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 multification 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 \( \dot{V}(t) = AV(t) + CU(t) \), 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 \( m \), and \( A \) and \( C \) are \( n \times n \) and \( n \times m \) matrices, 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:

\[
AV(t_n) = B(t_{n-1}) + CU(t_n)
\]  

(1)

where \( A = \begin{bmatrix} A' & (1/h)I \end{bmatrix}, \ h = t_n - t_{n-1}, \ B(t_{n-1}) = (1/h)I(t_{n-1}) \), \( I \) is the identity matrix and \( t_n \) is the time at the \( n \)th 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 multiprocessor 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 \( B(t_n) = B(t_{n-1}) + CU(t_n) \) 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 (LRS) 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 LRS 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 \( O(n^2) = n \times m \) to \( 4n - 3(n - 2) \) if \( n < n (m = n) \) 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.
\begin{align*}
B &= \mathbf{B'} + \mathbf{C} \times \mathbf{U}; \quad \mathbf{B'}: \text{n} \\
\mathbf{C}, \mathbf{B}: \text{n} \times \text{m}; \quad \mathbf{U}: \text{n} \times 1
\end{align*}

INPUT: \( \mathbf{C}(i,j), \mathbf{B'}(i), \mathbf{U}(i,j) = \mathbf{U}(i,j) \)

OUTPUT: \( \mathbf{B}(i,m) \)

FOR \( \text{ifrom 1 to n} \) {
    \( \mathbf{B}(i,0) = \mathbf{E}(i) \)
    FOR \( \text{ifrom 1 to m} \) {
        \( \mathbf{U}(i,j) = \mathbf{U}(i-1,j) \)
        \( \mathbf{S}(i,j) = \mathbf{B}(i,; - 1) + \mathbf{C}(i,; \times \mathbf{U}(i,j)) \)
    }
}

\( AX = BA' = [A | B'; A: n \times n, B: n \times 1, n = n + 1, \text{uniform}]/ \)

INPUT: \( \mathbf{A'}(i,j,0) = \mathbf{A'}(i,j) \)

OUTPUT: \( \mathbf{A'}(i,n,1), i = n \text{ to } n + 2 \)

FOR \( \text{ifrom 1 to n} \) {
    FOR \( \text{ifrom k to k + n - 1} \) {
        \( \mathbf{D}(i,k,k) = \mathbf{A'}(i,k,k - 1) \)
        IF \( \text{equal} \)
            FOR \( \text{ifrom k + 1 to n} \) {
                \( \mathbf{C}(i,j,k) = \mathbf{A'}(i,j,k - 1) / \mathbf{D}(i,j,k) \)
                \( \mathbf{A'}(i,j,k) = \mathbf{A'}(i,j,k - 1) - \mathbf{D}(i,j,k) \times \mathbf{C}(i,j,k) \)
            } \\
        ELSE
            FOR \( \text{ifrom k + 1 to n} \) {
                \( \mathbf{D}(i,j,k) = \mathbf{D}(i,j,k - 1) \)
                \( \mathbf{C}(i,j,k) = \mathbf{A'}(i,j,k - 1) \)
                \( \mathbf{A'}(i,j,k) = \mathbf{A'}(i,j,k - 1) - \mathbf{D}(i,j,k) \times \mathbf{C}(i,j,k) \)
            } \\
    }
}

Fig. 1 LRSA form
a Matrix-vector multiplication accumulation
b Uniform Gauss-Jordan algorithm

Fig. 2 Linked DG with tag code of solving eqns. 1 for the case of \( n = m = 4 \)

Fig. 3 Two-dimensional systolic-array system by projecting along \( i \) direction
a and e T (Tag): \( b = b' + c \times d \quad d' = a \)

To design the one-dimensional systolic-array system, one promising way is that the secondary projection is directly applied onto the two-dimensional systolic-array system which is shown in Fig. 3. By doing so, the same
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 times the number of data links along the projection direction.

Observing Fig. 3, there are two permissible directions, \( k \) and \( j \), for the second projection on the two-dimensional array, and the corresponding linear array structures are shown in Figs. 4a and b, respectively. Throughout the 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 \( i-j \) 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. 5a 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. 5a, 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. 4a 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 snapshot 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 \( n(n+1)+n+n^2+3n-1 \) and \( (n+1)(n+1)-1+n-1=n^2+3n-1 \) clocks, respectively. This is because in each elimination

---

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

**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

---


187
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...

