data dependence between coupled systems. Other techniques was also suggested for the search of an optimal systolic network in a restricted class of networks [11], and for the mapping of an acyclic program graph into a linear array [18].

Of the above techniques, reindexing seems to be the most promising and general one for mapping a given computation into a systolic implementation. It is described briefly as follows: First, the computation is written in the form of an algorithm consisting of nested loops or recurrence formulas. Each variable in the algorithm should be an element of an n+1 dimensional array, for some $n \geq 1$, and hence may be associated with a position in an n+1 dimensional space that we call here the "computation space". In this space, the "Dependence Vector" of a data item may be defined as the vector joining the positions at which the item is defined and used. One of the dimensions in the computation space is chosen to represent the "Time", and a specific space transformation is derived such that all the dependence vectors are mapped into new vectors that have positive components along the time dimension. The interconnection pattern of a network that may implement the given computation, and the speed of the data movement in the network are then determined by the components of the transformed dependence vectors.

The derivation of the space transformation from the dependence vector excludes any transformation that depends on the absolute position of the data in the computation space (called nonlinear transformations in [16]). In order to overcome this deficiency, Chen [2] suggested a technique in which the space transformation is accomplished through a point by point mapping. In addition, the Chen technique carries along the entire algorithm (first order recursive equations) during the design process, yielding a precise and complete specification of the systolic computation. This is a clear advantage over the previous reindexing techniques, where the specification of each cell and the description of the input have to be sought separately through a repeated application of the linear transformation to different points in the computation space.

In this paper, we present a technique that is based on the formal model of [14]. It is a reindexing technique in which the space transformation is derived from <defined,used> pairs of the data items instead of the dependence vectors. This allows transformations that are position dependent (non linear) and yet avoids the point by point mapping of the space.

As in [2], our technique carries along the entire description of the computation during the design process. More specifically, given a canonical algorithm, where each data item is associated with a position in the computation space, the data items along the "time" dimension(s) are compacted into data sequences. A sequence transformation is then applied to enforce "causality", a condition that ensures the positive increment of time. The resulting system of causal equations specifies precisely the topology of the network, as well as the operation of each cell and the description of the appropriate inputs.

The formal model [14] that supports our technique does not put any restriction on the topology of systolic networks. This allows the derivation of a wide range of systolic computations that may not be derived by any technique that imposes the condition of local communications at the algorithmic level (e.g. [2]). For example, the shortest path multistage network, derived in Section 6, may only be implemented on a network with global feed back. That is a ring-like architecture

Another advantage of the technique presented in this paper is the natural translation of multi-time dimensions into multistage networks. For example, if two dimensions of the design space are associated with time, then data items along these two dimensions may be easily packed into data sequences. The resulting system of equations then describes a multistage network where a coarse clock determines the beginning and end of each phase, and a fine clocks determines the cycles within each phase.

In the next section, we introduce systems of Causal Canonical Sequence equations CCS, and we show that any CCS specifies a systolic computation. In the following three sections, we describe the different steps involved in the transformation of a given algorithm into a CCS. These steps are illustrated by an example of a computation for the solution of banded, triangular linear systems. The multistage network derived in Section 6 shows the capability of the technique to handle multi-time dimensions and global feed-back loops, and the dynamic programming network of Section 7 is an example where non-linear sequence transformation may be applied.

2. Canonic Systems of Causal Equations

A systolic network is defined in [14] to be a network of cells (computational and I/O) where each communication link is unidirectional and each computational cell repeats indefinitely the execution of a specific cycle of the form: 1) Read data from the input links, 2) perform a specific computation, and 3) write the results on the output links. The initiation of the cycles in the different cells is synchronized by a global clock.

With this definition, any computation on a given systolic network N may be precisely specified as follows:

1) Assign to each cell in N a unique label $\ell \in I^n$, where I^n is the set of n-tuples of integers. If N is a linear or a two dimensional array, then the usual choice of n is 1 and 2, respectively.

2) Identify each link in N by a pair $\langle y,\ell\rangle$ (written as y_ℓ), where ℓ is the label of the cell at which the link terminates and y is a color assigned to the link. The only restriction on link colors is that links terminating at the same cell should have different colors. In this paper, links that are directed from a cell to itself will be allowed. This type of direct feed back may be used to store information from one cycle to the next, and thus models an internal register in the cell.

3) Associate with each link y_ℓ a data sequence η_ℓ (η is the greek letter corresponding to y). The ith element of η_ℓ, namely $\eta_\ell(i)$, is the data item that appears on y_ℓ at the beginning of cycle i. A special item 'δ' is used to indicate a "don't know" or a

"don't care" element.

4) For each computational cell v in N, specify the operation of v by a set E_v that contains one sequence equation for each output link of v. More specifically, if y_u is an output link of v, then include in E_v an equation of the form

$$\eta_u = \Gamma_v^u(\alpha_v, \beta_v, \gamma_v, \ldots) \tag{1}$$

where a_v, b_v, $c_v, \ldots,$ are input links to v, and Γ_v^u is a causal sequence operator that specifies, for any time t, the output item $\eta_u(t)$ in terms of the previous input items $\alpha_v(\tau)$, $\beta_v(\tau), \ldots,$ $\tau < t$. Many sequence operators are defined in [12] and [14]. In the appendix, we define the few operators that will be used in the examples of this paper.

5) Specify the elements of the sequences associated with the input links of the network (the output links of input cells).

6) Identify the output data items.

The system of equations obtained in 4, in addition to the input and output specifications described in 5 and 6, respectively, specify completely the systolic computation. It may be easily seen that this system of equations/input-output specifications satisfies the following conditions:

CS1: Each sequence in the system is indexed by a label $\ell \in I^n$, for a fixed n. Moreover, all the sequences that appear in the right side of any specific equation are indexed by the same label. (v in equation (1)).

CS2: The system is well defined and consistent. In other words, any sequence that appears in the system is defined exactly once, either in the input specification or as the left side of a sequence equation.

<u>Definition 1:</u> An equation of the form (1), and the associated operator Γ_u^v are called causal if, for any t, $t \geq 1$, $\eta_u(t)$ does not depend on any element $\alpha_v(\tau)$, $\beta_v(\tau)$, ..., for some $\tau \geq t$. □

<u>Definition 2:</u> A system of equations/input-output specifications is called canonic if it satisfies the above two conditions. If, in addition, each sequence equation in the system is causal, then the system is called a causal canonic system, denoted from now on by CCS. □

<u>Proposition 1:</u> Any CCS specifies a systolic computation.

<u>Proof:</u> We will obtain the systolic computation specified by the given CCS by constructing the underlying systolic network N as follows:

Let L be the set that contains all the indices of the sequences that appear in the CCS and construct for each index $v \in L$ a cell labeled v. Partition the equations in CCS into mutually exclusive sets of equations, where each set E_v contains the equations whose right side sequences are indexed by the index v. Now, consider each set E_v; By CS1, each equation in E_v has the form (1). For each such equation, construct a link directed from cell v to cell u. Finally, for each sequence η_i specified in the input specification part of the CCS, construct an input cell and a link directed from that cell to cell i. The label of the input cell may be assigned arbitrarily.

Given the topology of N, the operation of each cell v in N is then described by the equations in E_v. Condition CS2 guarantees that the input to each cell is an output of a cell in N (possibly an input cell), and that the output of each cell is uniquely defined. □

<u>Example:</u> Let $A=\{a_{i,j}; i=1,\ldots n,\ j=i-m,\ldots,i\}$ be a band lower triangular matrix, and let the vectors $b=\{b_i; i=1,\ldots n\}$ and $x=\{x_i; i=1,\ldots,n\}$ satisfy $Ax=b$ (see ALG1 in Sec. 3). Consider the following CCS:

INPUT{ $\bar{\gamma}_j = \Omega^j \ominus \gamma_j$, $j=1,\ldots,m+1$; $\bar{\beta}_{m+1} = \Omega^{m+1} \ominus \beta_{m+1}$; $\bar{\xi}_1 = \iota$;

where for $t=1,\ldots,n$,

$\gamma_j(t) = a_{t,t+j-m-1}$, $\beta_{m+1}(t) = b_t$, and $\iota(t) = 0$ };

$$\bar{\xi}_{j+1} = \Omega\,[\bar{\xi}_j + \bar{\gamma}_j * \bar{\zeta}_j\,] \qquad j=1,\ldots,m \qquad (2.a)$$

$$\bar{\zeta}_{j-1} = \Omega_0^j\, \Omega^{-j+1}\, \bar{\zeta}_j \qquad j=1,\ldots,m \qquad (2.b)$$

$$\bar{\zeta}_{j-1} = \Omega_0^j\, \Omega^{-j+1}\, [[\bar{\beta}_j - \bar{\xi}_j] / \bar{\gamma}_j\,] \qquad j=m+1 \qquad (2.c)$$

OUTPUT{ $x_i = \bar{\zeta}_m(m+1+2i)$; $i=1,\ldots,n$ }.

For the above CCS we have $L=\{1,\ldots,m+1\}$, E_i = { equ's (2.a/b) } for $i=1,\ldots,m$, and E_{m+1} = { equ (2.c) }. The corresponding network is shown in Fig. 1, where input/output cells are omitted. Note that the terms $\Omega_0^j\, \Omega^{-j+1}$ in (2.b/c) indicate that the first j elements on the link $\bar{z}_{j-1}$ should be forced to zero. This is important because it saves the values on the $\bar{x}$ links from destruction due to operations involving don't cares. By (2.a/b), cells $1,\ldots,m$ are multiply/add cells and cell m+1 is a subtract/divide cell. By the definition of $\bar{\gamma}_j$, the elements of the $(j-m-1)^{st}$ sub diagonal of A are supplied on the link $\bar{c}_j$ starting at time j+1 and separated from each other by one time unit. The inputs on $\bar{b}_{m+1}$ and $\bar{x}_1$, as well as the outputs on $\bar{x}_m$, are also specified precisely in the CCS. □

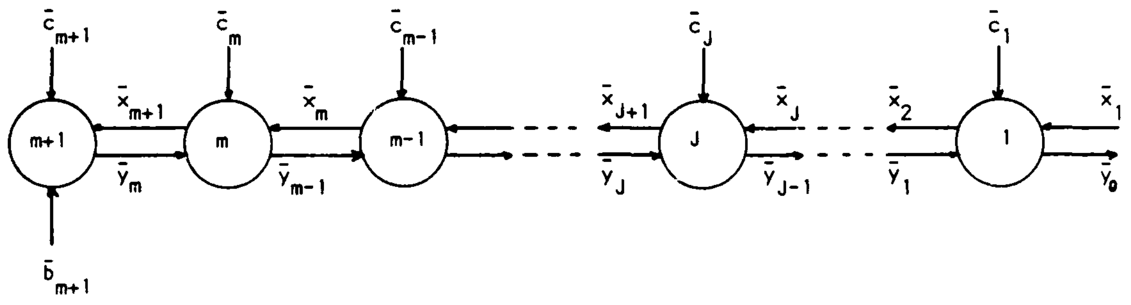


Fig. 1 - A network for forward substitution.

