Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

COMMENTS TO "A SYSTOLIC ARRAY FOR COMPUTING BA^{-1}

Jaime H. Moreno
Tomas Lang

July 1988
CSD-880048

Comments to "A Systolic Array for Computing BA^{-1}"

Jaime H. Moreno and Tomás Lang*
Computer Science Department
University of California Los Angeles
3680 Boelter Hall, Los Angeles, Calif. 90024

Abstract

An algorithm and a systolic array for computing BA^{-1} was presented in [1], where A and B are n by n and p by n matrices, respectively. Such an array computes BA^{-1} in $(4n+p-2)$ time units using $n(n+1)$ processing elements (PEs). In this correspondence, we apply a graph–based method for the design of systolic arrays to such algorithm. We systematically derive the array in [1] and another array that also performs the computation in the same time but using only $[\frac{1}{2}n(n+1)+pn]$ units. For $p < \frac{1}{2}(n+1)$, our array uses fewer PEs than the array in [1]. Moreover, our array exhibits throughput $(n+1)^{-1}$, high utilization of PEs, and $(n+2p)$ I/O ports, while the array in [1] has lower performance for these measures.

1 Introduction

In [1], Comon and Robert introduced a systolic array of $n(n+1)$ elementary processors that computes BA^{-1} in $(4n+p-2)$ time steps, where A is a dense (non–singular) n by n matrix, and B is a dense p by n matrix. Moreover, such an array can be directly extended to compute the vector $Y = BA^{-1}R$, where R is a vector with n components. These computations arise frequently in signal processing applications, as described by Comon and Robert and other researchers [2,3,4]. The algorithm used in [1] is shown in Figure 1.

In this correspondence, we apply a graph–based method for the design of systolic arrays [5] to the algorithm in Figure 1. Such method uses a fully–parallel dependency–graph as the description of the algorithm and performs transformations on the graph to derive an implementation. We derive the array in [1] and another array that computes the algorithm in the same time but using only $[\frac{1}{2}n(n+1)+pn]$ units. Moreover, this new array exhibits throughput $(n+1)^{-1}$ and requires $(n+2p)$ I/O ports, while these measures are $(n+p)^{-1}$ and $2n$ respectively for the the array in [1].

*J. Moreno has been supported by an IBM Computer Sciences Fellowship. This research has also been supported in part by the Office of Naval Research, Contract N00014–83–K–0493 "Specifications and Design Methodologies for High–Speed Fault–Tolerant Algorithms and Structures for VLSI"

For $k = 1$ to n
begin
$c_{kk}^{(k)} = 1/c_{kk}^{(k-1)}$
For $j = 1$ to n, $j \neq k$
$c_{kj}^{(k)} = -c_{kk}^{(k)} c_{kj}^{(k-1)}$
For $i = (k+1)$ to $(n+p)$
begin
For $j = 1$ to n, $j \neq k$
$c_{ij}^{(k)} = c_{ij}^{(k-1)} + c_{ik}^{(k-1)} c_{kj}^{(k)}$;
$c_{ik}^{(k)} = c_{ik}^{(k-1)} c_{kk}^{(k)}$;
end
end

$$C^0 = \begin{bmatrix} A \\ B \end{bmatrix} \quad , \quad C^n = \begin{bmatrix} F \\ D \end{bmatrix} \quad , \quad D = BA^{-1}$$

Figure 1: Algorithm to compute BA^{-1}

2 Systolic arrays for computing BA^{-1}

Figure 2 depicts the fully–parallel dependency–graph for the algorithm in Figure 1, where A is a 4 by 4 matrix and B is a 2 by 2 matrix. Such graph is obtained by tracing the execution of the sequential algorithm in Figure 1 (i.e., symbolic execution of the algorithm that tracks what variables are used and when). This graph is not suitable for an implementation, because it exhibits broadcasting of data, bi–directional data flow, too many nodes and $O(n^2)$ I/O bandwidth.

We first transform the fully–parallel dependency–graph in Figure 2 into a tri–dimensional graph. We refer to such tri–dimensional graph as a "multi–mesh dependency–graph." To achieve such transformation, we replace broadcasting by transmittent data [6], remove bi–directional data flow by moving nodes to one side of the sources of broadcasting, and add delay nodes so that dependencies are strictly between neighbor nodes. Specific procedures for these transformations have been previously described [5]. The resulting graph, shown in Figure 3, consists of parallel meshes of nodes that are dependent, although such meshes do not have the same number of nodes.

