Synthesis of a new systolic architecture for the algebraic path problem

Abdelhamid Benaini, Patrice Quinton, Yves Robert, Yannick Saouter, Bernard Tourancheau

INRIA

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1) 39 63 55 11

Rapports de Recherche

N° 1094

Programme 2
Structures Nouvelles d'Ordinateurs

SYNTHESIS OF A NEW SYSTOLIC ARCHITECTURE FOR THE ALGEBRAIC PATH PROBLEM

Abdelhamid BENAINI
Patrice QUINTON
Yves ROBERT
Yannick SAOUTER
Bernard TOURANCHEAU

Septembre 1989

RR-1094

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Synthesis of a New Systolic Architecture for the Algebraic Path Problem

Synthèse d'une nouvelle architecture systolique pour le problème du chemin algébrique

Abdelhamid BENAINI, Patrice QUINTON, Yves ROBERT, Yannick SAOUTER and Bernard TOURANCHEAU*

Soumis à la revue Science of Computer Programming

Publication Interne n° 486 - Juillet 1989 - 34 Pages

Résumé : Cet article concerne la synthèse automatique d'architectures systoliques. L'exemple cible choisi est une architecture systolique bidimensionnelle à structure toroïdale pour le problème du chemin algébrique. Nous expliquons d'abord informellement comment dériver cette architecture, puis nous montrons comment elle peut être synthétisée en utilisant une méthode basée sur les équations récurrentes uniformes. La synthèse fournit une preuve de la correction de l'architecture.

Mots-clés : architectures systoliques, réseau linéaire de processeurs, synthèse automatique, problème du chemin algébrique, équations récurrentes.

Abstract This paper deals with the systematic synthesis of systolic arrays. As a target example, we design a new 2D-toroidal systolic array for the algebraic path problem. First, we informally explain how to derive this new systolic architecture, then we show how to synthesize it using a systematic methodology based upon uniform recurrence equations. Such a synthesis provides a formal proof of the correctness of the architecture.

Key-words: systolic architecture, linear array of processors, automatic synthesis, algebraic path problem, recurrence equations

*A. Benaini, Y. Robert, and B. Tourancheau appartiennent au Laboratoire LIP-IMAG, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France. P. Quinton and Y. Saouter appartiennent à IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France.

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

1 Introduction

Since the seminal work of Kung and Leiserson ([18]), interest for systolic algorithms and architectures has never decreased, for the following reasons:

- systolic algorithms exist for a much larger class of algorithms than was initially expected, especially in the strategic domains of signal processing, numerical analysis, image processing, etc.
- with the advance of VLSI, it becomes more and more realistic to design directly into silicon such algorithms
- systolic algorithms represent also a basic paradigm for programming massively parallel architectures which embed easily the local communication model underlying systolic arrays
- last but not least, the regularity of systolic array make it possible to derive them from very high level specifications such as equations.

In this paper, we are mainly interested in the formal derivation of systolic algorithms. These last years, several methods for synthesizing systolic arrays appeared in the literature (see [27][8][4],[3], [22][35][29] among others). Most of these methods rely upon the early work of Karp et al. [14] on uniform recurrence equations and of Lamport [21] on the parallelization of do loops. Basically, it consists in describing the algorithms as a set of indexed calculations, each calculation being attached to a point of the space. By ordering the calculations in such a way that the order imposed by their dependences is preserved, and by projecting the domain of the equations along a direction of the space, one obtain a systolic array.

The approach we shall use here is based on the work presented in [29, 9, 10, 30, 31]. As a target example, we consider the Algebraic Path Problem (APP for short). It is defined as follows [11, 37]: given a weighted graph $G = (V, E, w)$ where V is a finite vertex set, E an arc set, w a function $E \rightarrow H$, $(H, \oplus, \otimes)$ a semiring with zero 0 and unity 1, find for all pairs of vertices (i, j) the quantities $d_{ij} = \oplus\{w(p); p \in M_{ij}\}$, where M_{ij} denotes the set of all paths from i to j. With the weighted graph (V, E, w) we associate the n by n weight matrix $A = (a_{ij})$, where $a_{ij} = w(i, j)$ if $(i, j) \in E$, and $a_{ij} = 0$ otherwise. We denote $M_{ij}^{(k)}$ the set of all paths from i to j which contain only vertices x with $1 \leq x \leq k$ as intermediate vertices. In practice, $a_{ij} = \oplus\{w(p); p \in M_{ij}^{(k)}\}$ is equal to the successive values of a_{ij} which we want to compute, starting from the initial value $a_{ij}^{(0)} = a_{ij}$ up to $a_{ij}^{(n)} = d_{ij}$. The solution to the APP problem can be obtained by a direct generalization of the Gauss-Jordan diagonalization algorithm to

compute the inverse of a real matrix [11]:

for $k = 1$ **to** n **do**
begin { phase k }
$a_{kk}^{(k)} := (a_{kk}^{(k-1)})^*$;
for $i = 1$ **to** n, $i \neq k$, **do** $a_{ik}^{(k)} := a_{ik}^{(k-1)} \otimes a_{kk}^{(k)}$
for $j = 1$ **to** n, $j \neq k$, **do**
begin
for $i = 1$ **to** n, $i \neq k$, **do** $a_{ij}^{(k)} := a_{ij}^{(k-1)} \oplus a_{ik}^{(k)} \otimes a_{kj}^{(k-1)}$
$a_{kj}^{(k)} := a_{kk}^{(k)} \otimes a_{kj}^{(k-1)}$;
end
end

In the computational procedure we have let $c^* := \oplus\{c^i; i \geq 0\}$ for $c \in H$. Applications of the APP are obtained by specializing the operations $\oplus$ and $\otimes$ in the appropriate semirings. Let us mention four of them: (i) determination of the inverse of a real matrix; (ii) shortest distances in a weighted graph; (iii) transitive and reflexive closure of a binary relation; (iv) maximum capacity matrix [11, 36]. This short list shows that path problems are ubiquitous in computer science. Several researchers have supplied the parallelization of the APP on systolic arrays (see [7][12][15][19][20] [28][32][33][34][36] among others).

The main result of this paper is a new systolic array for the APP whose area-time performances overcome those of the literature. We describe this new array in section 3. Beforehand, in section 2, we deal with a linear array of processors and we propose a parallel algorithm whose 2D space time extension will permit to retrieve the systolic solution. Finally in section 4, we show how to synthesize the new systolic array using a systematic methodology based upon recurrence equations. Such a synthesis provides a formal proof of the correctness of the array.

2 Using a Linear Array of Processors

We retrieve the generic algorithm for the APP by splitting it into elementary tasks, whose execution ordering is directed by precedence constraints: independent tasks can be processed simultaneously [5, 16, 23]. At step k, we let T_{kk} denote the task of updating column k and T_{kj} the task of combining column j with column k:

for $k = 1$ **to** n **do** {phase k}
task T_{kk}:
$< a_{kk}^{(k)} := (a_{kk}^{(k-1)})^*$;
for $i = 1$ **to** n, $i \neq k$, **do** $a_{ik}^{(k)} := a_{ik}^{(k-1)} \otimes a_{kk}^{(k)}$; $>$
for $j = 1$ **to** n, $j \neq k$, **do**
task T_{kj}:
$<$ **for** $i = 1$ **to** n, $i \neq k$, **do** $a_{ij}^{(k)} := a_{ij}^{(k-1)} \oplus a_{ik}^{(k)} \otimes a_{kj}^{(k-1)}$;
$a_{kj}^{(k)} := a_{kk}^{(k)} \otimes a_{kj}^{(k-1)}$; $>$

The precedence constraints are the following:

- T_{kk} precedes T_{jk} for all $j \geq k+1$ (the updating of column k must be completed before it can be combined with other columns)
- T_{kj} precedes $T_{k+1,j}$ for all $j \geq k+1$ (the updating of column j at step $k+1$ can only begin when its updating from step k is terminated).

The task graph model which can be constructed directly from these precedence constraints is depicted in figure 1. Assume that a task can be processed within a time unit. We easily solve a problem of size n on a linear array of processors, as illustrated in figure 2. Column j is input to the top processor at step j. The program of the processors can be detailed as follows (all column and processor indices are taken module n):

Processor P_i
if time $= 2i-1$ **then** receive column i from P_{i-1}
for time :=$2i$ **to** $2i+n-2$ **do**
begin
receive column $j = time - i + 1$ **from** P_{i-1};
execute task T_{ij};
send column j **to** P_{i+1}
end;
if time $= 2i+n-1$ **then send** column i **to** P_{i+1}

For simplicity, we have assumed that the rightmost processor P_n also sends a message to its right neighbor (i.e. to the external world).

Processor P_i, $1 \leq i \leq n$, executes task $T_{i,i+j}$ at time $t = 2i-1+j$. Assume that n is even for the sake of simplicity: we see that processor P_i executes its last task $T_{i,i-1}$ at time $2i+n-2$, that is one step before that $P_{i+n/2}$ initiates its first computation $T_{i+n/2,i+n/2}$. As a consequence, we can fold the array around its horizontal axis and assign to processor P_i the set of tasks $\{T_{ij}; 1 \leq j \leq n\} \cup \{T_{i+n/2,j} : 1 \leq j \leq n\}$. This leads to the parallel algorithm illustrated in figure 3: we solve a problem of size n on a ring of $n/2$ processors. Processor p_i, $1 \leq i \leq n/2$, executes task $t_{i,i+j}$ at time $2i-1+j$ and task $T_{i+n/2,i+n/2+j}$ at time $2i+n-1+j$.