Hence, by Proposition 1, the task of designing a systolic computation for a given algorithm is reduced to that of deriving a CCS equivalent to the algorithm. This derivation may be accomplished by first transforming the algorithm into a canonic form, then rewriting the canonic algorithm in the form of a canonic system of sequence equations/input-output specifications. If the system is not causal, then a sequence transformation may be applied to enforce causality and obtain a CCS that specifies a systolic computation. If more than one sequence transformation is possible, then, the one that reduces the execution time of the computation should be identified and chosen. In the next three sections, we explain each of the above steps in details.

3. Canonic algorithms

Kuhn [6], defines a naive algorithm as one that is written without regard to possible VLSI implementations. In order to design a systolic computation for a naive algorithm, we start by rewriting the algorithm in a caninic form:

Definition 3: A canonic algorithm is composed of an input statement (equivalent to a read statement), a body, and an output statement (equivalent to a write statement). The body of the algorithm is constructed from arbitrary nested DO loops that enclose assignment, or conditional assignment statements, where the latter is of the form "IF predicate THEN assignment". The following conditions should also be satisfied:

CA1: Each variable is an element of an n+1-dimensional array for some fixed n, $n \geq 1$, and each assignment statement is executed in the context of n+1 nested loops. Moreover, if S is an assignment statement that is executed at some instance $i_{1},\ldots,i_{n+1}$ of the n+1 loop indices, then each variable in the right side of S should be the $(i_1,\ldots,i_{n+1})^{th}$ entry of an array.

CA2: The value of each variable should be defined exactly once before it is used (via either an input statement or an assignment statement).

CA3: If S is an assignment statement that is executed at some instance $i_1,\ldots,i_{n+1}$ of the n+1 loop indices, and the variable in the left side of S is the $(j_1,\ldots,j_{n+1})^{th}$ entry of an array, then each j_k may depend only on i_k (possibly non linearly).

CA4: The predicate in a conditional assignment statement depends on the values of the loop indices and not on the values of the variables.

The n+1 loop indices define an (n+1)-dimensional space that we call the computation space. Accordingly, we may establish a one to one correspondence between the coordinates of the computation space and the dimensions of the arrays used in the canonic algorithm. More specifically, we associate with each $(j_1,...,j_{n+1})^{th}$ entry of an array, the position $(j_1,...,j_{n+1})$ in the computation space.

Algorithm transformations that satisfy conditions similar to CA1 and CA2 are called 'pipelining variable' in [16] , 'buffering variables' in [6] and 'massaging recursion variables' in [2]. Condition CA3 allows nonlinearity in the data dependence of the algorithm only along each dimension of the computation space. However, CA3 is less restrictive than the constant data dependence assumed in [16], and the first order recursion restriction of [2]. Finally, CA4 excludes from our design technique any network in which the operation of a specific cell depends on the value of its input. The design of this type of networks requires the definition of data dependent sequence operators, which we will not pursue in this paper.

As an example, consider the following algorithm for the solution of the linear system $A\ x = b$, where $A = \{a_{i,j}\}$ is an $n\times n$ lower triangular, banded matrix, with band-width $m+1$, and $b = \{b_i\}$ is an n-dimensional vector. In order to avoid loop bounds of the form $\max\{1,i-m\}$, we assume that $a_{i,j} = 0$ for $i\le m$, $j=i-m,\ldots,0$.

<u>ALG1: Naive forward substitution.</u>

```
    INPUT{ $x_i = 0$,          $i=1-m,\ldots,n$ ;
           $a_{i,j}$ , $b_i$        $i=1,\ldots,n$,    $j=i-m,\ldots i$ } ;
    DO i=1,n
     { DO j=1,m
           $x_i = x_i + a_{i,i+j-m-1} * x_{i+j-m-1}$ ;
       $x_i = (b_i - x_i) / a_{i,i}$ };
    OUTPUT{ $x_i$,        $i=1,\ldots,n$ }.
```

First, we rewrite the algorithm such that each statement is nested within two loops, and each variable is an element in a two dimensional array.

```
INPUT{ x(i,m+2)=0,  i=1-m,...,0 ; x(i,1)=0,  i=1,...,n ;
       a(i,j) = $a_{i,j}$, b(i,m+1) = $b_i$,  i=1,...,n,  j=i-m,...,i } ;
DO i=1,n
 DO j=1,m+1
 { IF j≤m THEN x(i,j+1) = x(i,j) + a(i,i+j-m-1) * x(i+j-m-1,m+2);
   IF j=m+1 THEN x(i,m+2) = ( b(i,j) - x(i,j) ) / a(i,i) } ;
OUTPUT{ $x_i$ = x(i,m+2),   i=1,...,n }.
```

Now, in order to satisfy CA1, we define the new variables c(i,j) = a(i,i+j-m-1) and z(i,j) = x(i+j-m-1,m+2). The first substitution is trivial, however, the second is an expansion of the column {x(k,m+2) ; k=1-m,...,n} into a two dimensional array z. Because the indices i and j are added in x(i+j-m-1,m+2), then, with the appropriate initial assignment, z may be expanded by using either z(i-1,j+1) = z(i,j) or z(i+1,j-1) = z(i,j). It may be shown that the first expansion leads to an algorithm where data are used before they are defined, thus violating CA2. Hence, we pursue the second expansion which is sketched in Fig 2. More precisely

```
z(1,j)       = x(j-m,m+2)        j=1,...,m
z(i+1,m)     = x(i,m+2)          i=1,...,n
z(i+1,j-1)   = z(i,j)            i=1,...,n,  j=1,...,m
```

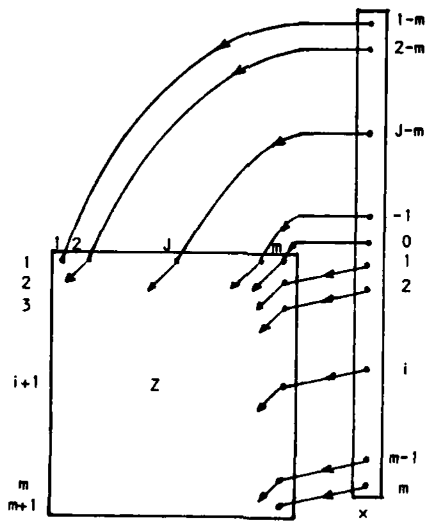


Fig 2 - expansion of a vector into a matrix

The incorporation of this expansion into the above algorithm gives the following:

<u>ALG2: Canonic forward substitution.</u>

```
INPUT( $z(1,j)=0$,  $j=1,\dots,m$ ;  $x(i,1)=0$,  $i=1,\dots,n$;
       $c(i,j)=a_{i,i+j-m-1}$, $b(i,m+1)=b_i$, $i=1,\dots,n$, $j=1,\dots,m+1$};
DO i=1,n
 DO j=1,m+1
 { IF $j \le m$ THEN { $x(i,j+1) = x(i,j) + c(i,j)\, z(i,j)$ ;
                  $z(i+1,j-1) = z(i,j)$ };
   IF $j = m+1$ THEN $z(i+1,m) = ( b(i,j) - x(i,j) ) / c(i,j)$ } ;
OUTPUT{ $x_i = z(i+1,m)$,   $i=1,\dots,n$ }.
```

4. The derivation of canonic sequence equations.

Conditions CA1 and CA3 of canonic algorithms establish a one to one correspondence between the loop indices $(i_1, \ldots, i_{n+1})$ and the dimensions of the arrays used in the algorithm. In other words, a given loop index i_k, may be used in the algorithm to select elements of arrays only along the k^th^ dimension. Hence, we may chose one loop index to represent the time, and project the variable arrays along the corresponding dimension by packing each n+1 dimensional array into an n dimensional sequence array. This transforms an algorithm which satisfies CA1 and CA2 into a system of sequence equations which satisfies CS1 and CS2. That is a canonic system of sequence equations.

For example, if we chose i to represent the 'time' in ALG2, then we may define the sequences ξ_j, ζ_j, γ_j, and β_{m+1} as follows

$$\xi_j(i) = x(i,j), \qquad \zeta_j(i) = z(i,j)$$

$$\gamma_j(i) = c(i,j), \qquad \beta_{m+1}(i) = b(i,m+1)$$

and rewrite ALG2 in the following form:

```
INPUT{ $\zeta_j(1) = 0$,  $j=1,..,m$,  ;  $\xi_1(i) = 0$,  $i=1,...,n$ ;
       $\gamma_j(i) = a_{i,i+j-m-1}$     $i=1,...,n$ , $j=1,...,m+1$        (3.a)
       $\beta_{m+1}(i) = b_i$            $i=1,...,n$ };                      (3.b)
DO i=1,n
   DO j=1,m+1
   { IF $j \le m$ THEN { $\xi_{j+1}(i) = \xi_j(i) + [\gamma_j(i) * \zeta_j(i)]$ ;
                         $\zeta_{j-1}(i+1) = \zeta_j(i)$ } ;
     IF $j = m+1$ THEN $\zeta_m(i+1) = [\beta_j(i) - \xi_j(i)] / \gamma_j(i)$ } ;
```

OUTPUT{ $x_i = \zeta_m(i+1)$, $i=1,\ldots,n$ }.