![Diagram](attachment:///image.png)

Fig. 6 Linear arrays with local FIFO queues designed from projection along $k$ direction of Fig. 3

<table>
<thead>
<tr>
<th>Tag</th>
<th>$a=k=4; b=k=3; c=2$</th>
<th>$d=k=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>$c = a; d = d; c = c$</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>$a = a; a = a; d = a; c = c$</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>$c = a; d = d; a = a$</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>$a = a; a = a; c = c$</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](attachment:///image.png)

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>$a = b + c + u$</th>
<th>$c = a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>$c = a; d = d; c = c$</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>$0 = b + b + u + x + u$</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>No operation</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>$0 = b + b + u + x + u$</td>
<td></td>
</tr>
</tbody>
</table>

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.

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 \(i\) direction instead of the \(j\) 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 \(\sum_{r=1}^{n} \frac{1}{n} + 1 - k + 2n - 1 = \frac{n^2 + 7n - 2}{2}\), 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 \(\frac{n^2 + 7n - 2}{2}\) which is two times faster than that in Fig. 6, but the PE number is \((2n - 1 + m)\) with \(\frac{3}{2}(n^2 - 1) - 2\) 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 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. 10a 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 \(B(t) = B(t) + C \times U(t)\) 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 a, b, c, d to sequentially pop out the data to the PE.

\[
\begin{array}{c}
a_{14} a_{33} b_{3} a_{44} a_{13} \\
a_{24} a_{34} a_{43} b_{24} a_{14} a_{23}
\end{array}
\]

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.

\[
\begin{array}{cccc}
a_{11} & 0 & 0 & 0 \\
a_{12} & a_{21} & 0 & 0 \\
a_{13} & a_{22} & a_{31} & 0 \\
a_{14} & a_{23} & a_{32} & a_{41}
\end{array}
\]

\[
\begin{array}{cccc}
a_{24} & a_{34} & a_{43} & b_{24} \\
a_{34} & a_{43} & a_{12} & 0 \\
b_{2} & a_{34} & a_{43} & b_{4}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
a_{31} & a_{32} & a_{33} & a_{34} \\
a_{41} & a_{42} & a_{43} & a_{44}
\end{array}
\]

\[
\begin{array}{cccc}
a_{11} & a_{12} & a_{13} & a_{14} \\
a_{21} & a_{22} & a_{23} & a_{24} \\
\text{Tag bit}
\end{array}
\]

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. 10a 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 \((\frac{1}{2}n^2 - n - 2)\) and \(n^2\) to \(NA + n\) and \(NZ + 2n - 1\), respectively. Furthermore, the systolic array in Fig. 10a is superior to Fig. 10b not only because it has shorter latency and less hardware requirement, but also the data \(b(b_{j})\) are needed after the \(n\)th time unit regardless of the sparseness of matrix \(A\), whereas in Fig. 10b it is dependent on the sparseness of \(A\). In real applications, especially when \(n\) is large, each row (column) of matrix \(A\) has only some nonzero entries (3 to 4), so \(NA\) can be expressed as \(r \times n\) and \(NZ + 5n - 2\), 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 \(AV = B\) 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 \(AV = B\), 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. 11a and b, respectively. Here, considering the overlapping of two stages for solving eqn. 1, matrix \( A \) is partitioned into \( \frac{n}{r} \times p \) submatrices of size \( n \times p \) by column. The partition scheme applied to the design of Fig. 10a 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. 10a. 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
be computed as follows:

LPGS:

\[
\text{latency} = \sum_{i=1}^{[n/p]} (N_{pi} + p - 1) \times ([n/p] - i)
\]

\[
= (r + 1)p - 1 \times \frac{[n/p][([n/p] + 1)]}{2}
\]

\[
\approx \frac{rn^4}{2p}
\]

LSGP:

\[
\text{latency} = \sum_{i=1}^{[n/p]} (r + p - 1) \times ([n/p] - i + 1)
\]

\[
= p(r + p - 1) \times \frac{[n/p]([n/p] + 1)}{2}
\]

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

where \(N_{pi} = rp\) is the number of nonzero elements in \(p\) column.

Due to the partition, the latency increases by a factor of \((n/2p)\), 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 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

11. JEN, C.W., and HSU, H.Y.: 'The design of a systolic array with tags input'. ISCAS, 1988, pp. 2263-2266