3 The new systolic array

3.1 Space-time extension of the ring algorithm

We move to smaller granularity from the linear array to the 2D-systolic architecture: rather than defining the tasks T_{kj} as a combination of full columns, we split such a combination into n elementary updates. Let O_{kji} be the operation performed on coefficient a_{ij} at step k.

Each processor P_k of the ring above is replaced by a ring array composed of n cells P_{kk}, $P_{k+1,k}$, ..., $P_{k-1,k}$ as shown in figure 4. These cells execute in a pipeline fashion the tasks

T_{kj} and $T_{k+n/2,j}$, $1 \leq j \leq n$, that were assigned to processor P_k. More precisely, cell P_{ik} executes successively the operations O_{kji}, $j = k, k+1, ..., k-1$ and the operations $O_{k+n/2,j,i}$, $j = k+n/2, k+n/2-1, ..., k+n/2-1$. In particular, cells P_{kk} and $P_{k+n/2,k}$ will compute the star operation O_{kkk} in task T_{kk} and $O_{k+n/2,k+n/2,k+n/2}$ in task $T_{k+n/2,k+n/2}$. Note that O_{kkk} is the first computation performed in the execution of T_{kk}. However, because of the ring topology, and in order to obtain local vertical communications between consecutive rows of the 2D array, we shift the cells so that P_{ij} is the j-th cell of row i of the array. Cells P_{ij} and $P_{i,i+n/2}$ will compute the star operation in task T_{ii} and $T_{i+n/2,i+n/2}$.

Note that Cosnard and Tchuente [6] propose other examples of space-time extensions of linear array algorithms into 2D systolic architectures. See also [1] for similar ideas.

The 2D torus of processors that we obtain using the previous extension is represented in figure 5. For a problem of size n, the array is composed of $\lceil \frac{n}{2} \rceil$ rows of n processors. Because columns of A were input to the vertical ring of processors, the coefficients of A are fed into the 2D array row by row (and in a skewed format, because of the pipeline execution of the tasks). The input format is shown in figure 5. The input control is also made explicit in figure 5. The operation of the processors is depicted in figures 6 and 7. For the sake of the exposition, we artificially particularize the action of the diagonal cells in figures 6 and 7. In fact, all processors should be viewed as identical, and there is no need of the additional diagonal connections for propagating the control. Systolic techniques for identifying the diagonal cells are well known (see Ullman [36] for the use of two boolean signals moving at full speed in one direction and at half speed in the othe for such an identification). The specialization of the diagonal cells makes it clearer when new phases of the algorithm are initialized, that is, when a new star operation is performed by some diagonal cell.

We state our main result:

Theorem 3.1 *Any instance of size n of the APP can be solved on a 2D-toroidal systolic array of* $\lceil \frac{n}{2} \rceil \times n$ *processors within* $5n - 2$ *time steps.*

We have validated the operation of the array using a program written in SISYC, which is a language for the simulation and the validation of systolic algorithms, based on the mathematical model for the specification and verification of systolic networks due to Melhem and Reinboldt [26, 25]. SISYC is an applicative language in the sense that all programs and all operators are functions of data sequences. In SISYC, we represent the data appearing on the same edge of the systolic network by a data sequence x, where $x(n)$ is equal to the data item that appeared on that edge at time n. The computation performed by each cell in the network is then modelled using operators on data sequences such as the shift, the arithmetic and the conditional operators. A SISYC compiler and environment tools developed by Benaini [2] were used to test our array on various instances of path problems, including matrix inversion.

4 Systematic derivation of the new architecture

In this section, we show how the architecture described in the previous section can be derived systematically. The method we use (see [29, 30, 31]) starts from an initial specification of

the algorithm using a system of *recurrence equations*. These equations, directly obtained form program 1 of subsection 1, are shown in subsection 4.1. Then, in subsection 4.2 and 4.3, we apply successively two translations to subdomains of the system. In subsection 4.4, we explain how to replace some of the non-uniform index functions by uniform ones (that is to say, index translations). This transformation, called uniformization, aims at "localizing" the data transfers between the points of the domains. Subsection 4.5 shows that all the indexes of the final system can be assigned an instant of time using a piecewise linear function, in such a way that the dependences between the equations be preserved. Finally, subsection 4.6 describes how the architecture can be derived by an appropriate projection of the domain.

4.1 Initial system of recurrence equations

The initial system of equations is obtained directly from the program 1:

Input Equations :

$$k = 0, 1 \leq i \leq n, 1 \leq j \leq n \quad \rightarrow \quad A(i,j,k) = a(i,j) \tag{1}$$

Computation Equations :

$$1 \leq k \leq n, i = j = k \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1)^* \tag{2}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, j = k \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1) \otimes A(k,j,k) \tag{3}$$

$$1 \leq k \leq n, 1 \leq j \leq n, j \neq k, i = k \quad \rightarrow \quad A(i,j,k) = A(i,k,k) \otimes A(i,j,k-1) \tag{4}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, 1 \leq j \leq n, j \neq k \quad \rightarrow \quad A(i,j,k) = A(i,j,k-1) \oplus A(i,k,k) \otimes A(k,j,k-1) \tag{5}$$

Output Equation:

$$1 \leq i \leq n, 1 \leq j \leq n \quad \rightarrow \quad R(i,j) = A(i,j,n) \tag{6}$$

All the equations are of the form

$$(i,j,k) \in C \rightarrow A(i,j,k) = f[...A(I(i,j,k))...]$$

where C, called the domain of the equation, is a convex polyhedron of $\mathbf{Z}^3$, and I is an affine mapping. An equation defines a finite set of equation instances, one for each point (i,j,k) of the domain of the equation.

In order to separate the equation describing proper computations and the communications of the algorithm, let us rewrite the equations by introducing new variables $P(i,j,k) = A(i,j,k-1)$, $AI(i,j,k) = A(k,j,k)$, $AJ(i,j,k) = A(i,k,k)$, and $AK(i,j,k) = A(k,j,k-1)$.

After substituting these new variables in equations (2) to (5), we obtain the new set of computation equations:

$$1 \leq k \leq n, i = j = k \quad \rightarrow \quad A(i,j,k) = P(i,j,k)^* \tag{7}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, j = k \quad \rightarrow \quad A(i,j,k) = P(i,j,k) \otimes AI(i,j,k) \tag{8}$$

$$1 \leq k \leq n, 1 \leq j \leq n, j \neq k, i = k \quad \rightarrow \quad A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) \tag{9}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k,$$

$$1 \leq j \leq n, j \neq k \rightarrow A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AK(i,j,k) \quad (10)$$
$$1 \leq k \leq n, 1 \leq i \leq n, 1 \leq j \leq n \rightarrow P(i,j,k) = A(i,j,k-1) \quad (11)$$
$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, 1 \leq j \leq n \rightarrow AI(i,j,k) = A(k,j,k) \quad (12)$$
$$1 \leq k \leq n, 1 \leq i \leq n, 1 \leq j \leq n, j \neq k \rightarrow AJ(i,j,k) = A(i,k,k) \quad (13)$$
$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, 1 \leq j \leq n, j \neq k \rightarrow AK(i,j,k) = A(k,j,k-1) \quad (14)$$

Notice that these equations are of two types: (7) to (10) express really computations, and equations (11) to (14) express only communications (we shall refer to them as *communications equations* in the following).

The domains of these equations are shown in figure 8a. In the following, given an equation numbered (x), we shall denote by $D(x)$ its domain. All the domains of the equations of the above system are enclosed in a cube of size N of the space.

4.2 Translation of the subdomain $j < k$

Our first transformation involves translating the part of the domain where $j < k$ by $(0, n, 0)$ (see figure 8b). In other words, we reindex the equation instances whose index satisfies $j < k$ by adding the vector $(0, n, 0)$. The following proposition explains precisely how this transformation must be done:

Proposition 4.1 *Consider a system of equations of the form*

$$z \in D \rightarrow A(z) = f[B_1(z), ..., B_q(z)] \quad (15)$$

or

$$z \in D \rightarrow A(z) = B(I(z)) \quad (16)$$

where D is a domain of $\mathbf{Z}^q$, and I is a mapping form $\mathbf{Z}^q$ to $\mathbf{Z}^l$, $l \leq q$. Let C be a subset of $\mathbf{Z}^q$, and θ be a vector of $\mathbf{Z}^q$. Denote $T_\theta(E)$ the translation of set E by vector θ. Assume moreover that $T_\theta(C)$ does not contain any point of D nor of $I(D)$, for all domain D and all dependence mapping I. We obtain an equivalent system of equations by replacing all equations of the form (15) by

$$z \in (D \setminus C) \rightarrow A(z) = f[B_1(z), ..., B_q(z)] \quad (17)$$
$$z \in T_\theta(D \cap C) \rightarrow A(z) = f[B_1(z), ..., B_q(z)] \quad (18)$$

and all equations of the form (16) by

$$z \in T_\theta(D \cap C \cap I^{-1}(C)) \rightarrow A(z) = B(I(z-\theta)+\theta) \quad (19)$$
$$z \in T_\theta((D \cap C) \setminus I^{-1}(C)) \rightarrow A(z) = B(I(z-\theta)) \quad (20)$$
$$z \in (D \setminus C) \cap I^{-1}(C)) \rightarrow A(z) = B(I(z)+\theta) \quad (21)$$
$$z \in D \setminus C \setminus I^{-1}(C) \rightarrow A(z) = B(I(z)). \quad (22)$$

Proof The new equations are easily obtained by considering the instances of the initial equations which are translated. In equation (15), all the arguments of the equation have the same index, and all are therefore translated or not translated. Therefore, we obtain two equations. In equation (16), since the left-hand side variable and the right-hand side argument do not have the same index, we have four cases to consider, depending on whether the index of the arguments are translated or not. ■

Let us apply such a transformation to the system, by taking C equal to the half-space $j < k$, and $\theta = (0, n, 0)$.

