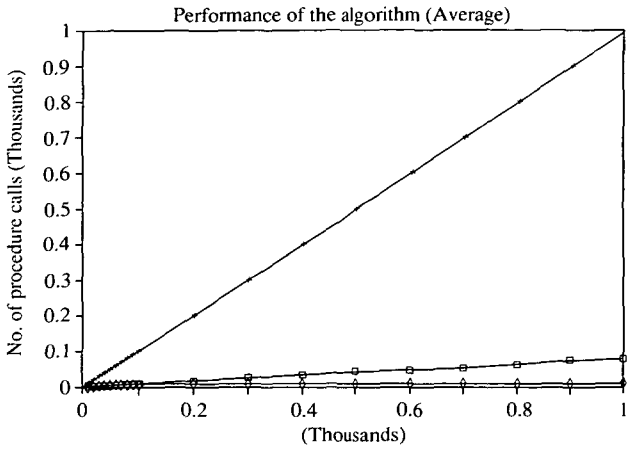

Figure 4. Number of nodes: □ swapping; + number of nodes; ◇ log (number of nodes).

in the tree. During the insertion process, the number of swapping required to insert a node at level $l, (0 \leqslant l \leqslant h)$, is $n-2^{h-(l-1)}-1$, $h = [\log_2(n+1)]$. Gerasch's algorithm[7] produces a nearly optimal tree with swapping as the basic operation. In this algorithm[7] imbalancing occurs at a node P if a subtree of P has more number of complete levels than the other subtree of P. In this case, P becomes the pivot node for rebalancing and keys are displaced in an 'inorder' fashion to make room for the new key. The key that is displaced out of the subtree of P replaces the key in P that in turn becomes the key to insert into the other subtree of P in order to complete the appropriate incomplete level of the subtree. Therefore for trees with nodes just above 2^k, for any k, Gerasch's algorithm runs approximately in logarithmic time. In a similar situation the present algorithm runs in linear time as a consequence of the strict balancing condition for maintaining the optimal shape. In average situations, the algorithm runs in between logarithmic and linear with the number of nodes in the tree. Empirical studies of the present algorithm reveal that the performance of the algorithm in average situation is close to $\log_2(n)$ (Fig. 4). Since the optimal shape is maintained through a series of local rearrangements, in average situations the tree reorganisation is limited to local changes.

4. Conclusion

We have presented a strategy to create an optimal binary search tree. Using swapping as the basic operation, the algorithm dynamically keeps the tree optimal during the updates. In worst case situations, the number of swappings during a single insertion is at most n, where n is the number of nodes in the tree. Hence the algorithm is significant while considering the static optimal binary search trees.

Acknowledgement

The authors are thankful to the referee for his valuable suggestions which improved the earlier version of this paper. One of the authors(APK) is grateful to the University Grants Commission(INDIA) for providing a research fellowship to carry out this research work.

A. P. KORAH and M. R. KAIMAL
Department of Computer Science,
University of Kerala,
Trivandrum, India 695 034

References

1. G. M. Adel'son-vel'skii and E. M. Landis, An algorithm for the organization of information. *Dokl. Akad. Naak, USSR* **146** (2), 263–266 (1962).
2. J. L. Bently, Multidimensional binary search tree used for associative searching, *Communications of the ACM* **18** (9), 509–517 (1975).
3. H. Chang and S. S. Iyangar, Efficient algorithms to globally balance a binary search tree. *Communications of the ACM* **27** (7), 695–702 (1984).
4. J. Culberson and J. I. Munro, Explaining the behaviour of binary search trees under prolonged updates: A model and simulations. *The Computer Journal* **32** (1), 68–75 (1989).
5. A. C. Day, Balancing a binary tree. *The Computer Journal* **19** (4), 360–361 (1976).
6. J. L. Eppinger, An empirical study of insertion and deletion in binary trees. *Communications of the ACM* **26** (9) (1983).
7. T. E. Gerasch, An insertion algorithm for a minimal internal path length binary search tree. *Communications of the ACM* **31** (5), 579–585 (1988).
8. G. H. Gonnet, Balancing binary tree by internal path reduction. *Communications of the ACM* **26** (12), 1074–1081 (1983).
9. D. E. Knuth, *The Art of Computer programming, vol. 3 Sorting and Searching*, Addison-Wesley, Reading Massachusetts (1973).
10. W. A. Martin and D. N. Ness, Optimal binary trees grown with a sorting algorithm. *Communications of the ACM* **15** (2), 88–93 (1972).
11. J. Nievergelt and E. N. Reingold, Binary search trees of bounded balance. *SIAM Journal of Computing* **2** (1), 33–43 (1973).
12. Q. F. Stout and B. L. Warren, Tree rebalancing in optimal time and space. *Communications of the ACM* **29** (9), 902–908 (1988).
13. N. Wirth, *Algorithms + Data structures = Programs*. Prentice Hall of India, New Delhi (1988).

