Design of one-dimensional systolic-array systems for linear state equations

C.-W. Jen
S.-J. Jou

Indexing terms: Electronic circuits, Matrix algebra

Abstract: To solve linear state equations, a two-dimensional systolic-array system has been proposed [9]. For the same purpose, various kinds of one-dimensional arrays are designed in the paper. The linear systolic-array system with first-in-first-out (FIFO) queues can be designed by applying double projections from the three-dimensional dependence graph (DG). As the array thus designed needs processors with multifunction operations and various input/output requirements, tag control bits are incorporated, and so make the overall computation more efficient. Furthermore, a linear systolic-array system with content addressable memory (CAM) is designed which can use the advantage of matrix sparseness to reduce the overall computation time. The partition scheme of the linear systolic-array system is also proposed to match the limitation of the *pin* number and the chip area. Finally, the cost and performance of all the class of systolic-array systems for solving linear state equations are illustrated.

1 Introduction

In many applications, such as the emulation of control system and the transient analysis in circuit simulation, it is necessary to solve the linear state equation $\dot{V}(t) = A'V(t) + CU(t)$, where $V(t)$ is a vector variable of size n, $\dot{V}(t)$ is the time derivative of $V(t)$, $U(t)$ is the input vector of size m, A' and C are $n \times n$ and $n \times m$ matrixes, respectively. To solve this linear state equation in a discrete-time system, the numerical integration method (for example, the backward Euler method) is often used to transform the differential equations into the following discrete-time matrix form:

$$AV(t_n) = B'(t_{n-1}) + CU(t_n) \qquad (1)$$

where $A = [A' + (1/h)I]$, $h = t_n - t_{n-1}$, $B'(t_{n-1}) = (1/h)V(t_{n-1})$, I is the identity matrix and t_n is the time at the nth step. The linear state equations have to be solved many times in some application algorithms because the linear state equations reside in the time loop of the algorithms. Therefore, the computational time for solving eqn. 1 is usually the dominant factor of the computational time of the algorithm, especially when n is large.

To speed up the computation of solving eqn. 1, parallel processing techniques may be used. As we know, the systolic array [1] is a synchronous VLSI computing network composed of many processor elements (PE) and local interconnection lines. It exploits the great potential of pipelining and multiprocessing which can solve computation-bound problems very efficiently. In the past few years, several systematic design methods [2–8] have been proposed to synthesise the systolic array.

To effectively solve eqn. 1, a two-dimensional systolic-array system has been successfully designed by the authors [9]. In this design, the matrix-vector multiplication–accumulation process was chosen to compute $B(t_n) = B'(t_n) + CU(t_n)$ and then the Gauss–Jordan algorithm was selected to solve $AV(t_n) = B(t_n)$. These two algorithms were described in a locally recursive single-assignment (LRSA) form, by which the geometry representations of each algorithm (i.e. the dependence graph (DG) [2, 3, 9]) can be easily derived and then linked together. Because of the different functions of nodes and input/output requirements on the DG, tag codes have been added to the index nodes [11]. The concepts of adding the tag codes are described in detail in Reference 9. Their LRSA form and the linked DG with a tag code for the case $n = m = 4$ are shown in Figs. 1 and 2, respectively. From this DG, a two-dimensional systolic-array system with tag bits has been designed by applying the time-scheduling and node-assignment projection procedure [16] along the i direction, as shown in Fig. 3. The computation time, i.e. latency, is reduced from $O[(n^3/2) + n \times m]$ to $4n - 3(4n - 2)$ if $m < n$ ($m = n$) and the block pipelining rate is n. But the two-dimensional systolic-array system uses $[(n^2 + n)/2 + m]$ PEs, $(n + 2m + 1)$ input ports and one output port, respectively. For large n, the PE number of a two-dimensional systolic-array system may be too large to be implemented in a VLSI chip.

The linear state equations may also be solved by a one-dimensional linear systolic array. Compared with the two-dimensional array, the linear array is better owing to its simple hardware, and it also possesses some merits such as easy extension, simple interconnections and easy incorporation of the fault tolerance design [10]. In this paper, several kinds of linear systolic-array systems are proposed. To design the linear systolic-array system by intuition, the original two-dimensional systolic-array system for solving the linear state equations is projected once again. Hence, double projections are applied onto the DG and the index space is reduced from three dimensions to one dimension [16]. As there are fewer PEs the latency should be increased.

Paper 7033G (E10), first received 15th November 1988 and in revised form 26th September 1989

The authors are with the Institute of Electronics, National Chiao Tung University, 75 Po-Ai St. Hsinchu, Taiwan, Republic of China

```
//$B = B' + C \times U$; $C, B: n \times m$; $U, B': n \times 1$//
INPUT: $C(i,j)$, $B'(i)$, $U(0,j) = U(j)$;
OUTPUT: $B(i,m)$;
FOR($i$ from 1 to $n$){
  $B(i,0) = B'(i)$;
  FOR($j$ from 1 to $m$){
    $U(i,j) = U(i-1,j)$;
    $B(i,j) = B(i,j-1) + C(i,j) \times U(i,j)$;
  }
}
```

