Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors

Juan J. Navarro, José M. Llaberia, and Mateo Valero
Facultad de Informática de Barcelona
Universidad Politécnica de Cataluña

The efficient solution of a large problem on a small systolic array requires good partitioning techniques to split the problem into subproblems that fit the array size.

Many scientific and technical applications require high computing speed; those involving matrix computations are typical. For applications involving matrix computations, algorithmically specialized, high-performance, low-cost architectures have been conceived and implemented. Systolic array processors (SAPs) are a good example of these machines.[1,2]

An SAP is a regular array of simple processing elements (PEs) that have a nearest-neighbor interconnection pattern. The simplicity, modularity, and expandability of SAPs make them suitable for VLSI/WSI implementation.

Algorithms that are efficiently executed on SAPs are called *systolic algorithms* (SAs). An SA uses an array of systolic cells whose parallel operations must be specified. When an SA is executed on an SAP, the specified computations of each cell are carried out by a PE of the SAP. An SA is a specification of

- the type of operation performed by each cell during each step,
- the communication pattern among the cells, and
- the data and control sequences for I/O in the boundary cells.

In an SA, communication between cells is regular and local, massive parallelism is exhibited, and a relatively low number of I/O operations is required. The properties of systolic algorithms and architectures limit their practical use to that of solving only regular, compute-bound problems with a high degree of parallelism.[2] Fortunately, many matrix problems are of this type.

A large set of SAs and SA design methodologies[3] has recently been published. Most of these SAs require a number of cells that depend on some dimension of the problem that is to be solved. We refer to these algorithms as *problem-size-dependent SAs*. Normally, these SAs solve the problem with just one pass of the data through the array. Moreover, the array topology is dependent on the type of problem to be solved.

The number of PEs in a real SAP is fixed. It depends on factors such as technology, desired speed, available host-SAP communication bandwidth, and system cost and complexity. Usually, the problem

0018-9162/87/0700-0077$01.00©1987IEEE

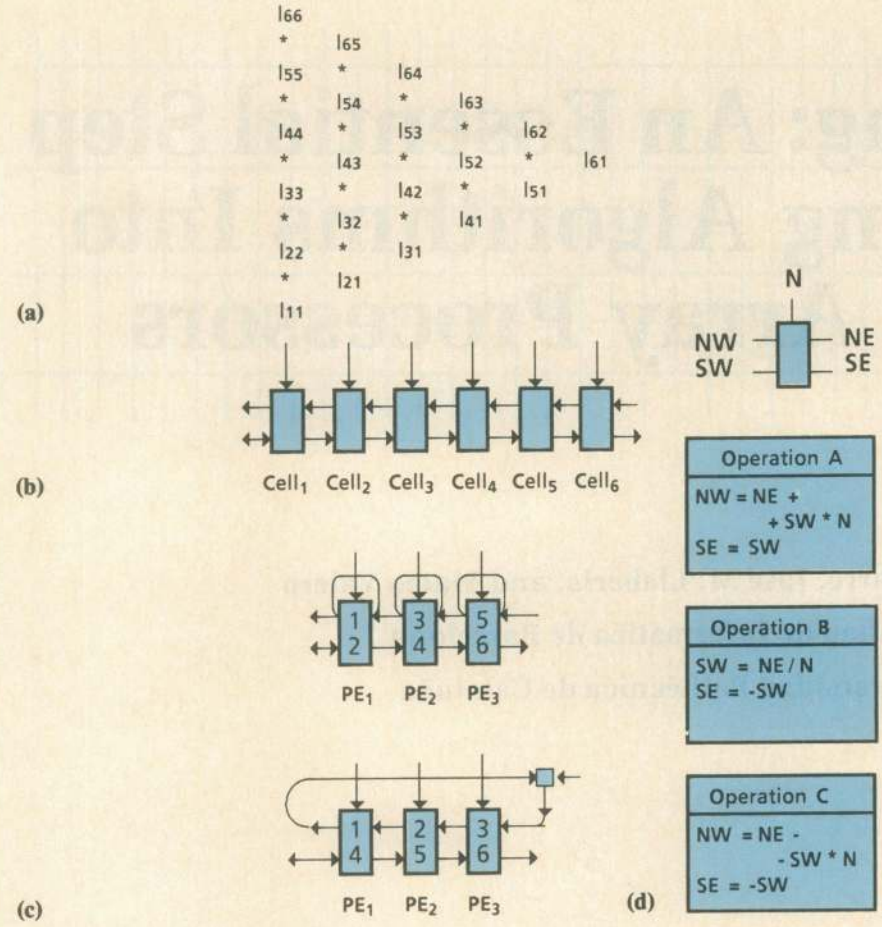


Figure 1. (a) A problem-size-dependent systolic algorithm for a lower triangular system of equations. (b) The SAP communication pattern that results from coalescent mapping. (c) The SAP communication pattern that results from cut-and-pile mapping. (d) Operations performed by PEs.

is too large to allow it to be solved directly by processing it on the array. In addition, if the system has fault-tolerant capabilities, the array is reconfigured in cases of PE failure, and the algorithm must be restructured to make it executable by the smaller array that results from such failure. Therefore, the original problem must be partitioned into subproblems whose individual sizes fit available SAP dimensions. It is therefore necessary to find general partitioning techniques suitable for SAPs and problems of any size.

Let us review here some previous work related to systolic partitioning. L. Johnsson[4] proposes partitioning techniques to aid in performing Gaussian elimination and matrix multiplication. R. Schreiber and P.J. Kucks[5] present a QR factorization that is executed on a modification of the triangular array proposed by W.M. Gentleman and H.T. Kung. D. Heller[6] describes partitioning methods that can be used in the QR factorization of band matrices. Using a trapezoidal extension of the SAP proposed by W.M. Gentleman and H.T. Kung, H.Y.H. Chuang and G. He[7] obtain a versatile SAP oriented to matrix computations that implement the Faddeeva Algorithm. H.D. Cheng and K.S. Fu[8] propose a partitioning method and a computational model that are based on the space-time domain expansion approach. And recently, D.I. Moldovan and J.A.B. Fortes[9] have extended their SAP design methodology to include fixed-size SAPs.

Usually, SAPs are *algorithm-specific,* and some of them are even direct hardware implementations of problem-size-dependent SAs. A typical application area for algorithm-specific problem-size-dependent SAPs is real-time digital signal processing.

To achieve wider applicability, it is useful to design *versatile SAPs,* which are SAPs that, when attached to a host, can execute several algorithms in a problem-size-independent manner.

To derive a versatile SAP, we propose the following steps.

For each problem in the problem set:

(1) find one or several SAs;
(2) select a problem-size-dependent SA with maximum similarity in cell operation and interconnection topology to the other SAs selected for problems;
(3) find partitioning algorithms for obtaining subproblems, and modify the selected SA so that the subproblems can be executed on a problem-size-independent SAP.

For the whole problem set:

(4) integrate the requirements (PE operations, interconnections, and control) for all the selected algorithms so that the versatile SAP can be built; and
(5) improve the versatile SAP with respect to speed (for example, by making use of pipelined PEs), array utilization, and fault-tolerant capabilities.

In the process described above, the selection of the SA and of the partitioning algorithms is the key to achieving the following goals:

• *Maximization of array utilization.* We define utilization as $U = T_1/n\,T_n$, where n is the number of PEs in the SAP, T_1 is the number of cycles needed to solve the problem with just one PE, and T_n is the number of cycles required to solve it on the SAP. Once n is fixed, maximizing U is equivalent to minimizing problem-execution time in the available SAP.

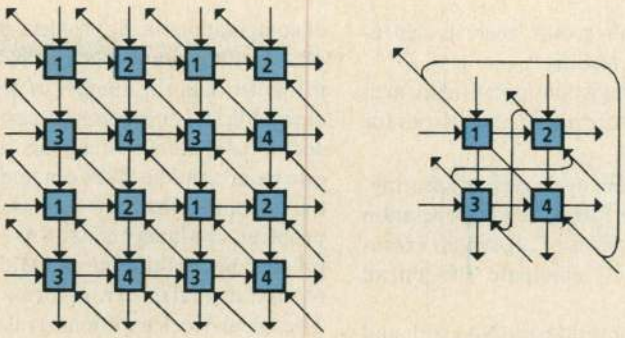
• *Minimization of the complexity of the SAP that results from the derivation process.* Important factors that determine complexity are the types of operations performed by the PEs and by the different classes of required PEs, the bandwidth between the SAP and the host, the amount of memory needed, the interconnection pattern, and the overall control of the array.

In this article, we present a technique for

obtaining the partitioning and the transformation of matrix problems; the technique is designed to minimize execution times for big problems in small arrays and to introduce very little additional complexity into the system. In the article, we apply the technique to solving both matrix-by-vector multiplication (M*V)[10] and triangular system equation (TSE) problems by means of a one-dimensional (1D) SAP; we also apply it to solving matrix-by-matrix multiplication (M*M), triangular matrix equations (TME), LU decomposition (LU), and other problems by means of a 2D SAP.[11] Actually, the proposed technique allows one to solve any problem on the preceding list on either on a 1D[12] or a 2D array.