On Equivalent Systolic Designs of LU Decomposition and Its Algebraic Representation

Algorithms which are to be mapped onto interconnecting processing elements in order to design a systolic array are conventionally represented by graphs or networks. This paper introduces the concept of algebraic representation and uses a generating function to represent a systolic array.

Received May 1989, revised September 1989

1. Introduction

Conventional design of systolic arrays is based on the mapping of an algorithm onto an interconnection of processing elements. These algorithms are typically described by graphs or networks, where nodes represent processing elements or registers and edges represent interconnections. Although for many purposes these conventional representations are adequate for specifying the VLSI algorithm, the algebraic representation is more suitable for supporting formal manipulation on designs than the graphic or network models.

Space-time recursion equations of parallel algorithm can be naturally represented by an algebraic representation. In this paper we use a generating function to represent a systolic array. It adapts the power series notation to a more algebraic form to aid the specification and design of systolic array. It also provides a global view on the data-interacting activity of a systolic array. Using a generating function as an algebraic representation of a systolic array, properties of linear algebra, such as velocity addition, can be applied to derive different but equivalent designs of a systolic array.

2. Generating function and systolic flow

The proposed generating function[2,9,10] as an algebraic notation for representing a systolic design consists of a collection of data streams. A 'data stream' represents the moving path of a data item, including the relation between the space coordinates of the data item and time. It can be considered as a moving path of a particle in kinetics. A data stream B can be represented by the following generating function:

$$B = \sum_{t \geqslant 0} bX^{i(t)}Y^{j(t)}Z^{k(t)}\tau^t$$

where X, Y, Z are the space axes, τ is the time axis, and b is the data item name of the data stream B. Data item b locates at space $(i(t), j(t), k(t))$ at time t. Let $B(t)$ denote $bX^{i(t)}Y^{j(t)}Z^{k(t)}\tau^t$. Implicitly, data item b of $B(t)$ carries a value which depends on t, but what we are concerned with here is just the position vector $(i(t), j(t), k(t))$ of the moving data item at time t, not how its value is modified in each *PE, hence we only use the data item name b,* not its value, to represent the data stream in the following discussion.

If data item b moves with constant velocity $\mathbf{V} = \Delta i\mathbf{x} + \Delta j\mathbf{y} + \Delta k\mathbf{z}$ from the beginning position $(i(0), j(0), k(0))$, where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are unit

vectors in X, Y, Z directions respectively, then at time t

$$B(t) = bX^{i(t)}Y^{j(t)}Z^{k(t)}\tau^t = bX^{i(0)+t\Delta i}Y^{j(0)+t\Delta j}Z^{k(0)+t\Delta k}\tau^{0+t}.$$

Hence the generating function of data stream B can be written as:

$$B = \sum_{t\geq 0} B(t) = \sum_{t\geq 0} bX^{i(0)+t\Delta i}Y^{j(0)+t\Delta j}Z^{k(0)+t\Delta k}\tau^{0+t}. \quad (2.1)$$

If we define

$$\frac{1}{1-\alpha^{\Delta i}\beta^{\Delta j}\gamma^{\Delta k}\tau^1}$$

as

$$1+\alpha^{\Delta i}\beta^{\Delta j}\gamma^{\Delta k}\tau^t + \alpha^{2\Delta i}\beta^{2\Delta j}\gamma^{2\Delta k}\tau^2 + \alpha^{3\Delta i}\beta^{3\Delta j}\gamma^{3\Delta k}\tau^3 + \ldots = \sum_{t\geq 0}\alpha^{t\Delta i}\beta^{t\Delta j}\gamma^{t\Delta k}\tau^t$$

and the operators α, β, γ are defined as

$$X^m\alpha^n \equiv X^{m+n},\ Y^m\alpha^n \equiv Y^{m+n},\ Z^m\alpha^n \equiv Z^{m+n},$$

hence

$$B = \sum_{t\geq 0} bX^{i(0)}Y^{j(0)}Z^{k(0)}\tau^0\alpha^{t\Delta i}\beta^{t\Delta j}\gamma^{t\Delta k}\tau^t = \frac{1}{1-\alpha^{\Delta i}\beta^{\Delta j}\gamma^{\Delta k}\tau^t}B(0). \quad (2.2)$$

Here, we use X, Y, Z to represent the space coordinates and use α, β, γ to represent the movement of data stream, i.e. they are related to the moving speed of data stream in X, Y, Z dimensions respectively. We call

$$\frac{1}{1-\alpha^{\Delta i}\beta^{\Delta j}\gamma^{\Delta k}\tau^1}$$

a moving operator. When it is applied to a data item, it will make the data item move with velocity

$$\mathbf{V} = \Delta i\mathbf{x} + \Delta j\mathbf{y} + \Delta k\mathbf{z}.$$

Hence, if a data item moves with velocity $\mathbf{V} = \frac{1}{2}\mathbf{x}$, the applicable moving operator is

$$\frac{1}{1-\alpha^{\frac{1}{2}}\beta^0\gamma^0\tau^1}.$$

It can be abbreviated as

$$\frac{1}{1-\alpha^{\frac{1}{2}}\tau^1}.$$

A 'data wave' is a collection of data streams, say B_p, moving in a uniform manner. A data wave can be represented by the following generating function.

$$W = \sum_p B_p \quad (2.3)$$

where p represents different data streams.