a

```
//$AX = B, A'' = [A \mid B']$, $A: n \times n, B: n \times 1, n1 = n + 1$, uniform//
INPUT: $A''(i,j,0) = A''(i,j)$;
OUTPUT: $A''(i,n1,n)$, $i = n$ to $2n - 1$;
FOR ($k$ from 1 to $n$){
  FOR ($i$ from $k$ to $n + k - 1$){
    $D(i,k,k) = A''(i,k,k-1)$;
    IF ($i$ equal $k$)
      FOR ($j$ from $k + 1$ to $n1$){
        $D(i,j,k) = D(i,j-1,k)$;
        $C(i,j,k) = A''(i,j,k-1)/D(i,j,k)$;
      }
    ELSE
      FOR ($j$ from $k + 1$ to $n1$){
        $D(i,j,k) = D(i,j-1,k)$;
        $C(i,j,k) = C(i-1,j,k)$;
        $A''(i,j,k) = A''(i,j,k-1) - D(i,j,k) \times C(i,j,k)$;
      }
  }
  FOR ($j$ from $k + 1$ to $n1$)
    $A''(n+k,j,k) = C(n+k-1,j,k)$;
}
```

b

Fig. 1 *LRSA form*

a Matrix-vector multiplication accumulation
b Uniform Gauss–Jordan algorithm

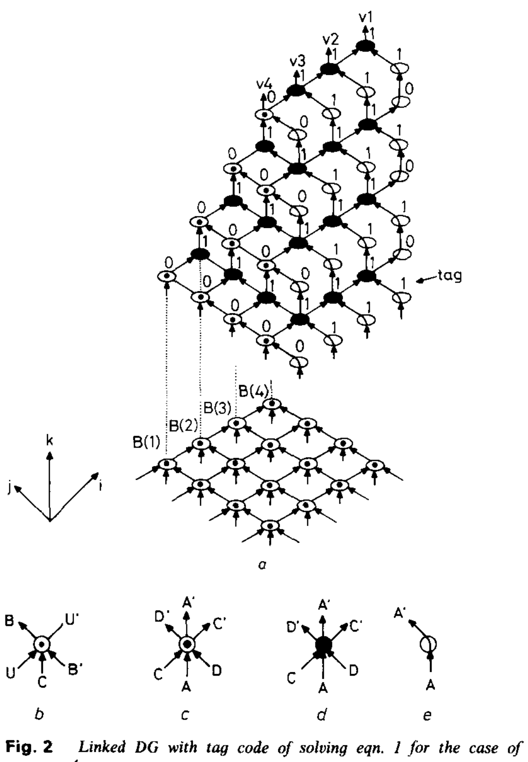


Fig. 2 *Linked DG with tag code of solving eqn. 1 for the case of $n = m = 4$*

b $B = B' + C \times U$, $U' = U$ *c* $C' = A/D$, $D' = D$, $A' = C$ *d* $A' = A - D \times C$, $C' = C$, $D' = D$ *e* $A' = A$

2 Linear systolic-array system with FIFO memory

To design the one-dimensional systolic-array system, one promising way is that the second projection is directly applied onto the designed two-dimensional systolic-array system which is shown in Fig. 3. By doing so, the same

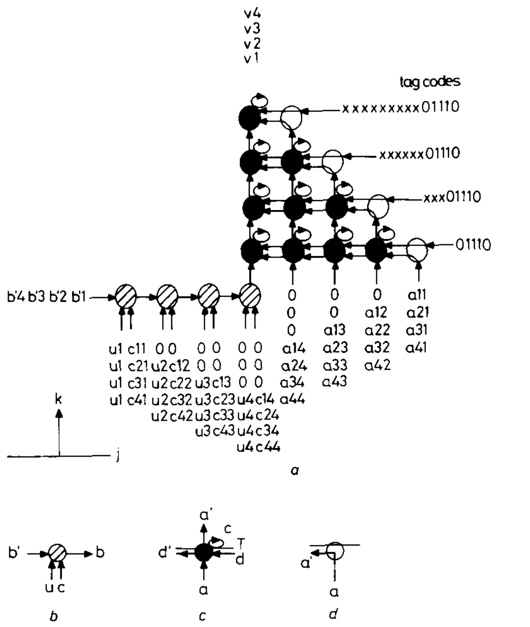


Fig. 3 *Two-dimensional systolic-array system by projecting along i direction*

a and *c* T (Tag):

$$0\begin{cases} a' = C \\ c = a/d \\ d' = d \end{cases} \qquad 1\begin{cases} c = c \\ a' = a - d \times c \\ d' = d \end{cases}$$

x — don't care

b $b = b' + cu$ *d* $a' = a$

projection design method as applied to the DG may be used. But first we have to attend to the difference between the array and the DG. In contrast to the array, the DG is an acyclic-directed graph which is the geometry representation of an algorithm and is constructed from a LRSA form [2, 3, 9]. Although there is much correspondence between the PEs and nodes, the iterated lines and dependence arcs in the array and graph, respectively, the major difference is that each PE usually processes a sequence of data, whereas the DG each node calculates one datum only. Thus the procedures of the second projection may therefore be the same as the first projection, except the time scheduling. To obtain the correct time scheduling during the second projection, one promising way is to first group the time for processing a sequence of data for each original PE in the two-dimensional array as a packeted time cluster, and then to make the global scheduling for different packeted time clusters associated with different original PEs. In this way, the local FIFO queue is required to store the intermediate results if iterated links exist along the secondary projection direction among the original PEs.

As a result, in the second projection, the computations associated with PEs along the node-assignment direction are projected into one new PE. Thus, the pipelining computations of the data through the PEs along the projection direction are replaced by sequential computation in one new PE. To maintain correct data processing, the pipelining data along the second projection direction in the two-dimensional systolic array must be saved in a sequence according to the second time-scheduling function, so that the PEs in the one-dimensional systolic array can re-use it at the right time to perform all the original computation along the projection direction. Note that, since during the projection the data computation sequence of the two-dimensional systolic array is still maintained, no extra controls or global lines are needed. Therefore, by applying this time-scheduling node-assignment projection procedure, a one-dimensional systolic array with a local FIFO stage has been derived. The largest size of FIFO memories can be determined by the number of data computation of a PE × the number of data links along the projection direction.

