# ROBUST SYSTOLIC ALGORITHMS FOR RELATIONAL DATABASE OPERATIONS

P. J. Varman and I. V. Ramakrishnan

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-220 March 1983

# ROBUST SYSTOLIC ALGORITHMS FOR RELATIONAL DATABASE OPERATIONS*

P. J. Varman
I. V. Ramakrishnan

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

## ABSTRACT

Systolic algorithms for relational database operations that are robust in the face of production flaws are presented. These algorithms execute on an underlying host network which is organized as a mesh array of simple processors. One processor in the network serves as the I/O port, through which all external communication occurs. Each processor consists of a comparator and shift registers. By appropriate interconnection of the shift registers, the processors of the host network are configured so that all non-faulty processors accessible from the I/O port can be utilized. Irrespective of the structure of the fault-free portion of the network obtained due to the random fault patterns, the behaviour at the I/O port is unchanged. The I/O bandwidth is independent of the problem size.

*This research was supported in part by the National Science Foundation under Grants MCS-8104017 and ONR Contract N00014-80-k-0987

# 1 INTRODUCTION

Advances in integrated-circuit technology have stimulated research on designing algorithms for solving specific computational problems directly on silicon. In [4, 5] systolic architectures were proposed as an attractive approach to solving compute-bound problems in a cost-effective manner using VLSI technology. A systolic architecture consists of a regular array of simple, identical processors which operate in synchronization to perform a single algorithm. The processors can be arranged in many forms, for instance a linear array, a rectangular mesh, a hexagonal mesh etc. The array is interfaced with the system bus of a host computer that drives the array as a peripheral.

Recently, Kung and Lehman [6] have proposed algorithms for relational database operations using a mesh array of systolic processors. However, the failure of a single link or processor in the mesh array, caused by production faults in the manufacturing process, would cause all these algorithms to fail. Since the probability of a fatal production flaw increases exponentially with the size of the circuit on the chip [9], building a sufficiently large mesh array as a single device for these algorithms becomes infeasible.

Several approaches to the problem of dealing with faults in VLSI arrays have been previously considered. An attempt to extract a fault-free mesh from a larger mesh with randomly distributed faults was considered in [8] and shown to result in the wastage of a large number of non-faulty processors. In [1, 2, 3, 8] various solutions for growing a linear array in a mesh with faults were presented. All of these solutions make relatively inefficient utilization of the fault-free area on the silicon. In this paper we present robust algorithms, based on the approach in [11], for implementing the various operations required in a relational database system. Robustness is achieved by these algorithms due to their ability to execute efficiently on any connected set of non-faulty processors in a faulty mesh array. No complicated addressing schemes are needed and the algorithm does not change with the topology of the fault-free component.

The remainder of the paper is organized as follows. In section 2, the network model on which the algorithms operate is described. Section 3 contains a description of the algorithms along with examples. In section 4 we consider some performance and implementation aspects of the algorithms. Proofs of correctness of the various algorithms are presented in the Appendix.

# 2 Network Model

The underlying host network is a set of identical modules arranged in the form of a two-dimensional mesh as shown in Figure 1(a). Each module consists of a *processor* and three *tie-points* each of which is the intersection point of a horizontal and vertical wire as shown. One module serves as the I/O port and all communication with the host computer driving the machine is done via this module.

During the manufacturing process some subset of the modules and communication links in the mesh will be faulty. We assume the existence of a testing mechanism that identifies all such faulty elements. The remaining modules in the host network that are accessible by some path from the I/O port are the ones that can be potentially utilized in performing the algorithm. Following the testing phase, these fault-free modules in the mesh are configured as in Figure 1(b) to obtain the computational structure on which our algorithms operate. This structure can be visualized as a one-dimensional pipeline 'wrapped' around the periphery of an arbitrary spanning tree that connects the non-faulty modules in the mesh. Algorithms for efficient configuration of such pipelines may be found in [11]. Communication links that are not part of this configured structure may be fused after the testing phase [7].

Each inter-module link in the mesh passes through a clocked shift-register that delays data passing through the link by one cycle. Note that the pipeline obtained by the configuration step discussed above differs from the 'standard' systolic array in that data passing through the pipeline encounters a *variable* delay (depending on the structure of the pipeline) in passing between logically adjacent processors. For instance $P_1$ and $P_2$ in Figure 1(b) are separated by 1 clock delay while $P_3$ and $P_4$ are separated by 4 clock delays. Since the pipeline structure depends on the fault distribution and the configuration algorithm, the exact delays between processors is *not known a-priori*. However, the algorithms described here have the property that irrespective of the pipeline configuration obtained, the behavior of the machine *as observed by the host* is always the same, although the internal workings of the machine may differ.

We now describe the individual processors in the network and their interconnection for the database operation algorithms. A conceptual model of a processor $P_i$ is shown in Figure 2(a). PE is a processing element that performs the same computation in every cycle using the elements at its *input-ports*, $I^i_A$ $I^i_B$ and $I^i_C$ and places the results at the corresponding *output-ports* $O^i_A$ $O^i_B$ and $O^i_C$ The computation performed by a PE in every cycle depends on the type of the relational database operation. $\underline{B}_i$ and $\underline{C}_i[1..k]$ are buffers (shift-registers) of size '1' and 'k' respectively. The value of 'k' depends on the problem and is explained further in Section 3. The input to the buffer $\underline{B}_i$ is the output of the buffer $\underline{b}_j$ in the inter-module link between $P_i$ and its immediate predecessor $P_j$ in the configured pipeline. The output of $\underline{B}_i$ is the input port $I^i_B$ of $P_i$, and the output port $O^i_B$ is connected to the buffer $\underline{b}_k$ in the inter-module link between $P_i$ and its immediate successor $P_k$ in the configured pipeline. Analogous connections are made for $\underline{C}_i[1..k]$, $I^i_C$ and $O^i_C$. $I^i_A$ is directly connected to the buffer $\underline{a}_j$ in the preceeding inter-module link and $O^i_A$ to the buffer $\underline{a}_k$ in the succeeding inter-module link. Figure 2(b) is a schematic of the machine corresponding to the pipeline configuration of Figure 1(b).