A 'systolic flow' is a collection of data waves, say W_q, which in turn can be represented by the following generating function.

$$G = \sum_q W_q = \sum_q \sum_p B_{p;q} = \sum_q \sum_p \sum_{t\geq 0} b_{p;q}X^{i(t)}Y^{j(t)}Z^{k(t)}\tau^t \quad (2.4)$$

where q represents different data waves. In this representation, although we use the same notation $b_{p;q}$ to represent all data items of the data stream $B_{p;q}$ all the time, it should be understood that the Σ_q operator applied on different data waves may modify its carrying value. This modification happens whenever all data waves have equal associative space and time coordinates.

3. Equivalent transformations for systolic array

We say that two systolic systems are equivalent if for a given input they can generate the same output. It is obvious that if interactions (or meetings) of input and output data streams are preserved, from one system to the other system, then both systems are able to generate the same output, and hence they are equivalent. Since interactions of data streams in a systolic system can be represented algebraically by a generating function (it will be illustrated by examples in Section 4), equivalent transformations which preserve the interactions can be found from the rules of linear algebra, such as velocity addition. The transformations can be applied on the generating function so that various complex designs can be derived from a simple one.

Li and Wah[7] have described a systematic design technique for deriving systolic array architecture. They have shown that for a given computation, one can write down a set of vector constraint equations which must be satisfied in any functionally corrected design. These equations relate to the parameters of a systolic array such as velocities of data flow, data distribution, periods of computation, and times of data access etc. These equations are:

$$t_{kx}\mathbf{x}_d + \mathbf{x}_{ks} = t_{kx}\mathbf{z}_d \quad (3.1)$$

$$t_{ky}\mathbf{y}_d + \mathbf{y}_{ks} = t_{ky}\mathbf{z}_d \quad (3.2)$$

$$t_i\mathbf{x}_d + \mathbf{x}_{is} = t_i\mathbf{y}_d \quad (3.3)$$

$$t_i\mathbf{z}_d + \mathbf{z}_{is} = t_i\mathbf{y}_d \quad (3.4)$$

$$t_j\mathbf{y}_d + \mathbf{y}_{js} = t_j\mathbf{x}_d \quad (3.5)$$

$$t_j\mathbf{z}_d + \mathbf{z}_{js} = t_j\mathbf{x}_d. \quad (3.6)$$

By the rule of vector addition, we can see that if $\mathbf{x}_d, \mathbf{y}_d, \mathbf{z}_d$ (velocities of data flow) are in the solution set of the above six equations, then the set of velocities $\mathbf{x}'_d, \mathbf{y}'_d, \mathbf{z}'_d$ such that

$$\mathbf{x}'_d = \mathbf{x}_d + \mathbf{x}_c$$
$$\mathbf{y}'_d = \mathbf{y}_d = \mathbf{x}_c$$
$$\mathbf{z}'_d = \mathbf{z}_d + \mathbf{x}_c$$

where $\mathbf{x}_c$ is an additional velocity, also satisfies the above six equations. Hence, $\mathbf{x}'_d, \mathbf{y}'_d$ and $\mathbf{z}'_d$ are also a valid solution to this computation. That is to say, velocity addition is an equivalent transformation. This type of transformation can be represented by generating function as follows.

Given a systolic system, if we let the whole system move at velocity

$$\mathbf{V}_1 = i_1\mathbf{x} + j_1\mathbf{y} + k_1\mathbf{z},$$

then the velocity of a data stream A which moves at velocity

$$\mathbf{V}_2 = i_2\mathbf{x} + j_2\mathbf{y} + k_2\mathbf{z},$$

originally, is changed to

$$\mathbf{V} = \mathbf{V}_1 + \mathbf{V}_2 = (i_1+i_2)\mathbf{x} + (j_1+j_2)\mathbf{y} + (k_1+k_2)\mathbf{z}.$$

Hence the generating function of data stream A is

$$A = \frac{1}{1-\alpha^{(i_1+i_2)}\beta^{(j_1+j_2)}\gamma^{(k_1+k_2)}\tau^t}A(0) = \sum_{t\geq 0} A(0)\alpha^{(i_1+i_2)t}\beta^{(j_1+j_2)t}\gamma^{(k_1+k_2)t}\tau^t = \sum_{t\geq 0} A(0)\alpha^{i_1 t}\beta^{j_1 t}\gamma^{k_1 t}\tau^t \oplus \sum_{t\geq 0} A(0)\alpha^{i_2 t}\beta^{j_2 t}\gamma^{k_2 t}\tau^t = \frac{1}{1-\alpha^{i_1}\beta^{j_1}\gamma^{k_1}\tau^1}A(0) \oplus \frac{1}{1-\alpha^{i_2}\beta^{j_2}\gamma^{k_2}\tau^1}A(0) \quad (3.7)$$

where '$\oplus$' is a special addition operator. It sums up the respective powers of α, β, and γ, of the same data item at time t. Hence, '$\oplus$' can be considered as vector addition of velocity or displacement vector of the data stream.