Observing Fig. 3, there are two permissable directions, k and j, for the second projection on the two-dimensional array, and the corresponding linear array structures are shown in Figs. 4*a* and *b*, respectively. Throughout the

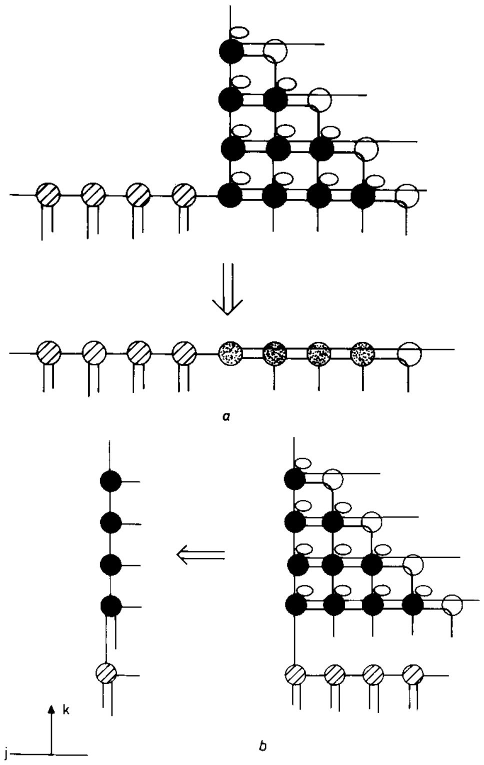


Fig. 4 *Linear array structure*
a Second projection direction is along the k axis
b Second projection direction is along the j axis

first and the second projections, a plane of index nodes on the DG are projected into one PE. Consequently, the index nodes with different functions will be mapped into a PE, and so the PE should have the capability to perform different functions. To do this, the appropriate control signals must be sent to the PE so that the correct function of the PE will be performed in good time. In our approach, tag codes [11] are added to the DG to distinguish the different functions on the index nodes. The tag code assignment is different from that of single projection in which only a line of index nodes are mapped into a PE. For double projections along the i and k or i and j directions, the index nodes of the k–i or j–i plane of the DG, respectively, will be mapped into a PE. Nodes with different functions in the same plane must therefore be assigned by different tag codes, so we assign 0′, 0″, 1, 2, 3′, 3″, 4 to the index nodes shown in Figs. 5*a* and *b*. Although the nodes marked with grey dots have the same function as others, their input data source is different from the others so we assign them with different codes (0′ and 1). In Fig. 5*a*, we add four transfer nodes to the top of the plane to pop out the data. Note that the functions of nodes 0′ and 0″, 3′ and 3″ (marked by 0 and 3, respectively) can be combined because no functional conflicts. Therefore, it needs five tag codes in these DGs in contrast to two tag codes in single projection cases. The final designs are shown in Figs. 6 and 7 which correspond to Figs. 4*a* and *b*, respectively. The FIFO length of each PE is four because the PEs in Fig. 6 and Fig. 7 need four data computation. The data sequence of inputs, tag bits and the snap shot of operation at some operation instants is also shown in Fig. 6.

The operating functions of each PE and tag codes are also shown in the inset of Figs. 6 and 7. With more functions mapped into one PE, it is accordingly a little more complex than Fig. 3. The latencies in Figs. 6 and 7 are still the same as they need $n(n+1)+n+n=n^2+3n-1$ and $(n+1)(n+1)-1+n-1=n^2+3n-1$ clocks, respectively. This is because in each elimination

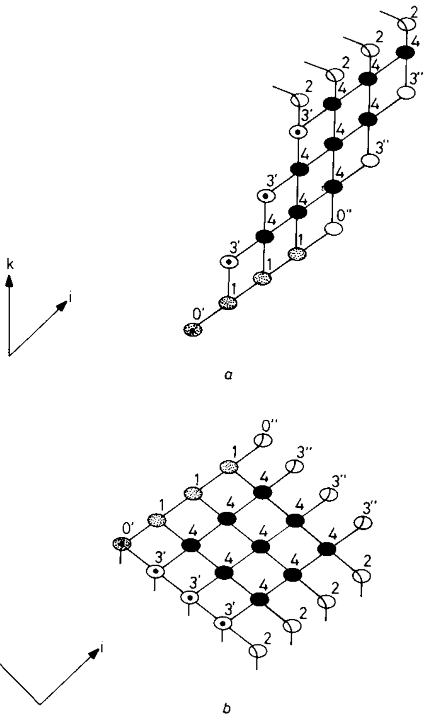


Fig. 5 *Tag code added for plane in DG*
a Double projection directions are along the i and k axes
b Double projection directions are along the i and j axes

step they need $n + 1$ operations in each PE, and between each elimination step one time unit is needed to propagate data. Although the latency increases from $(4n - 2)$ in Fig. 3 to $(n^2 + 3n - 1)$ in Figs. 6 and 7, the array structure is reduced from two dimensions to one dimension. From the comparison between Figs. 6 and 7, it is clear that the design shown in Fig. 6 will therefore need $(n + m)$ PEs with n^2 FIFO memory, $(n + 2m + 1)$ input ports and one output port and $2n$ input lines for tags, whereas the design in Fig. 7 needs $(n + 1)$ PEs with $(n^2 + n)$ FIFO memory, four input ports and one output port and $2n$ input lines for tags, respectively. With hardware costs taken into consideration, the latter design is superior to the former, but it takes $(n^2 + 3)$ time units to