The operation of the machine is as follows. In every clock cycle each element in a buffer moves forward to the next buffer in its path. Also in every clock cycle elements that are at the input ports of PEs are transformed and their new values clocked into the buffers connected to the output ports. This transformation depends on the type of database operation.

## 3 Algorithms

Herein we describe the algorithms for the various relational database operations. We assume some familiarity with the fundamentals of relational database theory (see, for example [10]).

A relation is a set of tuples. Each tuple consists of an ordered sequence of elements. It is these elements that are fed into the systolic array. The tuples in a relation, however, are not necessarily ordered in any particular fashion.

We will be using notations similar to that used in [6]. Relations are denoted by capital letters: A, B, C. Tuples that are members of these relations are denoted by subscripted lower-case letters. The $i^{th}$ tuple of A is denoted by $a_i$, or by $a_i \in A$, to indicate membership. In turn, elements in tuples are double subscripted: $a_{ik}$ is the $k^{th}$ element of $a_i$ and the whole tuple can be exhibited as $a_i = <a_{i1}, a_{i2}, .., a_{im}>$. Let [C] represent a Boolean matrix that contains the result of logical operations. The $(i,j)^{th}$ entry of [C], $c_{ij}$, is used to denote the result of a comparison between the $i^{th}$ and the $j^{th}$ tuples of the relations A and B respectively. $c_{ij}^k$ denotes the cumulative result of comparing k elements of the $i^{th}$ and $j^{th}$ tuple. $c_{ij}^0$ and $c_{ij}^{final}$ denote specific instances (the first and last) of $c_{ij}^k$. (We will use $c_{ij}$ to refer to $c_{ij}^k$ for any k when no confusion thereby occurs). Finally the notation $x_i$ is used to designate the result of some logical operation on all of the members of the $i^{th}$ row of [C], for example the OR or AND of $c_{ij}$ for all j.

### 3.1 Comparison Algorithm

Let A and B denote two relations with q attributes each and having cardinalities of p and r respectively, where $p \geq r$. Then the result of a COMPARISON operation on A and B which we denote as A * B is a pxr Boolean matrix [C] where $c_{ij}$ = True iff $\forall k$ such that $1 \leq k \leq q$, $a_{ik} = b_{jk}$.

The algorithm for the comparison operation follows. WC is a wild-card symbol that matches any element in the relations. The basic unit of time is a clock cycle. Time instants are denoted by 't', and the algorithm begins at t = 0.

The number of processors used by the algorithm is $N = p+q+r-2$. The length of the 1-bit wide buffer $\underline{C}[1..k]$ is $p+1$. The I/O port with the external host computer will be denoted as Port-A, Port-B and Port-C (for input) and Output-Port-A, Output-Port-B and Output-Port-C (for output) for the elements of A, B and [C] respectively.

Algorithm 1

1. Initialize all buffers $\underline{C}_k[1..p+1]$ ,k=1..,N and $\underline{c}_k$, k=1,..,2N to False.
2. $\forall$i and $\forall$j such that $1 \leq i \leq p$, $1 \leq j \leq r$ do:
   Pump $c_{ij}$ (value True) into Port-C at $t = (p+1)(j-1) + p(p-i)$.
   (At all other time instants pump False into Port-C).
3. $\forall$i and $\forall$j such that all $1 \leq i \leq p$, $1 \leq j \leq q$ do:
   Pump $a_{ij}$ into Port-A at $t = (p+1)r + p(p-1) + (p+1)(j-1) + (i-1)$.
4. $\forall$i and $\forall$j such that $1 \leq i \leq r$, $1 \leq j \leq q$ do:
   Pump $b_{ij}$ into Port-B at $t = p(p+r-1) + p(j-1) + (i-1)$.
5. Pump WC into Port-A for all $0 \leq t < r(p+1)+p(p-1)$ (which is the time when the first value of the relation A is pumped into the port) and for all $t > (p+1)(q+r-2)$ (which is the time when the last value of the relation A is pumped into the port).
6. $\forall$i and $\forall$j such that $1 \leq i \leq p$, $1 \leq j \leq r$ do:
   At $t = (p+1)(j-1) + p(p-i) + (p+q+r-2)(p+3)$, extract $c_{ij}^{final}$ from Output-Port-C.

At every cycle, the PE in each processor $P_k$ performs the following actions.

$O_A^k = I_A^k$

$O_B^k = I_B^k$

$O_C^k = I_C^k \wedge (I_A^k = I_B^k)$ where (X=Y) is a Boolean value equal to True iff X=Y.

An example for the Comparison Algorithm described above is presented below.

Example 1(a)

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \\ a_{41} & a_{42} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \qquad C = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

p=4, q=2, r=3.

The number of processors used in the computation is $p+q+r-2 = 7$. The size of the buffer $\underline{C}[1..p+1]$ is 5. We shall illustrate the operation of the algorithm on the machine shown in Figure 2(b), but the algorithm would be unchanged for any other seven processor machine. Note that although the actual time steps at which values are computed in various processors would change depending on the machine configured, the behavior at the I/O port (i.e the times at which elements are fed into and removed from the array) would be unchanged.