Since each data stream of the original system moves with an added velocity $\mathbf{V}_1$, although the interconnection or the number of PEs should be adapted to the new moving velocity of the data stream, it is obvious that the interactions of data streams are preserved, hence the velocity addition is an equivalent transformation.

By using the notation of the generating function, we will see that the velocity addition can be easily represented algebraically, so that equivalent systolic arrays can be simply derived. In this paper we choose LU-decomposition as example to demonstrate these points.

4. An illustrative example

Consider a matrix A which can be decomposed into a lower and an upper triangular matrices by Gaussian elimination without pivoting. VLSI computing structures for LU-decomposition problem have been proposed by Chen,[1] Kung,[4] Kung,[5,6] Hwang and Cheng,[3] Moldovan[8] and others. In this example it is shown that previously proposed architectures can be formally derived by using the generating function and the velocity addition equivalent transformation.

The algorithm for the LU-decomposition of a matrix $A = [a_{ij}]$ is expressed by the following program.[6,8]

```
for i = 1 to n do
  for j = 1 to n do
    a^0_ij = a_ij
for k = 1 to n do
  for j = k to n do
    for i = k to n do
      if i = k then u^i_kj = a^{k-1}_ij
        else u^i_kj = u^{i-1}_kj
      if j = k then l^j_ik = a^{k-1}_ij / u^i_kj
        else l^j_ik = β^{-1}
      a^k_ij = a^{k-1}_ij - l^j_ik * u^i_kj
```

The first design[8] is presented in Fig. 1. The data streams A_{ij} stay at processors while the data streams L_{ik} and U_{kj} which carry zero values initially move to the right and down respectively. Fig. 1 is the initial state of the systolic array for LU-decomposition. Assuming that data item a_{11} is located at the origin of the XY-planes, then the generating functions of the data waves A, L, U at time 0 can be described as follows.

$$\begin{aligned} A(0) &= a_{11}X^0Y^0\tau^0 + a_{12}X^1Y^0\tau^0 + a_{13}X^2Y^0\tau^0 \\ &\quad + a_{21}X^0Y^1\tau^0 + a_{22}X^1Y^1\tau^0 + a_{23}X^2Y^1\tau^0 \\ &\quad + a_{31}X^0Y^2\tau^0 + a_{32}X^1Y^2\tau^0 + a_{33}X^2Y^2\tau^0 \\ &= \sum_{i,j} a_{ij}X^{j-1}Y^{i-1}\tau^0 \end{aligned}$$

$$\begin{aligned} L(0) &= l_{11}X^{-1}Y^0\tau^0 + l_{21}X^{-2}Y^1\tau^0 \\ &\quad + l_{22}X^{-3}Y^1\tau^0 + l_{31}X^{-3}Y^2\tau^0 \\ &\quad + l_{32}X^{-4}Y^2\tau^0 + l_{33}X^{-5}Y^2\tau^0 \\ &= \sum_{i,k} l_{ik}X^{-i+1-k}Y^{i-1}\tau^0 \end{aligned}$$

$$\begin{aligned} U(0) &= u_{11}X^0Y^{-1}\tau^0 + u_{12}X^1Y^{-2}\tau^0 \\ &\quad + u_{22}X^1Y^{-3}\tau^0 + u_{13}X^2Y^{-3}\tau^0 \\ &\quad + u_{23}X^2Y^{-4}\tau^0 + u_{33}X^2Y^{-5}\tau^0 \\ &= \sum_{k,j} u_{kj}X^{j-1}Y^{-j+1-k}\tau^0 \end{aligned}$$

As times go on, the movement of the data waves A, L, U, are synchronized. Since the data streams A_{ij} stay at PEs and data streams l_{ik} and u_{kj} move to right and down respectively, the generating function for data waves A, L and U can be represented as follows:

$$\begin{aligned} A &= \frac{1}{1-\tau^1}A(0) \\ &= \sum_{t\geq 0} A(0)\tau^t \\ &= \sum_{t\geq 0}\sum_{i,j} a_{ij}X^{j-1}Y^{i-1}\tau^t \end{aligned} \tag{4.1}$$

$$\begin{aligned} L &= \frac{1}{1-\alpha^1\tau^1}L(0) \\ &= \sum_{t\geq 0} L(0)\alpha^t\tau^t \\ &= \sum_{t\geq 0}\sum_{i,k} l_{ik}X^{-i+1-k+t}Y^{i-1}\tau^t \end{aligned} \tag{4.2}$$

$$\begin{aligned} U &= \frac{1}{1-\beta^1\tau^1}U(0) \\ &= \sum_{t\geq 0} U(0)\beta^t\tau^t \\ &= \sum_{t\geq 0}\sum_{k,j} u_{kj}X^{j-1}Y^{-j+1-k+t}\tau^t \end{aligned} \tag{4.3}$$