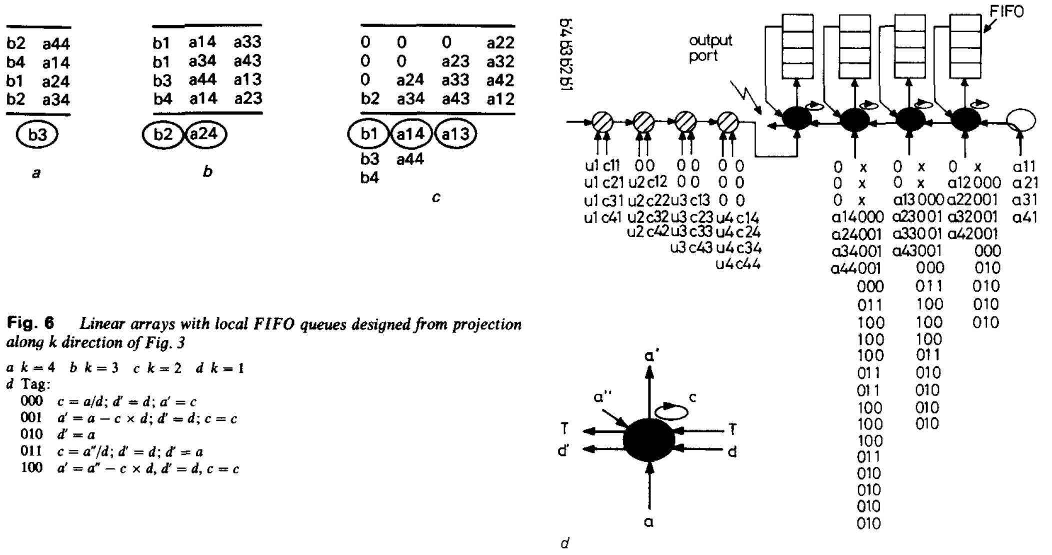


Fig. 6 *Linear arrays with local FIFO queues designed from projection along k direction of Fig. 3*
a $k = 4$ *b* $k = 3$ *c* $k = 2$ *d* $k = 1$
d Tag:
000 $c = a/d;\ d' = d;\ a' = c$
001 $a' = a - c \times d;\ d' = d;\ c = c$
010 $d' = a$
011 $c = a''/d;\ d' = d;\ d' = a$
100 $a' = a'' - c \times d,\ d' = d,\ c = c$

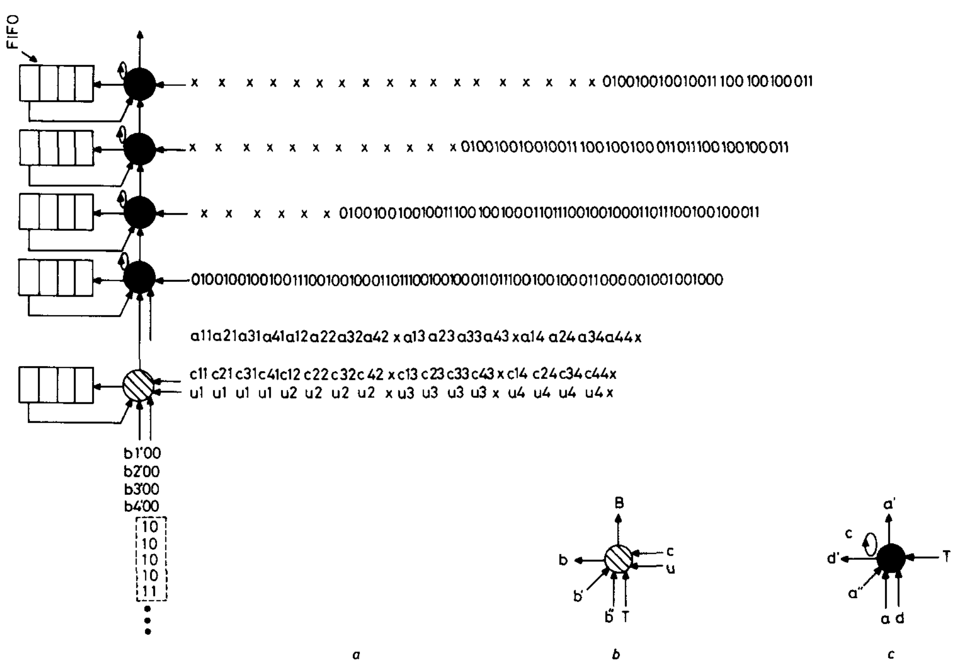


Fig. 7 *Linear arrays with local FIFO queues designed from projection along j direction of Fig. 3*

b Tag:
00 $b = b' + c \times u$
10 $B = b = b'' + c \times u$
11 No operation

c Tag:
000 $c = a/a'';\ a' = c$
001 $a' = a - a'' \times c;\ c = c$
010 $d' = d$
011 $c = d/a'';\ d' = a'';\ a' = c$
100 $a' = d = a'' \times c,\ d' = a'',\ c = c$

input the data, whereas the design shown in Fig. 6 needs only $(2n - 1)$. Observing the sequence of tag bits in Figs. 6 and 7, we see that if we change the tag bit value and the connection condition of the PE when the tag bit is pumped through the array, the function of codes 0 and 3, 1 and 4 are the same since their connections are suitably rearranged. This kind of modified design applied to Fig. 6 is shown in Fig. 8, where only two tag bits are required. The design shown in Fig. 7 can also be modified using a similar method.