We select, in Step 2 of the previously described process, problem-size-dependent SAs proposed by H.T. Kung and C.E. Leiserson.[1] We do not explicitly consider Step 4; however, implementing it is simple because we make appropriate choices in Step 2.

Finally, we make some comments related to the use of pipelined PEs.



(c)

NW N W E S SE

Operation A	Operation B	Operation C
NW = SE + + N * W S = N E = W	N = SE * W NW = 0 S = - SE * W E = W	N = SE / W NW = 0 S = -SE / W E = 1 / W

Operation D	Operation E	Operation F	Operation G
NW = SE - - N * W S = - N E = W	N = SE W = 1 S = 1 / S E E = 1	NW = 0 W = SE * N S = N E = SE * N	NW = 0 W = SE / N S = 1 / N E = SE / N

Figure 2. (a) A 2D array of cells. (b) The spiral SAP communication pattern that results from cut-and-pile mapping. (c) Operations performed by PEs.

Systolic algorithms for matrix computations

Several problem-size-dependent SAs can be derived for a matrix problem, depending on the speed and direction of dataflows as well as on the types of matrix substructures (rows, columns, or diagonals) that form the data sequence that enters or leaves through each I/O link of the array. We discuss two types of SAs according to their I/O matrix substructures: *band SAs*, in which the matrices enter or leave by diagonals, and *dense SAs*, in which the I/O substructures are rows or columns. We do not consider here hybrid algorithms (one type is the class of algorithms in which one matrix enters by rows or columns and another by diagonals).

In problem-size-dependent band SAs, the number of cells is related to the matrix bandwidth. When these algorithms are executed on an SAP, maximum array utilization is achieved when problems with band matrices (*band problems*) are being solved. In problem-size-dependent dense SAs, the number of cells depends on the number of rows or columns in the matrices involved. Maximum array utilization is achieved in the case of *dense problems* (that is, problems with dense matrices).

When the structure of the matrices (band or dense) involved in the problem does not fit the type of SA (dense or band) that is being used to solve the problem, array utilization decreases dramatically. Nevertheless, this drawback can be overcome if the original algorithm is modified. For example, Partial Row Translations[13] have been proposed for the solution of a dense matrix-by-vector multiplication problem executing a problem-size-dependent band SA. On the other hand, Systolic Rings[14] are well suited for the efficient solution of band problems executing problem-size-dependent dense SAs.

Another important factor in SAs is the relative direction of dataflows. Accordingly, we differentiate between SAs with "data contraflow" and SAs without it. Figures 1a and 2a show, respectively, one 1D topology and one 2D topology, both with data contraflow. For example, by changing the direction of the diagonal connections in Figure 2a, we would obtain a 2D topology without data contraflow. Generally, in 1D (or 2D) SAs with data contraflow, only one of every two (or three) consecutive cells is active during each step. In these cases, to achieve maximum array utilization, every PE of the SAP would have to execute the computations of two (or three) consecutive cells.

In the set of problems considered in this article, we distinguish between two groups. The first includes M*V and M*M multiplication. In the second group we have all other problems (TSE, TME, LU). Problems in the first group are *homogeneous*; that is, all the operations to be performed on data are of the same type. This fact makes it possible to have only one type of cell, which is used to construct a *homogeneous array*. In the second group, the problems are *nonhomogeneous*. For example, the operations (division and change of sign) that are performed on the elements in the main diagonal are different from those operations (multiplication and addition) that are performed on the other elements. For this reason, the array may need different types of cells. Also, in

problems in this group, there is dependency between results.

For the problems considered in this article, we identify certain characteristics for band SAs.

- All data move by traversing the array, and each cell performs the same operation during all the cycles of algorithm execution; these facts eliminate PE-control requirements.
- There are several band SAs with and without data contraflow for solving problems of the first group.
- For the second group, because of data dependencies, the band SAs require data contraflow; in these cases, all the cells perform multiplication and addition except one boundary cell that must perform division and change of sign.

We can also identify certain characteristics for dense SAs.

- For both groups of problems, one of the data sets (either the operands or the partial results) remains static within the cells as the computation proceeds. Consequently, additional control is necessary to establish the difference between array load (and/or array unload) operations and calculation operations.
- Data contraflow is not required for either group of problems.
- For problems in the second group, the dense SAs require that either all the cells (in the case of a 1D array) or all the cells in the main diagonal (in the case of a 2D array) must be able to perform division and change of sign, besides addition and multiplication. Consequently, the complexity of the array increases.

In summary, dense SAs require greater array complexity than do band SAs, but no data contraflow. The absence of data contraflow usually produces conditions favorable to fault tolerance and to implementation with pipelined PEs.[14] Efficient partitioning for dense SAs leads to the usual decomposition into square or rectangular submatrices.

The implementation of band SAs requires lower complexity in the array and less control than does the implementation of dense SAs. In this article, we use band SAs and propose both partitioning techniques that incorporate triangular submatrices and the utilization of pipelined PEs.

Partitioning and DBT transformation

A problem-size-dependent SA defines a good space-time mapping between the set of computations in the problem and the set of cells. If the number of PEs in the array is smaller than the number of cells in the algorithm, it is necessary to carry out an additional space-time mapping. In this section we present the SAPs obtained by spatial mapping that are used to execute problem-size-independent SAs. We also present the partitioning and DBT (Dense to Band matrix transformation by Triangular-block partitioning) algorithms, which serve to transform a problem-size-dependent SA—by temporal mapping—into a problem-size-independent SA.

Spatial mapping. The spatial cell-to-PEs mapping provides the set of computations to be performed by each PE; that is, it defines the different types of operations to be performed by each element, the array interconnection pattern, and the location of memory units. A good spatial mapping must preserve the topological properties of the SA. To illustrate this, let us suppose that we want to solve a triangular system of equations: $\mathbf{LX} = \mathbf{B}$, where $\mathbf{L}$ is a lower triangular c-by-c matrix and $\mathbf{X}$ and $\mathbf{B}$ are column vectors. A one-pass solution can be achieved with the band SA[1] shown in Figure 1a, if we assume that $c = 6$. All the cells perform multiplication and addition except cell_1, which performs division and change of sign.

Let us consider two types of cell-to-PE mappings: *coalescent* and *cut-and-pile*, both of which preserve regularity and locality in communications. We assume that the available SAP has w PEs.

In the coalescing mapping, cell_i for $1 \leq i \leq c$ is mapped to PE_k, where $k = \lceil i/\lceil c/w \rceil \rceil$ for $1 \leq k \leq w$. (See Figure 1b for an illustration of the communication pattern when $w = 3$.) In coalescing mapping, $\lceil c/w \rceil$ consecutive cells are assigned to one PE, so the PE requires feedback links to itself; these links are implemented with local memory. Each data item that has entered into a PE remains in it for $\lceil c/w \rceil$ cycles. Hence, each PE must have a local memory size proportional to $\lceil c/w \rceil$. In addition, the computing load is not uniformly distributed among PEs. Because of this, PEs 1, 2, and 3 in Figure 1b have to carry out 11, 7, and 3 computations, respectively.

In the cut-and-pile mapping, cell_i is mapped to PE_k, where $k = 1 + (i-1) \bmod w$ (Figure 1c). Each PE is functionally equivalent to a cell in the SA. One feedback line between the first and the last PEs is required. In general, the size of the memory needed in the feedback loop is proportional to $\lceil c/w \rceil$. With our technique, attaining sizes proportional to w alone is possible. This mapping distributes the PEs' computing load evenly among the PEs, and therefore the computing time is smaller. The loads of PEs 1, 2, and 3 in Figure 1c are 9, 7, and 5 computations, respectively. We chose cut-and-pile mapping for the partitioning methodology we propose.

When we consider using SAs with 2D topology to solve problems such as LU decomposition,[1] we see that the spatial cut-and-pile mapping is similar to the mapping just described. In Figure 2, an array with 4-by-4 cells and the SAP with 2-by-2 PEs that results from the mapping are shown. The feedback lines that can be seen in the figure must be added to satisfy the SA communication requirements. This bidimensional array is named *spiral SAP*.[15] The feedback lines in the horizontal and vertical flows, which would form a torus, are not required because the information that flows in these directions is not modified in the problems under discussion. For the same reason, the feedback line from the last to the first PE in Figure 1c has been eliminated. It is also possible to find mappings from the 2D topology to a 1D array.[12] Figures 1c and 2b illustrate the 1D and 2D SAP topologies considered in this article, and in Figures 1d and 2c, the set of all the necessary operations to be performed by PEs is shown. In our descriptions of each algorithm, we will indicate the particular set of operations to be carried out by each PE.

Temporal mapping. The original problem must be partitioned into subproblems whose data structures fit into the available SAP dimensions. The subproblem data structures are conditioned by the spatial mapping. These subproblems must be executed one after another in a manner that respects the data dependencies. The temporal mapping defines the time at which each computation assigned to a PE must be performed. We define this temporal mapping by means of the input data sequence (that is, by the order in which subproblems are executed).