The above algorithm uniquely defines those elements of ξ_j, η_j, ζ_j and β_{m+1}, that are used in the algorithm. However, by CA2, any element of a sequence that is not defined in the algorithm is not used, and hence may be set to the don't care element δ or assigned an arbitrary value. For example, ζ_j, $j=1,\ldots,m-1$, are defined by $\zeta_j(1) = 0$ and $\zeta_{j-1}(i+1) = \zeta_j(i)$, $i=1,\ldots,n$. Given that $\zeta_{j-1}(i+1)$ is not defined in the algorithm for $i > m$, we may compact the definitions of $\zeta_{j-1}(i)$ in the form of the sequence equation $\zeta_{j-1} = \Omega_0\, \zeta_j$. Repeating this for all the sequences gives the following canonic system of sequence equations:

INPUT{ $\xi_1 = \iota$, where $\iota(t)=0$ for any t ;
γ_j, $j=1,\ldots,m+1$ and β_{m+1}, as in (3.a/b) } ;

$$\xi_{j+1} = \xi_j + \gamma_j * \zeta_j \qquad j=1,\ldots,m \tag{4.a}$$

$$\zeta_{j-1} = \Omega_0\, \zeta_j \qquad j=1,\ldots,m \tag{4.b}$$

$$\zeta_{j-1} = \Omega_0\, [\, [\beta_j - \xi_j)/\gamma_j \,] \qquad j=m+1 \tag{4.c}$$

OUTPUT{ $x_i = \zeta_m(i+1)$, $i=1,\ldots,n$ }.

The conditional assignment statements in ALG2 does depend on j, and hence the resulting system of equations contains different equations for different values of j. On the other hand, if the conditional assignment statement in the canonic algorithm depends on the index chosen to represent the 'time' then the multiplexing operator has to be used in order to express the algorithm in sequence form. For example, if the index j in ALG2 is chosen to represent the 'time', and the sequences ξ_i, ζ_i, γ_i and β_i, are defined for i=1,...,n by

$$\xi_i(j) = x(i,j), \quad \zeta_i(j) = z(i,j)$$

$$\gamma_i(j) = c(i,j) \tag{5.a}$$

$$\beta_i(j) = \begin{cases} b(i,m+1) & \text{if } j=1 \\ \delta & \text{if } j>1 \end{cases} \tag{5.b}$$

Then, from ALG2, $\zeta_{i+1}(j-1)$ is equal to $\zeta_i(j)$ if $j \le m$ and to $[\beta_i(j)-\xi_i(j)]/\gamma_i(j)$, if $j=m+1$. Adding to this $\zeta_{i+1}(j-1)=\delta$ for $j>m+1$ (not defined by the algorithm), we get

$$\zeta_{i+1} = \Omega^{-1} M^{m,1,\infty}(\zeta_i \ , \ [\beta_i-\xi_i]/\gamma_i \ , \ \delta^*)$$

Similarly, we may define ξ_i, and derive the following canonic system of equations in which the sequence δ^* is defined by $\delta^*(t)=\delta$ for any $t\ge 1$:

INPUT{ $\eta_1 = \iota$; γ_i and β_i, i=1,...,n, as in (5.a/b) } ;

$$\xi_i = \Omega_0 M^{m,\infty} (\xi_i+[\gamma_i * \zeta_i] \ , \ \delta^*) \qquad i=1,...,n \tag{6.a}$$

$$\zeta_{i+1} = \Omega^{-1} M^{m,1,\infty}(\zeta_i \ , \ [\beta_i-\xi_i]/\gamma_i \ , \ \delta^*) \qquad i=1,...,n \tag{6.b}$$

OUTPUT{ $x_i = \zeta_{i+1}(m)$, i=1,...,n }.

The main difference between an algorithm and a system of sequence equations is that some order of evaluation is imposed in the algorithm, while no order is imposed on the evaluation of the elements of the sequences in a system of sequence equations. However, when a CCS is evaluated in a systolic network, the order of evaluation is such that the tth elements of all the sequences in the system are evaluated simultaneously, and the evaluation proceeds in the order t=1,2,.... We call this order an element-wise evaluation.

Given that variables in a canonic algorithm cannot be overwritten (see CA2), it is clear that the order of evaluation imposed by the algorithm is only important because it guarantees that each variable is defined before it is used. Clearly, this property is preserved in the element-wise evaluation of the equivalent system of sequence equations only if the system is causal.

5. Enforcing the causality condition.

Consider the sequence equation

$$\rho_{a(i),b(j)} = \Gamma(\alpha_{i,j}\ ,\ \beta_{i,j},\ldots) \qquad (7)$$

where Γ is a sequence operator and a(i) and b(j) are functions of i and j, respectively. The more general form of (7) may involve n dimensional sequence arrays. However, for simplicity, we restrict our discussion to the case n=2. The extension to higher dimensions should be obvious.

Definition 4: The causality factor $\phi(t)$ of equation (7) at any t is defined as the minimum integer such that $\rho_{a(i),b(j)}(t)$ does not depend on any $\alpha_{i,j}(\tau)$, $\beta_{i,j}(\tau),\ldots,$ for $\tau > t-\phi(t)$. The minimum causality factor ϕ_m of equation (1) is defined by $\phi_m = \min\{\phi(t);\ t\geq 1\}$. Clearly, if (7) is causal, then $\phi_m > 0$. □

Any data item in (7) may be associated with a position in a 3-dimensional computation space. For example $\rho_{a(i),b(j)}(t)$ is associated with the position (t,a(i),b(j)). Moreover, if $\phi(t)$ is the deficiency factor of (7) at t, then only data items associated with the positions (τ,i,j), $\tau = 1,\ldots,t-\phi(t)$ may be used to define $\rho_{a(i),b(j)}(t)$. This motivates the following definitions:

Definition 5: The dependence pair of equation (7) at any $t\geq 1$ is a pair of vectors $\langle v,u\rangle$, where $v = (t,a(i),b(j))$ and $u = (t-\phi(t),i,j)$. The minimum dependence pair of (7) is the pair $\langle v,u_m\rangle$, where $u_m = (t-\phi_m,i,j)$. □

Definition 6: The difference vector of equation (7) at any $t\geq 1$ is the vector $v-u = (\phi(t), a(i)-i, b(j)-j)$. The minimum difference vector of (7) is the vector $v-u_m = (\phi_m, a(i)-i, b(j)-j)$. □

Note that any non linearity in the difference vector along the t dimension is absorbed in the minimum difference vector by assuming the worst case. Note also that the first component of the difference vector is equal to the deficiency factor.

If equation (7) is not causal, then the first component of the minimum difference vector is not positive. However, it may be possible to enforce causality by the application of some sequence transformation to (7). We consider two types of transformations. Namely

<u>Sequence spreading:</u> A spread of equation (7) by a constant s, $s>0$, is a substitution of each sequence $\sigma_{i,j}$ in (7) (here $\sigma = \rho, \alpha, \beta, \ldots$) by another sequence $\bar{\sigma}_{i,j} = \Theta^s \sigma_{i,j}$.

<u>Sequence skewing:</u> A skew of equation (7) by a function $w(i,j)$ is a substitution of each sequence $\sigma_{i,j}$ in (7) ($\sigma = \rho, \alpha, \beta, \ldots$) by another sequence $\bar{\sigma}_{i,j} = \Omega^{w(i,j)} \sigma_{i,j}$.

<u>Theorem 1:</u> Let ϕ_m be the minimum deficiency factor of equation (7). If the following equation

$$\bar{\bar{\rho}}_{a(i),b(j)} = \bar{\Gamma}(\bar{\bar{\alpha}}_{i,j}, \bar{\bar{\beta}}_{i,j}, \ldots) \tag{8}$$

is obtained by first spreading (7) by s and then skewing it by $w(i,j)$, that is by the substitution of

$$\bar{\bar{\rho}}_{a(i),b(j)} = \Omega^{w(a(i),b(j))} \Theta^s \rho_{a(i),b(j)}$$

$$\bar{\alpha}_{i,j} = \Omega^{w(i,j)}\ \Theta^{s}\ \alpha_{i,j}\ ,\quad \bar{\beta}_{i,j} = \Omega^{w(i,j)}\ \Theta^{s}\ \beta_{i,j}\ldots$$

then, the minimum deficiency factor of equation (8) is given by

$$\bar{\phi}_m = (s+1)\phi_m + w(a(i),b(j)) - w(i,j). \tag{9}$$

Proof: By the definitions of the operators Θ and Ω, if $\bar{\sigma}_{i,j} = \Omega^{w(i,j)}\ \Theta^{s}\ \sigma_{i,j}$, then $\sigma_{i,j}(t) = \bar{\sigma}_{i,j}((s+1)t-s+w(i,j))$. That is the above transformation maps the position (t,i,j) in the computation space into the position ((s+1)t-s+w(i,j), i, j) in the same space. Hence, the minimum dependence pair of equation (8) is $\langle \bar{v}, \bar{u}_m\rangle$, where

$$\begin{aligned}\bar{v} &= ((s+1)t-s+w(a(i),b(j)),\ a(i),\ b(j))\\ u_m &= ((s+1)(t-\phi_m)-s+w(i,j),\ i,\ j)\end{aligned}$$

From which we directly find that the first component of the minimum difference vector, and thus the minimum deficiency factor are given by (9). □

For the special case of linear transformation, we may prove the following result by direct substitution in (9).

Corollary: In Theorem 1, let $a(i)=i+a_o(i)$ and $b(j)=j+b_o(j)$, and let $w(i,j) = c_1 i + c_2 j$ be a linear function, then the minimum deficiency factor of equation (8) is given by

$$\bar{\phi}_m = (s+1)\phi_m + c_1\ a_o(i) + c_2\ b_o(j). \qquad \square$$

Now, given a non causal system of n canonic sequence equations, let the minimum dependence pair of the kth equation in the system be:

$$\langle (t,\ i+a_k(i),\ j+b_k(j))\ ,\ (t-\phi_k,\ i,\ j)\rangle, \qquad k = 1,\ldots,n$$

where not all ϕ_k, k=1,...,n are positive. In order to transform the given system into a causal system, we first attempt to find a constant s and a linear function $w(i,j)=c_1 i + c_2 j$ such that

$$\begin{bmatrix} \phi_1 & a_1(i) & b_1(j) \\ \phi_2 & a_2(i) & b_2(j) \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \phi_n & a_n(i) & b_n(j) \end{bmatrix} \begin{bmatrix} (s+1) \\ c_1 \\ c_2 \end{bmatrix} > \begin{bmatrix} 0 \\ 0 \\ \\ \\ 0 \end{bmatrix} \qquad (10)$$

where the relation > is applied element-wise. If this is possible, then, we have found a linear sequence transformation that will transform our system into a CCS. On the other hand, if equation (10) does not have a solution, then we should seek a non linear function w(i,j) such that

$$(s+1)\phi_k + w(a(i),b(j)) - w(i,j) > 0 \quad \text{for } k = 1,...,n \qquad (11)$$

In many cases, there may be more than one constant s and one function w(i,j) which satisfy (10) or (11). In such cases, we may choose s and w to minimize the execution time of the network.