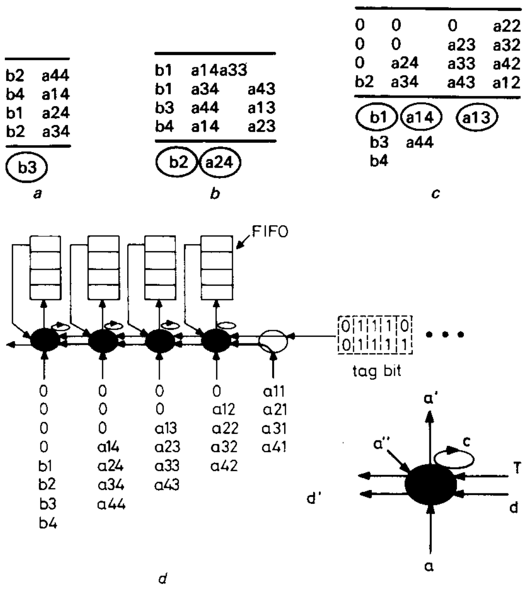


Fig. 8 *Modified design of Fig. 6*
a $k = 4$ *b* $k = 3$ *c* $k = 2$
d $k = 1$
Tag:
00 $c = a/d;\ d' = d;\ a' = a$
10 $a' = a - c \times d;\ d' = d;\ c = c$
01 $a' = c$; set d' - - a''; Tag = 11; $c = c$
11 $a' = c$; set a - - a''; $d' = d;\ c = c$

As to latency, the systems obtained by applying the second projection on Fig. 3 are not the best. Now, let us turn to the original 3-D DG which is redrawn in Fig. 9A. If the first projection is along the j direction instead of the i direction, then during each elimination step k it only needs $(n + 1 - k)$ operations in the PEs as we can see in the DG, where only $(n + 1 - k)$ computations are projected into PEs in the k direction. So, if the second projection is along the k direction, the latency is $\sum_{k=1}^{n} (n + 1 - k) + 2n - 1 = (n^2 + 7n - 2)/2$, where $(2n - 1)$ is the time unit to propagate data between PEs. Thus, a more efficient design is to choose the first projection to be along the j direction, and the resulting two-dimensional systolic-array system has the same latency $4n - 2$ as in Fig. 3, but the PE number becomes n^2. Then the second projection is along the k direction using the same procedures as in Figs. 6 and 7. The linear systolic array thus obtained is shown in Fig. 9B, which stores the elements of the same row in one PE. The latency of this design is $(n^2 + 7n - 2)/2$ which is two times faster than that in Fig. 6, but the PE number is $(2n - 1 + m)$ with $(\frac{3}{2}n^2 - n - 2)$ FIFO memory and four output ports which are larger than those of Fig. 6.

3 Linear systolic array with content addressable memory

In many real applications, matrix A in eqn. 1 is sparse especially when n is large, i.e. most entries of A are zeros. So if we fully utilise this property to modify the systolic array in Figs. 6, 7 and 9, to avoid trivial operations

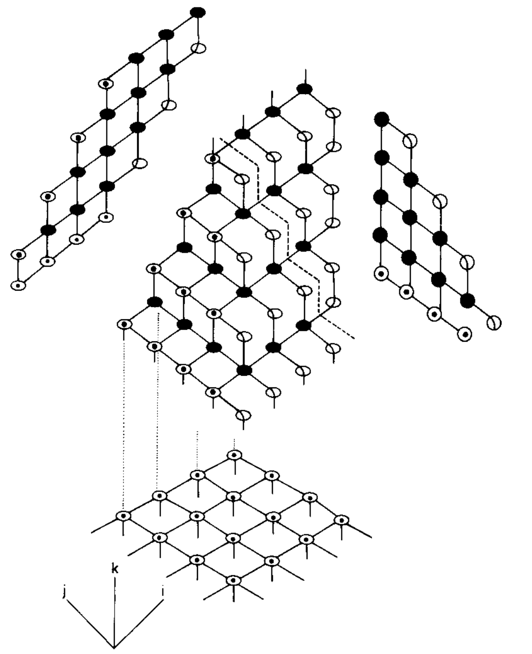
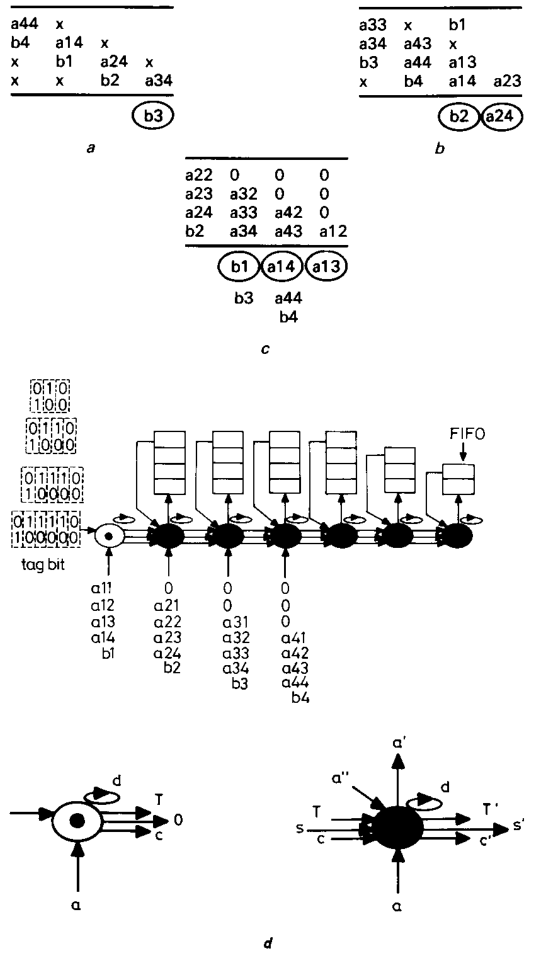


Fig. 9A *DG of solving eqn. 1 and array structures after projection along i and j axes, respectively*

involving zero elements, and do not store the zero elements, then the computation of eqn. 1 may be speeded up and the storage amount be reduced.