Tables 1.A, 1.B and 1.C below, specify the behavior at the I/O port. The time-instants at which $a_{ij}$, $b_{ij}$ and $c_{ij}$ values are fed into and extracted from the array are obtained from Algorithm 1, and displayed in Tables 1.A, 1.B and 1.C respectively. The upper number in Table 1.C represents the input and the lower number the output time for the corresponding $c_{ij}$.

| i \ j | 1 | 2 |
|---|---|---|
| 1 | 27 | 32 |
| 2 | 28 | 33 |
| 3 | 29 | 34 |
| 4 | 30 | 35 |

TABLE 1.A

Pump WC into Port-A for $0 \leq t < 27$ and $t > 35$.

| i \ j | 1 | 2 |
|---|---|---|
| 1 | 24 | 28 |
| 2 | 25 | 29 |
| 3 | 26 | 30 |

TABLE 1.B

| i \ j | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 12<br>61 | 17<br>66 | 22<br>71 |
| 2 | 8<br>57 | 13<br>62 | 18<br>67 |
| 3 | 4<br>53 | 9<br>58 | 14<br>63 |
| 4 | 0<br>49 | 5<br>54 | 10<br>59 |

TABLE 1.C

We will trace the history of a particular element (say $c_{41}$) from the time it enters the array till it leaves. Snapshots of the computation are shown in Table 2. The explanations for the various columns is as follows. Column 1 (T) is the time instant of interest. Column 2 (Present-at) is the location of $c_{41}$ at that time and Column 3 (Value) is the value of the variable $c_{41}$ at that time. Column 4 (i) gives the index of the processor at whose input port $c_{41}$ is present at that time instant, and Columns 5 and 6 ($I^i_A$ and $I^i_B$) indicate the elements present at the $I_A$ and $I_B$ ports of processor $P_i$ at that time. Finally, Columns 7 and 8 ($T_a$, $T_b$) are the time instants at which the elements which are at $I^i_A$ and $I^i_B$ at that instant, entered the array.

TABLE 2: Snapshots of the Comparison Algorithm

| T | Present-at | Value | i | $I_A^i$ | $I_B^i$ | $T_a$ | $T_b$ | Comments |
|---|---|---|---|---|---|---|---|---|
| 0 | $\underline{c}_1$ | TRUE | | | | | | |
| 5 | $I_C^1$ | TRUE | 1 | WC | | 5 | | |
| 6 | $\underline{c}_2$ | TRUE | | | | | | |
| 11 | $I_C^2$ | TRUE | 2 | WC | | 10 | | |
| 12 | $\underline{c}_3$ | TRUE | | | | | | |
| 17 | $I_C^3$ | TRUE | 3 | WC | | 15 | | |
| 18 | $\underline{c}_4$ | TRUE | | | | | | |
| 19 | $\underline{c}_5$ | TRUE | | | | | | |
| 20 | $\underline{c}_6$ | TRUE | | | | | | |
| 21 | $\underline{c}_7$ | TRUE | | | | | | |
| 26 | $I_C^4$ | TRUE | 4 | WC | | 20 | | |
| 27 | $\underline{c}_8$ | TRUE | | | | | | |
| 28 | $\underline{c}_9$ | TRUE | | | | | | |
| 33 | $I_C^5$ | TRUE | 5 | WC | | 25 | | |
| 34 | $\underline{c}_{10}$ | TRUE | | | | | | |
| 39 | $I_C^6$ | TRUE | 6 | $a_{41}$ | $b_{11}$ | 30 | 24 | |
| 40 | $\underline{c}_{11}$ | $c_{41}^1$ | | | | | | $c_{41}^1 = (a_{41} = b_{11})$ |
| 45 | $I_C^7$ | $c_{41}^1$ | 7 | $a_{42}$ | $b_{12}$ | 35 | 28 | |
| 46 | $\underline{c}_{12}$ | $c_{41}^2$ | | | | | | $c_{41}^2 = c_{41}^1$ |
| 47 | $\underline{c}_{13}$ | $c_{41}^2$ | | | | | | $\wedge (a_{41} = b_{11})$ |
| 48 | $\underline{c}_{14}$ | $c_{41}^2$ | | | | | | |
| 49 | | | | | | | | Remove from I/O Port. |

### 3.2 Intersection

Let A and B be two relations having q attributes each and cardinalities of p and r respectively. The result of INTERSECTING A and B which we denote as $A \cap B$ is a $p \times 1$ Boolean vector X such that $\forall i$ such that $1 \leq i \leq p$, $x_i =$ True iff $a_i = b_k$ for some k such that $1 \leq k \leq r$.

Using the notation developed for Comparison, $x_i = \bigvee_{i=1}^{q} c_{ij}^{final}$.

For clarity in exposition we will assume that each processor is augmented with additional data paths to carry the values of the vector X. Specifically, we expand the processor model shown in Figure 2(a) to contain additional input and output ports $I_X^i$ and $O_X^i$ respectively and allow a buffer $\underline{x}_i$ identical to the buffers $\underline{a}_i$ used earlier. Furthermore, we assume that there are ports Port-X and Output-Port-X at the I/O port, for input and output of the X-vector values. In practice the values of vector X can be appended as a 1-bit field to the $a_{ij}$ values of Relation A and use the same data paths. However, for notational clarity we shall be assuming an independent set of data paths as explained above.

The computation performed on the x values in the PE of processor $P_s$ is given by

$$O_X^s = I_X^s \vee [I_C^s \wedge (I_A^s = I_B^s)]$$

Note that the expression in the square brackets is the $c_{ij}$ value computed by the comparison array at $P_s$.

Algorithm 2

(Steps 1-6 of Algorithm Comparison).

7. $\forall i$ such that $1 \leq i \leq p$ do:
   Pump $x_i^0$ (value False) into Port-X at $t = (p+1)(p+q+r-2) - (p-i)$
8. Extract $x_i^{final}$ from the Output Port-X at $t = (p+3)(p+q+r-2) - (p-i)$

An example of the operation of the algorithm for the relations A and B of Example 1(a) is presented below. The machine on which it is executed is assumed to be that of Figure 2(b) as before.