<u>Definition 7:</u> Given any system of sequence equations/input-output specifications, let S_o be the set that contains the positions (in the computation space) of the data items in the output specification part of the system. If a spread by s followed by a skew by w(i,j) transform the given system into a CCS, then each position in S_o is mapped into a new position. Let $\bar{S}_o$ contain these new positions. The execution time T_e of the systolic computation corresponding to the CCS is then defined by

$$\begin{aligned} T_e &= \max\{ t \ ; \ (t,i,j) \in \bar{S}_o \} \\ &= \max\{ (s+1)t-s+w(i,j)^o ; \ (t,i,j) \in S_o \}. \end{aligned} \qquad (12)$$

□

Hence, the optimal choice of s and w(i,j) is the one that minimizes T_e.

For example, consider the system of equations (4). The computation space for this system is two dimensional and the minimum dependence pairs for its equations are

$$\begin{array}{ll} \langle (t\,,\, j+1)\,,\,(t\,,\,j)\rangle & \text{for } j=1,\ldots,m \\ \langle (t\,,\, j-1)\,,\,(t-1\,,\,j)\rangle & \text{for } j=1,\ldots,m \\ \langle (t\,,\, j-1)\,,\,(t-1\,,\,j)\rangle & \text{for } j=m+1 \end{array}$$

which shows that the system is not causal. Hence, we look for a constant s and a linear function $w(j) = c_2 j$ such that

$$\begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} s+1 \\ c_2 \end{bmatrix} > \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

and $T_e = \max\{(s+1)(t+1)-s+c_2 j \;;\; j=m,\ t=1\ldots,n\} = (s+1)n+mc_2+1$ is minimum, which in this case means the smallest s and c_2. Clearly, s=1 and c_2=1 satisfy the above conditions, and hence, we use the linear transformation

$$\bar{\xi}_j = \Omega^j \,\Theta\, \xi_j \quad ; \quad \bar{\zeta}_j = \Omega^j \,\Theta\, \zeta_j \qquad (13.a)$$

$$\bar{\gamma}_j = \Omega^j \,\Theta\, \gamma_j \quad ; \quad \bar{\beta}_{m+1} = \Omega^{m+1} \,\Theta\, \beta_{m+1} \qquad (13.b)$$

More specifically, if we multiply both sides of (4.a), (4.b) and (4.c) by $\Omega^{j+1}\Theta$, $\Omega^{j-1}\Theta$ and $\Omega^m\Theta$, respectively, and use property P1 from the Appendix, we may get

$$\bar{\xi}_{j+1} = \Omega\, \Omega^j \,\Theta\, [\xi_j + \gamma_j * \zeta_j], \qquad j=1,\ldots,m \qquad (14.a)$$

$$\bar{\zeta}_{j-1} = \Omega^{j-1}\, \Omega_0\, \Omega \,\Theta\, \zeta_j, \qquad j=1,\ldots,m \qquad (14.b)$$

$$\bar{\zeta}_{j-1} = \Omega^{j-1}\, \Omega_0\, \Omega \,\Theta\, [[\beta_j - \xi_j]/\gamma_j], \qquad j=m+1 \qquad (14.c)$$

Next, we replace Ω^{j-1} in (14.b/c) by Ω_0^{j-1}, and insert $\Omega^{-j}\,\Omega^{j}$ in the same equations, and finally use the definitions (13) to obtain the CCS (2) that was introduced in Section 2. In general, it is safe to replace a don't care by a specific value for the sake of simplifying the expressions. However, the converse is not true. In other words we are not allowed to replace a specific value, which may be defined in the original algorithm, by a don't care. This is why we could not simplify (14.b/c) by changing Ω_0 into Ω.

The system (6) of Section 4 provides another example of a non causal system. Its dependence pairs are $\langle(t,i),(t-1,i)\rangle$ and $\langle(t,i+1),(t+1,i)\rangle$ and the output set $S_o = \{(m,i+1)\ ;\ i=1,\dots,n\}$. For this system, a linear transformation with $s=0$ and $w(i)=2i$ is optimal. Hence, we let

$$\bar{\sigma}_i = \Omega_0^{2i}\,\sigma_i, \qquad \sigma = \xi,\zeta,\gamma,\beta$$

and multiply (6.a) and (6.b) by Ω_0^{2i} and $\Omega_0^{2(i+1)}$, respectively. Then we use property 2 form the appendix to obtain the following CCS:

INPUT{ $\bar{\bar{\eta}}_1 = \iota\ ;\ \bar{\gamma}_i = \Omega_0^{2i}\,\gamma_i\ ;\ \bar{\beta}_i = \Omega_0^{2i}\,\beta_i$

where γ_i and β_i, $i=1,\dots,n$, are as in (5.a/b) };

$$\bar{\xi}_i = \Omega_0\, M^{m,\infty}_{2i+1,[0]}\left(\ \bar{\xi}_i+[\bar{\gamma}_i*\bar{\zeta}_i]\ ,\ \delta^*\right) \qquad i=1,\dots,n$$

$$\bar{\bar{\zeta}}_{i+1} = \Omega_0\, M^{m,1,\infty}_{2i+1,[0]}\left(\ \bar{\bar{\zeta}}_i\ ,\ [\bar{\bar{\beta}}_i-\bar{\xi}_i]/\bar{\gamma}_i\ ,\ \delta^*\ \right) \qquad i=1,\dots,n$$

OUTPUT{ $x_i = \bar{\zeta}_{i+1}(m+2i+2)$, $i=1,\dots,n$) }.

This CCS specifies the network of Figure 3 which has n computational cells. Each cell i starts, at time 2i+1, the computation of the value of x_i in an internal register. The content of this register is described by the sequence $\bar{\xi}_i$ associated with the feed back link $\bar{x}_i$. After m+1 time units, the cell terminates its computation and the computed value of x_i is passed to the following cells i+1,...,n, on the output link $\bar{y}_{i+1}$.

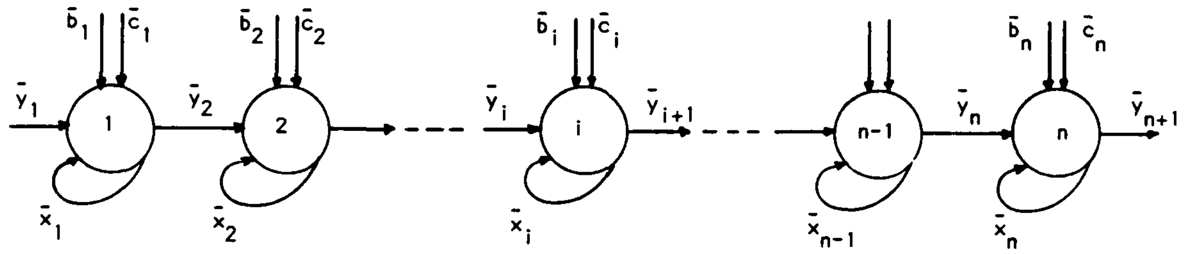


Fig 3 - A forward substitution network with n computational cells.

6. Example 1: A multistage shortest path network

Consider an S stage graph where each stage s, $0 \le s \le S$ consists of n_s nodes, with $n_0 = n_S = 1$. For each edge directed from a node j, $1 \le j \le n_{s-1}$, in stage s-1 to a node i, $1 \le i \le n_s$, in stage s, we are given a cost $a^s_{i,j}$, and the problem is to find the minimum cost of a path from the initial node (node 1 in stage 0) to the terminal node (node 1 in stage S).

In order to solve the problem, we let $m = \max\{n_s : s=0,\dots,S\}$, and we assume that $a^s_{i,j} = \infty$ if there is no path from node j in stage s-1 to node i in stage s, or if $n_{s-1} < j \le m$ and/or $n_s < i \le m$. That is if either of the two nodes does not exist.

In the following algorithm, the solution proceeds by finding at each stage s and for each node i in s the minimum cost C^s_i of a path from the initial node to node i. Each C^s_i is computed progressively in y(i,j,s). (We denote min{x,y} by x@y).

```
INPUT{ y(i,m+1,0) = 0,      i=1,...,m ;
       a(i,j,s)   = $a^s_{i,j}$   i,j=1,...,m ,   s=1,...,S } ;
DO s=1,S
 DO i=1,m
  DO j=1,m
  { IF j=1 THEN y(i,j+1,s) = y(j,m+1,s-1) * a(i,j,s);
    IF j>1 THEN y(i,j+1,s) = y(i,j,s) @ y(j,m+1,s-1) * a(i,j,s)};
OUTPUT{ $C^S_1$ = y(1,m+1,S) }.
```

Although each variable in the above algorithm is an element of a three dimensional array, the algorithm does violate CA1 of canonic algorithms. Namely, $y(j,m+1,s-1)$ is not an $(i,j,s)^{th}$ element of an array. Hence, we let $y(j,m+1,s-1) = x(i,j,s)$, and use the expansion

$$x(1,j,s+1) = y(j,m+1,s) \quad ; \quad x(i+1,j,s) = x(i,j,s)$$

This gives the following algorithm:

```
INPUT{ $x(1,j,1) = 0$,          $j=1,...,m$ ;
       $a(i,j,s) = a^s_{i,j}$,   $i,j=1,...,m$,  $s=1,...,S$ } ;
DO s=1,S
 DO i=1,m
  DO j=1,m
  { $x(i+1,j,s) = x(i,j,s)$ ;
    IF $j = 1$ THEN $y(i,j+1,s) = x(i,j,s) + a(i,j,s)$ ;
    IF $1<j<m$ THEN $y(i,j+1,s) = y(i,j,s)$ @ $(x(i,j,s) + a(i,j,s))$ ;
    IF $j = m$ THEN $x(1,i,s+1) = y(i,j,s)$ @ $(x(i,j,s) + a(i,j,s))$};
OUTPUT{ $C^S_1 = x(1,1,S+1)$ }.
```