We notice that $D(7)$, and $D(8)$ do not intersect the half-space $j < k$. Therefore, equations (7), and (8) are not modified.

By the transformation, equations (9) and (10) give only one equation, as the index functions of the right-hand side elements is the identity. We have thus:

$$1 \leq k \leq n, k+1 \leq j \leq n+k-1, i = k \quad \rightarrow \quad A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) \tag{23}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, k+1 \leq j \leq n+k-1 \quad \rightarrow \quad A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AK(i,j,k) \tag{24}$$

The transformation of the communication equations is more involved, as the index mapping is not the identity. Consider for example equation (11) that we recall here:

$$1 \leq k \leq n, 1 \leq i \leq n, 1 \leq j \leq n \quad \rightarrow \quad P(i,j,k) = A(i,j,k-1).$$

This equation is split in two equations, one with subdomain $j < k$ is translated, and the other one, where $j \leq k$ is not:

$$1 \leq k \leq n, 1 \leq i \leq n, 1 \leq j < k \quad \rightarrow \quad P(i,j,k) = A(i,j,k-1)$$
$$1 \leq k \leq n, 1 \leq i \leq n, k \leq j \leq n \quad \rightarrow \quad P(i,j,k) = A(i,j,k-1).$$

The first equation, after translation becomes: equations:

$$1 \leq k \leq n, 1 \leq i \leq n, n+1 \leq j < n+k-1 \quad \rightarrow \quad P(i,j,k) = A(i,j,k-1)$$
$$1 \leq k \leq n, 1 \leq i \leq n, j = n+k-1 \quad \rightarrow \quad P(i,j,k) = A(i,j-n,k-1).$$

Putting together the equations which result from the transformation, we obtain the new definition of P:

$$1 \leq k \leq n, 1 \leq i \leq n, 1 \leq j < k \quad \rightarrow \quad P(i,j,k) = A(i,j,k-1)$$
$$1 \leq k \leq n, 1 \leq i \leq n, k \leq j \leq n+k-1 \quad \rightarrow \quad P(i,j,k) = \begin{cases} \text{if } j < n+k-1 \text{ then } A(i,j,k-1) \\ \text{if } j = n+k-1 \text{ then } A(i,j-n,k-1). \end{cases} \tag{25}$$

A similar treatment, once applied to (12), (13), and (14) gives:

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, \quad k+1 \leq j \leq n+k-1 \quad \rightarrow \quad AI(i,j,k) = A(k,j,k) \tag{26}$$

$$1 \leq k \leq n, 1 \leq i \leq n, k+1 \leq j \leq n+k-1 \quad \rightarrow \quad AJ(i,j,k) = A(i,k,k) \tag{27}$$

$$1 \leq k \leq n, 1 \leq i \leq n, i \neq k, \quad k+1 \leq j \leq n+k-1 \quad \rightarrow \quad AK(i,j,k) = \begin{cases} \text{if } j < k+n-1 \text{ then } A(k,j,k-1) \\ \text{if } j = k+n-1 \text{ then } A(k,j-n,k-1) \end{cases} \tag{28}$$

Finally, the output equation (6) has to be rewritten

$$1 \leq i \leq n, 1 \leq j \leq n \rightarrow R(i,j) = A(i,j+n,n) \tag{29}$$

in order to take into account the reindexing of the results $A(i,j,n)$.

4.3 Translation by $(0, p, -p)$

Let $n = 2p$. We now apply a translation of $(0, p, -p)$ to the part of the domain such that $k > p$. This transformation is sketched in figure 9. All equations but (1) are affected by this transformation. Applying the translation to (7), (8), (27) and (24) provides the following new computation equations:

$$1 \le k \le p, i = j = k \rightarrow A(i,j,k) = P(i,j,k)^* \quad (30)$$

$$1 \le k \le p, i = k+p, j = 2p+k \rightarrow A(i,j,k) = P(i,j,k)^* \quad (31)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, j = k \rightarrow A(i,j,k) = P(i,j,k) \otimes AI(i,j,k) \quad (32)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, j = 2p+k \rightarrow A(i,j,k) = P(i,j,k) \otimes AI(i,j,k) \quad (33)$$

$$1 \le k \le p, k+1 \le j \le 2p+k-1, i = k \rightarrow A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) \quad (34)$$

$$1 \le k \le p, i = k+p, k+2p+1 \le j \le 4p+k-1 \rightarrow A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) \quad (35)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, k+1 \le j \le 2p+k-1 \rightarrow A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AK(i,j,k) \quad (36)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, k+2p+1 \le j \le k+4p-1 \rightarrow A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AK(i,j,k) \quad (37)$$

On the other hand, the communication equations (25), (26), (27), and (28) become:

$$1 \le k \le p, 1 \le i \le 2p, k \le j \le 2p+k-1 \rightarrow P(i,j,k) = \begin{cases} \text{if } j < 2p+k-1 \text{ then } A(i,j,k-1) \\ \text{if } j = 2p+k-1 \text{ then } A(i,j-2p,k-1) \end{cases} \quad (38)$$

$$1 \le k \le p, 1 \le i \le 2p, k+2p \le j \le k+4p-1 \rightarrow P(i,j,k) = \begin{cases} \text{if } j < 4p+k-1 \text{ and } k > 1 \text{ then } A(i,j,k-1) \\ \text{if } j < 4p+k-1 \text{ and } k = 1 \text{ then } A(i,j-p,p) \\ \text{if } j = 4p+k-1 \text{ and } k > 1 \text{ then } A(i,j-2p,k-1) \\ \text{if } j = 4p+k-1 \text{ and } k = 1 \text{ then } A(i,j-3p,p) \end{cases} \quad (39)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, k+1 \le j \le 2p+k-1 \rightarrow AI(i,j,k) = A(k,j,k) \quad (40)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, k+2p+1 \le j \le k+4p-1 \rightarrow AI(i,j,k) = A(k,j,k) \quad (41)$$

$$1 \le k \le p, 1 \le i \le 2p, k+1 \le j \le 2p+k-1 \rightarrow AJ(i,j,k) = A(i,k,k) \quad (42)$$

$$1 \le k \le p, 1 \le i \le 2p, k+2p+1 \le j \le k+4p-1 \rightarrow AJ(i,j,k) = A(i,k,k) \quad (43)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, k+1 \le j \le 2p+k-1 \rightarrow AK(i,j,k) = \begin{cases} \text{if } j \le k+2p-2 \text{ then } A(k,j,k-1) \\ \text{if } j = k+2p-1 \text{ then } A(k,j-2p,k-1) \end{cases} \quad (44)$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, k+2p+1 \le j \le k+4p-1$$

$$\rightarrow \quad AK(i,j,k) = \begin{cases} \text{if } j < k+4p-1 \text{ and } k > 1 \text{ then } A(k,j,k-1) \\ \text{if } j < k+4p-1 \text{ and } k = 1 \text{ then } A(p+1,j-p,p) \\ \text{if } j = k+4p-1 \text{ and } k > 1 \text{ then } A(k,j-2p,k-1) \\ \text{if } j = k+4p-1 \text{ and } k = 1 \text{ then } A(p+1,j-3p,p) \end{cases} \tag{45}$$

Finally, the output equation becomes

$$1 \leq i \leq 2p, 1 \leq j \leq 2p \quad \rightarrow \quad R(i,j) = A(i,j+3p,p) \tag{46}$$

4.4 Uniformization

In this section, we replace some of the linear dependence mapping by uniform ones, that is, we replace some of the dependences which are not translations by new ones involving only translations. This transformation concerns equations (40), (41), (42), and (43).

As far as equation (40) is concerned, we note that all points belonging to a line parallel to the i axis share the same right-hand side term $A(k,j,k)$. We choose a uniformization scheme where the value $A(k,j,k)$ is propagated to points $(k+1,j,k)$, ... , (N,j,k) then wrapped around to point $(1,j,k)$ and propagated to $(k-1,j,k)$.