The temporal mapping directly influences the array utilization. A lack of global communication capability combined with a high degree of pipelining in the SAP may produce low array utilization in loading and unloading of subproblems. Maximum utilization may be achieved if the matrix structures of subproblems and the execution order of subproblems allow

each subproblem's unloading to overlap with the next subproblem's loading.

The solution of each subproblem is arrived at by execution of a band SA. The execution sequence for solving all the subproblems may be viewed as the execution of a new band problem, which we denote as the *transformed problem*. The bandwidth of the transformed problem fits the available SAP size.

Overlapping the loads and unloads of subproblems is equivalent to achieving maximum juxtaposition of the submatrices that constitute the band of the transformed problem (that is, it is equivalent to getting maximum density in the band). Thus, the total execution time is minimized.

A set of rules for constructing the transformed problem's band with maximum density must be defined for each type of problem. The DBT proposed here achieves the transformation of a homogeneous problem, such as a matrix-by-vector or a matrix-by-matrix multiplication, into a band problem of the same type (such a band problem is called a *homogeneous transformed problem*). The following rules transform the original N-by-M matrix $\mathbf{A}$ into a band matrix $\underline{\mathbf{A}}$ with bandwidth w. N, M, and w can have any value (usually $N, M >> w$). However, we assume (and the assumption does not result in loss of generality), that $N = \underline{N}w$ and $M = \underline{M}w$. If $\mathbf{A}$ does not have these dimensions, it is augmented with rows and /or columns of zeros until it does.

Rules for triangular-block partitioning.

(1) Split matrix $\mathbf{A(N,M)}$ into $\underline{N}$-by-$\underline{M}$ square submatrices $\mathbf{A_{i,j}(w,w)}$.

(2) Decompose each submatrix $\mathbf{A_{i,j}}$, in turn, into three submatrices: $\mathbf{A_{i,j}} = \mathbf{A_{Li,j}} + \mathbf{A_{Di,j}} + \mathbf{A_{Ui,j}}$, where $\mathbf{A_{Li,j}}$ is the strictly lower triangular part of $\mathbf{A_{i,j}}$; $\mathbf{A_{Ui,j}}$ is the strictly upper triangular part of $\mathbf{A_{i,j}}$; and $\mathbf{A_{Di,j}}$ is formed with the main diagonal of $\mathbf{A_{i,j}}$. From these matrices we can define $\mathbf{A_{LDi,j}} = \mathbf{A_{Li,j}} + \mathbf{A_{Di,j}}$ and $\mathbf{A_{DUi,j}} = \mathbf{A_{Di,j}} + \mathbf{A_{Ui,j}}$, which are, respectively, the lower and upper triangular submatrices of $\mathbf{A_{i,j}}$.

Rules for dense matrix to band matrix transformation in inner-product-based problems.

(3) Build band matrix $\underline{\mathbf{A}}$ by alternately juxtaposing $\mathbf{A_{LDi,j}}$ and $\mathbf{A_{Ui,j}}$ submatrices, or by juxtaposing $\mathbf{A_{Li,j}}$ and $\mathbf{A_{DUi,j}}$, to fill up the band of $\underline{\mathbf{A}}$. Depending on which submatrix is chosen to be first, matrix $\underline{\mathbf{A}}$ may be a lower- or an upper-band matrix. The following refers to the case $\mathbf{A_{i,j}} = \mathbf{A_{Li,j}} + \mathbf{A_{DUi,j}}$, where $1 \le i \le \underline{N}$ and $1 \le j \le \underline{M}$ and $\underline{\mathbf{A}}$ is an upper-band matrix. The other cases can be easily derived. The rules are

(a) For $1 \le k \le \underline{N}\,\underline{M}$, if $\underline{\mathbf{A}}_{k,k}$ is equal to $\mathbf{A_{DUi,j}}$, then $\underline{\mathbf{A}}_{k,k+1}$ must be equal to $\mathbf{A_{Li,m}}$ for any m such that $1 \le m \le \underline{M}$.

(b) For $1 \le k \le \underline{N}\,\underline{M} - 1$, if $\underline{\mathbf{A}}_{k,k+1}$ is equal to $\mathbf{A_{Li,j}}$, then $\underline{\mathbf{A}}_{k+1,k+1}$ must be equal to $\mathbf{A_{DUn,j}}$ for any n such that $1 \le n \le \underline{N}$.

Several $\underline{\mathbf{A}}$ matrices can be obtained by the application of the preceding rules. However, as we shall note later, in the selection of a specific $\underline{\mathbf{A}}$, implementation factors must be taken into account.

When the original matrix $\mathbf{A}$ is dense, the transformed matrix $\underline{\mathbf{A}}$, with maximum band density and containing all the submatrices of $\mathbf{A}$, can always be obtained. The dimensions of $\underline{\mathbf{A}}$ are $\underline{N}\,\underline{M}w$-by-$(\underline{N}\,\underline{M}+1)w$ if $\underline{\mathbf{A}}$ is an upper-band matrix, or $(\underline{N}\,\underline{M}+1)w$-by-$\underline{N}\,\underline{M}w$ if $\underline{\mathbf{A}}$ is a lower-band matrix. When the DBT algorithm is applied to nondense matrices (for example, to band matrices), the dimensions of $\underline{\mathbf{A}}$ and the density of its band depend on the way the sparsity in $\mathbf{A}$ is structured.

In a subsequent section entitled "Partitioning and execution technique," we present the partitioning and transformation algorithm used to transform the STE, TME, and LU problems.

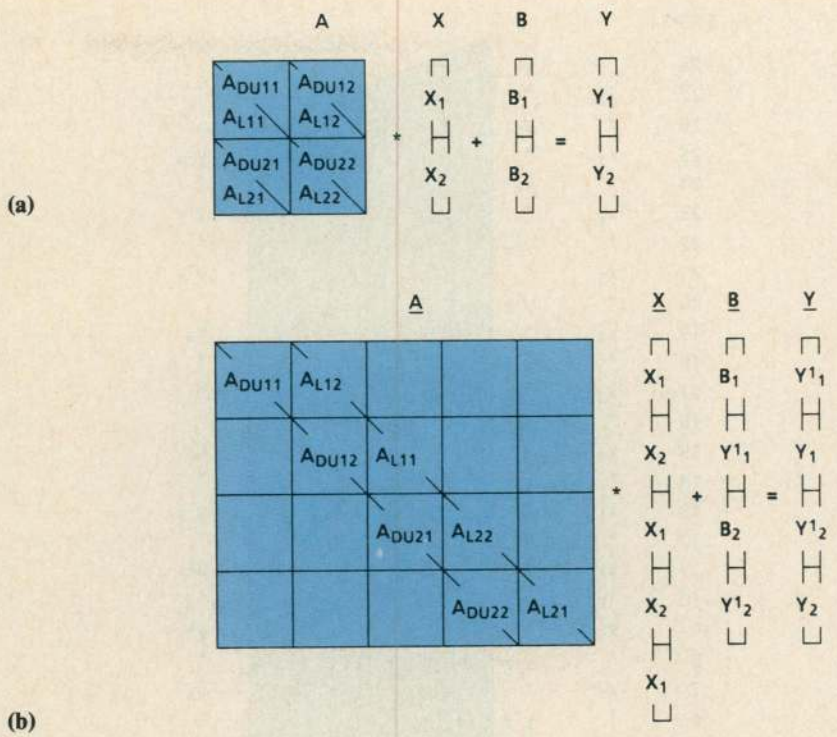


Figure 3. (a) Triangular-block partitioning of a matrix-by-vector problem. (b) A problem transformed by DBT algorithm application.

The matrix-by-vector problem

Here we address the problem of computing vector $\mathbf{Y}$, where $\mathbf{Y} = \mathbf{AX} + \mathbf{B}$, where $\mathbf{A}$ is an N-by-M matrix, and $\mathbf{X}$ and $\mathbf{B}$ are vectors whose respective dimensions are M and N. We assume that our 1D SAP has w PEs. $\underline{\mathbf{Y}} = \underline{\mathbf{A}}\,\underline{\mathbf{X}} + \underline{\mathbf{B}}$ is the transformed problem. In Figure 3a, the partitioning of the original problem is shown for $\underline{N} = \underline{M} = 2$. Now, each square block is decomposed as $\mathbf{A_{i,j}} = \mathbf{A_{Li,j}} + \mathbf{A_{DUi,j}}$. One of the possible DBT transformations that builds up matrix $\underline{\mathbf{A}}$ by following the preceding rules is shown in Figure 3b. Transformed vectors $\underline{\mathbf{X}}$ and $\underline{\mathbf{B}}$ are determined by the selected DBT because the