This algorithm, however, violates condition CA3 because, for $j=m$, the index i is used to select an element of the x array along the second dimension, which is associated with the index j. In order to overcome this problem, we may use the y and x arrays, alternatively, to accumulate the partial costs at successive stages. More specifically, we rewrite the algorithm in the following canonic form:

```
INPUT{ $x(1,j,1) = 0$,        $j=1,\dots,m$ ;
       $a(i,j,s) = a^s_{i,j}$,   $i,j=1,\dots,m$,   $s=1,3,5,\dots$ ;
       $b(i,j,s) = a^s_{j,i}$,   $i,j=1,\dots,m$,   $s=2,4,6,\dots$  } ;
DO $s=1,S$
{ IF ($s$ = odd) THEN
    DO $i=1,m$
     DO $j=1,m$
     {$x(i+1,j,s) = x(i,j,s)$;
      IF $j=1$ THEN $y(i,j+1,s) = x(i,j,s) + a(i,j,s)$;
      IF $1<j<m$ THEN $y(i,j+1,s) = y(i,j,s)$ @ $(x(i,j,s) + a(i,j,s))$;
      IF $j=m$ THEN $y(i,1,s+1) = y(i,j,s)$ @ $(x(i,j,s) + a(i,j,s))$ };
  IF ($s$ = even) THEN
    DO $j=1,m$
     DO $i=1,m$
     {$y(i,j+1,s) = y(i,j,s)$;
      IF $i=1$ THEN $x(i+1,j,s) = y(i,j,s) + b(i,j,s)$;
      IF $1<i<m$ THEN $x(i+1,j,s) = x(i,j,s)$ @ $(y(i,j,s) + b(i,j,s))$;
      IF $i=m$ THEN $x(1,j,s+1) = x(i,j,s)$ @ $(y(i,j,s) + b(i,j,s))$ }
} ;
OUTPUT{ $C^S_1$ = IF $S$ is odd THEN $y(1,1,S+1)$ ELSE $x(1,1,S+1)$ }.
```

Now, we may chose both i and s to represent the time and compress the arrays along these dimensions. More specifically, we first compress the arrays along the i dimension by defining the sequences

$$\xi^s_j(i) = x(i,j,s) \qquad j=1,\dots,m, \quad s=1,\dots,S$$

$$\eta_j^s(i) = y(i,j,s) \qquad j=1,\dots,m, \quad s=1,\dots,S$$

$$\gamma_j^s(i) = \begin{cases} a(i,j,s) & j=1,\dots,m, \quad s=1,3,\dots \\ b(i,j,s) & j=1,\dots,m, \quad s=2,4,\dots \end{cases} \tag{15.a}$$

and then compress the sequences ξ_j^s, η_j^s and γ_j^s, $s=1,\dots,S$ along the s dimension by defining the sequences

$$\xi_j = P^m_{s=1,S}(\ \xi_j^s\ ) \qquad j=1,\dots,m$$

$$\eta_j = P^m_{s=1,S}(\ \eta_j^s\ ) \qquad j=1,\dots,m$$

$$\gamma_j = P^m_{s=1,S}(\ \gamma_j^s\ ) \qquad j=1,\dots,m \tag{15.b}$$

Note that the elements of the sequences η_1^s for s =odd are not defined by the canonic algorithm. These elements, however, are not used in the algorithm, and hence may be set to the don't care element δ. With this, the two step compression leads to the following canonic system of of sequence equations:

INPUT{ γ_j, $j=1,\dots,m$, as given by (15)} ;

$$\xi_j = \Omega_0\ M^{m,1,m-1}(\ \xi_j\ ,\ \eta_j+\gamma_j\ ,\ \xi_j@[\eta_j+\gamma_j])\qquad j=1,\dots,m \tag{16.a}$$

$$\eta_{j+1} = M^{m,m}(\ \xi_j+\gamma_j\ ,\ \eta_j) \qquad j=1 \tag{16.b}$$

$$\eta_{j+1} = M^{m,m}(\ \eta_j@[\xi_j+\gamma_j]\ ,\ \eta_j\ ) \qquad j=2,\dots,m-1 \tag{16.c}$$

$$\eta_{j-m+1} = M^{m,m}(\delta^*\ ,\ \Omega^m\eta_j@[\Omega^m\xi_j + \Omega^m\gamma_j]\ ) \qquad j=m \tag{16.d}$$

OUTPUT{ C_1^S = IF S is odd THEN $\eta_1(mS+1)$ ELSE $\xi_1(mS+1)$ }.

The system (16) is not causal. More specifically, its equations have the following dependence pairs:

$$\langle(\ t\ ,\ j\ )\quad ,\ (\ t-1\ ,\ j\ )\rangle \qquad j=1,\dots,m \tag{17.a}$$

$$\langle (t, j+1), (t, j) \rangle \qquad j=1 \qquad (17.b)$$
$$\langle (t, j+1), (t, j) \rangle \qquad j=2,\dots,m-1 \qquad (17.c)$$
$$\langle (t, j-m+1), (t-m, j) \rangle \qquad j=m \qquad (17.d)$$

In order to enforce causality via a linear sequence transformation, we must find two constants p and c_2, such that

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ m & -m+1 \end{bmatrix} \begin{bmatrix} p+1 \\ c_2 \end{bmatrix} > \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and $T_e = (p+1)(mS+1)-p+c_2$ is minimum. Clearly, $p=0$ and $c_2 = 1$ satisfy the above conditions. That is causality may be enforced in (16) via the following substitutions:

$$\bar{\xi}_j = \Omega^j \xi_j \;;\; \bar{\eta}_j = \Omega^j \eta_j \;;\; \bar{\gamma}_j = \Omega^j \gamma_j \,, \quad j=1,\dots,m \qquad (18)$$

More specifically, we first multiply (16.a) by Ω^j, (16.b/c) by Ω^{j+1} and (16.d) by Ω. Then, we use property P2 from the Appendix to interchange the Ω and M operators, and finally, we use (18) to obtain the following CCS:

INPUT{ $\bar{\bar{\gamma}}_j = \Omega^j \gamma_j$, $j=1,\dots,m$ };

$$\bar{\xi}_j = \Omega^j \Omega_0 \Omega^{-j} M^{m,1,m-1}_{j+1}(\bar{\xi}_j , \bar{\eta}_j , \bar{\xi}_j @[\bar{\gamma}_j+\bar{\eta}_j]) \quad j=1,\dots,m \qquad (19.a)$$

$$\bar{\bar{\eta}}_2 = \Omega M^{m,m}_2(\bar{\bar{\xi}}_1 , \bar{\bar{\eta}}_1) \qquad j=1 \qquad (19.b)$$

$$\bar{\eta}_{j+1} = \Omega M^{m,m}_{j+1}(\bar{\eta}_j @[\bar{\xi}_j+\bar{\bar{\gamma}}_j] , \bar{\bar{\eta}}_j) \qquad j=2,\dots,m-1 \qquad (19.c)$$

$$\bar{\bar{\eta}}_1 = \Omega M^{m,m}_1(\delta^* , \bar{\eta}_j @[\bar{\bar{\gamma}}_j+\bar{\bar{\xi}}_j]) \qquad j=m \qquad (19.d)$$

OUTPUT{ C^S_1 = IF S is odd THEN $\bar{\eta}_1(mS+2)$ ELSE $\bar{\bar{\xi}}_1(mS+2)$ }.

The above CCS describes a linear network of m cells (see Fig 4), where each cell j contains an accumulator (call it X_j) whose

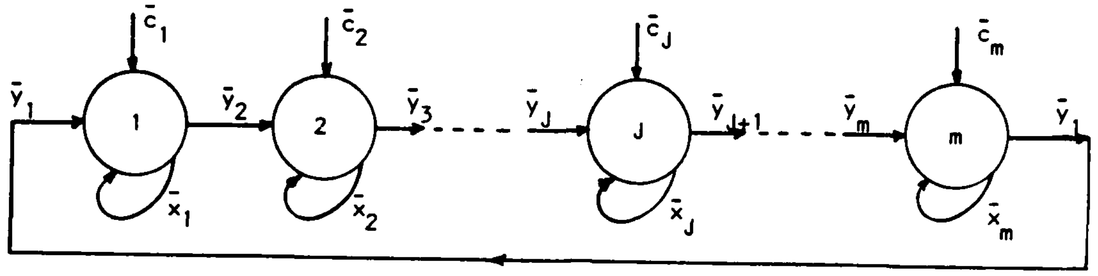


Fig 4 - A multistage, shortest path network

content is described by $\bar{\xi}_j$. The operation of each cell j alternates between two phases; In an odd phase, C_j^s is stored in X_j and the cell contributes in the computation of C_i^{s+1}, i=1,...,m, where each C_i^{s+1} is computed progressively on the $\bar{y}$ links by picking up contributions from the different cells. In the next phase, the computed C_i^{s+1}, i=1,...,m, circulate unchanged on the $\bar{\bar{y}}$ links while cell j computes C_j^{s+2} in its accumulator. The precise operation of each cell is given by (19).

Note that the term $\Omega^j\Omega_0\Omega^{-j}$ in equation (19.a) indicates that the content of the accumulator at cell j is reset to zero at the j+1th cycle. In order to simplify this equation, we may reset the accumulator to zero at the first cycle and maintain this zero for the first j+1 cycles. This is equivalent to the replacement of (19.a) in the CCS (19) by

$$\bar{\bar{\xi}}_j = \Omega_0 \, M^{m,1,m-1}_{j+1,[0]}(\bar{\bar{\xi}}_j \, , \, \bar{\eta}_j \, , \, \bar{\xi}_j @ [\bar{\bar{\eta}}_j + \bar{\gamma}_j] \,) \qquad (19.e)$$

A network very similar to the one described in this section is given in [19].

7. EXAMPLE 2: A network for dynamic programming.

Consider the following optimal parenthesization problem [4]: Given $c_{i,i}$, $i=1,\dots,n$, find $c_{1,n}$, where

$$c_{i,i+j} = \min\{\ f(c_{i,i+k-1}\ ,\ c_{i+k,i+j})\ ;\ k \in K=\{1,\dots,j\}\ \} \quad (20)$$

for $j=1,\dots,n-1$, $i=1,\dots,n-j$ and a given function f.