Let us create a new variable AIP defined by:

$$\begin{aligned} &1 \leq k \leq p, 1 \leq i \leq 2p, \\ &\quad k \leq j \leq 2p+k-1 \\ &\qquad \rightarrow \quad AIP(i,j,k) = \begin{cases} \text{if } i = k \text{ then } A(i,j,k) \\ \text{if } i = 1 \text{ then } AIP(2p,j,k) \\ \text{otherwise } AIP(i-1,j,k) \end{cases} \end{aligned} \tag{47}$$

$$\begin{aligned} &1 \leq k \leq p, 1 \leq i \leq 2p, \\ &k+2p \leq j \leq 4p+k-1 \\ &\qquad \rightarrow \quad AIP(i,j,k) = \begin{cases} \text{if } i = k \text{ then } A(i,j,k) \\ \text{if } i = 1 \text{ then } AIP(2p,j,k) \\ \text{otherwise } AIP(i-1,j,k) \end{cases} \end{aligned} \tag{48}$$

It is clear that for all (i,j,k) of the domain, AIP has exactly the $A(k,j,k)$ value which is needed in the definition of $AI(i,j,k)$. Therefore, by replacing all the terms of the form $A(k,j,k)$ in the definition of AI by the corresponding value of AIP, we obtain from (40)

$$1 \leq k \leq p, 1 \leq i \leq 2p, i \neq k, k+1 \leq j \leq 2p+k-1 \quad \rightarrow \quad AI(i,j,k) = AIP(i,j,k) \tag{49}$$

and from (41)

$$1 \leq k \leq p, 1 \leq i \leq 2p, i \neq k+p, k+2p+1 \leq j \leq k+4p-1 \quad \rightarrow \quad AI(i,j,k) = AIP(i,j,k) \tag{50}$$

A similar operation can be done on the definition of AK (equations (44) and (45) by creating a new variable AKP.

The propagation scheme for AJ is simpler: value $A(i,k,k)$ is propagated from point to point $(i,k+1,k)$, ... ,$(i,N+k-1,k)$.

The final system of equations is summarized in figure 10, 11, and 12.

4.5 Ordering the calculations

In this section, we order the index points of the domain in such a way that the arguments of the left-hand side of the equations be ordered after the arguments of their right-hand side. In other words, we define a mapping t, called a timing-function, from $\mathbf{Z}^3$ to $\mathbf{N}$, which gives the instant of time when a equation attached to an index point is to be evaluated. This mapping must preserve the dependence between the equations. The timing-function we use is a piecewise linear mapping of the space. The total domain of the system is

$$\mathcal{D} = \{1 \leq k \leq p, 1 \leq i \leq 2p, k \leq j \leq 4p + k - 1\}$$

The function we choose is:

$$t(i,j,k) = \begin{cases} \text{if } i \geq k \text{ and } j < 2p + k \text{ then } i + j + k - 3 \\ \text{if } i < k \text{ and } j < 2p + k \text{ then } i + j + k - 2p \\ \text{if } i \geq k + p \text{ and } j \geq 2p + k \text{ then } i + j + k - 3 \\ \text{if } i < k + p \text{ and } j \geq 2p + k \text{ then } i + j + k - 2p \end{cases} \tag{51}$$

which can be checked to have the above property. This choice is illustrated in figure 13.

4.6 Synthesis of the final architecture

The final architecture is obtained by projecting the domain onto the plane $j = 0$. Each integral point of the projection of the domain is to be interpreted as a cell of the final architecture. In other word, the calculation associated with a point (i, j, k) is executed on processor (i, k), at time $t(i, j, k)$. We can check that this projection ensures that a cell has at most one calculation to perform at any given instant of time. The projected domain is shown in figure 16: the final architecture is a rectangle of length $2p$ and height p.

We consider successively two phases: the first phase corresponds to the subdomain of the system where $j < 2k + p$, and the second phase corresponds to the remaining of the domain.

4.6.1 First phase

The structure of each cell is deduced from the computation equations (52) to (59) of the final system of equations. In fact, each cell has four different behaviours, depending on its location and on the time, as shown in figure 14. For example, the star operation is done only by cells of the array such that $i = k$.

The communications are described by equations (60) to (69). We examine successively the communications for P, AI, AJ, AK, AIP and AKP.

- P is defined by equation (60). If $j < 2p + k - 1$, two cases are to be considered:
  - if $i \neq k + 1$, the P port of cell (i, k) is connected directly to the A port of the cell $(i, k - 1)$. Indeed, the timing function is the same for the left-hand side and the right-hand side argument.

- if $i = k+1$, the timing-function is $i+j+k-3$ for the left-hand side, and $i+j+k+2p$ for the right-hand side. Therefore, there is a latency of $2p-2$ between the production of A and its use.

If $j = 2p + k - 1$, again two cases are to be considered. If $i \neq k + 1$, the P port of cell (i, k) is connected to the A port of cell $(i, k - 1)$, but not directly, since P must receive the value produced $t(i, j, k) - t(i, j - 2p, k - 1) = 2p - 2$ cycles before. If $i = k + 1$, then the delau is $4p - 2$. In summary, the mechanism for communicating the P value has to be controlled by a boolean signals which select the right register or the direct link, depending whether $j = 2p+k-1$ or not and whether $i = k+1$ or not. We shall examine later on how these signals are created.

- AI is defined by (62). It is therefore always connected directly to AIP.
- AJ is defined by equation (64). If $j > k$, AJ stays inside a given cell (i, k), that is, it is stored in a register. This register is initialized by loading A of cell $(i, k - 1)$, when $j = k$.
- AK is defined by equation (66). If $j < k + 2p - 1$, the port AK of cell (i, k) is connected directely to AKP. When $j = k + 2p - 1$, AK is connected to a register which is loaded with AKP $2p$ cycles earlier, that is when $j = k$. This mechanism is very similar to the communication of P.
- AIP is defined by equation (68). If $i \neq 1$ and $i \neq k$, the AIP port of a cell is connected to the AIP port of its left neighbor. There is a wrap around connection for the case $i = 1$, and for the case $i = k$, AIP is connected to the port A of the cell (i, k).
- Finally, AKP is defined by equation (70). The definition is identical to that of AIP, except that when $i = k$, AKP is connected to the A port of cell $(i, k - 1)$

Figure 16 summarizes the structure of the systolic array for the first phase of the algorithm. Crossed squares represent the cells where the star operation is to be done.

4.6.2 Second phase

The computation done by the cell is the same than previously seen, except that the Star operation is done by cells (i, k) such that $i = k + p$.

The communication pattern is more complicated. We have to distinguish the case $k = 1$ from the case $k > 1$. When $k > 1$, the communication scheme is the same as previously. Let us detail the cas when $k = 1$ which concerns only the bottom row of the array.

- P is connected to the cell (i, p) of the top row in a wrap-around fashion. Depending on whether $j = 4k + p - 1$ or not, there is a buffer, just as was explained earlier.
- for the special case of processor $(p + 1, 1)$, AKP is connected to the A port of processor (p, p)

The structure of the array for this second phase is shown in figure 17. By merging together the structure for the first and the second phase, we obtain (up to a symmetry) the solution described in figure 5.

In the version we have sketched, the control of the operation of the array depend on the evaluation of linear inequalities involving the original indexes of the recurrence system. All these conditions can be replaced in a systematic way by the test of boolean control variables which circulate in a systolic fashion between the cells of the architecture. Indeed, linear inequalities define convex polyhedral subdomains of the total domain of the system. By adding finitely many new boolean variables which are propagated from the boundaries of the domain to all the inside points, it is possible to replace any equation by a conditional function, whose condition depends only on the evaluation of boolean variables. The interested reader may find in [35] or [36] more details on this mechanism.

5 Conclusion

The price to pay for our new array is a period of $\delta = 4n$ against a period of $\delta = n$ for Kung-Lo-Lewis version [20]. Also, partitioning issues are more involved, since acyclic implementations usually exhibit more favorable characteristics with respect to fault-tolerance, two-level pipelining, and problem decomposition in general ([13, 17])... However, the number of cells in our new array is half that of Kung-Lo-Lewis, and this is a great saving.

We have seen that this array can be derived in a systematic way by using conceptually simple transformations of recurrence equations. This approach, which can be partly mechanized, has the main interest of capturing the essential features of systolic algorithms, i.e., their geometrical regularity and their locality.

References

[1] H.M. Ahmed, J.M. Delosme, and M. Morf. Highly concurrent computing structures for matrix arithmetic and signal processing. *Computer* 15, 1, (1982), 65-82.

[2] A. Benaini. Conception et validation des algorithmes systoliques. "Thèse de l'Institut National Polytechnique de Grenoble", September 1988.

[3] P.R. Cappello and K. Steiglitz. "Digital signal processing applications of systolic algorithms," *VLSI Systems and Computations*, H.T.Kung, B. Sproull et G. Steel éditeurs, Computer Science Press (1981), 245-254.

[4] M.C. Chen. A design methodology for synthesizing parallel algorithms and architectures. *Journal of Parallel and Distributed Computing*, 461–491, December 1986.

[5] E.G. Coffman and P.J. Denning. *Operating System Theory.* Prentice Hall, 1972.

[6] M. Cosnard and M. Tchuente. Designing systolic algorithms by top-down analysis. In *Third International Conference on Supercomputing*, Boston (U.S.A), May 1988.