Thus, the systolic flow of LU-decomposition is $G = A + L + U$. In order to perform the LU-decomposition correctly, the data streams A_{ij}, L_{ik}, and U_{kj} of Fig. 1 must meet in the right place at the right time. That is, both the space and time coordinates of these data streams must be identical. Hence, from the x-coordinate of (4.1)–(4.3) we get $-i+k+t = j-1$ and from the y-coordinate of (4.1)–(4.3) we have $i-1 = -j+l-k-t$. The result is the following indicial equation

$$k = -i-j+2+t \tag{4.4}$$

Here k is a monotonic function of t for given

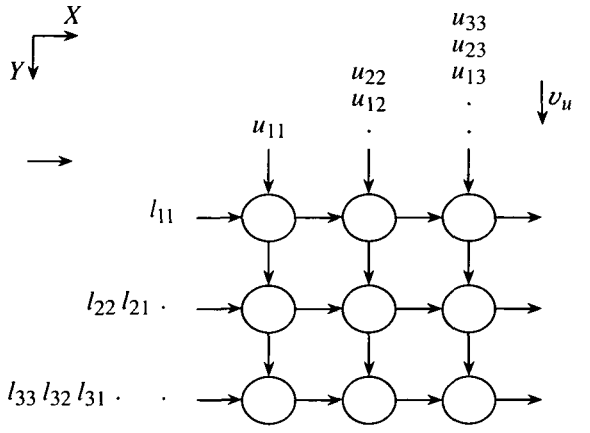

Figure 1. Systolic array for LU-decomposition.

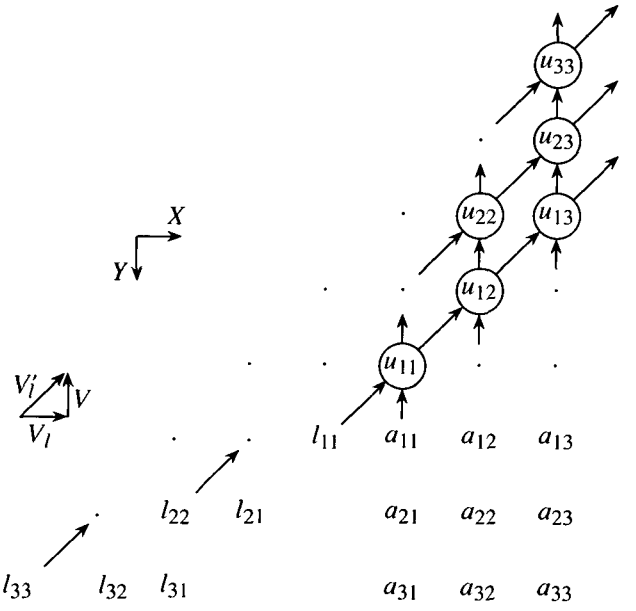

Figure 2. Systolic array for LU-decomposition with added velocity V′ = −y.

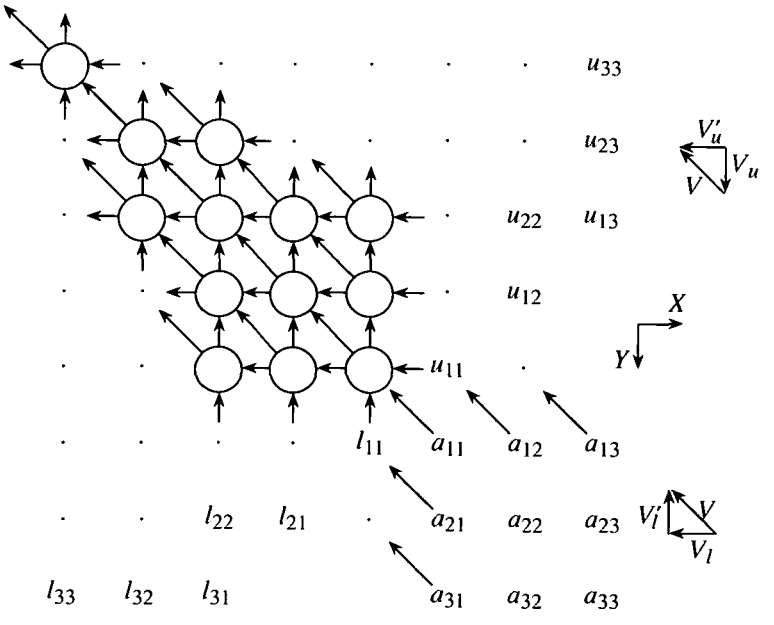

Figure 3. Systolic array for LU-decomposition with added velocity V = −x−y.

i,j. Hence for all i,js there must exist ts such that k has values of 1, 2, 3, ... respectively. Hence the correctness of the first design (Fig 1) can be verified by using (4.4).