The order at which the minimum is evaluated over the set of indices $K = \{1,\dots,j\}$ is not specified by the problem. The simplest order is the sequential order $k=1,\dots,j$. It is possible to write an algorithm using this order and then apply our technique to derive an equivalent systolic computation. The resulting network, however, does not overlap the computation of $c_{i,i+j}$ for different j, and hence has an execution time $T_e = O(n^2)$.

An alternative order for the evaluation of (20) is to start from the middle of the interval [1,j], namely from $\ell = (j+1) \div 2$, and proceed towards the boundaries of [1,j] in the two directions $\ell-k$ and $\ell+k$, $k=1,2,\dots$, simultaneously. This is described, more precisely, by the algorithm shown in Fig 5, where $x@y$ denotes $\min\{x,y\}$, and $x\div y$ denotes the quotient of x/y.

The algorithm of Fig 5 is not canonic. However, it may be rewritten in the canonic form shown in Fig 6 by using the following substitutions along with the appropriate expansions:

$$m(i,j,k) = h^{k}_{i,i+j}$$

$$z(i,j,k) = c_{i,i+(j\div 2)+k-1} \quad ; \quad w(i,j,k) = c_{i+(j\div 2)+k,i+j}$$

```
INPUT{ $c_{i,i}$   i=1,...,n }
DO j=1,n-1
 DO i=1,n-j
  DO k=1, $\ell$ +1          /* $\ell$ is the quotient of (j+1)÷2 */
   {IF (j = odd) THEN
    { IF k = 1 THEN $h^{k+1}_{i,i+j} = f(c_{i,i+\ell-1}, c_{i+\ell,i+j})$ ;
      IF 1<k≤$\ell$ THEN $h^{k+1}_{i,i+j} = h^{k}_{i,i+j}$ @ $f(c_{i,i+\ell-k}, c_{i+\ell-k+1,i+j})$
                                    @ $f(c_{i,i+\ell+k-2}, c_{i+\ell+k-1,i+j})$ ;
      IF k=$\ell$+1 THEN $c_{i,i+j} = h^{k}_{i,i+j}$
    } ;
    IF (j = even) THEN
    { IF k = 1 THEN $h^{k+1}_{i,i+j} = f(c_{i,i+\ell-k}, c_{i+\ell-k+1,i+j})$
                                    @ $f(c_{i,i+\ell+k-1}, c_{i+\ell+k,i+j})$ ;
      IF 1<k≤$\ell$ THEN $h^{k+1}_{i,i+j} = h^{k}_{i,i+j}$ @ $f(c_{i,i+\ell-k}, c_{i+\ell-k+1,i+j})$
                                    @ $f(c_{i,i+\ell+k-1}, c_{i+\ell+k,i+j})$ ;
      IF k=$\ell$+1 THEN $c_{i,i+j} = h^{k}_{i,i+j}$
    } ;
   } ;
OUTPUT{ $c_{1,n} = h^{L+1}_{1,n}$, where L = n ÷ 2 }
```

Fig. 5 - An algorithm for dynamic programming.

```
INPUT{ z(i,1,1) = $c_{i,i}$ , y(i,1,1) = $c_{i+1,i+1}$ ,   i=1,...,n-1 };
DO j=1,n-1
 DO i=1,n-j
  DO k=1, $\ell$ +1
   {IF (j = odd) THEN
    { IF k = 1 THEN { x(i,j+1,k) = z(i,j,k) ; w(i-1,j+1,k) = y(i,j,k) ;
            m(i,j,k+1) = f(z(i,j,k) , y(i,j,k)) };
      IF 1<k≤$\ell$ THEN { x(i,j+1,k) = x(i,j,k) ; w(i-1,j+1,k) = w(i,j,k) ;
            z(i,j+1,k-1) = z(i,j,k) ; y(i-1,j+1,k-1) = y(i,j,k) ;
            m(i,j,k+1) = m(i,j,k) @ f(x(i,j,k),y(i,j,k))
                                  @ f(z(i,j,k),w(i,j,k)) };
      IF k=$\ell$+1 THEN { z(i,j+1,k-1)=m(i,j,k) ; y(i-1,j+1,k-1)=m(i,j,k)};
    } ;
     IF (j = even) THEN
    { IF k = 1 THEN { x(i,j+1,k+1)=x(i,j,k) ; w(i-1,j+1,k+1)=w(i,j,k) ;
            z(i,j+1,k) = z(i,j,k) ; y(i-1,j+1,k) = y(i,j,k) ;
            m(i,j,k+1) = f(x(i,j,k),y(i,j,k)) @ f(z(i,j,k),w(i,j,k)) };
      IF 1<k≤$\ell$ THEN {
x(i,j+1,k+1)=x(i,j,k) ; w(i-1,j+1,k+1)=w(i,j,k) ;
            z(i,j+1,k) = z(i,j,k) ; y(i-1,j+1,k) = y(i,j,k) ;
            m(i,j,k+1) = m(i,j,k) @ f(x(i,j,k),y(i,j,k))
             @ f(z(i,j,k),w(i,j,k))};
      IF k=$\ell$+1 THEN { z(i,j+1,k) = m(i,j,k) ; y(i-1,j+1,k) = m(i,j,k)};
    } ;
   } ;
OUTPUT{ $c_{1,n}$ = m(1,n-1,L+1), where L = n÷2 }.
```

Fig 6 - A canonic algorithm for dynamic programming.

$\langle (t, i+a_k(i), j+b_k(j)) , (t-\phi_k, i, j) \rangle, \quad k = 1,\ldots,n$

$$x(i,j,k) = c_{i,i+\ell-k} \quad ; \quad y(i,j,k) = c_{i+\ell-k+1,i+j}$$

Next, we chose k to represent the time and we define the sequences $\mu_{i,j}$, $\xi_{i,j}$, $\eta_{i,j}$, $\zeta_{i,j}$ and $\omega_{i,j}$ to contain the elements of the arrays m, x, y, z and w, respectively, along the k dimension. We also define the element-wise sequence operator ϕ such that $[\phi(\xi,\eta)](t) = f(\xi(t) , \eta(t))$. With this, we may compact the canonic algorithm along the kth dimension and obtain the following canonic system of sequence equations:

$$\text{INPUT}\{ \zeta_{i,1}(1)=c_{i,i} \; , \; \eta_{i,1}(1) = c_{i+1,i+1} \; i=1,\ldots,n-1; \quad \zeta_{i,1}(t)=\eta_{i,1}(t)=\delta,\; t>1,\; i=1,\ldots,n-1\}; \tag{21}$$

FOR $j=1,\ldots,n-1$ and $i=1,\ldots,n-j$

$$\mu_{i,j} = \begin{cases} \Omega\, M^{1,\ell-1,\infty}(\phi(\zeta_{i,j},\eta_{i,j}) \;,\; \mu_{i,j} \oplus \psi_{i,j} \;,\; \delta^*) & \text{if } j \text{ is odd} \\ \Omega\, M^{1,\ell-1,\infty}(\psi_{i,j} \;,\; \mu_{i,j} \oplus \psi_{i,j} \;,\; \delta^*) & \text{if } j \text{ is even} \end{cases} \tag{22}$$

where $\ell=(j+1)\div 2$ and $\psi_{i,j} = \phi(\xi_{i,j},\eta_{i,j}) \oplus \phi(\zeta_{i,j},\omega_{i,j})$

$$\xi_{i,j+1} = \begin{cases} M^{1,\ell-1,\infty}(\zeta_{i,j} \;,\; \xi_{i,j} \;,\; \delta^*) & \text{if } j \text{ is odd} \\ \xi_{i,j} & \text{if } j \text{ is even} \end{cases} \tag{23}$$

$$\omega_{i-1,j+1} = \begin{cases} M^{1,\ell-1,\infty}(\eta_{i,j} \;,\; \omega_{i,j} \;,\; \delta^*) & \text{if } j \text{ is odd} \\ \omega_{i,j} & \text{if } j \text{ is even} \end{cases} \tag{24}$$

$$\zeta_{i,j+1} = \begin{cases} M^{\ell-1,1,\infty}(\Omega^{-1}\zeta_{i,j} \;,\; \Omega^{-1}\mu_{i,j} \;,\; \delta^*) & \text{if } j \text{ is odd} \\ M^{\ell,1,\infty}(\zeta_{i,j} \;,\; \mu_{i,j} \;,\; \delta^*) & \text{if } j \text{ is even} \end{cases} \tag{25}$$

$$\eta_{i-1,j+1} = \begin{cases} M^{\ell-1,1,\infty}(\Omega^{-1}\eta_{i,j} \;,\; \Omega^{-1}\mu_{i,j} \;,\; \delta^*) & \text{if } j \text{ is odd} \\ M^{\ell,1,\infty}(\eta_{i,j} \;,\; \mu_{i,j} \;,\; \delta^*) & \text{if } j \text{ is even} \end{cases} \tag{26}$$

$\text{OUTPUT}\{ c_{1,n} = \mu_{1,n-1}(L+1)$, where $L = n \div 2$ };

The dependence pairs for equations (22), (23) and (24) are, respectively,

$$\langle (t, i, j), (t-1, i, j)\rangle \qquad j=1,2,\ldots$$
$$\langle (t, i, j+1), (t, i, j)\rangle \qquad j=1,2,\ldots$$
$$\langle (t, i-1, j+1), (t, i, j)\rangle \qquad j=1,2,\ldots$$

The dependence pairs for equations (25) are

$$\langle (t, i, j+1)\rangle, (t+1, i, j)\rangle \qquad j=1,3,\ldots \qquad (27.a)$$
$$\langle (t, i, j+1)\rangle, (t, i, j)\rangle \qquad j=2,4,\ldots \qquad (27.b)$$

and, the dependence pairs of equations (26) are

$$\langle (t, i-1, j+1)\rangle, (t+1, i, j)\rangle \qquad j=1,3,\ldots \qquad (28.a)$$
$$\langle (t, i-1, j+1)\rangle, (t, i, j)\rangle \qquad j=2,4,\ldots \qquad (28.b)$$

As indicated by the dependence pairs, all the equations in the system are not causal. However, by applying Corollary 1 of Section 5, we may check that a linear skew of the equations with Ω^{2j} transforms the system into a causal one that has an execution time $T_e = 2n+L-1$, where $L=n\div 2$.