[7] J.M. Delosme. A parallel algorithm for the algebraic path problem. In M. Cosnard et al., editor, *Int. Workshop on Parallel and Distributed Algorithms*, North-Holland, October 1988.

[8] J.M. Delosme and I.C.F. Ipsen. An illustration of a methodology for the construction of efficient systolic architectures in VLSI. In *International Symposium on VLSI Technology, Systems and Applications, Taipei, Taiwan, R.O.C.*, pages 268–273, May 1985.

[9] P. Frison, P. Gachet, and P. Quinton. Designing systolic arrays with DIASTOL. In S.Y. Kung, R.E. Owen, and J.G. Nash, editors, *VLSI Signal Processing II*, pages 93–105, IEEE Press, November 1986.

[10] P. Gachet, B. Joinnault, and P. Quinton. Synthesizing systolic arrays using DIASTOL. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 25–36, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[11] M. Gondran and M. Minoux. *Graphs and Algorithms.* Wiley and Sons, 1984.

[12] L.J. Guibas, H.T. Kung, and C.D. Thompson. "Direct VLSI implementation of combinatorial algorithms," *Proc. Caltech Conf. on VLSI: Architecture, Design, Fabrication*, California Institute of Technology, Pasadena, CA, U.S.A. (1979), 509-525.

[13] K. Hwang and Y.H. Chen. " Partitioned matrix algorithm for VLSI arithmetic systems," *IEEE Trans. Computers* 31, 12 (1982), 1215-1224.

[14] R.M. Karp, R.E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, 14(3):563–590, July 1967.

[15] M.R. Kramer and J. Van Leeuwen. "Systolic computation and VLSI," *in Foundations of Computer Science IV*, J.W. DeBakker et al. eds (1983), 75-103.

[16] S.P. Kumar. *Parallel algorithms for solving linear equations on MIMD computers.* PhD thesis, Washington State University, 1982.

[17] H.T. Kung and M.S. Lam. Fault-tolerance and two-level pipelining in VLSI systolic arrays. *Journal of Parallel and Distributed Computing*, 1(1):32–63, 1984.

[18] H.T. Kung and C.E. Leiserson. "Systolic arrays for (VLSI)", in [24] chapitre 8.3.

[19] S.Y. Kung and S.C. Lo. "A spiral systolic architecture/algorithm for transitive closure problems," *Proc. IEEE Int. Conf. on Computer Design ICCD'85*, New-York, USA (1985), 622-626.

[20] S.Y. Kung, S.G. Lo, and P.S. Lewis. Optimal systolic design for the transitive closure and the shortest path problem. *IEEE Trans. Computers*, C-36(5):603–614, 1987.

[21] L. Lamport. Parallel execution on array and vector computers. In *1975 Sagamore Computer Conference on Parallel Processing*, IEEE, 1975.

[22] G.H. Li and B.W. Wah. "The design of optimal systolic arrays," *IEEE Trans. Computers* 34, 1 (1985), 66-77.

[23] R.E. Lord, J.S. Kowalik, and S.P. Kumar. Solving linear algebraic equations on an mimd computer. *J.ACM*, 30(1):103–107, 1983.

[24] C.A. Mead and L.A. Conway. *Introduction to VLSI systems*, Addison-Wesley, Reading, Mass., USA, 1980.

[25] R. G. Melhem. "A language for the simulation of systolic architectures," *Proc. IEEE 12-th Int. Sym. on Computer Architecture*, Boston, MA, USA (1985), 310-314.

[26] R.G. Melhem and W.C. Rheinboldt. A mathematical model for the verification of systolic networks. *SIAM J. Conput.*, 13(3):541–565, August 1984.

[27] D.I. Moldovan. "On the analysis and synthesis of VLSI algorithms," *IEEE Trans. Computers* 31, 11 (1982), 1121-1126.

[28] J.G. Nash and S. Hansen. "Modified Faddeev algorithm for matrix manipulation," *Proc. SPIE (Society of Photo-Optical Instrumentation Engineers)*, Real-time Signal Processing VII (1984), 39-46.

[29] P. Quinton. Automatic synthesis of systolic arrays from recurrent uniform equations. In *11th Annual Int. Symp. Computer Arch., Ann Arbor*, pages 208–214, June 1984.

[30] P. Quinton. Mapping recurrences on parallel architectures. In *Third International Conference on Supercomputing*, Boston (U.S.A), May 1988.

[31] P. Quinton and V. Van Dongen. "The Mapping of Linear Recurrence Equations on Regular Arrays", To appear in *The Journal of VLSI Signal Processing*, 1989
Quinton89".

[32] Y. Robert. M. Tchuente, "Résolution systolique de systèmes linéaires denses," *RAIRO Modélisation et Analyse Numérique* 19, 2 (1985), 315-326.

[33] Y. Robert and D. Trystram. Systolic solution of the algebraic path problem. In W. Moore, A. McCabe, and R. Urquhart, editors, *International Workshop on Systolic Arrays*, pages 171–180, Adam Hilger, University of Oxford, UK, July 2-4 1986.

[34] G. Rote. "A systolic array algorithm for the algebraic path problem (shortest paths; matrix inversion)," *Computing* 34 (1985), 191-219.

[35] S.K.Rao. Regular iterative algorithms and their implementations on processor arrays, PhD Thesis, Information Systems laboratory, Standford University, U.S.A., Octobre 1985.

[36] J.D. Ullman. *Computational aspects of VLSI*, Chapitre 5: Systolic algorithms, Computer Science Press, Rockville, Maryland, U.S.A, 1984.

[37] U. Zimmermann. "Linear and combinatorial optimization in ordered algebraic structures", *Ann. Discrete Math.* 10 (1981), 1-38.

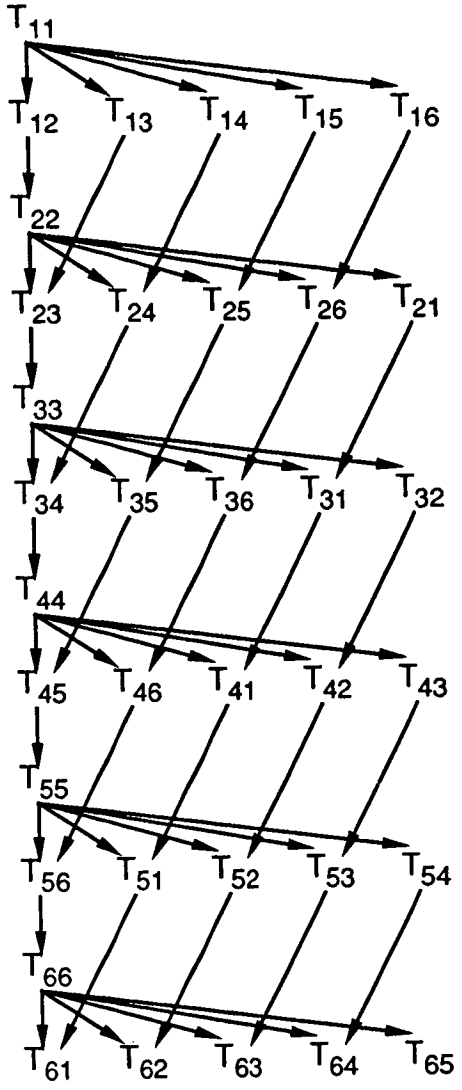


Figure 1: Task graph for the APP ($n = 6$)