The time instants at which the X values are fed into Port-X and extracted from Output-Port-X are given in Table 3. Snapshots of a trace of $x_4$ from the time it enters till the time it leaves the array are given in Table 4.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad x_i = \text{TRUE iff } (a_i = b_1) \vee (a_i = b_2) \vee (a_i = b_3)$$

Table 3: Input-Output times of x values (from Algorithm 2)

| Input Time | | Output Time | |
|---|---|---|---|
| $x_1$ | 32 | $x_1$ | 46 |
| $x_2$ | 33 | $x_2$ | 47 |
| $x_3$ | 34 | $x_3$ | 48 |
| $x_4$ | 35 | $x_4$ | 49 |

Column headings are similar to those in Table 2.Column 1 (T) is the time-instant of interest, Column 2 (Present-at) is the location of $x_4$ and Column 3 (Value) the Value of $x_4$ at that instant. Column 4 (i) is the index of the processor at whose input port $I^i_X$, $x_4$ is present, Column 5 ($I^i_C$) is the element at the corresponding input port $I^i_C$ and Column 6 ($T_c$) gives the time instant at which the element now at $I^i_C$ entered the array. The correctness of the Comparison Algorithm ensures that the *values* of the elements at $I^i_A$ $I^i_B$ and $I^i_C$ are those needed for the intersection algorithm.

### 3.3 Difference

The difference of the two relations A and B can be simply found by taking the complement of the values of the X vector computed by the interection algorithm described above.

### 3.4 Duplicate Removal

Removal of duplicate tuples in a relation can be accomplished by the intersection (slightly modified) of the relation with itself.
Let $X = A \cap A$ and $[C] = A * A$.

TABLE 4: Snapshots of the Intersection Algorithm

| T | Present-at | Value | i | $I_C^i$ | $T_c$ | Comments |
|---|---|---|---|---|---|---|
| 35 | $I_X^1$ | FALSE | 1 | FALSE | 30 | |
| 36 | $I_X^2$ | FALSE | 2 | FALSE | 25 | |
| 37 | $I_X^3$ | FALSE | 3 | FALSE | 20 | |
| 38 | $\underline{x}_4$ | FALSE | | | | |
| 39 | $\underline{x}_5$ | FALSE | | | | |
| 40 | $\underline{x}_6$ | FALSE | | | | |
| 41 | $I_X^4$ | FALSE | 4 | FALSE | 15 | |
| 42 | $\underline{x}_8$ | FALSE | | | | |
| 43 | $I_X^5$ | FALSE | 5 | $c_{43}^1$ | 10 | |
| 44 | $I_X^6$ | $x_4^1$ | 6 | $c_{42}^1$ | 5 | $x_4^1 = c_{43}^2$ |
| 45 | $I_X^7$ | $x_4^2$ | 7 | $c_{41}^1$ | 0 | $x_4^2 = c_{43}^2 \vee c_{42}^2$ |
| 46 | $\underline{x}_{12}$ | $x_4^3$ | | | | $x_4^3 = c_{43}^2 \vee c_{42}^2 \vee c_{41}^2$ |
| 47 | $\underline{x}_{13}$ | $x_4^3$ | | | | |
| 48 | $\underline{x}_{14}$ | $x_4^3$ | | | | |
| 49 | | | | | | Remove from I/O Port. |

Then $x_i = (\bigvee_{i \neq k} c_{ik})(\vee\, c_{ii})$

If $c_{ii}$ is ensured to be False, then $x_i$=True iff $a_i$ is a duplicate tuple. Also, to detect the duplicate tuple unambigously, we must ensure that only one of the duplicates is removed. That is only one of $x_i$ and $x_j$ is set True if $a_i = a_j$. We do this by ensuring that $c_{ij}$ is False for $i \geq j$.

Since a False value of $c_{ij}$ can be ensured by pumping an initial value of False for $c_{ij}^0$ into the array, Step 2 of the Algorithm is modified as shown below for Duplicate Removal.

Algorithm 3

Replace Step 2 of Algorithm 1 as follows.

$\forall i$ and $\forall j$ such that $i < j \leq r$ and $1 \leq i \leq p$ do:

Pump $c_{ij}^0$ (value True) into Port-C at $t=(p+1)(j-1) + p(p-i)$.

(At all other time instants pump False).

The components of Vector X that are set to True on completion of the algorithm correspond to the duplicate tuples in the relation.

### 3.5 Union

The Union of two relations is found by merging the two relations and then removing duplicates.

### 3.6 Projection

Achieved by projecting over the desired columns followed by duplicate removal.

### 3.7 Join

Achieved by intersection of the appropriate columns of the two relations.

## 4 Discussion

We have presented systolic algorithms for relational database operations such as intersection, remove-duplicates, union, difference and joins. A unique feature of these algorithms is their robustness which permit fault tolerant realizations in VLSI technology. It is therefore possible to concieve of a large wafer-sized array of these processors in which the non-faulty ones are efficiently utilized in performing the computation. Such a device may be interfaced to the system bus of existing general purpose computers to improve the overall system performance.

In wafer-scale integration of array processors each processor in the array can occupy as large an area as a present-day chip. Using present-day NMOS technology an estimate of 1000 bit shift register per chip was provided in [6]. Hence the arrays used by our algorithms can handle relations having $10^3$ tuples.

Lastly we provide some time complexity measures for our algorithms. In particular we examine comparison of tuples in two relations having p tuples each and q attributes per tuple. The time complexity (which is the number of cycles between the time at which the first element in [C] is pumped into the array and the time at which the last element from [C] emerges out of the array) is seen from step 2 and step 6 of Algorithm 1 in section 3.1 to be $O(p^2)$. Clearly this is a speedup of O(q) over a sequential algorithm. The time complexity for the other algorithms are the same as the algorithms are essentially similar.

## REFERENCES

[1] R.C. Aubusson and I. Catt.
Wafer-Scale Integration - a Fault-Tolerant Procedure.
*IEEE J. Solid-State Circuits* SC-13(3), June, 1978.

[2] D.S. Fussell and P.J. Varman.
Fault-Tolerant Wafer-Scale Architectures for VLSI.
In *Proc. 9th Annual Symposium on Computer Architecture*, pages 190-198. , 1982.