According to our graph–based method, we now transform the multi–mesh dependency–graph into a two–dimensional graph (i.e., a mesh–dependency graph) by collapsing *parallel paths* of the graph onto single nodes. Each path is collapsed onto a different node. In other words, nodes in a path of the graph are grouped into a single node whose functionality and computation time are given by the primitive nodes. Such grouping can be regarded as projecting the three–dimensional dependency–graph onto a two–dimensional graph along any of the three dimensions. Consequently, there are three alternatives while performing such grouping, one in each dimension, leading to arrays that exhibit different performance measures as discussed below. Those measures are a consequence of the quantity and type of primitive nodes that are collapsed onto a single node. For example, if all groups have the same number of primitive nodes, then the resulting array will exhibit optimal utilization.

We present now two alternative groupings, along axes X and Y. Grouping along the Z–axis is

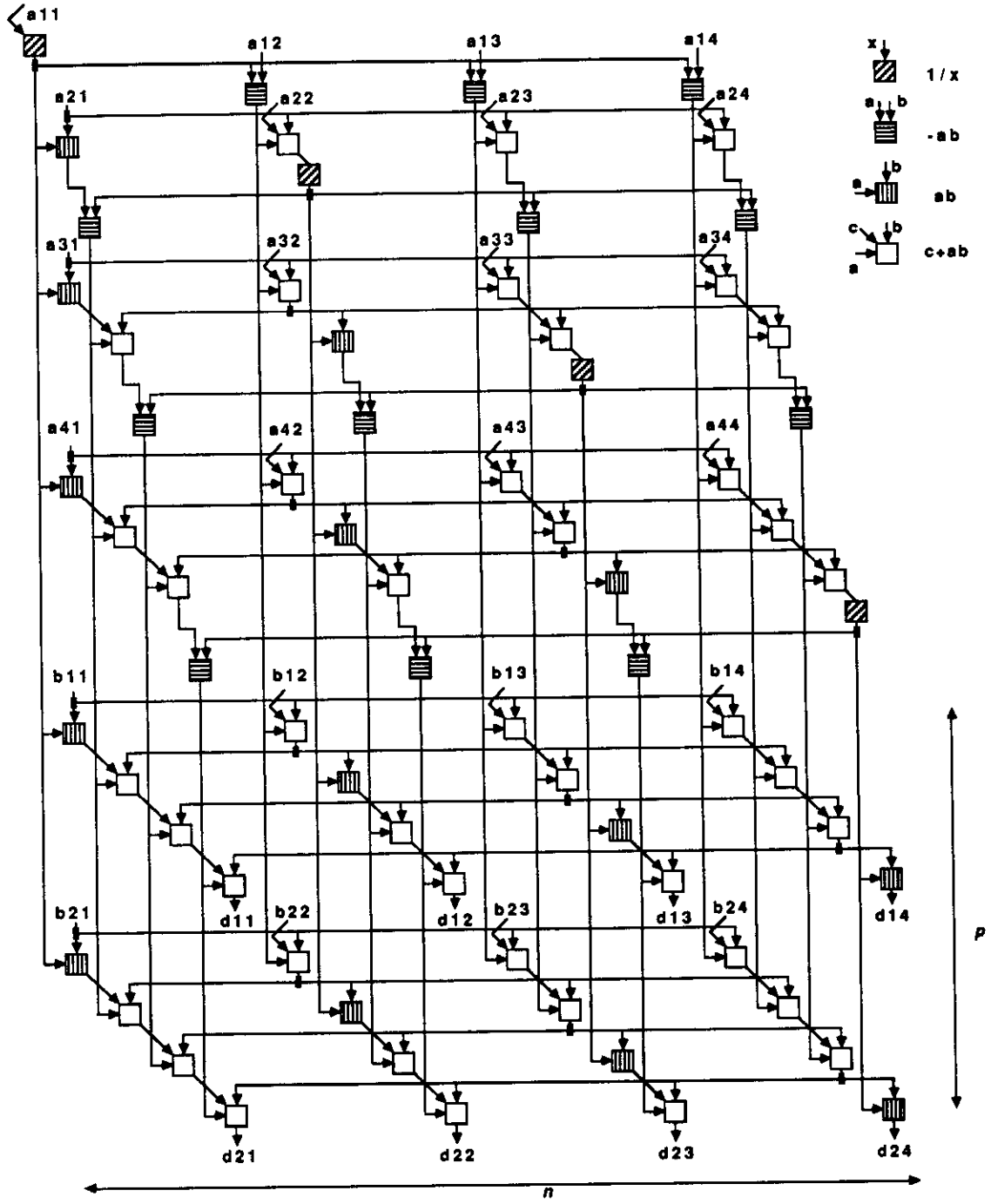

Figure 2: Fully–parallel dependency–graph for computing BA^{-1}

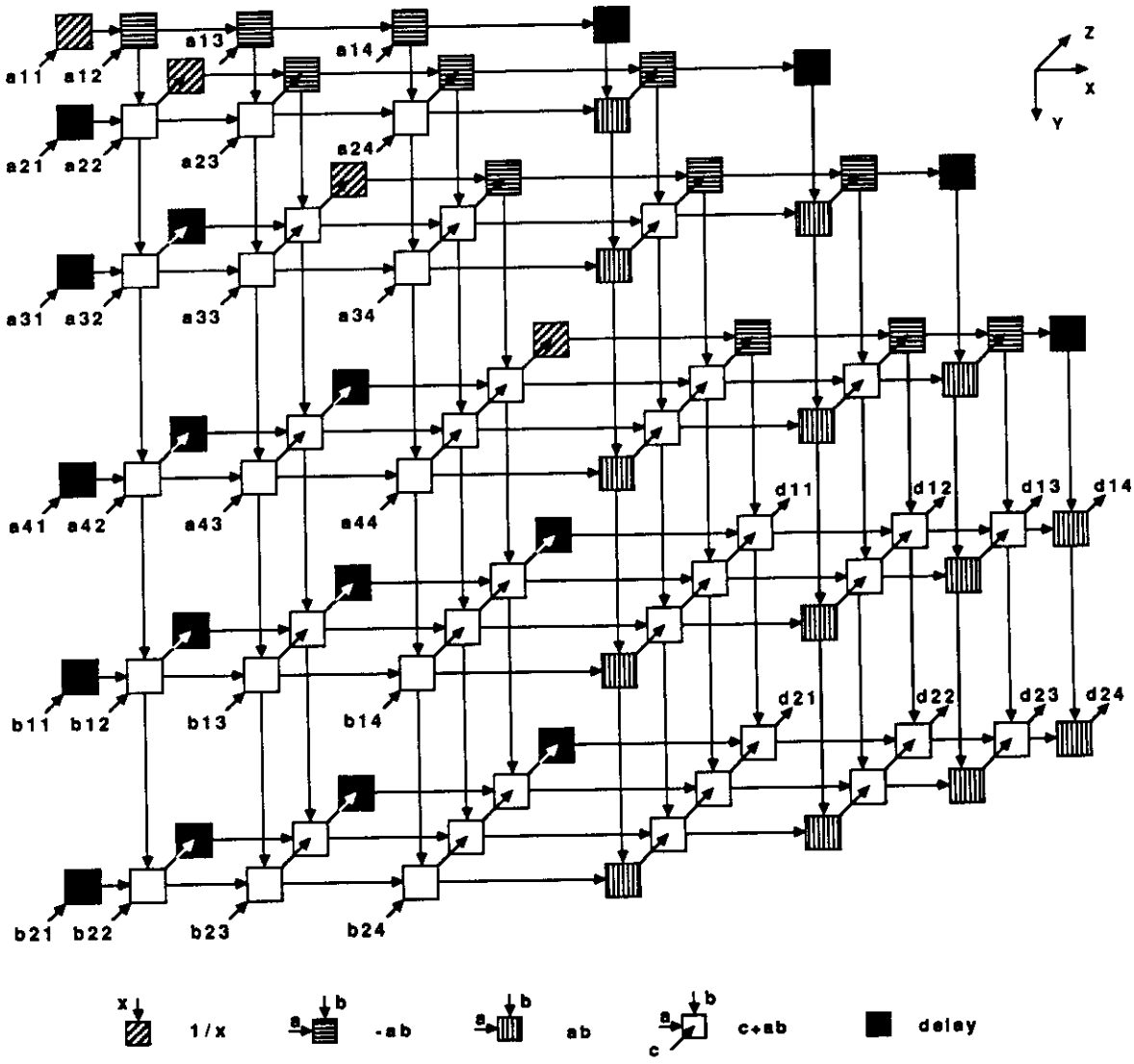

Figure 3: Multi-mesh dependency-graph for computing BA^{-1}

significantly less efficient so that it doesn't merit discussing it here.