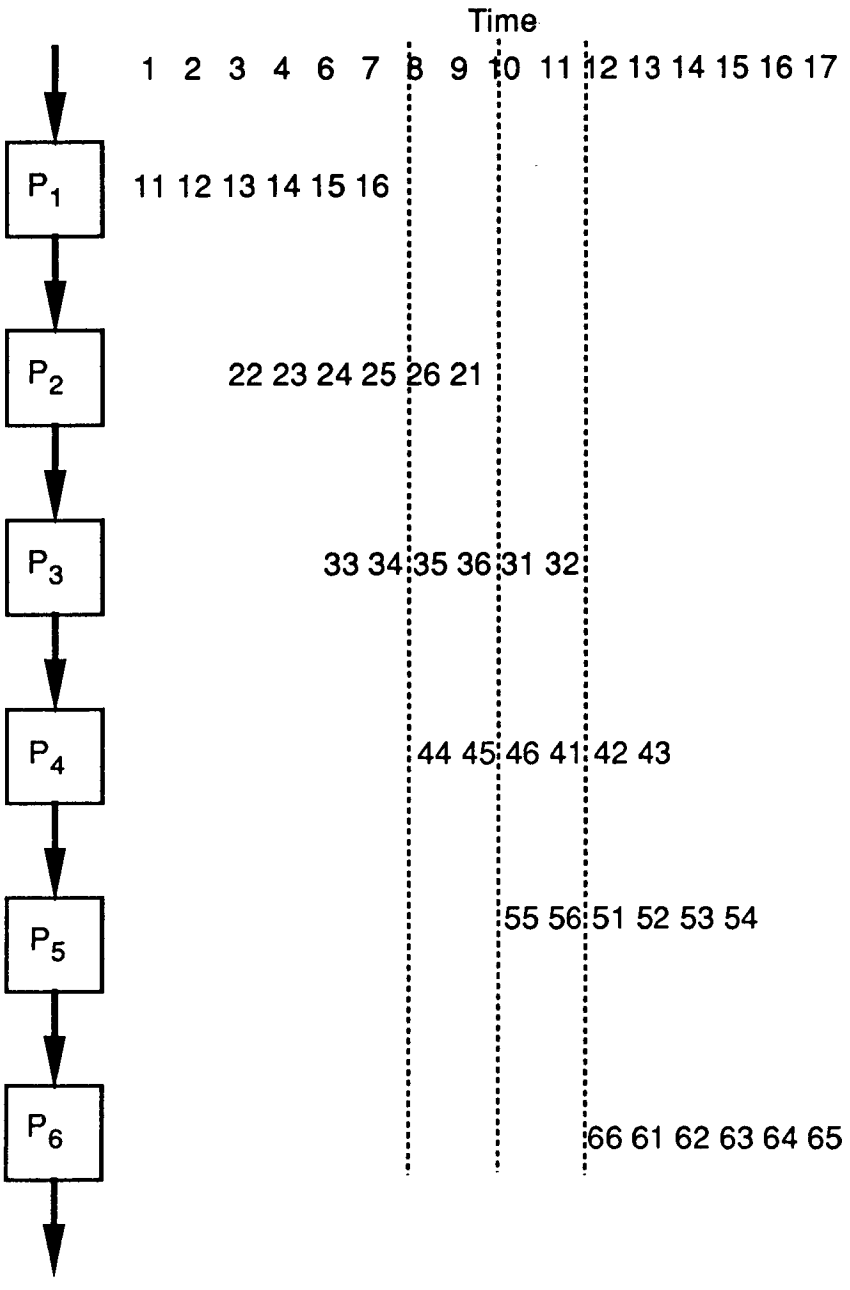


Figure 2: Solving a problem of size n on a linear array of n processors ($n = 6$)

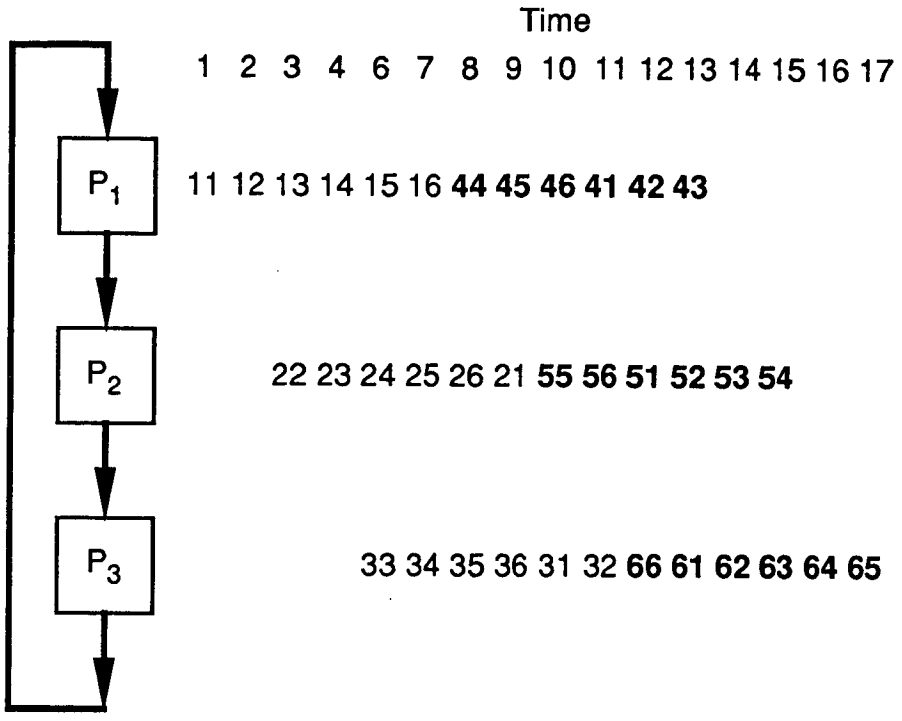


Figure 3: Solving a problem of size n on a ring of $n/2$ processors ($n = 6$)

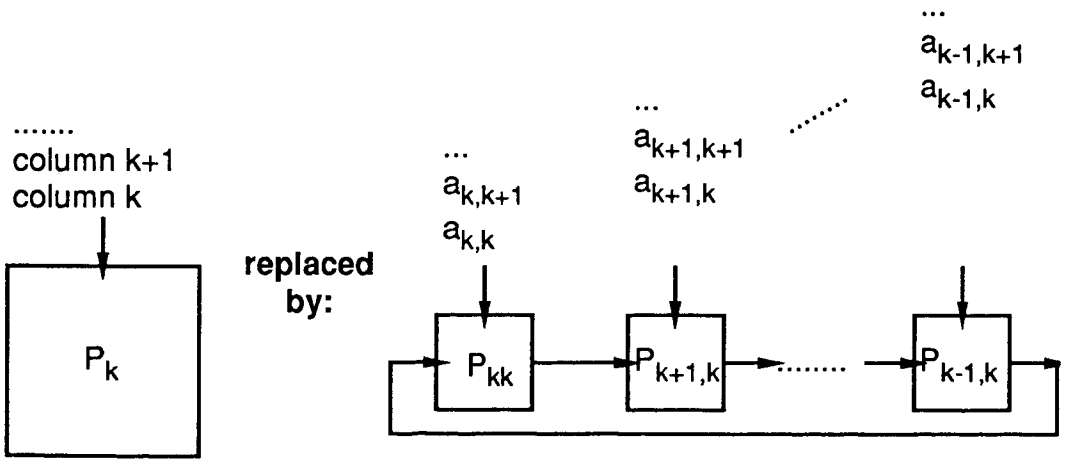


Figure 4: Replacing P_i by a ring of n elementary cells

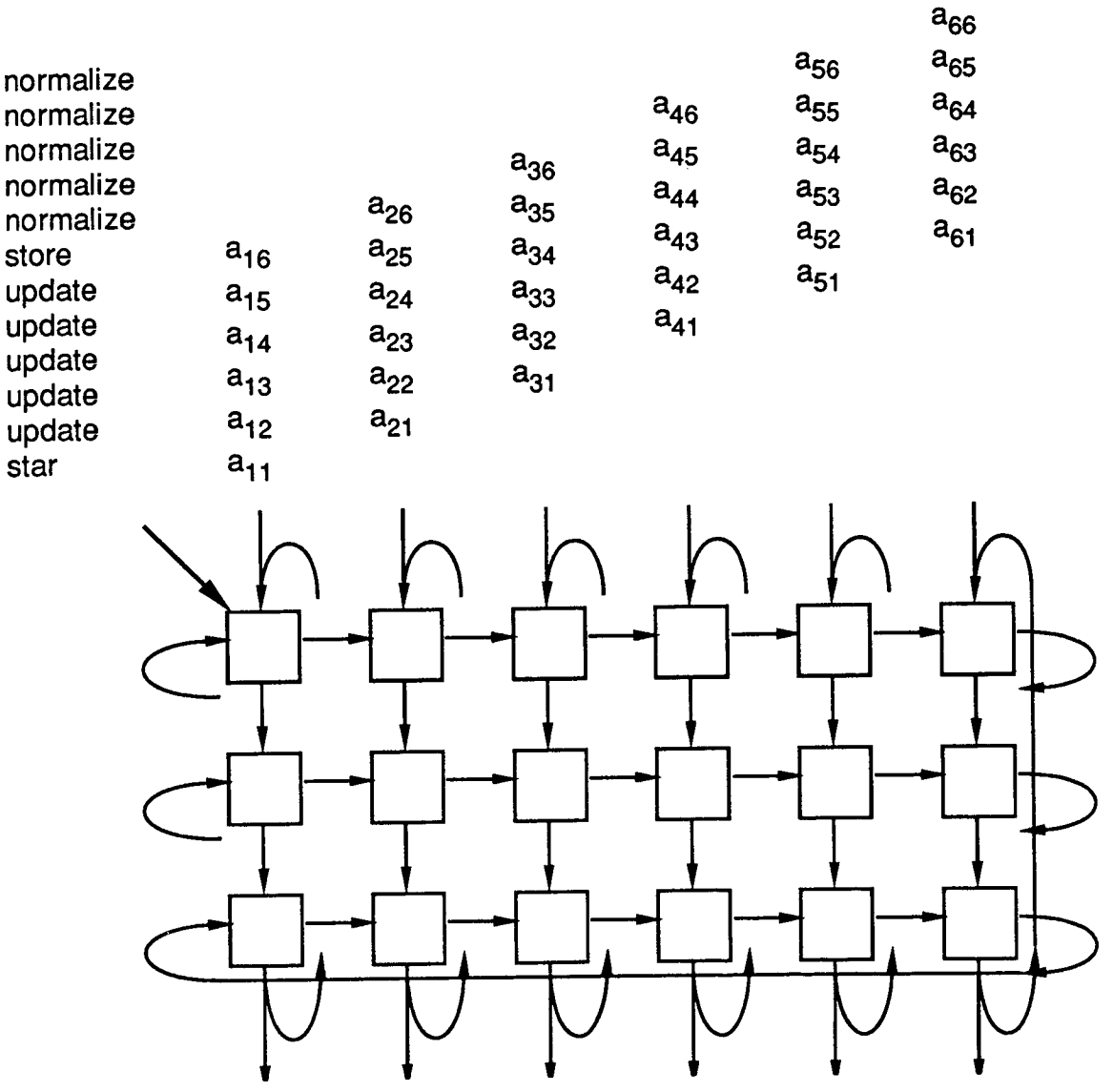
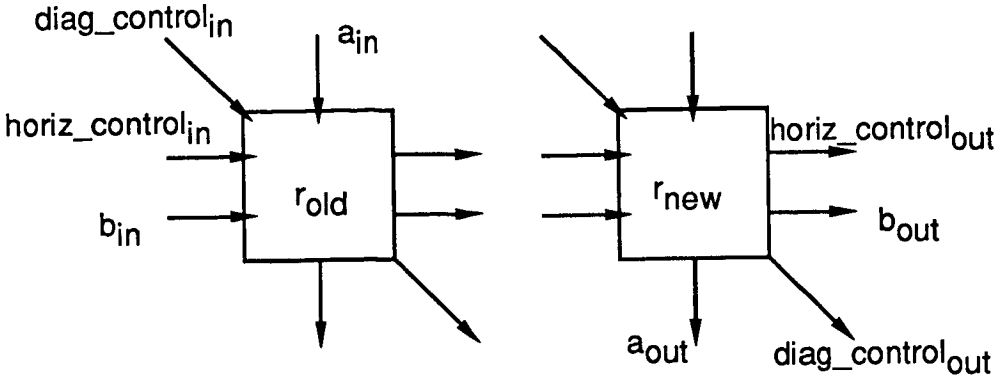