But if we only store the nonzero elements of A, the data sequence that feeds the right PE at the right time will be destroyed. Thus, at each PE when the elimination step is carried out and the data are pipelining through PEs to perform the elimination, we must search for the right data that have been stored in the local memory when doing the division or multiplication computations. This search requirement can be achieved by using the content addressable memories (CAM), in which part of the memory contents are used to search for the right data instead of using address. In this case, each PE stores the associated nonzero column (or row) elements in their local CAM.

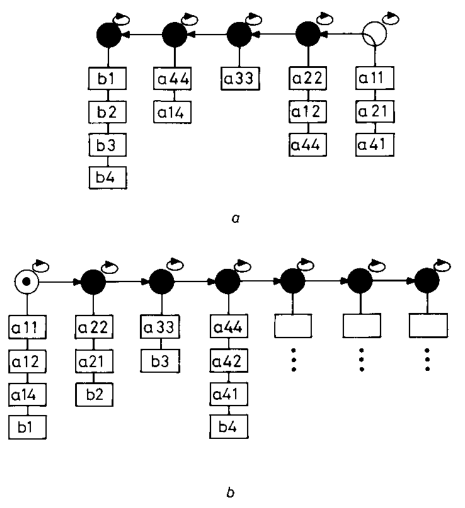
The idea of using CAM for linear sparse matrix computation was first proposed in Reference 12. Figs. 10*a* and *b* are the CAM systolic arrays designed from the modification of Figs. 6 and 9, respectively, with data format in arbitrary sparseness. The arrays for solving $\boldsymbol{B}(t_n) = \boldsymbol{B}'(t_{n-1}) + \boldsymbol{C} \times \boldsymbol{U}(t_n)$ is not shown here for simplicity. In this design, each data word must include four fields: the value of an element, its row (column) index, tag bits and an end-of-column (or end-of-row) indication bit. The tag bits have the same meaning as in Figs. 6 and 9. Because the data sequence is destroyed and the data cannot be piped into the array, they must be preloaded into the CAM by a host computer. Also the tags can no longer be pipelined into the associated PE. So we attach



tag	PE A	PE B
0 0	$d = a$	$d = a,\ T' = T,\ s' = s$
1 0	$c' = a/d$	$a' = (a\bar{s} + a'' \cdot s) - c \times d,\ c' = c,\ d = d,\ T' = T,\ s' = s$
0 1		$d = a'',\ T' = T,\ s' = s$
1 1		$c' = a''/d,\ T' = 10,\ s' = \bar{s}$

Fig. 9B *Linear arrays with local FIFO queues designed from projections along j and k axes*
a $k = 4$ *b* $k = 3$ *c* $k = 2$ *d* $k = 1$

the tag bits to the word to control the computation. The end-of-column (end-of-row) indication bit is used to signal the arriving of the last element of a column (row) so as to start the elimination steps started by the next column (row) elements. The row (column) index is used for CAM to search for the needed data in another column (row). The computation flow of Figs. 10*a* and *b* is shown in Fig. 10*c*. Note that the diagonal elements of each column (row) are stored first in each PE so that, when popping out the data of column (row) k to carry out the kth elimination step, the correct data sequence can be obtained. This is because, in the elimination step k, the first job is to divide the row elements by the diagonal elements. Thus the CAM must have the ability of using the row (column) index to search for the data and can sequentially pop out the data to the PE.



```
FOR (i from 1 to n){
  Pop out elements of PE i;
  Transport the elements through the left(right) PEs;
  According to the column(row) index to get data from CAM;
  Doing the multiplication or division according to the tag bits;
  }
```

c

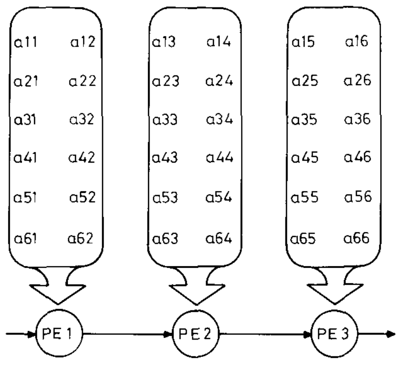
Fig. 10 *CAM systolic arrays*
Designed from modification of
a Fig. 6
b Fig. 9*c* by using the sparseness property of matrix *A*

The latency of the whole system is reduced from $(n^2 + 2n + 1)$ and $(n^2 + 7n - 2)/2$ to $NA + n$ and $NZ + 2n - 1$ (excludes the time of loading input data) for Figs. 10*a* and *b*, respectively, because the trivial computations are avoided, where NA is the number of nonzero elements of matrix A and NZ is the number of nonzero elements along and below the diagonal. The memory requirement is also reduced from $(\frac{3}{2}n^2 - n - 2)$ and n^2 to $NA + n$ and $NA + NZ + 2n - 1$, respectively. Furthermore, the systolic array in Fig. 10*a* is superior to Fig. 10*b* not only because it has shorter latency and less hardware requirement, but also the data b(.) are needed after the nth time unit regardless of the sparseness of matrix A, whereas that in Fig. 10*b* is dependent on the sparseness of A. In real applications, especially when n is large, each row (column) of matrix A has only a few nonzero entries (3 to 4), so NA and NZ can be expressed as $r \times n$ and $(r \times n + n)/2$, respectively, where r denotes the average number of nonzero elements in each row. Consequently, if we compare the computation time of the systolic array with FIFO, i.e. $(n^2 + 7n - 2)/2$, we find that much time is saved. In Wing [13], the LU decomposition method is used to solve $\boldsymbol{AV} = \boldsymbol{B}$ and it took $NZ + 5n - 2$ time units to complete the computation, whereas in our design of solving eqn. 1, which is much complex than solving $\boldsymbol{AV} = \boldsymbol{B}$, it takes only $NA + n$ time units (which is less than $NZ + 5n - 2$), since $NA + n$ is less than $NZ + 5n - 2$ as long as $r < 9$.