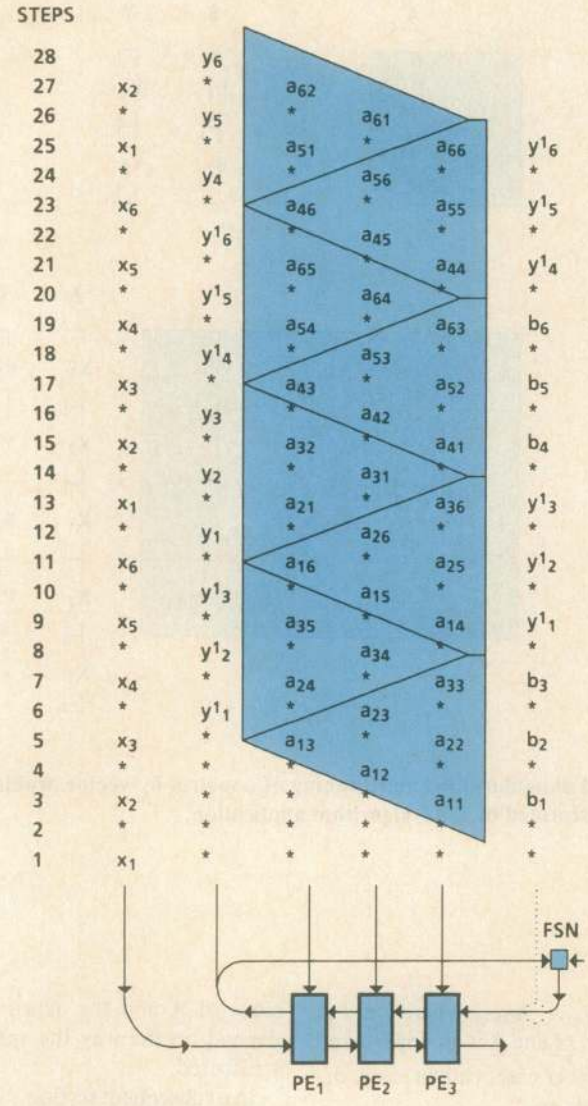


Figure 4. I/O data sequencing for the solution of the matrix-by-vector transformed problem.

transformed solution $\mathbf{Y} = (\underline{\mathbf{Y}}_1, \underline{\mathbf{Y}}_2, \underline{\mathbf{Y}}_3, \underline{\mathbf{Y}}_4)$ must include the original solution $\mathbf{Y} = (\mathbf{Y}_1, \mathbf{Y}_2)$. As we can see in Figure 3b, the transformed vectors are built from the subvectors of w elements in which the original **X** and **B** vectors have been split. Note that $\underline{\mathbf{B}}$ and $\underline{\mathbf{Y}}$ have some subvectors that are the same. By means of these recurrences, it is possible to make subvectors $\underline{\mathbf{Y}}_2$ and $\underline{\mathbf{Y}}_4$ of $\underline{\mathbf{Y}}$ equal to subvectors $\mathbf{Y}_1$ and $\mathbf{Y}_2$ of **Y**, respectively.

The I/O data sequences for the execution of the transformed problem for $N = M = 6$ and $w = 3$ are shown in Figure 4. Each PE performs Operation A (Figure 1d); this operation is equivalent to the one carried out in all the cells of the original SA.[1] Note that the recurrences between $\underline{\mathbf{B}}$ and $\underline{\mathbf{Y}}$ are implemented by means of the feedback link between PE_1 and PE_w. All the elements of $\underline{\mathbf{Y}}$ that must be introduced as elements of $\underline{\mathbf{B}}$ through the NE input of PE_w are available w cycles before they are needed in the NW output of PE_1. This fixed delay can be achieved by the insertion of w equally spaced registers in the feedback path; such insertion preserves local communication in the array. The feedback selection node (labeled "FSN" in Figure 4) controls the NE data input of PE_w so that the right computations can be made. In the FSN, only a multiplexer is needed to select either partial results or the elements of vector $\underline{\mathbf{B}}$. The multiplexer's "Select" signal is easily generated. The Select signal is defined by the DBT transformation used. We can observe that the i,j index sequences of the matrix **A** elements that enter through ports N of the SAP obey a regular pattern. Figure 4 shows the simple address generation required for the execution of the transformed problem.

Because the band of matrix $\underline{\mathbf{A}}$ is dense, it is only during $2w - 2$ load cycles and $w - 1$ unload cycles that array utilization is below the maximum of ½ for the SAP. The computation time is given by $T = (2N\,M/w) + 2w - 3$. Nevertheless, to reach a utilization value near unity, we should group into a physical PE the computations assigned to two consecutive PEs in the theoretical contraflow array. If we do this and if the number of available physical PEs is w_f, w must be equal to $2w_f$ for the DBT transformation, and the total computing time will be $T \approx N\,M/w_f$ with physical array utilization of $U \approx 1$. It is also possible to execute the transformed problem $\underline{\mathbf{Y}} = \underline{\mathbf{A}}\,\underline{\mathbf{X}} + \underline{\mathbf{B}}$ directly on a 1D SAP that has w PEs and is without data contraflow. In such a case, the computing time is $T = (N\,M/w) + 2w - 2$, and utilization approaches 1 without PE grouping.

Regular DBTs

All DBT transformations originate matrices with a dense band of minimum length. The transformations allow maximum array utilization and minimum computation time. However, the complexity of both input-data-address generation and of the feedback selection node depend on the DBT type used. In general, extra memory is required in the feedback selection node to store partial results during some cycles. The size of this memory and the input-data-address generation depend on the selected DBT algorithm. We call DBT algorithms "regular" when they permit global designs that have minimum complexity. Some regular DBTs that we will discuss below are shown in Figure 5. Regular DBT algorithms are classified into two groups: *standard* and *transposed*. In the standard group (Figure 5a), $\underline{N}\,\underline{M}$ square blocks from **A(N,M)** have been decomposed as $\mathbf{A}_{i,j} = \mathbf{A}_{Li,j} + \mathbf{A}_{DUi,j}$, for $1 \le i \le \underline{N}$ and $1 \le j \le \underline{M}$. The transformed matrix $\underline{\mathbf{A}}$ is an $\underline{N}\,\underline{M}$-by-$\underline{N}\,\underline{M} + 1$ block upper-band matrix. Figure 5a illustrates the order to follow in the selection of tri-

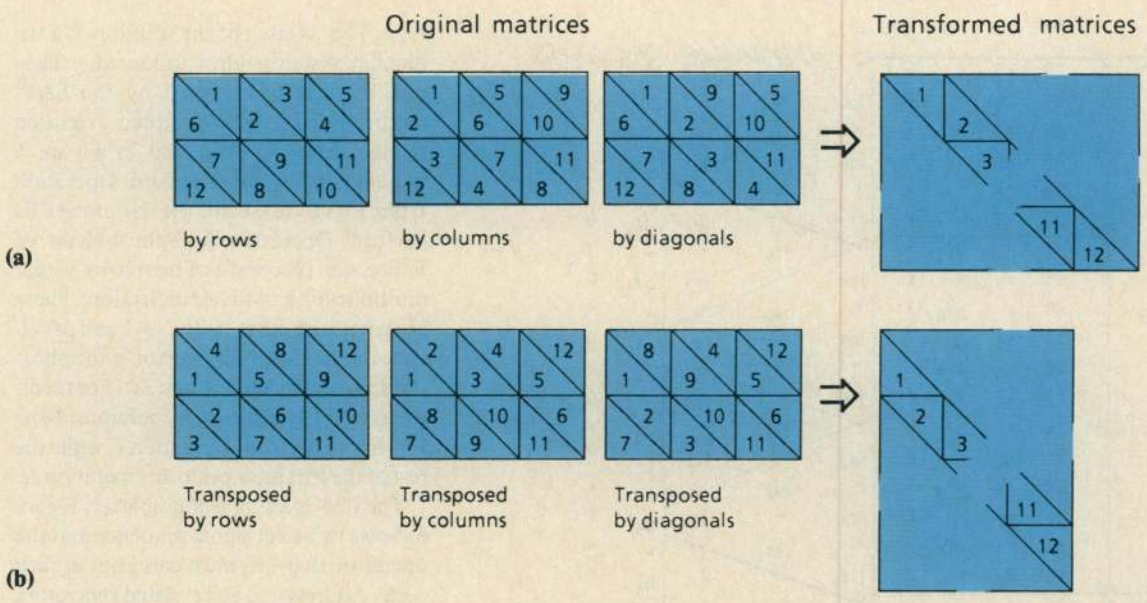


Figure 5. Regular DBT transformations. (a) Standard DBT algorithms. (b) Transposed DBT algorithms.

angular blocks when one is building matrix $\underline{\mathbf{A}}$ for different algorithms, whether by rows, columns, or diagonals, when $\underline{N}=2$ and $\underline{M}=3$. DBT transformations carried out along diagonals originate a dense band-transformed matrix with maximum band density only if $\underline{N}$ and $\underline{M}$ are relatively prime. In the previous section, DBT transformation carried out along rows was the applied algorithm.

We say that the second group of regular DBT algorithms is *transposed by rows, columns, or diagonals*. For this group, the decomposition is $\mathbf{A}_{i,j} = \mathbf{A}_{LDi,j} + \mathbf{A}_{Ui,j}$ for $1 \le i \le \underline{N}$ and $1 \le j \le \underline{M}$. $\underline{\mathbf{A}}$ is a lower-band matrix with $\underline{N}\,\underline{M}+1$-by-$\underline{N}\,\underline{M}$ blocks. The triangular-block selection order is illustrated in Figure 5b. We can obtain the same result by applying a standard DBT to the transposed matrix $\mathbf{A}$ and then transposing the resulting matrix. Note that the technique of Partial Row Translations[13] is an instance of regular DBT transformation if $\underline{N}$ and $\underline{M}$ are equal to 1.