2.1 Grouping along Y–axis

The first alternative that we consider consists of grouping nodes in the multi–mesh dependency–graph by vertical paths, that is, each vertical path in the graph is collapsed onto a different node. The resulting graph is shown in Figure 4a, which leads to the array shown in Figure 4b. This array corresponds to the one proposed by Comon and Robert. The performance of such an array is discussed later.

2.2 Grouping along X–axis

A second alternative consists of grouping nodes in the multi–mesh by horizontal paths, that is, each horizontal path in the graph is collapsed onto a different node. The graph obtained from such grouping is shown in Figure 5a, which can be mapped onto the array shown in Figure 5b. Performance of this array is described below.

2.3 Comparison of the arrays

Table 1 summarizes the characteristics and performance of the two arrays derived here. From such table, we conclude that the array derived from grouping nodes along the X–axis, shown in Figure 5, has better performance measures than Comon and Robert's scheme (obtained by grouping along the Y–axis) because of the following reasons:

- It computes the algorithm in the same number of steps but using fewer units (if $p < n/2$). Such improvement is achieved by having all cells highly utilized, while this is not the case in Comon and Robert's scheme.
- Throughput of the array while computing successive instances of the algorithm (i.e., computation of the algorithm for different sets of data), is independent of p.
- Lower I/O bandwidth.

However, Comon and Robert's scheme has the advantage that it can compute BA^{-1} for successive matrices $B_1, B_2, \ldots$ on the same array, without modifications. Such a case corresponds to extending the multi–mesh graph shown in Figure 3 along the Y–axis so that added nodes become part of existing groups when grouping along the Y–axis. This is in contrast to grouping along the X–axis, where the added nodes lead to more cells (successive matrices B should be handled as a partitioning problem in such a case, as mentioned below).

The graphs in Figures 4a and 5a are also suitable for partitioning the algorithm (i.e., computing a large problem on a small size array), using the partitioning method described in [7]. However, for this case the graph derived from grouping along the X–axis is more convenient because of the identical computation time of all nodes.