Figure 5: The 2D-toroidal systolic array with its input data and control



Step t **Step t+1**

case diag_control$_{in}$ **of**

star : **begin** horiz_control$_{out}$:= diag_control$_{in}$;
a_{out} := r_{old} ; b_{out} := a_{in}* ; r_{new} := nil ; **end**

update : **begin** horiz_control$_{out}$:= diag_control$_{in}$;
b_{out} := a_{in} ; a_{out} := nil ; r_{new} := nil ; **end**

store : **begin** horiz_control$_{out}$:= diag_control$_{in}$;
a_{out} := r_{old} ; b_{out} := nil ; r_{new} := b_{in} ; **end**

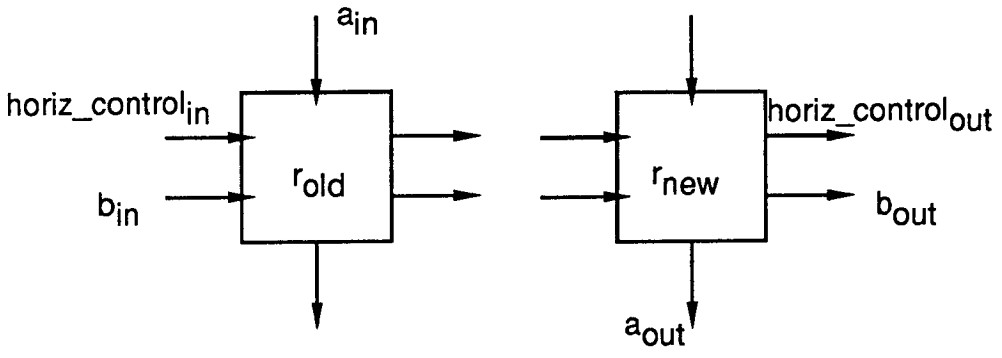
normalize : **begin** horiz_control$_{out}$:= diag_control$_{in}$;
a_{out} := r_{old} (x) b_{in} ; b_{out} := nil ; r_{new} := r_{old} ; **end**

nil : act as non-diagonal cell controlled by horiz_control$_{in}$

esac

diag_control$_{out}$:= diag_control$_{in}$

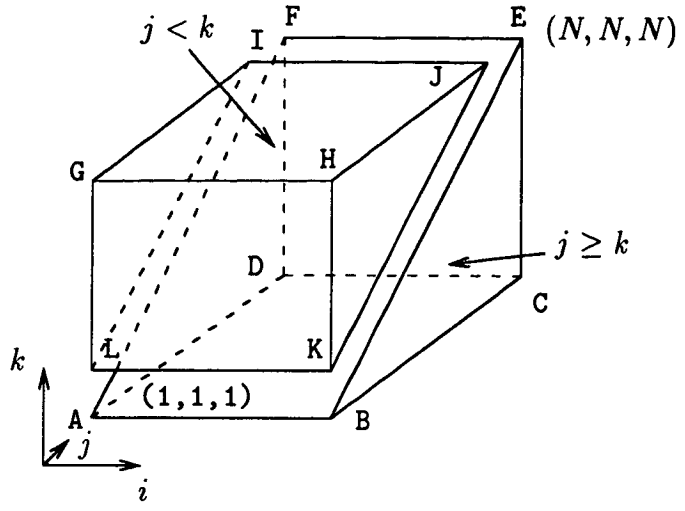
Figure 6: Operation of the diagonal processors



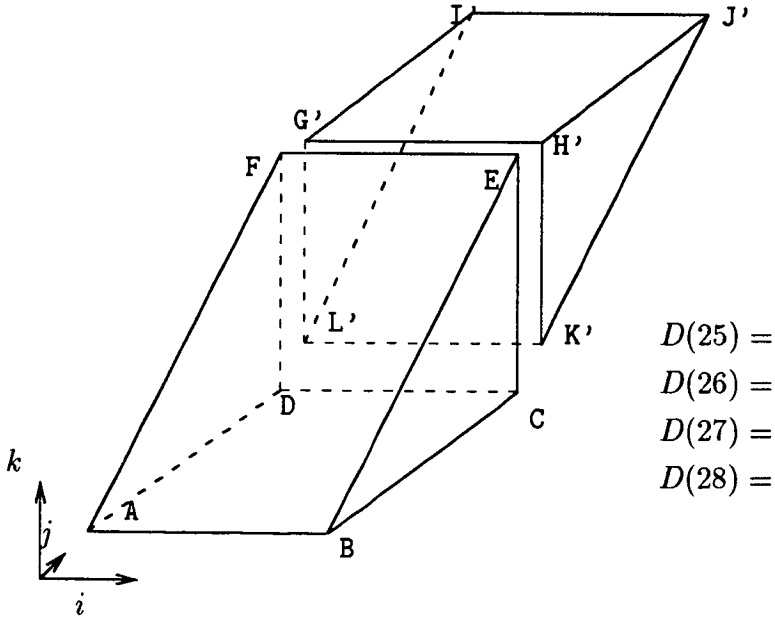
Step t **Step t+1**

case $horiz\_control_{in}$ **of**
star : **begin** $a_{out} := r_{old}$; $b_{out} := b_{in}$; $r_{new} := a_{in}$ (x) b_{in} ; **end**
update : **begin** $b_{out} := b_{in}$; $a_{out} := a_{in}$ (+) r_{old} (x) b_{in} ; $r_{new} := r_{old}$; **end**
store : **begin** $a_{out} := r_{old}$; $b_{out} := nil$; $r_{new} := nil$; **end**
normalize : **begin** $a_{out} := nil$; $b_{out} := nil$; $r_{new} := nil$; **end**
esac
$horiz\_control_{out} := horiz\_control_{in}$

Figure 7: Operation of the non-diagonal processors



$$D(7) = AE$$
$$D(8) = ABEF - AE$$
$$D(9) = AHED - AE$$
$$D(10) = ABCDEFGH - AHED - ABEF$$
$$D(11) = ABCDEFGH$$
$$D(12) = ABCDEFGH - AHED$$
$$D(13) = ABCDEFGH - ABEF$$
$$D(14) = D(10)$$