An interesting remark is that, in the absence of the pairs (27.a) and (28.a), a skew of the form Ω^{j} is sufficient to enforce causality. In other words, the factor of two is only needed for the case 'j=odd'. For this reason we may try to apply a non linear skew of the form $\Omega^{q(j)}$, where

$$q(j) = 3j\div 2 - 1 = \begin{cases} q_1(j) = (3j-1)\div 2 - 1 & \text{if } j \text{ is odd} \\ q_2(j) = 3j\div 2 - 1 & \text{if } j \text{ is even} \end{cases}$$

The application of Theorem 1 indicates that a skew of the system (22)-(26) with q(j) will enforce causality. For example, the pairs (28.a/b) are mapped to

$$\langle (t+q_2(j+1)\ ,\ i-1\ ,\ j+1)\ ,\ (t+1+q_1(j)\ ,\ i\ ,\ j)\rangle \qquad j=1,3,\ldots$$
$$\langle (t+q_1(j+1)\ ,\ i-1\ ,\ j+1)\ ,\ (t+q_2(j)\ ,\ i\ ,\ j)\rangle \qquad j=2,4,\ldots$$

from which we find that the minimum deficiency factors of equations (26), after transformation, are $q_2(j+1)-q_1(j)-1=1$, and $q_1(j+1)-q_2(j)=1$, for j=odd and j=even, respectively. Similarly, we can show that the minimum deficiency factors of the other equations are all equal to unity. The execution time of the transformed system is given by $T_e = L+1+q(n-1) = 2n-2$.

Hence, a substitution of the form

$$\bar{\bar{\sigma}}_{i,j} = \Omega^{q(j)}\ \sigma_{i,j} \qquad \sigma = \mu\ ,\ \xi\ ,\ \omega\ ,\ \zeta \text{ or } \eta$$

in the system (22)-(26) gives the following CCS:

INPUT{ $\bar{\zeta}_{i,1} = \zeta_{i,1}$, $\bar{\eta}_{i,1} = \eta_{i,1}$,i=1,...,n-1,
where $\zeta_{i,1}$ and $\eta_{i,1}$ are as in (21) };

FOR j=1,...,n-1 and i=1,...,n-j

$$\bar{\bar{\mu}}_{i,j} = \begin{cases} \Omega\ M^{1,\ell-1,\infty}_{q1(j)+1}(\phi(\bar{\zeta}_{i,j},\bar{\eta}_{i,j})\ ,\ \bar{\mu}_{i,j}@\bar{\psi}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is odd} \\ \Omega\ M^{1,\ell-1,\infty}_{q2(j)+1}(\bar{\bar{\psi}}_{i,j}\ ,\ \bar{\mu}_{i,j}@\bar{\psi}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is even} \end{cases}$$

where $\ell=(j+1)\div 2$ and $\bar{\bar{\psi}}_{i,j} = \phi(\bar{\bar{\xi}}_{i,j},\bar{\bar{\eta}}_{i,j})@\phi(\bar{\bar{\zeta}}_{i,j},\bar{\bar{\omega}}_{i,j})$

$$\bar{\bar{\xi}}_{i,j+1} = \begin{cases} \Omega^2\ M^{1,\ell-1,\infty}_{q1(j)+1}(\bar{\bar{\zeta}}_{i,j}\ ,\ \bar{\bar{\xi}}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is odd} \\ \Omega\ \bar{\bar{\xi}}_{i,j} & \text{if } j \text{ is even} \end{cases}$$

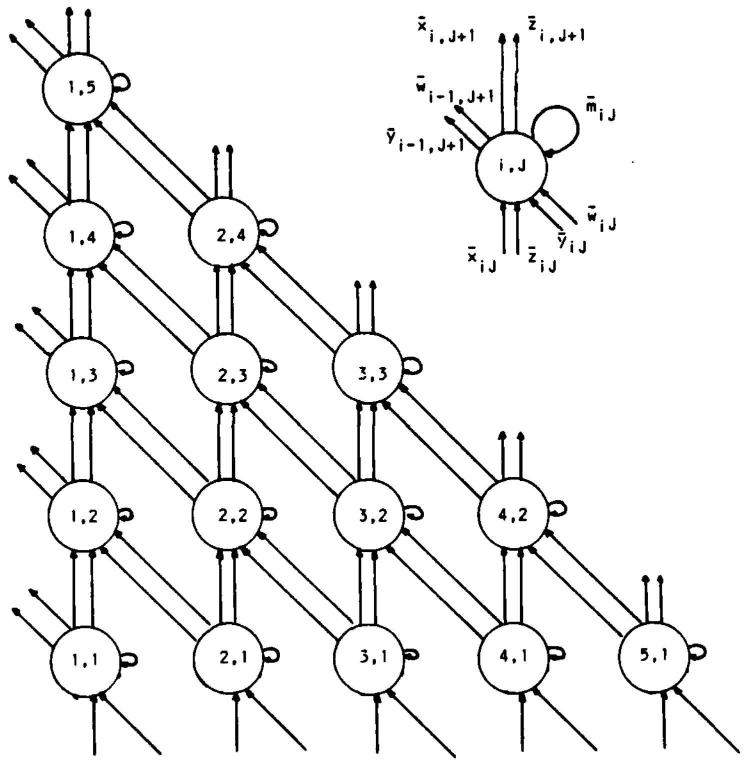
$$\bar{\bar{\omega}}_{i-1,j+1} = \begin{cases} \Omega^2\ M^{1,\ell-1,\infty}_{q1(j)+1}(\bar{\bar{\eta}}_{i,j}\ ,\ \bar{\bar{\omega}}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is odd} \\ \Omega\ \bar{\bar{\omega}}_{i,j} & \text{if } j \text{ is even} \end{cases}$$

$$\ddot{\zeta}_{i,j+1} = \begin{cases} \Omega\ M^{\ell-1,1,\infty}_{q1(j)+2}(\bar{\bar{\zeta}}_{i,j}\ ,\ \bar{\bar{\mu}}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is odd} \\ \Omega\ M^{\ell,1,\infty}_{q2(j)+1}(\bar{\zeta}_{i,j}\ ,\ \bar{\mu}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is even} \end{cases}$$

$$\bar{\eta}_{i-1,j+1} = \begin{cases} \cap\, M^{\ell-1,1,\infty}_{q1(j)+2}(\bar{\eta}_{i,j}\ ,\ \bar{\mu}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is odd} \\ \cap\, M^{\ell,1,\infty}_{q2(j)+1}(\bar{\eta}_{i,j}\ ,\ \bar{\mu}_{i,j}\ ,\ \delta^*) & \text{if } j \text{ is even} \end{cases}$$

OUTPUT{ $c_{1,n} = \bar{\mu}_{1,n-1}(2n-2)$ }.

The above CCS specifies the network shown in Fig 7.a, which was first introduced by Kung and Guibas in [4]. The structure of each cell may be directly derived from the operators in the causal equations. As an example, we show in Fig 7.b the internal details of a cell (i,j), j=even. Note that the circuits for the outputs on $\bar{z}_{i,j+1}$ and $\bar{w}_{i-1,j+1}$ are similar to those for $\bar{y}_{i-1,j+1}$, and $\bar{x}_{i,i+1}$, respectively, and, hence, are not shown in the figure.



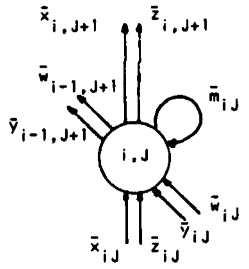


Fig 7.a - A systolic network for linear programming

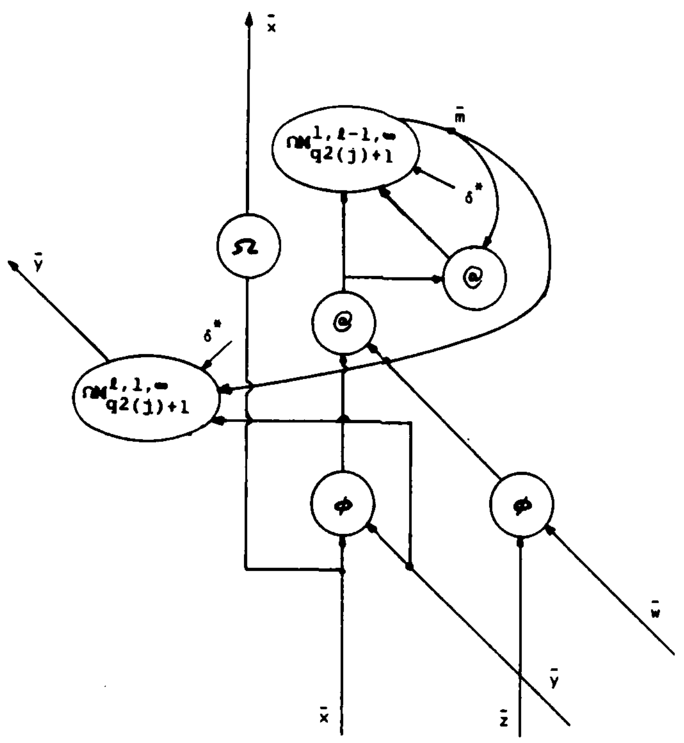


Fig 7.b - The details of a cell (i,j), j=even.

8. Concluding remarks

The sequence model introduced in [14] for the verification of systolic computations is applied in this paper to the systematic design of such computations. Given an algorithm for the solution of a specific problem, the first step in the design technique is the transformation of the algorithm into a canonic form. Then, the algorithm is rewritten as a system of sequence equations and finally, a sequence transformation is used to enforce causality and produce a complete specification of a network which executes the original algorithm.

The technique is applicable to self-timed computations as well as systolic computations. More specifically, it was shown in [13] that self timed networks may be specified by systems of weakly causal equations, where the minimum deficiency factor ϕ_m of each equation is non-negative rather than positive. Hence, in the last step of our technique, a sequence transformation that enforces only weak causality should produce the specification of a self-timed computation.