In the following, we will describe other systolic designs for LU-decomposition. We have found that each of these designs can be derived from the first design by simply adding a constant velocity, say $\mathbf{V} = u\mathbf{x} + v\mathbf{y}$, to all the data streams $A_{ij}(t)$, $L_{ik}(t)$, and $U_{kj}(t)$. After the velocity addition, the data streams $A_{ij}(t)$, $L_{ik}(t)$ and $U_{kj}(t)$ are transformed into

$$A = \frac{1}{1-\tau^t} A(0) \oplus \frac{1}{1-\alpha^u\beta^v\tau^t} A(0)$$

$$= \frac{1}{1-\alpha^u\beta^v\tau^1} A(0)$$

$$= \sum_{t \geqslant 0} A(0)\alpha^{ut}\beta^{vt}\tau^t$$

$$= \sum_{t \geqslant 0} \sum_{i,j} a_{ij} X^{j-1+ut} Y^{i-1+vt} \tau^t \qquad (4.5)$$

$$L = \frac{1}{1-\alpha^1\tau^1} L(0) \oplus \frac{1}{1-\alpha^u\beta^v\tau^1} L(0)$$

$$= \frac{1}{1-\alpha^{1+u}\beta^v\tau^1} L(0)$$

$$= \sum_{t \geqslant 0} L(0)\alpha^{(1+u)t}\beta^{vt}\tau^t$$

$$= \sum_{t \geqslant 0} \sum_{i,k} l_{ik} X^{-i+1-k+(1+u)t} Y^{i-1+vt} \tau^t \qquad (4.6)$$

$$U = \frac{1}{1-\beta^1\tau^1} U(0) \oplus \frac{1}{1-\alpha^u\beta^v\tau^1} U(0)$$

$$= \frac{1}{1-\alpha^u\beta^{1+v}\tau^1} U(0)$$

$$= \sum_{t \geqslant 0} U(0)\alpha^{ut}\beta^{(1+v)t}\tau^t$$

$$= \sum_{t \geqslant 0} \sum_{k,j} u_{kj} X^{j-1+ut} Y^{-j+1-k+(1+v)t} \tau^t. \qquad (4.7)$$

From (4.5)–(4.7) we can derive its corresponding indicial equation $k = -i-j+2+t$. Since this derived indicial equation is the same as (4.4), we see that the new design preserves the execution sequence of the first design in Fig. 1, hence it preserves the computation. Therefore, they are equivalent.

Figures 2–5 are some systolic designs for LU-decomposition which are transformed from Fig. 1 with an added velocity of

$$\mathbf{V} = -\mathbf{y}, \quad \mathbf{V} = -\mathbf{x}-\mathbf{y},$$
$$\mathbf{V} = -\tfrac{1}{3}\mathbf{x}-\tfrac{1}{3}\mathbf{y}$$

and

$$\mathbf{V} = -\tfrac{1}{2}\mathbf{x}-\tfrac{1}{2}\mathbf{y}, \text{ respectively.}$$

5. Discussion

Theoretically, the searching space for finding a feasible added velocity vector is infinite, hence we can select the velocity with any direction and with any speed. Since the computation can only be performed when data streams are met in each *PE*, if we let the interconnection primitives of our array structure to be 8-neighbour bidirectional connected and the distance of the directly connected *PE* is less than or equal to 1 in *X*-, *Y*-axis respectively, the speeds of data streams after the operation of velocity addition are still less than 1. Otherwise there must have a longer link to propagate data, and this is outside the scope of our discussion.