[3] I. Koren.
A Reconfigurable and Fault-Tolerant VLSI Multiprocessor Array.
In *Proc. 8th Annual Symposium on Computer Architecture*. , 1981.

[4] H.T. Kung.
Why Systolic Architectures.
*IEEE Computer* 15(1):37-46, January, 1982.

[5] H.T. Kung and C.E. Leiserson.
Systolic Arrays (for VLSI).
In *Sparse Matrix Proc.*, pages 256-282. SIAM, 1978.

[6] H.T. Kung and P.L. Lehman.
Systolic (VLSI) Arrays for Relational Database Operations.
In *Proc. SIGMOD*, pages 105-116. , 1980.

[7] T. E. Mangir and A. Avizienis.
Fault-Tolerant Design for VLSI: Effect on Interconnect Requirement on Yield Improvement of VLSI Designs.
*IEEE Trans. on Computers* C-31(7):609-616, July, 1982.

[8] F.B. Manning.
An Approach to Highly Integrated, Computer-Maintained Cellular Arrays.
*IEEE Trans. on Computers* C-26(6):536-552, June, 1977.

[9] C.A. Mead and L.A. Conway.
*Introduction to VLSI Systems*.
Addison-Wesley, Reading, Massachusetts, 1980.

[10] Jeffrey D. Ullman.
*Principles of Database Systems*.
Computer Science Press, Potomac, Maryland, 1980, .

[11] P. J. Varman.
*Fault Tolerant VLSI Computation (in preparation)*.
PhD thesis, Departmant of Electrical Engineering, University of Texas at Austin, 1983.

## Appendix

We now develop the correctness proof for the algorithm described earlier.

Definition A Processor $P_k$ has distance 'd' if there are d inter-module links between $P_1$ and $P_k$ in the path configured.

Proof of the Comparison Algorithm

**Lemma 4-1:** All the $c_{ij}$ ($a_{ij}$,$b_{ij}$) values enter Port-C (Port-A, Port-B) at distinct times.

**Proof:** We shall prove the result only for the $c_{ij}$ values, the rest of the proofs being similar.

Suppose $c_{ij}$, $c_{mn}$ enter the port simultaneously. From Step 2 of the algorithm it follows that

$(p+1)(j-1) + p(p-i) = (p+1)(n-1) + p(p-m)$

Hence, $(p+1)(j-n) = p(i-m)$

$(i-m) = (j-n) + (j-n)/p.$

Since $|j-n| < r \leq p$, the above integer equation can only be satisfied if $j-n = 0$.

Thus $j=n$ and correspondingly $i = m$.

□

**Lemma 4-2:** Let $t_a$, $t_b$ and $t_c$ be the times when $a_{ij}$, $b_{ij}$ and $c_{ij}$ are pumped into their respective Input-Ports. Then the times $t_A$, $t_B$ and $t_C$ at which they reach the corresponding input-ports of processor $P_s$ (where $P_s$ has distance 'd') is given by:

1. $t_A = t_a + d.$
2. $t_B = t_b + s + d.$
3. $t_C = t_c + (p+1)s + d.$

**Proof:** We shall prove only (3), the rest of the proofs being similar.

From the interconnection it can be seen that $c_{ij}$ will traverse the buffers $\underline{C}_1[1..p+1]$, $\underline{C}_2[1..p+1]$, ......, $\underline{C}_s[1..p+1]$, incurring a total delay of $(p+1)s$. In addition, exactly 'd' of the $\underline{c}$ buffers will also be traversed, incurring an extra delay of 'd'.

Summing the delays: $t_C = t_c + (p+1)s + d.$

□

**Lemma 4-3:** The initial value of $c_{ij}$ (value True) remains unchanged till it reaches $I^s_C$, where $s = r + i - j$.

**Proof:** Let $1 \leq k \leq r+i-j-1$, and $P_k$ have distance 'd'. Let $t_c$, $t_C$ be the times when $c_{ij}$ is pumped into Port-C and at which it reaches $I_C^k$ respectively.

From Lemma 4.2, $t_C = t_c + (p+1)k + d$.

Suppose the element at $I_A^k$ at $t_C$ is x and let x have been pumped into Port-A at $t_x$.

Then $t_C = t_x + d$

If $t_x < r(p+1)+p(p-1)$ then x = WC, and $c_{ij}$ will be unchanged at $P_k$.

Hence we must show that:

$t_x = t_C - d < r(p+1)+p(p-1)$

Substituting for $t_c$ from Algorithm 1 (Step 2) and simplifying we need to show that

$k < r+i-j - (i-1)/(p+1)$

Since $(i-1)/(p+1) < 1$ and $k \leq r+i-j-1$, the result follows.

□

**Lemma 4-4:** The final value of $c_{ij}$ ($c_{ij}^q$) remains unchanged from $O_C^s$ (where $s = r+q+i-j-1$) till it reaches the Output-Port-C.

**Proof:** Let $r+q+i-j \leq k \leq p+q+r-2$ and let $P_k$ have distance 'd'. Let $t_C$, $t_c$, $t_x$ be as defined in Lemma 4.3.

If $t_x > (p+1)(r+p+q-2)$ then x = WC, and $c_{ij}$ will be unchanged. Thus we require to show that

$t_x = t_C - d > (p+1)(r+p+q-2)$ Substituting for $t_C$ and simplifying we get

$k > (r+q+i-j) - (i+1)/(p+1)$

Since $k \geq r+q+i-j > (r+q+i-j) - (i+1)/(p+1)$ the result follows.

□

**Lemma 4-5:** $a_{ik}$, $b_{jk}$ and $c_{ij}$ reach the input ports $I_A^s$ $I_B^s$ and $I_C^s$ of $P_s$ ($s = r+k+i-j-1$) at the same time.