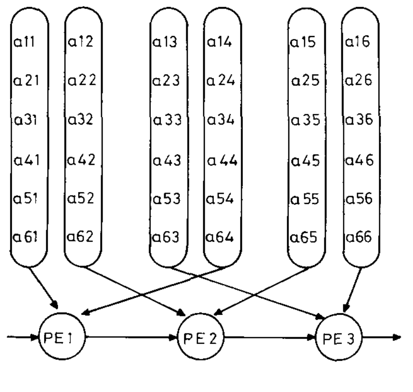
4 Partition of linear systolic-array system

With the advance of VLSI technology, more PEs can be integrated into one chip, but there are also physical limitations imposed by the number of I/O pins and yield. A natural solution is to divide the computation problem into smaller problems with a fixed size.

Many partition methods have been proposed to solve this problem [6, 14, 15]. Roughly, according to the computation sequence of the data, they can be categorised into two types: i.e. local-serial-global-parallel (LSGP) and local-parallel-global-serial (LPGS) [6] which are illustrated in Figs. 11*a* and *b*, respectively. Here, considering the overlapping of two stages for solving eqn. 1, matrix A' is partitioned into $\lceil n/p \rceil$ submatrices of size $n \times p$ by column. The partition scheme applied to the design of Fig. 10*a* is shown in Fig. 12A. As there are two independent iterated arcs on the DG so that the data will be changed iteratively along their propagation through the index nodes, global memories thereby become a necessity to save the intermediate results. The temporary results are popped out from the last PE into global memories and then fed into the first PE. Therefore, the data sequence can be maintained. By doing so, only global FIFO memories are required and one global feedback line and a switch box are needed in this partition scheme. The systolic array with local CAM and global FIFO is demonstrated in Fig. 12B, which is easily derived from Fig. 10*a*. The data flow of the LPGS and LSGP partition schemes are shown in Figs. 12C and D, respectively. The size of the FIFO queue is determined by the nonzero elements of matrix A for the LPGS scheme or by the maximum number of nonzero elements in the column vector of matrix A for the LSGP scheme. The latency can



a



b

Fig. 11 *Concept of partitioning and scheduling*
a LSGP
b LPGS

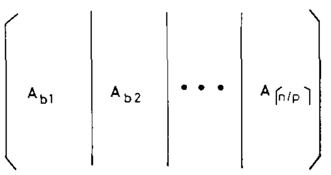


Fig. 12A *Data partition of matrix A″*

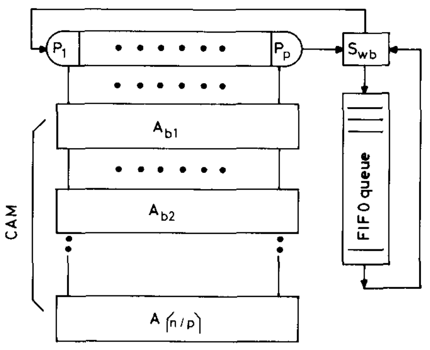


Fig. 12B *Fixed size (p) one-dimensional systolic array with local CAM and global FIFO*

```
FOR($i$ from 1 to $\lceil n/p \rceil$){
  FOR($j$ from $i$ to $\lceil n/p \rceil$){
    IF($j$ equal $i$)
      FOR(column $k$ from 1 to $p$ of $A_{bi}$){
        Sequently pop elements of the $(i-1) \times \lceil n/p \rceil + k$ column
        out from CAM of PE $k$ to $p$;
        Normal Gauss-Jodan operation for matrix column
        from $(j-1) \times \lceil n/p \rceil + 1$ to $j \times \lceil n/p \rceil$;
      }
    ELSE
      FOR(each element in FIFO queue){
        Sequently pop elements out from FIFO;
        Normal Gauss-Jordan operation for matrix
        column from $(j-1) \times \lceil n/p \rceil + k$ to $j \times \lceil n/p \rceil$;
      }
  }
}
```

Fig. 12C *LPGS data flows*

```
FOR($i$ from 1 to $n$){
  Sequently pop elements of matrix column $i$ from CAM of
  PE $((i-1) \bmod p) + 1$;
  Normal Gauss-Jordan operation for column from $i$ to $\lceil n/p \rceil \times p$;
  FOR($j$ from $\lceil n/p \rceil + 1$ to $\lceil n/p \rceil$){
    Sequently pop elements out from FIFO;
    Normal Gauss-Jordan operation for matrix columns from
    $(j-1) \times p$ to $j \times p$;
  }
}
```

Fig. 12D *LSGP data flow*

Table 1: Analyses of various systolic arrays solving eqn. 1

Algorithm	Data link	Concurrency	T	BT	PE	M	O/I	T × (PE/M)
Fully parallel (Fig. 2)	Global link	n^2	$2n+m$	1	$\frac{n^3}{2}$	0	n/n^2	n^4
Local parallel (Fig. 3)	Local link	$K \times n$	$4n$	n	$\frac{n^2}{2}$	0	$1/n$	$2n^3$
Local memory parallel (Fig. 8)	Local FIFO	n	$\frac{n^2}{2}$	$\frac{n^2}{2}$	n	n^2	$1/n$	$\frac{n^2}{2} \times (n/n^2)$
Local memory parallel (Fig. 10)	Local CAM	n	***NA***	***NA***	n	***NA***	$1/n$	***NA*** × (n/***NA***)
Serial	Local memory	1	$\frac{n^3}{2}$	$\frac{n^3}{2}$	1	n^2	1/1	$\frac{n^3}{2} \times (1/n^2)$