Partitioning and execution technique

The DBT transformation rules we have considered so far are used to solve homogeneous problems, such as matrix-by-vector multiplications. The execution of a large nonhomogeneous problem (for example, a TSE) on a small SAP, which is performed by partitioning the problem into subproblems that are subsequently chained, may be viewed as the execution of a band problem in which different types of operations on different frames of the band are defined.

Our partitioning and execution technique consists of three steps.

(1) Split the problem into subproblems that, in accordance with the precedences, are executed one after another. The submatrices need not be triangular ones.

(2) Find a band SA for each subproblem. Execute directly those subproblems whose size fits the size of the available array dimensions.

(3) There are subproblems whose dimensions do not allow direct subproblem execution.

(a) If possible, execute them after they have been transformed to band subproblems. When the subproblems are matrix-by-vector or matrix-by-matrix types, some of the previously described DBT algorithms can be used. Otherwise, new transformation rules must be devised.

(b) When the procedures given in Steps 2 and 3a are not applicable, the subproblems must be partitioned once more, starting at Step 1.

In summary, all the subproblems are either directly executed or are executed after some sort of transformation. Our technique for finding the best partitioning is heuristic, and it must obtain the maximum possible overlapping between the loading and unloading of subproblems.

Triangular system of equations

In this section our concern is the solution of the triangular system equation $\mathbf{LX}=\mathbf{B}$. $\mathbf{L}$ is a lower triangular matrix with N-by-N dimensions. $\mathbf{X}$ and $\mathbf{B}$ are column vectors with N elements. The unknowns x_i for $1 \le i \le N$ are computed by means of forward substitutions. The problem is solved on a 1D SAP with w PEs; the SAP is illustrated in Figure 1c.

Hereafter in this article, we will use the following notation to denote submatrices of an N-by-M matrix $\mathbf{A}$. For example, block-row $\mathbf{A}_{i,a:b}$ refers to the juxtaposition of consecutive $b-a+1$ blocks of w-by-w elements from the same block-row:

$$\mathbf{A}_{i,a:b} = (\mathbf{A}_{i,a}\ \mathbf{A}_{i,a+1} \cdots \mathbf{A}_{i,b}).$$

Similarly, $\mathbf{A}_{a:b,j}$ is the block-column built up by the vertical juxtaposition of consecutive blocks:

$$\mathbf{A}_{a:b,j} = (\mathbf{A}_{a,j}\ \mathbf{A}_{a+1,j} \cdots \mathbf{A}_{b,j})^T.$$

The original problem is partitioned and executed by applying the previously

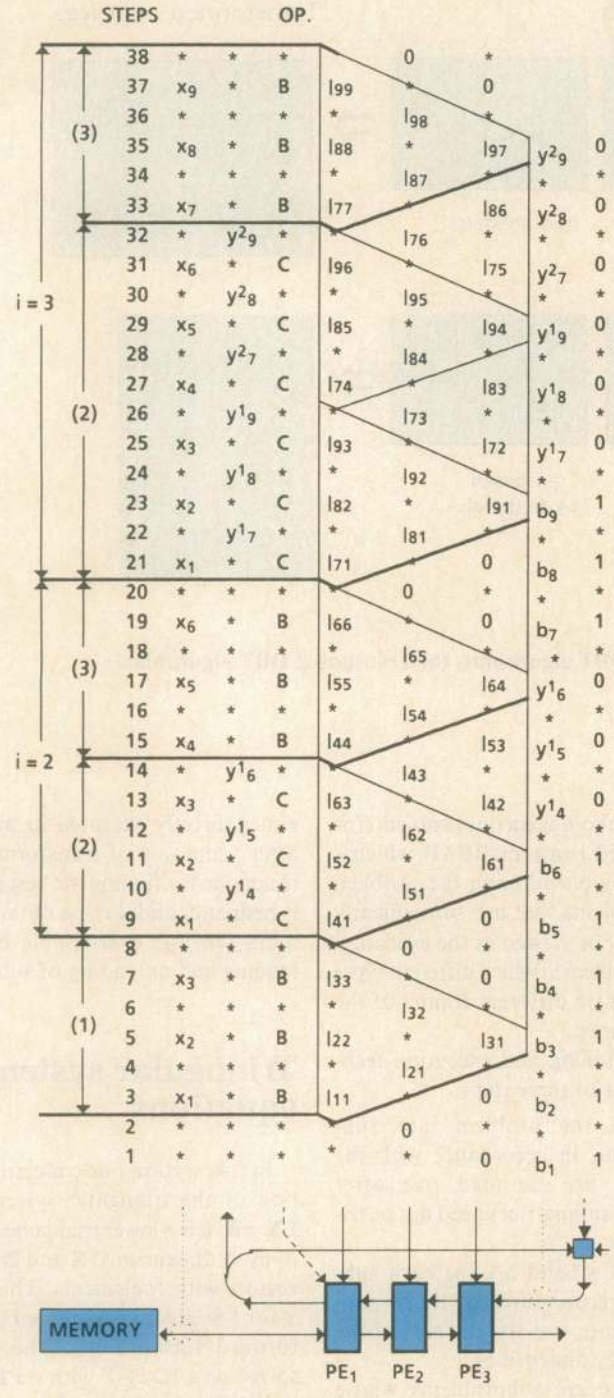


Figure 6. Chained execution of subproblems, and I/O data sequencing for a triangular system of equations.

described steps. The first-level partitioning algorithm and the order of execution of the subproblems is Step 1 as described in the section entitled "Partitioning and execution technique":

Compute X_1 from $L_{1,1}\, X_1 = B_1$ (1)
For i = 2 to $\underline{N}$ do
$B_i = B_i - L_{i,1:i-1}\, X_{1:i-1}$ (2)
Compute X_i from $L_{i,i}\, X_i = B_i$ (3)
End for

The I/O data sequences for executing these subproblems on a linear SAP with $N=9$ and $w=3$ are shown in Figure 6. Subproblems (1) and (3) are of the same type. They consist of the solution of a triangular system with w unknowns. They can be directly executed by the SAP[1] (according to Step 2). The direct execution by the array as presented in Figure 6 implies that PE_1 must perform Operation B (see Figure 1d), while the rest of the PEs perform Operation A. Subproblems of Expression (2) consist of matrix-by-vector multiplications with actualization. These are particular cases, with $\underline{N}=1$ and $\underline{M}=(i-1)$, of the matrix-by-vector multiplications presented above (Step 3a). For them, we use DBT transposed by columns. Now PE_1 has to perform Operation C while the rest of the PEs must perform Operation A.

The FSN is a simple multiplexer. Figure 6 shows its Select signal sequence and the operation that PE_1 must carry out in each cycle. Address and Select signal generation is simple. The number of cycles needed to solve the problem is $T=(N^2/w)+N+w-2$.

Matrix-by-matrix operation

We now consider the operation $\mathbf{E}=\mathbf{FG}+\mathbf{H}$ to be performed on the spiral systolic array processor (SSAP) with w-by-w PEs (Figure 2b); **F**, **G**, and **H** are, respectively, M-by-N, N-by-P, and M-by-P matrices. First we split the problem into $\underline{M}\,\underline{P}$ disjoint subproblems according to the following algorithm (this corresponds to Step 1 in the "Partitioning and execution technique" section):

For m = 1 to $\underline{M}$ do
For p = 1 to $\underline{P}$ do
$E_{m,p}=F_{m,1:\underline{N}}G_{1:\underline{N},p}+H_{m,p}$ (4)
End for
End for

The $\underline{M}\,\underline{P}$ subproblems (Expression (4)) are solved one after another on the SSAP. Every subproblem is of the type $\mathbf{D}=\mathbf{AB}+\mathbf{C}$, where **A**, **B**, and **C** are, respectively, w-by-N, N-by-w, and w-by-w submatrices. By means of DBT algorithms, the problem $\mathbf{D}=\mathbf{AB}+\mathbf{C}$ is transformed to a banded one: $\underline{\mathbf{D}}=\underline{\mathbf{A}}\,\underline{\mathbf{B}}+\underline{\mathbf{C}}$ (Step 3); see Figure 7 (matrices **C** and $\underline{\mathbf{C}}$ have been omitted in the figure). By applying DBT transposed by columns to matrix **A**, and DBT by columns to matrix **B**, matrices $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$ are obtained. Matrix $\underline{\mathbf{A}}$ is an $(\underline{N}+1)$-by-$\underline{N}$ block lower-band matrix; $\underline{\mathbf{B}}$ is an $\underline{N}$-by-$(\underline{N}+1)$ block upper-band matrix with blocks of w-by-w elements. We define, now, $\underline{\mathbf{C}}$ as a tridiagonal $(\underline{N}+1)$-by-$(\underline{N}+1)$ block

matrix in which $\underline{C}_{i,j} = \mathbf{C}$ for $i = j = 1$ and $\underline{C}_{i,j} = 0$ otherwise.