**Proof:** Let $P_s$ have distance 'd', and $a_{ik}$, $b_{jk}$ and $c_{ij}$ reach the input ports of $P_s$ at $t_A$, $t_B$ and $t_C$ respectively.

From the Algorithm and Lemma 4.2

$t_A = (p+1)r + p(p-1) + (p+1)(k-1) + (i-1) + d.$

$t_B = p(r+p-1) + p(k-1) + (j-1) + s + d.$

$t_C = (p+1)(j-1) + p(p-i) + (p+1)(s) + d.$

Substituting for s, each of the above expressions reduces to

$p^2 + pk + pr + k + d + r - 2p + i - 2$.

Hence $t_A = t_B = t_C$.

□

**Lemma 4-6:** $\forall k$ such that $1 \leq k \leq q$, the value of $c_{ij}$ ($c_{ij}^k$) at $O_C^s$ where $s=r+k+i-j-1$, is $\bigwedge_{m=1}^{k}(a_{im}=b_{jm})$.

**Proof:** (By induction on k).

Base Case: k=1

By Lemma 4.3, the value of $c_{ij}$ at $I_C^u$ ($u=r+i-j$) is True. By Lemma 4.5, $a_{i1}$, $b_{j1}$ and $c_{ij}^0$ (=True) arrive at the Input-Ports of $P_u$ at the same time.

Thus, the value at $O_O^u$ which is $c_{ij}^1 = \bigwedge_{m=1}^{1}(a_{im}=b_{jm})$

A straightforward induction step will complete the proof.

□

**Theorem 4-1:** $c_{ij} = \bigwedge_{m=1}^{q} (a_{im}=b_{jm})$ when it leaves the Output Port C.

**Proof:** Follows from Lemmas 4.6 and 4.4.

□

Proof of the Intersection Algorithm

**Lemma 4-7:** $\forall k$ such that $i \leq k \leq p$, the value of $x_k$ can only change at $P_s$ where $q+k-1 \leq s \leq r+q+k-1$.

**Proof:** If $I_C^s$ has the value False then the value of $x_k$ cannot change at $P_s$ at that cycle. We will show that $I_C^s$ has a True value only if s lies in the above range.

Value True can reach $I_C^s$ only if it was pumped into Port-C at the times specified in the Algorithm (Step 2). For $x_k$ and some $c_{ij}$ $1 \leq i \leq p$, $1 \leq j \leq$ to arrive simultaneously at $P_s$, where $P_s$ has distance 'd'

$(p+1)(j-1) + p(p-i) + s(p+1) + d = (p+1)(p+q+r-2) - (p-k) + d$.

Simplifying, $s = r+q+i-j-1 + (k-i)/(p+1)$

Since $|k-i| < p$, and all other quantities are integers, k-i=0.

Therefore, s=r+q+i-j-1 and since k=i and $i \leq j \leq r$, the range of s for $x_k$ to meet any $c_{ij}$ is given by
$q+k-1 \leq s \leq r+q+k-2$.

□

**Lemma 4-8:** $x_i$ and $c_{ij}^{q-1}$ arrive simultaneously at the input-ports $I_X^s$ and $I_C^s$ of $P_s$, where s=r+q+i-j-1, $\forall$i and $\forall$j such that $1 \leq i \leq p$, $1 \leq j \leq r$.

**Proof:** $x_i$ is pumped into Port-X at the same time as $a_{iq}$ (Steps 3 and 7 of the Algorithm). Both encounter the same delays. From Lemma 4.6, $a_{iq}$ and $c_{ij}$ arrive simultaneously at $P_s$ (s=r+q+i-j-1) and from Lemma 4.6 its value is $c_{ij}^{q-1}$. Therefore the same holds for $x_i$.

□

**Theorem 4-2:** $\forall$k such that $1 \leq k \leq p$, $x_k = \bigvee_{j=1}^{r} c_{ij}$ when it leaves the Output Port X.

**Proof:** From Lemma 4.7, the value of $x_k$ does not change till it reaches $I_X^s$ where s=q+k-1. Using Lemma 4.8 and the fact that the initial value of $x_k$ is False, it can be shown by induction that its value at $O_X^s$(s=r+q+k-2) is $\bigvee_{j=1}^{r} c_{ij}^{q}$. Invoking Lemma 4.7 again this value remains unchanged till it reaches the Output-Port-X.