T = latency, BT = block pipelining period, PE = processor element, M = memory, K = variable from 1 to n
The number in this table is the order of the complexity

be computed as follows:

LPGS:

$$\text{latency} = \sum_{i=1}^{\lceil n/p \rceil} (N_{bi} + p - 1) \times (\lceil n/p \rceil - i)$$

$$= ((r+1)p - 1) \times \left(\frac{\lceil n/p \rceil (\lceil n/p \rceil + 1)}{2} \right)$$

$$\approx \frac{rn^2}{2p}$$

LSGP:

$$\text{latency} = p \sum_{i=1}^{\lceil n/p \rceil} (r + p - 1) \times (\lceil n/p \rceil - i + 1)$$

$$= p(r + p - 1) \left(\frac{\lceil n/p \rceil (\lceil n/p \rceil + 1)}{2} \right)$$

$$\approx \frac{(r+p)n^2}{2p}$$

where $N_{bi} = rp$ is the number of nonzero elements in p column.

Due to the partition, the latency increases by a factor $(n/2p)$, and a small array size will therefore pay a greater latency. The advance of using sparseness properties is to reduce the latency by a factor of (n/r).

5 Discussion

The one-dimensional linear systolic-array system with local FIFO, the linear systolic-array system with local CAM and the one-dimensional fixed-size linear systolic-array system with local CAM and global FIFO are all successfully designed by our DG approach. Which systolic array system is suitable to solve the problem is an interesting issue and would be determined by some practical considerations. Table 1 summarises the comparisons of the features and performance for the various systolic-array systems. Besides those designs in this paper, the Table also includes a fully parallel design, which corresponds to a three-dimensional systolic-array system, and one processor system executing a serial algorithm, which may agree with the design obtained by triple projection on the three-dimensional DG.

6 Acknowledgment

This work was supported by the National Science Council, Taiwan ROC under Grant NSC77-0201-E009-01.

7 References

1 KUNG, H.T., and LEISERSON, C.E.: 'Systolic arrays for VLSI'. Proceedings of SIAM, Sparse Matrix Symposium, 1978, pp. 256–282
2 KUNG, S.Y., LO, S.C., and LEWIS, P.A.: 'Optimal systolic design for the transitive closure and the shortest path problems', *IEEE Trans.*, 1987, **C-36**, (5), pp. 603–614
3 GACHET, P., JOINNAULT, B., and QUITTON, P.: 'Synthesizing systolic arrays using DIASTOL'. International Workshop of Systolic Arrays, University of Oxford, Department for External Studies, 2nd–4th July 1986, pp. F4.1–F4.12
4 CHEN, M.C.: 'Synthesizing systolic design'. Proceedings of International Symposium on VLSI Technology, Systems and Applications, Taiwan ROC, May 1985, pp. 209–215
5 MOLDOVAN, D.I.: 'ADVIS: a software package for the design of systolic arrays', *IEEE Trans.*, 1987, **CAD-6**, (1), pp. 33–40
6 MOLDOVAN, D.I., and FORTES, J.A.B.: 'Partition and mapping algorithm into fixed size systolic arrays', *IEEE Trans.*, 1986, **C-35**, (1), pp. 1–12
7 FORTES, J.A.B., FU, K.S.,' and WAH, B.W.: 'Systematic approaches to the design of algorithmic specified systolic arrays'. Proceedings of IEEE ICASSP, Piscataway, NJ, USA, 1985, pp. 891–895
8 LI, G.L., and WAH, B.W.: 'The design of optimal systolic arrays', *IEEE Trans.*, 1985, **C-34**, (1), pp. 66–77
9 JOU, S.J., and JEN, C.W.: 'The design of systolic array system for solving linear state equations', *IEE Proc. G*, 1988, **135**, (5), pp. 211–218
10 RAMAKRISHNAN, I.V., and VARMAN, P.J.: 'Synthesis of an optimal family of matrix multiplication algorithms on linear array', *IEEE Trans.*, 1986, **C-35**, (11)
11 JEN, C.W., and HSU, H.Y.: 'The design of a systolic array with tags input'. ISCAS, 1988, pp. 2263–2266
12 KIECKHAFER, R.M., and POTTLE, C.: 'A processor array for factorization of unstructured sparse matrices'. IEEE International Conference on Circuits and Computers, 1982, pp. 380–382
13 WING, O.: 'A content-addressable systolic array for sparse matrix computation', *J. Parallel Distrib. Comput.*, 1985, pp. 170–181
14 NELIS, H., and DEPRETTERE, E.F.: 'A systematic method for mapping algorithms of arbitrarily large dimensions onto fixed size systolic arrays'. IEEE International Symposium on Circuits and System, 1987, **2**, pp. 559–563
15 NAVARRO, J.J., and LLABERIA, J.M., and VALERO, M.: 'Partition: An essential step in mapping algorithms into systolic array processors', *IEEE Comput.*, 1987, pp. 77–88
16 LIU, C.-M., and JEN, C.-W.: 'Design of algorithm-based fault-tolerant VLSI array processor', *IEE Proc. E*, 1989, **136**, (6), pp. 539–547