Figure 7a shows the triangular-block partitioning for the problem $\mathbf{D} = \mathbf{AB} + \mathbf{C}$, and Figure 7b shows the band problem $\underline{\mathbf{D}} = \underline{\mathbf{A}}\,\underline{\mathbf{B}} + \underline{\mathbf{C}}$. The result $\underline{\mathbf{D}}$ is a tridiagonal $(\underline{N}+1)$-by-$(\underline{N}+1)$ block matrix that can be evaluated by means of a 2D SAP with w-by-w PEs.[1] This evaluation can be accomplished by the SSAP if all PEs perform Operation A. Matrix $\mathbf{D}$ can be derived from $\underline{\mathbf{D}}$ if one adds the $\mathbf{D}_{i,j}$ blocks for $1 \le y, j \le \underline{M}+1$. This computation may be performed inside the array by means of the spiral feedback lines without producing any time overhead. The elements on the main diagonal of $\mathbf{D}_{i,j}$ are required $2w$ cycles later than their appearance in the array output. The other elements must be delayed w cycles after their appearance in the output of the North and West ports before being input into the South and East ports. Hence, we insert $2w$ equally spaced registers in the diagonal feedback path and w registers in the other feedbacks. In this way, we preserve the local communication requirement.

The original problem is solved by chaining $\underline{M}\,\underline{P}$ subproblems, as in the problem considered above. The selected transformation requires the insertion of two zero blocks between subproblems; one is an L block and the other is a U block. In this case, the total computation time is $(3MPN/w^2) + (3MP/w) + w$. A technique to chain subproblems without zero blocks already exists.[10] The improved computation time is $(3MPN/w^2) + 4w$, but control is slightly more complex. For the SSAP with contraflow, the maximum theoretical utilization of 1/3 is reached when $NMP >> w^3$. Utilization reaches 1 if we group the computations performed by three theoretical PEs into one physical PE. The original problem can also be solved in an SAP without contraflow, where it achieves $U \approx 1$ without PE grouping.

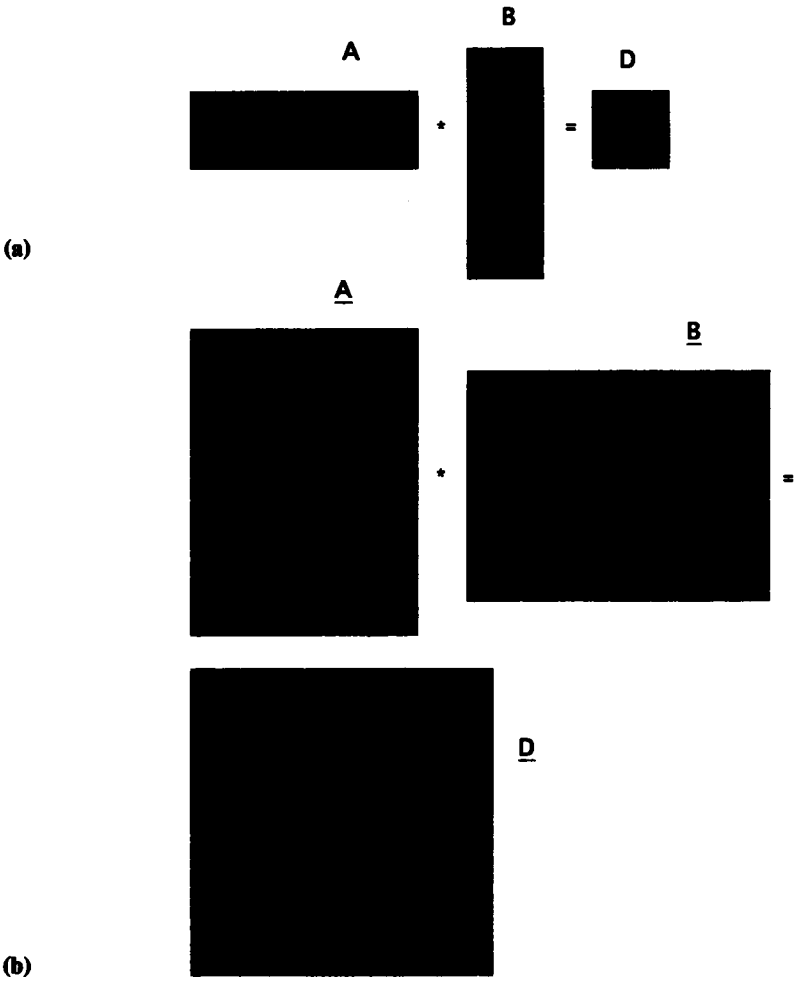


Figure 7. (a) Triangular-block partitioning of a matrix-by-matrix problem. (b) The DBT-transformed problem.

Triangular matrix equations

A *matrix equation* is a set of linear systems, all sharing the coefficient matrix but with different right-hand-side vectors. In this section, we are concerned with the solution of the lower triangular matrix equation $\mathbf{LY} = \mathbf{B}$ to be performed on a w-by-w SSAP; $\mathbf{L}$ is an N-by-N lower triangular matrix and $\mathbf{Y}$ and $\mathbf{B}$ are N-by-M matrices. In the first level of partitioning, the problem is split into $\underline{M}$ independent subproblems, each one with $\mathbf{Y}$ and $\mathbf{B}$ of dimensions N-by-w. They are executed one after another. Each N-by-w system is partitioned and executed following the algorithm

Compute $\mathbf{Y}_1$ from $\mathbf{L}_{1,1}\mathbf{Y}_1 = \mathbf{B}_1$ (5)

For i = 2 to $\underline{N}$

$\mathbf{B}_i = \mathbf{B}_i - \mathbf{L}_{i,1:i-1}\mathbf{Y}_{1:i-1}$ (6)

Compute $\mathbf{Y}_i$ from $\mathbf{L}_{i,i}\mathbf{Y}_i = \mathbf{B}_i$ (7)

End for

(This corresponds to Step 1.)

Subproblems (5) and (7) are the solution of w-by-w matrix equations. Consider, for example, Subproblem (5). The matrix equation size does not allow a direct execution, which corresponds to Step 3. For this reason, Subproblem (5) is decomposed as follows:

Compute $\mathbf{Y}_{DU1}$ from

$(\mathbf{L}_{1,1}\mathbf{Y}_{DU1})_{DU} = \mathbf{B}_{DU1}$ (8)

$\mathbf{B}'_{L1} = \mathbf{B}_{L1} - (\mathbf{L}_{1,1}\mathbf{Y}_{DU1})_L$ (9)

Compute $\mathbf{Y}_{L1}$ from $\mathbf{L}_{1,1}\mathbf{Y}_{L1} = \mathbf{B}'_{L1}$ (10)

Subproblems (8) and (9) are computed simultaneously by the SSAP when the boundary PEs are programmed in such a way that

- $PE_{1,1}$ performs Operation C,
- $PE_{1,j}$ for $2 \le j \le w$ performs Operation B, and
- the rest of the PEs perform Operation A.

In order to get the desired result, $\mathbf{L}_{1,1}$ must be input through the West boundary PEs, and $\mathbf{B}_1$ through the South and East PEs. Result $\mathbf{Y}_{U1}$ is obtained from the North PEs. At the same time, matrix $\mathbf{B}'_{L1}$ is obtained from West PEs. Subproblem (10) is perfectly chained to the previous

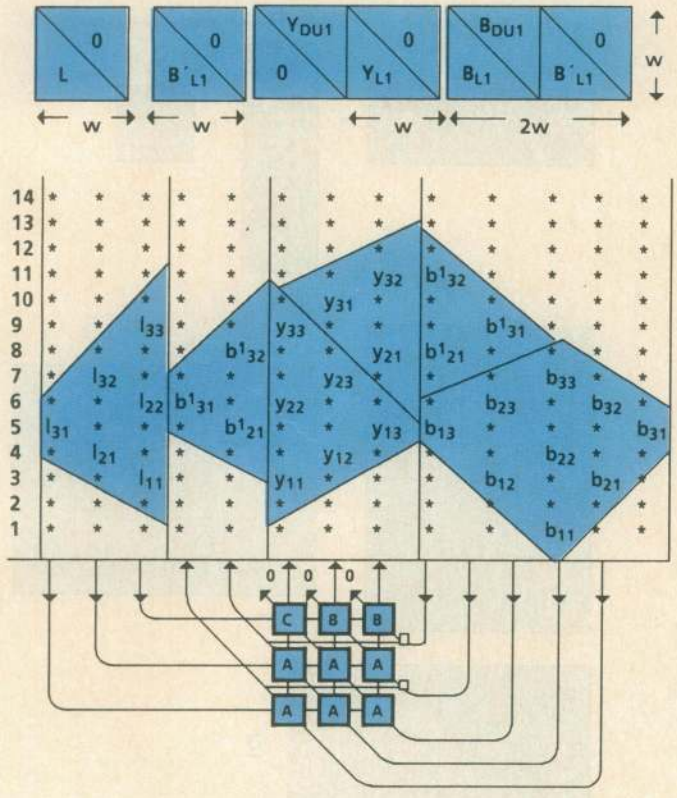


Figure 8. Chained execution of and I/O data sequencing for the three subproblems that result from partitioning of the *w*-by-*w* triangular matrix equation.

one if $\mathbf{B'}_{L1}$ is input through the PEs, and $\mathbf{Y}_{L1}$ is obtained from North PEs. Figure 8 shows the chaining of Subproblems (8), (9), and (10). Subproblem (6) is a matrix actualization that is solved by means of DBT. Observe that now $PE_{1,j}$ for $1 \leq j \leq w$ must perform Operation D (Figure 2c) to change the sign of the previously computed matrix, $\mathbf{Y}_{1:i-1}$. The computation time needed to find the unknown matrix **Y** with *N*-by-*M* elements is $T = (3MN^2/2w^2) + (3MN/2w) + w$.