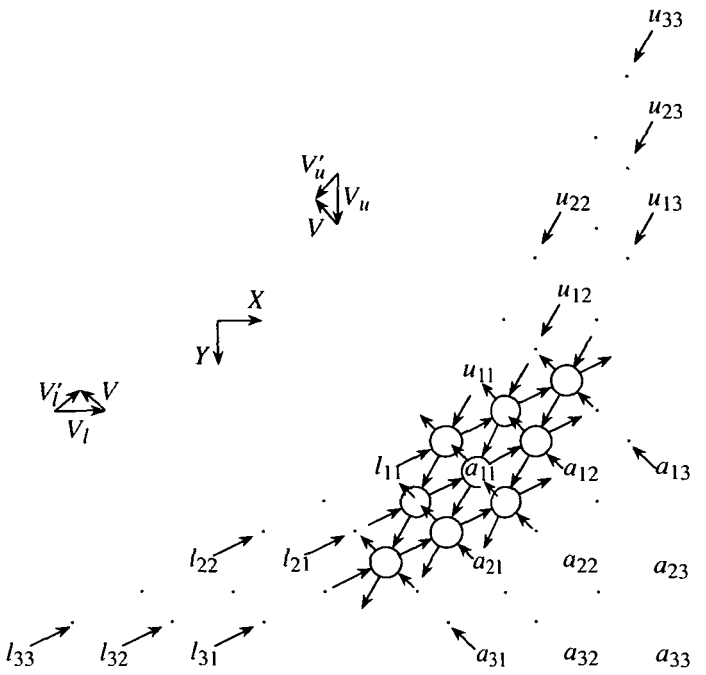

Figure 4. Systolic array for LU-decomposition with added velocity $\mathbf{V}' = -\frac{1}{3}\mathbf{x} - \frac{1}{3}\mathbf{y}$.

Hence from (4.5)–(4.7), we can get the constraint equations for feasible velocity addition operation. From *X*-coordinate we get

$$|u| \leqslant 1$$
$$|1+u| \leqslant 1$$

and from *Y*-coordinate we have

$$|v| \leqslant 1$$
$$|v+1| \leqslant 1$$

The results are $-1 \leqslant u \leqslant 0$ and $-1 \leqslant v \leqslant 0$. This means the feasible solution of velocity addition operation are northwestward. That is why all feasible designs proposed by various researchers can be found by adding a velocity with northwest direction.

For comparison, the performance of various equivalent systolic designs for LU-decomposition is summarized in Table 1.

The array size of Fig. 1 is n^*n. All the *PE*s have the same architecture. However, their function at any given moment may differ. Its advantages are that the pipelining period (δ) is equal to 1 and there is no diagonal connection.

Figure 2 is the same as Fig. 1 in terms of *PE* function, pipelining period, and connection number. But it needs only half the *PE*s of design 1 and thus saves hardware.

The pipelining period of Fig. 3 is also equal to 1. But the array size is equal to $n^2+(n-1)^2$.

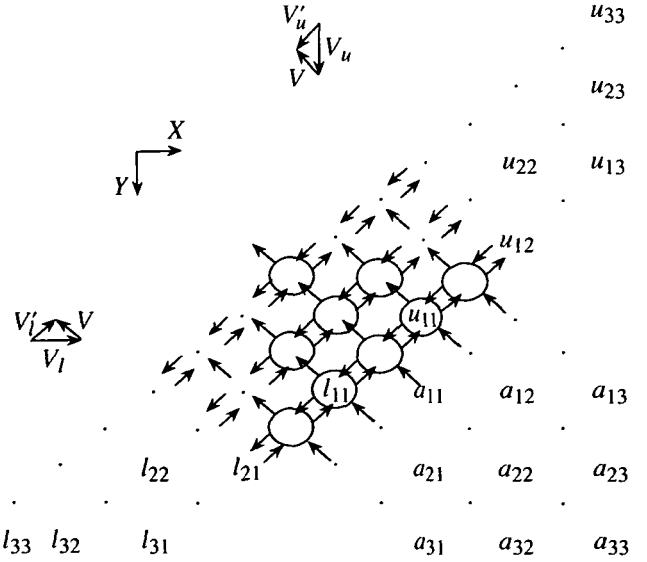

Figure 5. Systolic array for LU-decomposition with added velocity $\mathbf{V}' = -\frac{1}{2}\mathbf{x} - \frac{1}{2}\mathbf{y}$.

Table 1. Performance comparison of systolic arrays for LU-decomposition

Fig.	# *PE*	δ	Diagonal connection	Preloading	Execution	Draining
1	n^2	1	No	n	$3n-2$	$n-1$
2	$n(n+1)/2$	1	No	0	$3n-2$	$n-1$
3	$n^2+(n-1)^2$	1	Yes	0	$3n-2$	1
4	n^2	3	Yes	$n-1$	$3n-2$	1
5	$n(n+1)/2$	2	No	0	$3n-2$	1

Hence it needs more hardware cost. But there is no need for preloading time of A_{ij}, hence this design has the minimum computation time.

The array size of Fig. 4 is equal to n^2, but there is no need for processor reprogramming. The function of *PE*s are fixed all the time, i.e. the first column *PE*s perform division, the first two *PE*s only pass data, and the inner *PE*s perform the multiply-and-add operation. Its disadvantage is that the pipelining period is equal to 3.

The function of each *PE* in Fig. 5 may change at one given moment and its pipelining period is equal to 2. But it needs only $n*(n+1)/2$ *PE*s and there is no need for preloading time; it also has the advantage of diagonal connection.

6. Conclusion

Conventional design of systolic arrays is based on the mapping of an algorithm onto an interconnection of processing elements. This mapping is done in an *ad hoc* manner. In this paper we present a notation of the generating function which is a mathematical formal approach to represent systolic arrays. With the properties of linear algebra, it supports the transformation between various equivalent systolic designs.