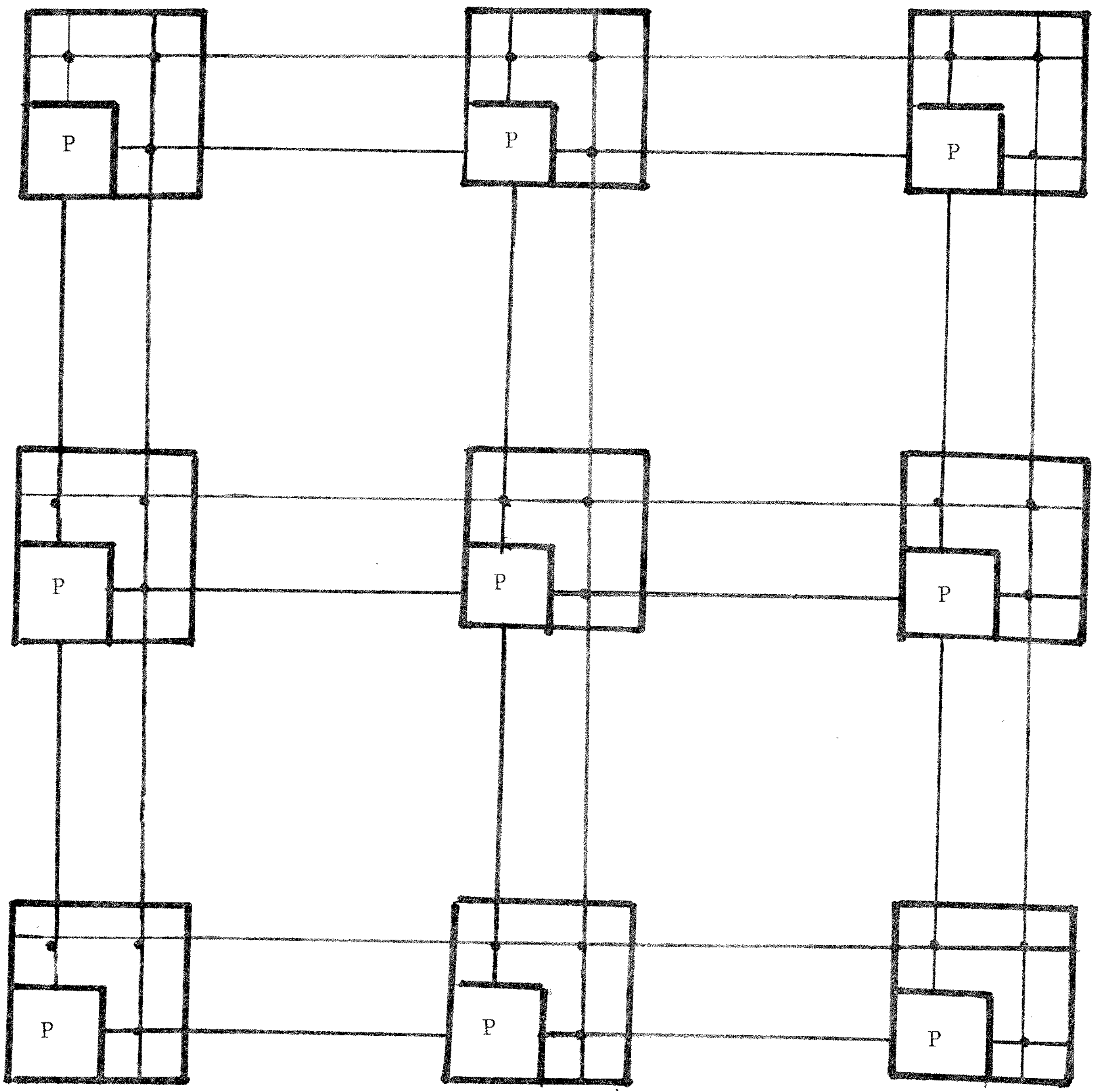
□

P: Processor in Module

—— : Link

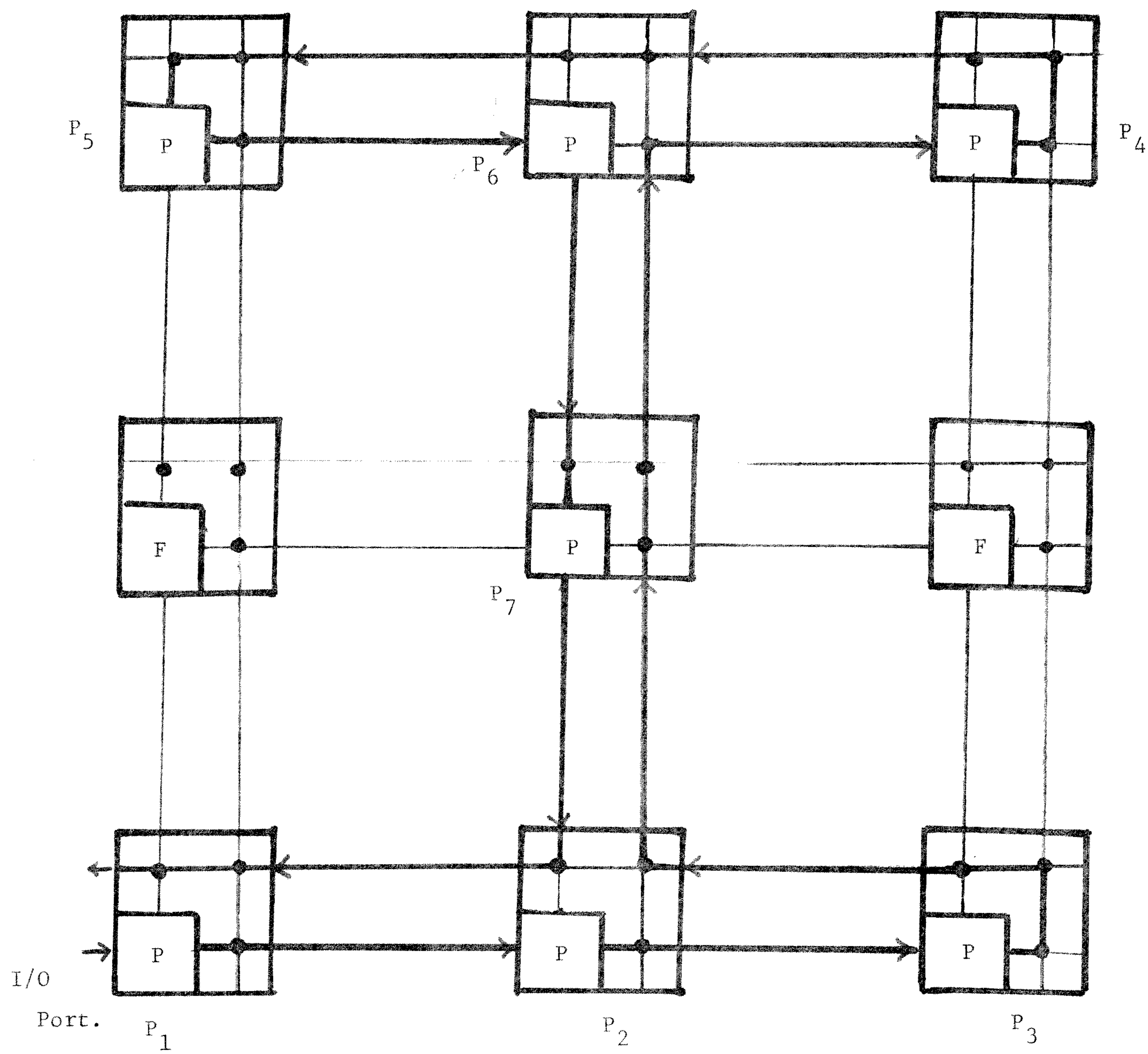
-+- : Tie-Point

Figure 1(a): Layout of modules as a two dimensional mesh



P: Processor in Module
: Tie-Point
F: Faulty processor
: Links in configured machine