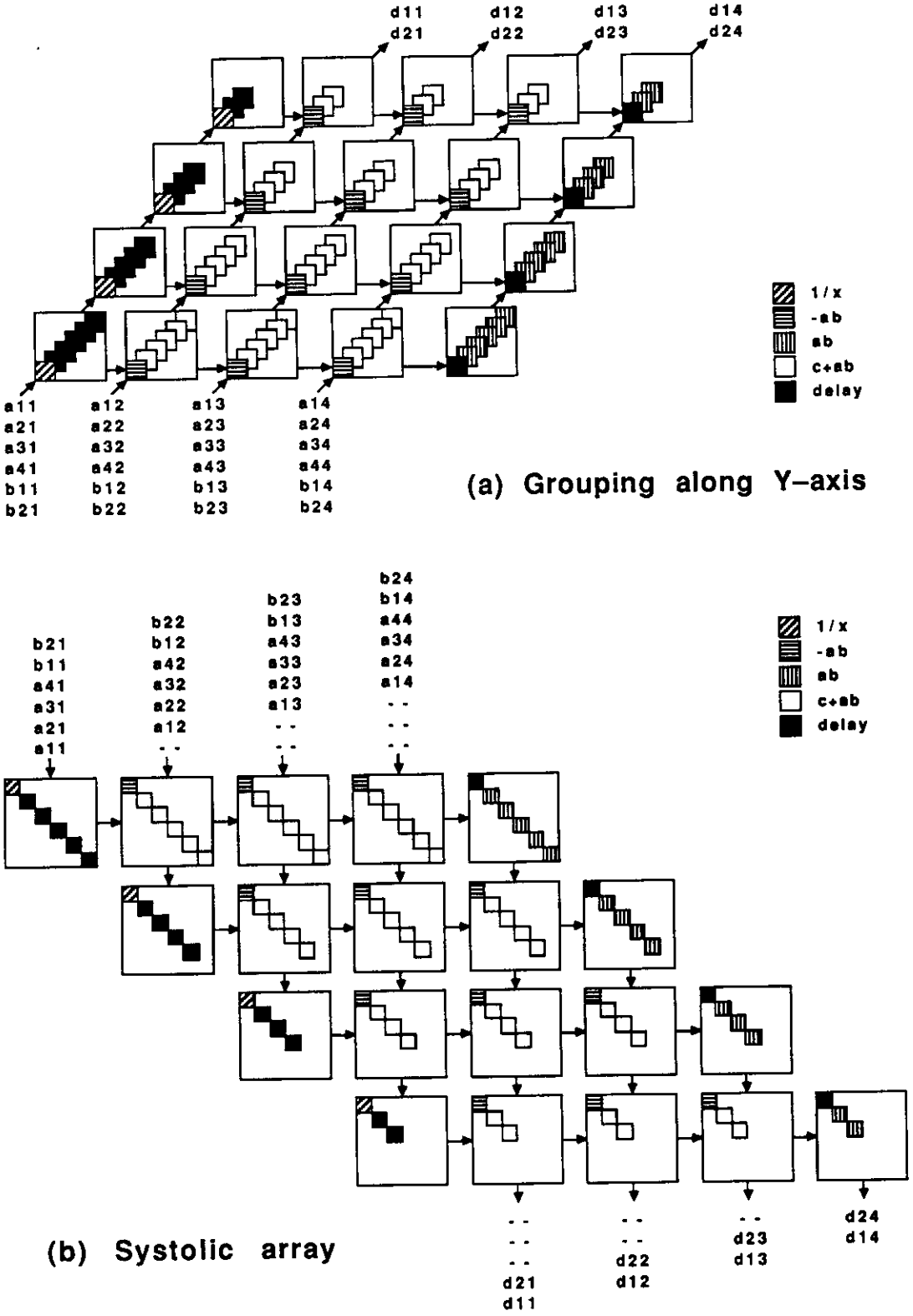

Figure 4: Grouping nodes along Y-axis

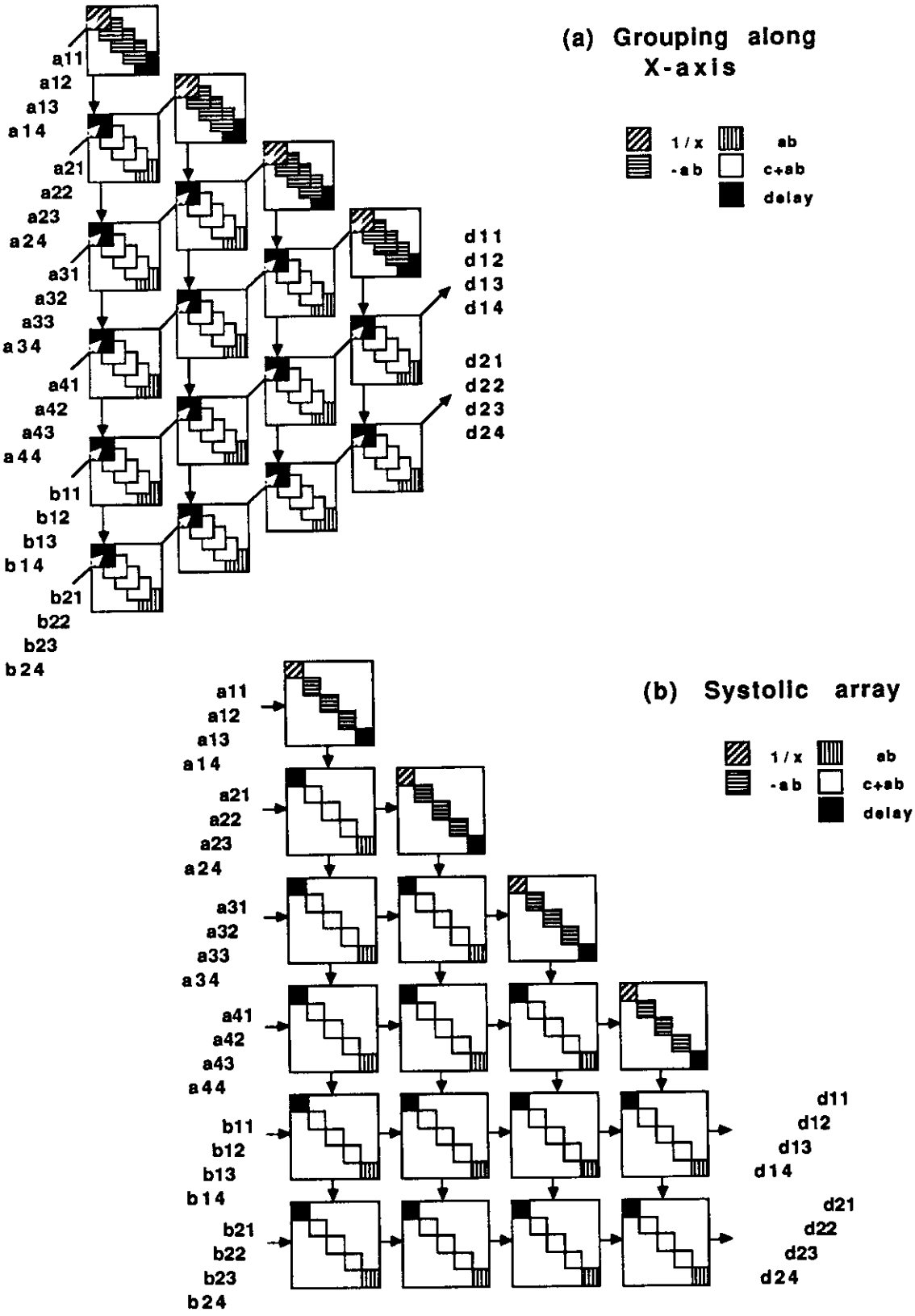

Figure 5: Grouping nodes along X-axis

Table 1: Performance measures of systolic arrays to compute BA^{-1}

	Y–axis (Comon – Robert)	X–axis
Computation time	$4n + p - 2$	$4n + p - 2$
Throughput	$(n + p)^{-1}$	$(n + 1)^{-1}$
Number of cells	$n(n + 1)$	$\frac{1}{2}n(n + 1) + pn$
I/O ports	$2n$	$n + 2p$
Utilization	$\frac{n(n+2p+1)}{2(n+1)(n+p)}$	$\frac{n^2}{n(n+1)} \rightarrow 1$
Cells complexity	3 types $1/x, [-ab, c + ab], ab$	2 types $[1/x, -ab], [c + ab, ab]$

3 Conclusions

We have applied a graph–based method for the design of systolic arrays to an algorithm that computes BA^{-1}. We have systematically derived two systolic structures for such algorithm, among them one previously proposed by Comon and Robert. We concluded that the array shown in Figure 5 has better performance than Comon and Robert's scheme.

This exercise has shown that the application of a graph–based design method for systolic arrays is powerful, producing results that are new and more efficient than others devised in ad–hoc manners. Moreover, such arrays are systematically derived giving consideration to a set of performance measures.

References

[1] P. Comon and Y. Robert, "A systolic array for computing BA^{-1}," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-35, pp. 717–723, June 1987.

[2] H. Kung, "Why systolic architectures?," *IEEE Computer*, vol. 15, pp. 37–46, Jan. 1982.

[3] H. Ahmed, J. Delosme, and M. Morf, "Highly concurrent computing structures for matrix arithmetic and signal processing," *IEEE Computer*, vol. 15, pp. 65–82, Jan. 1982.

[4] J. Speiser and H. Whitehouse, "Parallel processing algorithms and architectures for real–time signal processing," in *SPIE Real–Time Signal Processing IV*, pp. 2–9, 1981.

[5] J. Moreno and T. Lang, "Design of special–purpose arrays for matrix computations: preliminary results," in *SPIE Real–Time Signal Processing X*, pp. 53–65, 1987.
[6] S. Kung, *VLSI Array Processors.* Prentice Hall, 1988.
[7] J. Moreno and T. Lang, "Graph–based partitioning of matrix algorithms for systolic arrays: application to transitive closure," in *International Conference on Parallel Processing*, 1988.