Y-C. HOU* and J-C. TSAY
Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C.

* To whom correspondence should be addressed.

References

1. M. C. Chen, Synthesizing systolic designs, *International Symposium on VLSI Technology, System and Applications*, pp. 209–215 (1985).
2. Y. C. Hou and J. C. Ysay, An algebraic model for representing equivalent designs of systolic arrays. *Proc ISMM International Symposium*, Florida, pp. 163–166 (1988).
3. K. Hwang and Y. H. Cheng, VLSI computing structures for solving large scale linear system of equations, *Proc. Parallel Processing Conference*, pp. 217–227 (1980).
4. H. T. Kung, Highly concurrent systems, *Introduction to VLSI System*, edited C. A. Mead and L. A. Conway, Reading, MA; Addison-Wesley (1980).
5. S. Y. Kung, On supercomputing with systolic wavefront array processors, *Proc. IEEE*, **72** (7), 867–884 (1984).
6. S. Y. Kung, *VLSI Array Processors*. Prentice-Hall (1988).
7. G. J. Li and B. W. Wah, The design of optimal systolic arrays. *IEEE Trans. on Computers*, **C-34**, 66–77 (1985).
8. D. I. Moldovan, On the design of algorithms for VLSI systems. *Proc. IEEE*, **71** (1),m 113–120 (1983).
9. J. C. Tsay and Y. C. Hou, Generating function and equivalent transformation for systolic arrays, to appear in *Parallel computing*, North-Holland.
10. S. Yuan, *An Algebraic Notation for the Design and Verification of Systolic Arrays*, M.S. Thesis, National Chiao Tung University, Hsinchu, R.O.C. (1987).

Reverse Chaining for Answering Temporal Logical Queries

A possible structure for the past data of a partitioned temporal database is reverse field chaining. Under this technique field versions are chained by descending time. Exact analysis derives the expected number of block accesses when logical queries against a partitioned temporal database with reverse chaining are satisfied. Numerical results are given.

Received November 1989, revised November 1990

1. Reverse chaining

A very efficient structure for answering queries based on secondary key values against a temporal database is *reverse chaining*, according to which field versions are chained by decreasing time. The motivation of the present work is a previous effort on the performance analysis of such systems,[1] where a formula was given deriving the cost of answering logical queries by using reverse chaining. However, the analysis was approximate and resulted in a pessimistic evaluation. Here, we give an improved mechanism for traversing many reverse chains by using some sort of parallelism instead of a one-a-time chain processing.

Suppose that the original layout of a record from a personel file of a conventional database is:

EMP(empno,name,position,salary).

This file in database applications with time support should have a different layout. If field version chaining is applied and the user queries concern the fields 'position' and 'salary' then the previous layout should be:

EMP(empno,name,position,*p_chain*,
salary,*s_*chain,time)

Figure 1 is a graphical representation of this layout.

Suppose that the following logical query is posed in this file form: 'Retrieve the salaries and the positions of empno = x for the past three years'. This logical query is decomposed in two simple queries as: 'Retrieve the salaries of empno = x for the past three years' and 'Retrieve the positions of empno = x for the past three years'.

In the absence of any indexes for accessing history data the relevant records should be retrieved by following the two chains. In Ref. 1 it was assumed that the two chains are followed independently from each other. Therefore the total cost was considered to be the sum of the costs for answering each single query, where the cost of each single query is given by the ubiquitous formula by Yao[3] for the expected number of block accesses when a number of fixed length file records is randomly selected.

The two chains of records may pass through the same blocks. If great buffer space is available then block accesses may be saved by navigating simultaneously through the chains instead of searching the chains one after the other. These savings are possible because the past data file, in essence, may be considered as a sequential heap with the attribute 'time', standing for 'valid time from' or 'transaction time', as the primary key record.

The term simultaneous navigation needs some explanation. Every block appended to the heap file has greater address than the predecessor blocks ones'. This block, also, contains records with 'time' field values greater than the corresponding values of the previous blocks. The search in one chain is performed by following the pointer to the next version in the chain (previous in time). The chain to be processed next will be chosen by following the rule: pick the chain pointing to a block with address greater than the address pointed by the other chain. Evidently, it is not certain that any two consecutive searches do correspond to the same chain. If the block to be fetched resides already in main memory, because it belongs to the other chain too, then no I/O cost is paid. Instead, negligible CPU cost is paid to examine the records of the specific block, read the two chain pointers and decide which one to follow. This method is similar to the traversing of multilist files in parallel as reported by Claybrook[2]. In addition if the cost has to be computed in advance for taking decisions on query optimization, then the cost of independent chain search should be replaced by the cost of simultaneous chain search. This is the contribution of our work.

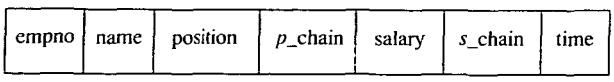

Figure 1. Temporal database record layout.