Figure 1(b): Configuration of the non-faulty modules



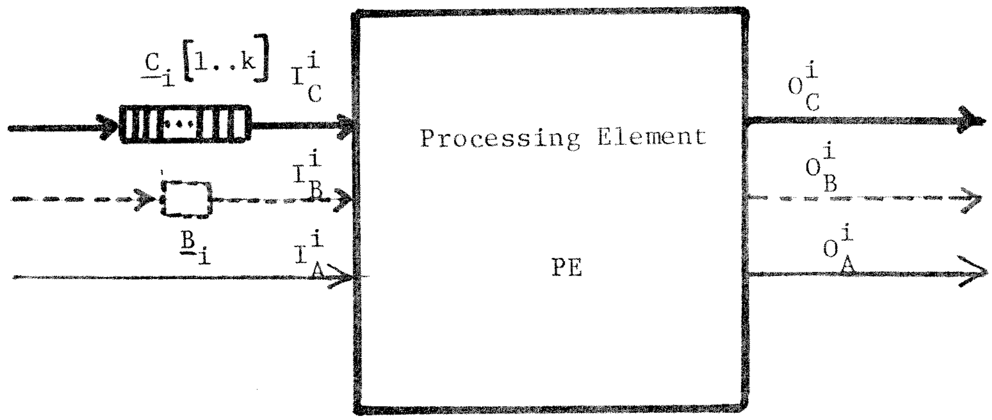



Figure 2(a): Schematic of a Processor $P_i$.

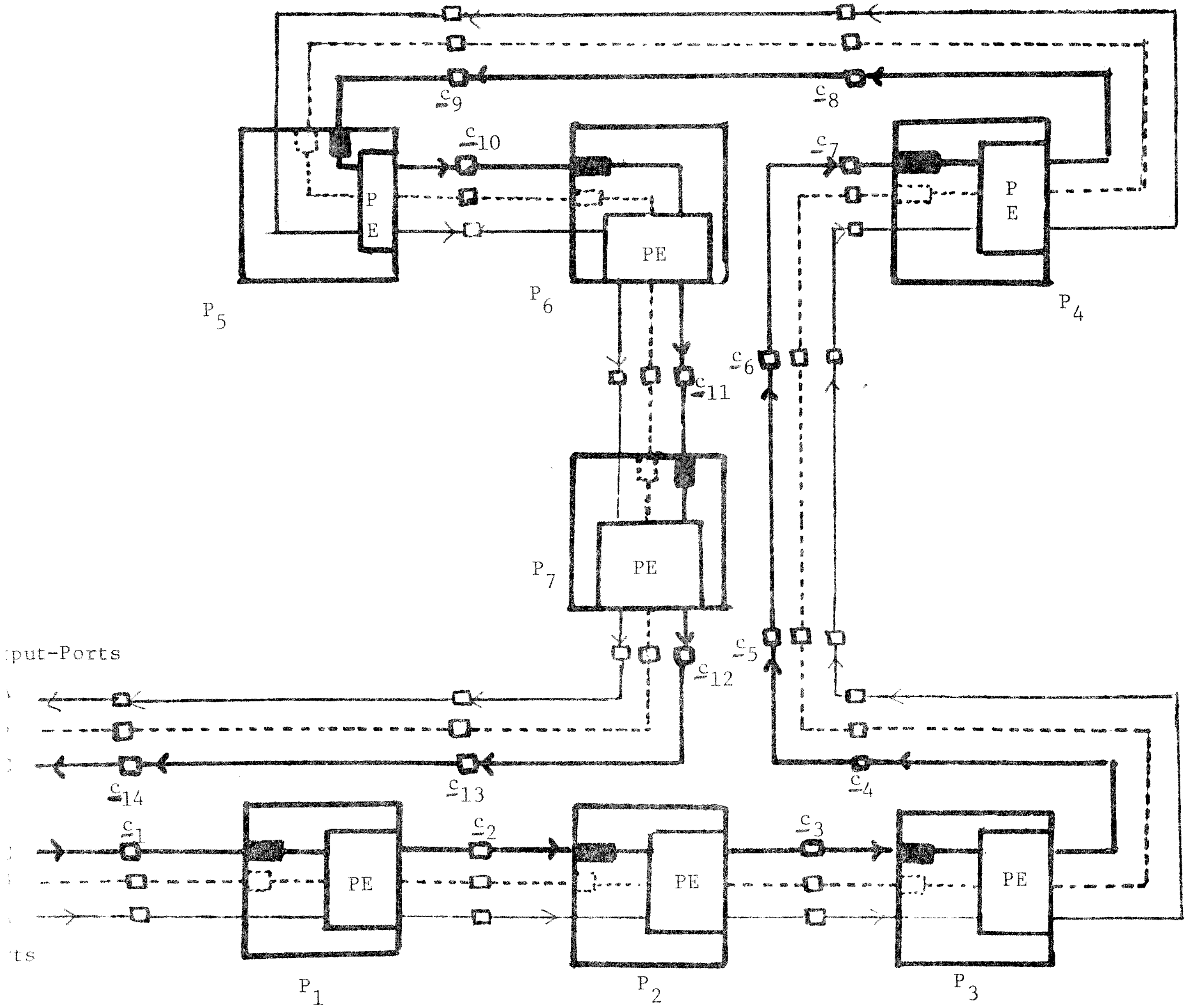



Figure 2(b): Configuration of shift registers and processing elements corresponding to Figure 1(b).