LU decomposition

Given a square matrix **A** with dimension *N*, two triangular matrices, **L** (lower) and **U** (upper), must be found to verify that **A** = **LU**. We assume that **A** can be decomposed and that each of the diagonal elements of **L** is equal to 1. The LU decomposition problem is carried out in the *w*-by-*w* SSAP.

In the first level of partitioning, matrix **A** is divided into square blocks, $\mathbf{A}_{i,j}$; each block has *w*-by-*w* elements. Additionally, matrices **L** and **U** are split into their respective blocks; each block has *w* rows and columns. The partitioning method for and the execution sequence of the resulting subproblems are specified in the following algorithm:

Compute $L_{1,1}$ and $U_{1,1}$ from

$$A_{1,1} = L_{1,1}U_{1,1} \quad (11)$$

For i = 2 to N do

Compute $U_{1:i-1,i}$ from

$$L_{1:i-1,1:i-1}U_{1:i-1,i} = A_{1:i-1,i} \quad (12)$$

Compute $L_{i,1:i-1}$ from

$$L_{i,1:i-1}U_{1:i-1,1:i-1} = A_{i,1:i-1} \quad (13)$$

Compute $A_{i,i} =$

$$A_{i,i} - L_{i,1:i-1}U_{1:i-1,i} \quad (14)$$

Compute $L_{i,i}$ and $U_{i,i}$ from

$$A_{i,i} = L_{i,i}U_{i,i} \quad (15)$$

End for

(The algorithm corresponds to Step 1.)

Subproblems (11) and (15) are of the same type: The LU decomposition of a matrix $\mathbf{A}_{i,i}(\mathbf{w},\mathbf{w})$. Both can be directly solved by the SSAP (corresponding to Step 2). Matrix $\mathbf{A}_{i,i}$ must enter the array through East and South PEs, and matrices $\mathbf{L}_{i,i}$ and $\mathbf{U}_{i,i}$ go out, respectively, through West and North PEs. The operations performed by each PE are as follows (Figure 2c): $PE_{1,1}$ performs Operation E; $PE_{1,j}$ for $2 \leq j \leq w$ perform Operation B; $PE_{j,2}$ for $2 \leq i \leq w$ perform Operation F; and the rest of PEs perform Operation A.

Subproblems (12) and (13) are of the same type: One is the transposition of the other. Each one is the solution of a triangular system of matrix equations, **LX** = **B**; this system of equations has been addressed previously. The subproblem in Expression (14) is a matrix-by-matrix multiplication with actualization. We have commented before that such a subproblem is solved by applying DBT transformation (Step 3a).

In the chaining of Subproblems (15), (12), (13), and (14), some blocks of zeroes are required, but they are not necessary between Subproblems (14) and (15). The total execution time is given by $T = (N^3/w^2) + (3N^2/2w) + (N/2) + w$.

Other problems

Given the problems whose solutions we have already demonstrated, it is easy to solve other problems, such as matrix equations and matrix inversions. For instance, to compute the inverse of a matrix **A**, we can apply the following algorithm.[11]

(a) Perform LU decomposition of matrix **A**.

(b) Solve the lower triangular matrix equation, which is $\mathbf{LL}^{-1} = \mathbf{I}$ where $\mathbf{L}^{-1}$ is the inverse of the triangular matrix **L**, and **I** is the identity matrix. The regular sparsity of matrices **L** and $\mathbf{L}^{-1}$ is taken into account in the application of DBT transformations; such application allows us to optimize computing time to $T = (N^3/2w^2) + (3N^2/2w) + N + w$.

(c) Solve the upper-triangular matrix equation, which is $\mathbf{U}^{-1}\mathbf{U} = \mathbf{I}$; this step is analogous to Step (b).

(d) Compute the matrix product $\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$. The computing time, which is improved as in Step (b) by exploiting the triangularity of the involved matrices, is $T = (N^3/w^2) + (3N^2/2w) + (N/2) + w$. The resulting matrix-inversion time is $T = (3N^3/w^2) + (6N^2/w) + (3N) + w$.

Pipelined processing elements

In the design of SAPs, whether they have or do not have data contraflow, it is

desirable to use pipelined arithmetic units to increase throughput and, consequently, to decrease computing time. When arrays with neither intra-PE cycles nor data contraflow are used, it is easy to attain an efficient use of two-level pipelining.[14] If SAPs with data contraflow or intra-PE cycles are considered, two techniques can be used to take advantage of pipelining when we execute big problems on small arrays:

(a) grouping the input of adjacent data streams into an individual physical PE and
(b) overlapping (that is, parallel) execution of independent or chainable subproblems that result from the original problem's decomposition.

The number of grouped streams in (a) and the number of parallel subproblems executed in (b) are proportional to the number of pipeline stages in the arithmetic units.

The structures present in contraflow SAPs are diagonals. They are grouped after the DBT algorithm is applied. This algorithm balances the load allocated to each processor and reduces to a minimum loading and unloading times. The required local memory is a function only of the available SAP dimensions, since it is independent of the size of the problem to be solved.

Subproblems considered in Technique Step (b) are obtained from a first-level partition. These subproblems fit one or two dimensions of the available array. They are transformed in a later step.

Matrix-by-vector problem execution that involves the application of a DBT transposed by columns algorithm and diagonal groupings (Technique (a)) is depicted in Figure 9a. The matrix size is 8-by-8, and the array has $w = 2$ PEs. Each PE includes an adder and a multiplier, both of which have three pipelining stages. There are $4w$ registers in the feedback path.

At present, systolic array processors are a viable choice for solving a wide class of matrix problems. Systolic algorithms have usually been conceived with the assumption that an unlimited number of processing elements are available. The necessity for partitioning problems appears when the algorithm requires more processing elements than exist in the array. Partitioning is an essential step in mapping algorithms into both algorithm-