The order of associating operands to operator in the original algorithm is crucial and may lead to different systolic computations that solve the same problem. For example, in the dynamic programming algorithm of Section 7, different networks may be obtained by considering different orders for the evaluation of equ (20) over the set of indices $K \leftarrow \{1,\ldots,j\}$. Also, in ALG1 of Section 2, if the order of the summation is reversed, that is $x_i \leftarrow \sum_{j=1}^{m} a_{i,i+j-m-1}\, x_{i+j-m-1}$ is replaced by $x_i \leftarrow$

$\sum_{j=m}^{l} a_{i,i+j-m-1}\, x_{i+j-m-1} = \sum_{k=1}^{m} a_{i,i-k}\, x_{i-k}$, then the resulting algorithm does not have any systolic realization. In order to overcome this problem, it is essential to find a suitable notation to express generic algorithms, namely algorithms in which the orders of evaluation of the operations are not specified, and then to introduce a design technique which derives the order that leads to the optimal design.

Finally, we should note that the sequence transformations used in this paper are time independent. More specifically, we used transformations of the form $\bar{\sigma}_{i,j} = \Omega^{w(i,j)}\theta^{s}\sigma_{i,j}$, where s is a constant. A more general transformation may be obtained by assuming that s=s(t) is a function of time, that is the elements of the sequences are spread non uniformly. However, we did not find any example where this time dependent spreading is useful and hence we have chosen to simplify our notation by keeping s constant.

Acknowledgement

I would like to thank Concettina Guerra for the long discussions that stimulated this research. The use of the multistage shortest path network as an example was also her idea.

Appendix

In this appendix, we define the sequence operators that are used in the paper and we introduce some of their properties. Let $\overline{R}_\delta$ be the set of all sequences defined on $R \cup \{\delta\}$, where R is the set of real numbers, and δ is a special element called the don't care element.

1) δ-regular, element-wise operators: Any binary operator 'op' defined on R may be extended to $\overline{R}_\delta$ by applying it, element wise to elements of sequences, with δ being the result of any operation involving δ. More specifically,

$$[\xi \text{ 'op' } \eta](t) = \begin{cases} \delta & \text{if } \xi(t)=\delta \text{ or } \eta(t)=\delta \\ \xi(t) \text{ 'op } \eta(t) & \text{otherwise} \end{cases}$$

2) The shift operator; $\Omega_x^r : \overline{R}_\delta \to \overline{R}_\delta$, is defined by

$$[\Omega_x^r \zeta](t) = \begin{cases} x & \text{if } t \le r \\ \zeta(t-r) & \text{if } t > r \end{cases}$$

More descriptively, if r is positive, then Ω_x^r inserts r elements, each equal to x, at the beginning of its operand. For example, if

$$\zeta = z_1 , z_2 , z_3 , \ldots \qquad (29)$$

then, $\Omega_x^2\zeta = x,x,z_1,z_2,\ldots$. Note that Ω_x^r may be used to model a cell which maintains x on its output for r time units, and delays its input by r units. For simplicity, we omit r if it is unity, and x if it is δ.

On the other hand, if r is negative, then Ω_x^r trims the first r elements of its operand. For example, with ζ of (29), $\Omega^{-2} \zeta =$

$z_3, z_4, \ldots$. Note that Ω^{-r} may be used as an inverse to Ω^r. More specifically, $\Omega^{-r}\, \Omega^r_x\, \zeta = \zeta$. However, the converse is not always true, that is $\Omega^r_x\, \Omega^{-r}\, \zeta = \zeta$ only if the first r elements of ζ are equal to x.

<u>3) The spread operator;</u> $\Theta^s : \bar{R}_\delta \rightarrow \bar{R}_\delta$ is defined as follows:

$$[\Theta^s\, \zeta](t) = \begin{cases} \zeta((t+s)\div(s+1)) & t=1, s+2, 2s+3, \ldots, (i-1)s+i, \ldots \\ \delta & \text{otherwise} \end{cases}$$

In other words, Θ^s inserts s don't care elements between Successive elements of its operand. With ζ of (29), we have $\Theta^2 \zeta = z_1, \delta, \delta,\ z_2, \delta, \delta,\ z_3, \ldots$.

<u>4) The Multiplexing operator:</u> $M^{w1,\ldots,wn}_{r,[x]}$; $[\bar{R}_\delta]^n \rightarrow \bar{R}_\delta$, is defined to model a multiplexer that has n inputs. It starts operation at time r (1 if r is omitted), and, periodically, samples its inputs with the ratio $w_1:\ldots:w_n$. The output for the first r-1 time units is set to x (δ if x is omitted). More specifically, if $K=w_1+\ldots+w_n$ is the multiplexing period, then

$$[M^{w1:\ldots:wn}_{r,[x]}\ (\xi_1, \ldots, \xi_n)](t) = \begin{cases} x & \text{if } t<r \\ \xi_e(t) & \text{if } t\geq r \end{cases}$$

where e is the largest integer between 1 and n such that the remainder of $(t-r)\div K$ is less than $w_1+\ldots+w_e$. For example, with ζ as in (29), and

$$\eta = y_1, y_2, y_3, \ldots \tag{30}$$

we have $M^{2,1}_{2,[x]}(\zeta,\eta) = x, z_2, z_3, y_4, z_5, z_6, y_7, \ldots$. Note that if $w_n=\infty$, then $M^{w1,\ldots,wn}$ may be used to model an n-phase cell, where each phase $e=1,\ldots,n-1$ executes for w_e time units, and the last phase executes from time $r+w_1+\ldots+w_{n-1}$ until infinity.

5) The piping operator: $P_n^k(\xi_1,...\xi_n)$: $[\bar{R}_\delta]^n \rightarrow \bar{R}_\delta$, concatenates the first k elements of its operands $\xi_1,...,\xi_n$, into one long sequence. For example if ζ and η are as in (29) and (30), respectively, then

$$P_2^3(\zeta,\eta) = z_1,z_2,z_3,y_1,y_2,y_3,\delta,\delta,...$$

For simplicity, we write $P_n^k(\xi_1,...,\xi_n)$ as $P^k_{e=1,n}(\xi_e)$.

The following properties, may be directly verified from the definitions of the sequence operators:

Property P1: $\Theta^s\ \Omega_x\ \zeta = \Omega_x\ \Omega_\delta^s\ \Theta^s\ \zeta$

Property P2:

$$\Omega_x^r\ M_1^{w1,..,wn}(\zeta_1,...,\zeta_n) = M_{r+1,[x]}^{w1,..,wn}(\Omega_x^r\ \zeta_1,...,\Omega_x^r\ \zeta_n)$$

Property P3: If 'op' is an element-wise operation, then

$$\Omega_s^r\ [\zeta\ \text{'op'}\ \eta] = \Omega_s^r\zeta\ \text{'op'}\ \Omega_s^r\eta$$
$$\Theta^s\ [\zeta\ \text{'op'}\ \eta] = \Theta^s\zeta\ \text{'op'}\ \Theta^s\eta$$

REFERENCES

[1] Cappello, P. and Steiglitz, K. "Unifying VLSI Design with Geometric Transformations," Proc. Int. Conf. on Parallel Processing, pp. 448-457, (1982).

[2] Chen, M. "Synthesizing Systolic Designs," Tec. Report DCS/RR-374, Yale University, (March 1985). (Also in Proc. Int. Sypm. on VLSI Technology, Systems and Applications, Taipei-Taiwan, May 1985).

[3] Delosme, J. and Ipsen, I. "Efficient Systolic Arrays for the Solution of Toeplitz Systems: An Illustration of a Methodology for the Construction of Systolic Architectures in VLSI," Tec. Report DCS/RR-370, Yale University, (June 1985).

[4] Guibas, L.J., Kung, H.T. and Thompson, C.D. "Direct VLSI Implementation of Combinatorial Algorithms," Proc. of Caltech on VLSI, (1979).

[5] Johnsson, L. and Cohen, D. "A Mathematical Approach to Modelling Flow of Data and Control in Computational Networks," in VLSI Systems and Computations, ed. by H.T. Kung, B. Sproull and G. Steele, Computer Science Press, pp. 213-225, (1981).

[6] Kuhn, R. "Transforming Algorithms for Single Stage and VLSI Architectures." Proc. Workshop on Interconnection Networks for Parallel and Distributed Processing, pp. 11 19, (April 1980).

[7] Kung, S.Y., Arun, K., Gab-ezer, R. and Rao, B. "Wavefront Array Processor, Language, Architecture and Applications," IEEE Trans. on Computer, C-31, pp. 1054-1066, (1982).

[8] Kung, H.T. and Lin, W. "An Algebra for VLSI Algorithm Design," Proc. Conf. on Elliptic Problem Solvers, (1983).

[9] Lam, M. and Mostow, J. "A Transformational Model of VLSI Systolic Design," Computer, pp. 42-52, (February 1985).

[10] Leiserson, C. and Saxe, J. "Optimizing Synchronous

Systems," J. VLSI and Computer Systems, vol 1, pp. 41-68, (1983).

[11] Li, G. and Wah, B. "The Design of Optimal Systolic Arrays," IEEE Trans. on Computers, C-34, pp. 66-77, (January 1985).

[12] Melhem, R. "Formal Analysis of a Systolic System for Finite Element Stiffness Matrices," Journal of Computer and System Sciences, Vol 31-1, pp.1-27, (Aug. 1985).

[13] Melhem, R. "Verification of a Class of Deadlock-free, Self-timed Computational Networks", TR ICMA-84-76, The University of Pittsburgh, (Aug. 1984).

[14] Melhem, R. and Rheinboldt, W. "A Mathematical Model for the Verification of Systolic Networks," SIAM J. on Computing, 13(3), pp. 541-565, (August 1984).

[15] Miranker, W. and Winkler, A. "Space Time Representation of Computational Structures," Computing, 32, pp. 93-114, (1984).

[16] Moldovan, D. "On the Analysis and Synthesis of VLSI Algorithms," IEEE Trans. on Computers, C-31(11), pp. 1121-1126, (November 1982).

[17] Quinton, P. "Automatic Synthesis of Systolic Arrays from Uniform Recurrent Equations," Proc. 11th Annual Symp. on Computer Architecture, pp. 208-214, (1984).

[18] Ramakrishnan, I., Fussel, D. and Silberschatz, A. "On Mapping Homogeneous Graphs on a Linear Array Processor Model," Proc. Int. Conf. on Parallel Processing, pp. 440-447, (1983).

[19] Wah, B., Li, G. and Yu, C. "Multiprocessing of Combinatorial Search Problems," Computer, pp. 93-108, (June 1985).