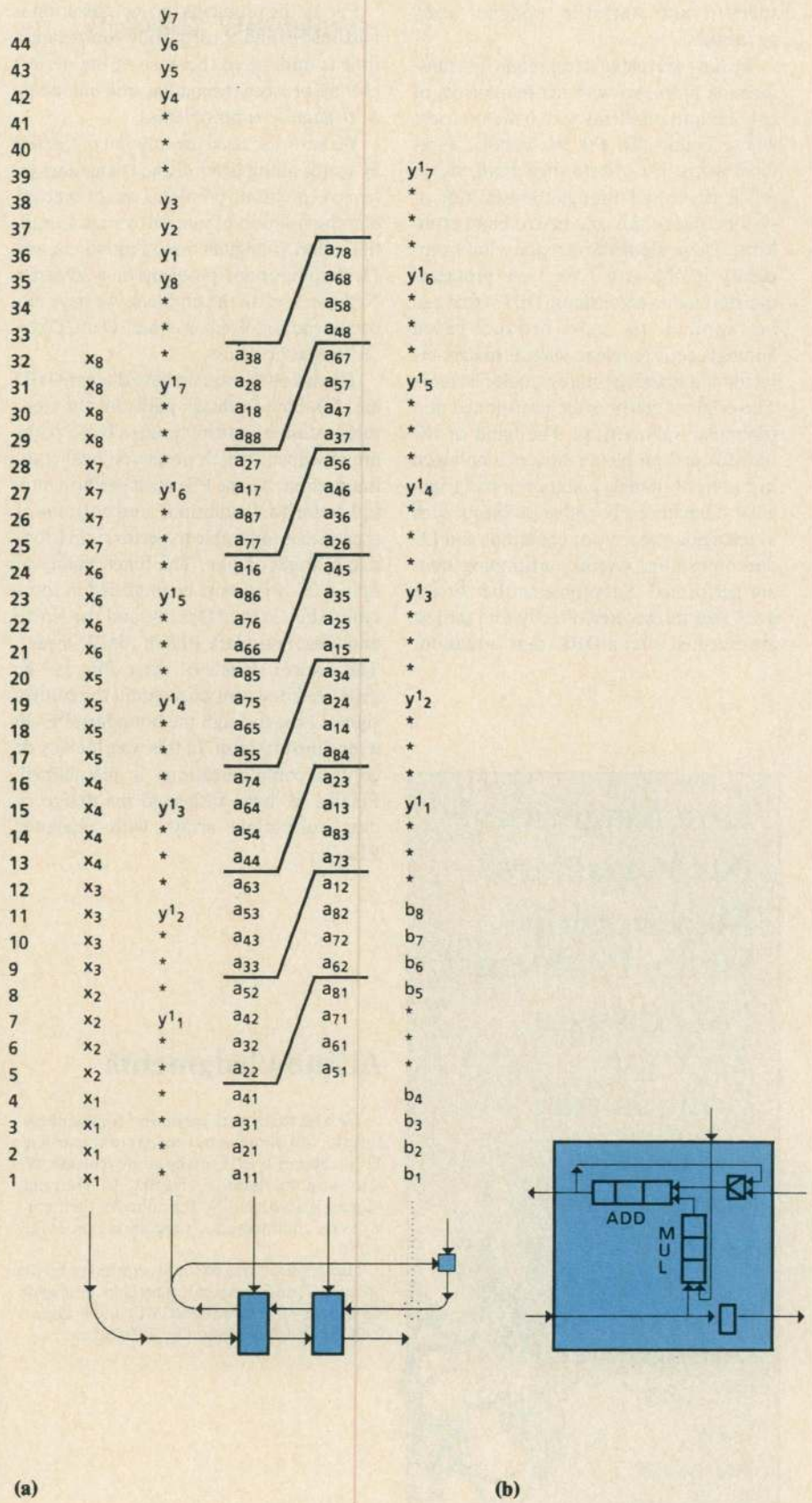


Figure 9. (a) I/O data sequencing for matrix-by-matrix execution in a 1D systolic array with pipelined PEs. (b) A pipelined PE.

specific and versatile systolic array processors.

We have presented a technique for transforming problems with dense matrices of any size into problems with band matrices whose bandwidth fits the systolic array dimensions. The transformed problems are efficiently solved through the execution of systolic algorithms oriented to band problems. These algorithms require low complexity in the array. We have proposed transformation algorithms (DBTs) that can be applied to inner-product-based homogeneous problems such as matrix-by-vector and matrix-by-matrix multiplication. The original matrices are partitioned into triangular submatrices. The band of the transformed-problem matrices is obtained by means of suitable juxtaposition of triangular submatrices. For other problems, such as triangular systems of equations and LU decomposition, several partitioning steps are performed. Subproblems that fit the array size are executed directly, and the rest are executed after a DBT transformation.

For all the problems, array utilization is maximized, and consequently computation time is minimized, because of the perfect overlap between the loading and unloading of triangular subproblems.

We have discussed the solution of matrix-by-vector multiplication and triangular system of equations problems on a 1D array, and the solution of matrix-by-matrix multiplication, triangular matrix equations, and LU decomposition problems on a 2D array. Nevertheless, all the problems we have discussed can be solved on either 1D or 2D versatile systolic arrays.

The systolic arrays we have discussed (1D and 2D) have feedback paths for the communication of partial results. These paths are pipelined, which preserves local communication. All the PEs must perform multiplications and additions, and only one of them needs to be able to perform divisions and changes of sign. The functionality of only a few PEs must be modified in some cycles (PE_1 in the 1D array, and the North and West boundary PEs in the 2D array). The control signals of these PEs can be generated by a centralized unit; the control signals flow through the boundary PEs in a pipelined fashion. In this way, locality of control communications is maintained. Finally, we have addressed the design of data contraflow arrays with pipelined PEs.□

Acknowledgments

We wish to thank the reviewers for their helpful criticism of previous versions of this article. Their comments led to many improvements. We also wish to thank F. Nuñez, E. Herrada, Miguel Valero, and N. Torralba for their constructive comments and suggestions about this work.

This research was partially supported by the Spanish Comisión Asesora para la Investigación Científica y Técnica (CAICYT) under Grants 2906-83 and 314-85, and by Telefónica.

References

1. H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI)," *Sparse Matrix Proc. 1978,* 1979, Society for Industrial and Applied Mathematics (SIAM), pp. 256-282. (A slightly different version appears in the text *Introduction to VLSI Systems,* Section 8.3, C.A. Mead and L.A. Conway, eds., 1980, Addison-Wesley, Reading, Mass.)
2. H.T. Kung, "Why Systolic Architectures?" *Computer,* Vol. 15, No. 1, Jan. 1982, pp. 37-46.
3. J.A.B. Fortes, S.Y. Fu, and B.W. Wah, "Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays," *Proc. Int'l Conf. Acoustics, Speech, and Signal Processing,* 1985, pp. 300-303.
4. L. Johnsson, "Computational Arrays for Band Matrix Equations," tech. report 4287, May 1981, Computer Science Dept., California Institute of Technology, Pasadena, Calif.
5. R. Schreiber and P.J. Kucks, "Systolic Linear Algebra Machines in Digital Signal Processing," in Chapter 22 of *VLSI and Modern Signal Processing,* S.Y. Kung, H.J. Whitehouse, and T. Kailath, eds., 1985, Prentice-Hall, Englewood Cliffs, N.J., pp. 389-405.
6. D. Heller, "Partitioning Big Matrices for Small Systolic Arrays," in Chapter 11 of *VLSI and Modern Signal Processing,* S.Y. Kung, H.J. Whitehouse, and T. Kailath, eds., 1985, Prentice-Hall, Englewood Cliffs, N.J., pp. 185-199.
7. H.Y.H. Chuang and G. He, "A Versatile Systolic Array for Matrix Computations," *Proc. 12th Int'l Symp. Computer Architecture,* 1985, pp. 315-322.
8. H.D. Cheng and K.S. Fu, "Algorithm Partition for Fixed-Size VLSI Architecture Using Space-Time Domain Expansion," *Proc. Seventh Symp. Computer Arithmetic,* 1985, pp. 126-132.
9. D.I. Moldovan and J.A.B. Fortes, "Partitioning and Mapping Algorithms Into Fixed Size Systolic Arrays," *IEEE Trans. Computers,* Vol. C-35, No. 1, Jan. 1986, pp. 1-12.
10. J.J. Navarro, J.M. Llaberia, and M. Valero, "Computing Size-Independent Matrix Problems on Systolic Array Processors," *13th Int'l Symp. Computer Architecture,* 1986, pp. 271-279.
11. J.J. Navarro, J.M. Llaberia, and M. Valero, "Solving Matrix Problems with No Size Restriction on a Systolic Array Processor," *Int'l Conf. Parallel Processing,* Aug. 1986, pp. 676-683.
12. J.J. Navarro et al., "LU Decomposition with No Size-Restriction Using a One-Dimensional Systolic Array Processor," *Proc. Second Int'l Conf. Supercomputing,* May 1987, Vol. 3, p. 218.
13. R.W. Priester et al., "Signal Processing with Systolic Arrays," *Proc. Int'l Conf. Parallel Processing,* 1981, pp. 207-215.
14. H.T. Kung and M. Lam, "Wafer-Scale Integration and Two-Level Pipelined Implementations of Systolic Arrays," *J. Parallel and Distributed Processing,* Vol. 1, No. 1, Aug. 1984, pp. 32-63.
15. S.Y. Kung, "VLSI Array Processors," *IEEE ASSP Magazine,* Vol. 2, No. 3, July 1985, pp. 4-22.

Juan J. Navarro is an assistant professor at the Facultad de Informática de Barcelona of the Universidad Politécnica de Cataluña, Spain, and a member of the Computer Architecture Dept.

His current research interests include VLSI architectures, parallel processors and algorithms, and architectures for digital signal processing.

Navarro earned engineering (1982) and PhD degrees (1986) from the Universidad Politécnica de Cataluña in telecommunication engineering and computer science, respectively.

José M. Llaberia is an assistant professor at the Facultad de Informática de Barcelona of the Universidad Politécnica de Cataluña.

His teaching and research activities are developed at the Computer Architecture Dept.; these involve systolic array processors, the design of VLSI architectures, and design and performance evaluation of interconnection networks for multiprocessor systems.

Llaberia received the telecommunication engineering degree in 1979, computer science degree in 1981, and the PhD in computer science in 1983, all from the Universidad Politécnica de Cataluña.

He is a member of IEEE.

Mateo Valero was an assistant professor at the Universidad Politécnica de Cataluña (Escuela de Telecommunicatión) from 1974 to 1980. Since October 1980, he has been at the Facultad de Informática de Barcelona. In 1983 he was made a full professor of computer architecture. Between May 1984 and December 1985 he was the dean of the Facultad de Informática. He is currently head of the Computer Architecture Dept.

Valero's teaching and research interests are in computer architecture, with emphasis on the design and performance evaluation of interconnection networks for multiprocessor systems, for local area networks, for systolic array processors, and for RISC-like processors.

He received the telecommunication engineering degree from the Universidad Politécnica de Madrid in 1974 and his PhD in telecommunication engineering from the Universidad Politécnica de Cataluña in 1980.

Valero is a member of IEEE and is vice president of the Spanish Chapter of the Computer Society of the IEEE.

Readers may write to Juan J. Navarro at the Departamento de Arquitectura de Computadores, Facultad de Informática (UPC), Pau Gargallo, 5.08028 Barcelona, Spain; phone 34 (3) 333-83-08.