$$D(25) = AEJ'D - AE$$
$$D(26) = ABCJ'I'FD - AEJ'D - ABEF$$
$$D(27) = ABCJ'I'FD$$
$$D(28) = ABCJ'I'FD - ABEF$$

Figure 8: Translation of the subdomain $j < k$

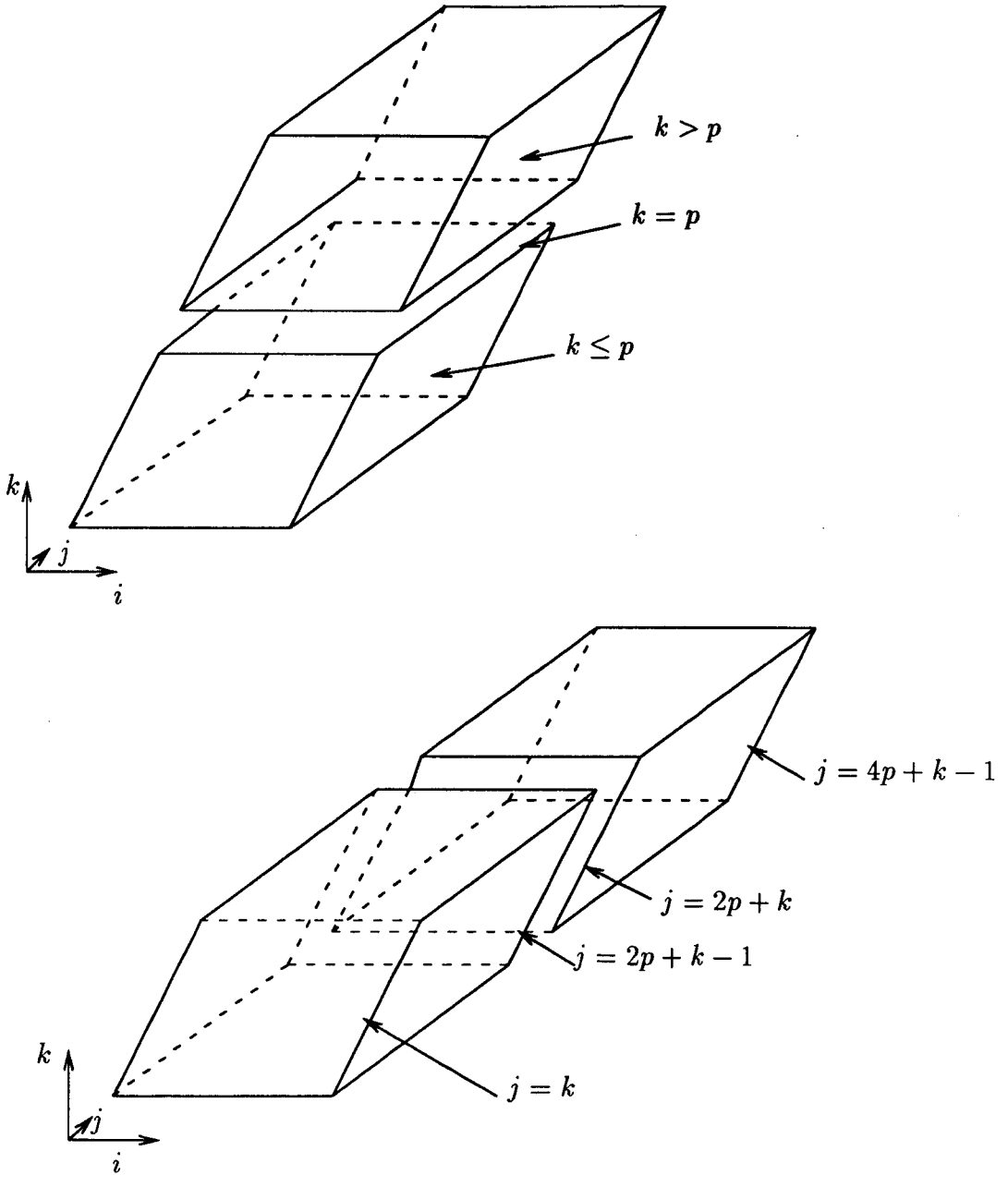


Figure 9: Translation of the subdomain $k > p$

Computation equations

$$
\begin{aligned}
1 \leq k \leq p, i = j = k &\rightarrow A(i,j,k) = P(i,j,k)^* && (52)\\
1 \leq k \leq p, i = k+p, j = 2p+k &\rightarrow A(i,j,k) = P(i,j,k)^* && (53)\\
1 \leq k \leq p, 1 \leq i \leq 2p,& \\
i \neq k, j = k &\rightarrow A(i,j,k) = P(i,j,k) \otimes AI(i,j,k) && (54)\\
1 \leq k \leq p, 1 \leq i \leq 2p,& \\
i \neq k+p, j = 2p+k &\rightarrow A(i,j,k) = P(i,j,k) \otimes AI(i,j,k) && (55)\\
1 \leq k \leq p, k+1 \leq j \leq 2p+k-1, i = k &\rightarrow A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) && (56)\\
1 \leq k \leq p, i = k+p,& \\
k+2p+1 \leq j \leq 4p+k-1 &\rightarrow A(i,j,k) = AJ(i,j,k) \otimes P(i,j,k) && (57)\\
1 \leq k \leq p, 1 \leq i \leq 2p, i \neq k,& \\
k+1 \leq j \leq 2p+k-1 &\rightarrow A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AI(i,j,k) && (58)\\
1 \leq k \leq p, 1 \leq i \leq 2p, i \neq k+p,& \\
k+2p+1 \leq j \leq k+4p-1 &\rightarrow A(i,j,k) = P(i,j,k) \oplus AJ(i,j,k) \otimes AI(i,j,k) && (59)
\end{aligned}
$$

Figure 10: Final system of equations: computation equations

Communication Equations

$$1 \le k \le p, 1 \le i \le 2p, \quad k \le j \le 2p+k-1$$
$$\rightarrow \quad P(i,j,k) = \begin{cases} \text{if } j < 2p+k-1 \text{ then } A(i,j,k-1) \\ \text{if } j = 2p+k-1 \text{ then } A(i,j-2p,k-1) \end{cases} \tag{60}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k+2p \le j \le k+4p-1$$
$$\rightarrow \quad P(i,j,k) = \begin{cases} \text{if } j < 4p+k-1 \text{ and } k > 1 \text{ then } A(i,j,k-1) \\ \text{if } j < 4p+k-1 \text{ and } k = 1 \text{ then } A(i,j-p,p) \\ \text{if } j = 4p+k-1 \text{ and } k > 1 \text{ then } A(i,j-2p,k-1) \\ \text{if } j = 4p+k-1 \text{ and } k = 1 \text{ then } A(i,j-3p,p) \end{cases} \tag{61}$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, \quad k+1 \le j \le 2p+k-1$$
$$\rightarrow \quad AI(i,j,k) = \begin{cases} \text{if } j < k+2p-1 \text{ then } AIP(i,j,k) \\ \text{if } j = k+2p-1 \text{ then } AIP(i,j-2p,k) \end{cases} \tag{62}$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, \quad k+2p-1 \le j \le k+4p-1$$
$$\rightarrow \quad AI(i,j,k) = \begin{cases} \text{if } j < k+4p-1 \text{ then } AIP(i,j,k) \\ \text{if } j = k+4p-1 \text{ then } AIP(i,j-2p,k) \end{cases} \tag{63}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k \le j \le 2p+k-1$$
$$\rightarrow \quad AJ(i,j,k) \begin{cases} \text{if } j > k \text{ then } AJ(i,j-1,k) \\ \text{if } j = k \text{ then } A(i,j,k) \end{cases} \tag{64}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k+2p \le j \le k+4p-1$$
$$\rightarrow \quad AJ(i,j,k) = \begin{cases} \text{if } j > k+2p \text{ then } AJ(i,j-1,k) \\ \text{if } j = k+2p \text{ then } A(i,j,k) \end{cases} \tag{65}$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k, \quad k+1 \le j \le 2p+k-1$$
$$\rightarrow \quad AK(i,j,k) = \begin{cases} \text{if } j < k+2p-1 \text{ then } AKP(i,j,k) \\ \text{if } j = k+2p-1 \text{ then } AKP(i,j-2p,k) \end{cases} \tag{66}$$

$$1 \le k \le p, 1 \le i \le 2p, i \ne k+p, \quad k+2p-1 \le j \le k+4p-1$$
$$\rightarrow \quad AK(i,j,k) = \begin{cases} \text{if } j \le k+4p-1 \text{ then } AKP(i,j,k) \\ \text{if } j = k+4p-1 \text{ then } AKP(i,j-2p,k) \end{cases} \tag{67}$$

Figure 11: Final system of equations

$$1 \le k \le p, 1 \le i \le 2p, \quad k \le j \le 2p+k-1$$

$$\rightarrow \quad AIP(i,j,k) = \begin{cases} \text{if } i = k \text{ then } A(i,j,k) \\ \text{if } i = 1 \text{ then } AIP(2p,j,k) \\ \text{otherwise } AIP(i-1,j,k) \end{cases} \tag{68}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k+2p \le j \le 4p+k-1$$

$$\rightarrow \quad AIP(i,j,k) = \begin{cases} \text{if } i = k+p \text{ then } A(i,j,k) \\ \text{if } i = 1 \text{ then } AIP(2p,j,k) \\ \text{otherwise } AIP(i-1,j,k) \end{cases} \tag{69}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k \le j \le 2p+k-1$$

$$\rightarrow \quad AKP(i,j,k) = \begin{cases} \text{if } i = k \text{ then } A(i,j,k-1) \\ \text{if } i = 1 \text{ then } AKP(2p,j,k) \\ \text{otherwise } AKP(i-1,j,k) \end{cases} \tag{70}$$

$$1 \le k \le p, 1 \le i \le 2p, \quad k+2p \le j \le 4p+k-1$$

$$\rightarrow \quad AKP(i,j,k) = \begin{cases} \text{if } i = k \text{ and } k > 1 \text{ then } A(i,j,k-1) \\ \text{if } i = k+p \text{ and } k = 1 \text{ then } A(k+p, j-p, k+p-1) \\ \text{if } i = 1 \text{ then } AKP(2p,j,k) \\ \text{otherwise } AKP(i-1,j,k) \end{cases} \tag{71}$$

Figure 12: Final system of equations (continued)

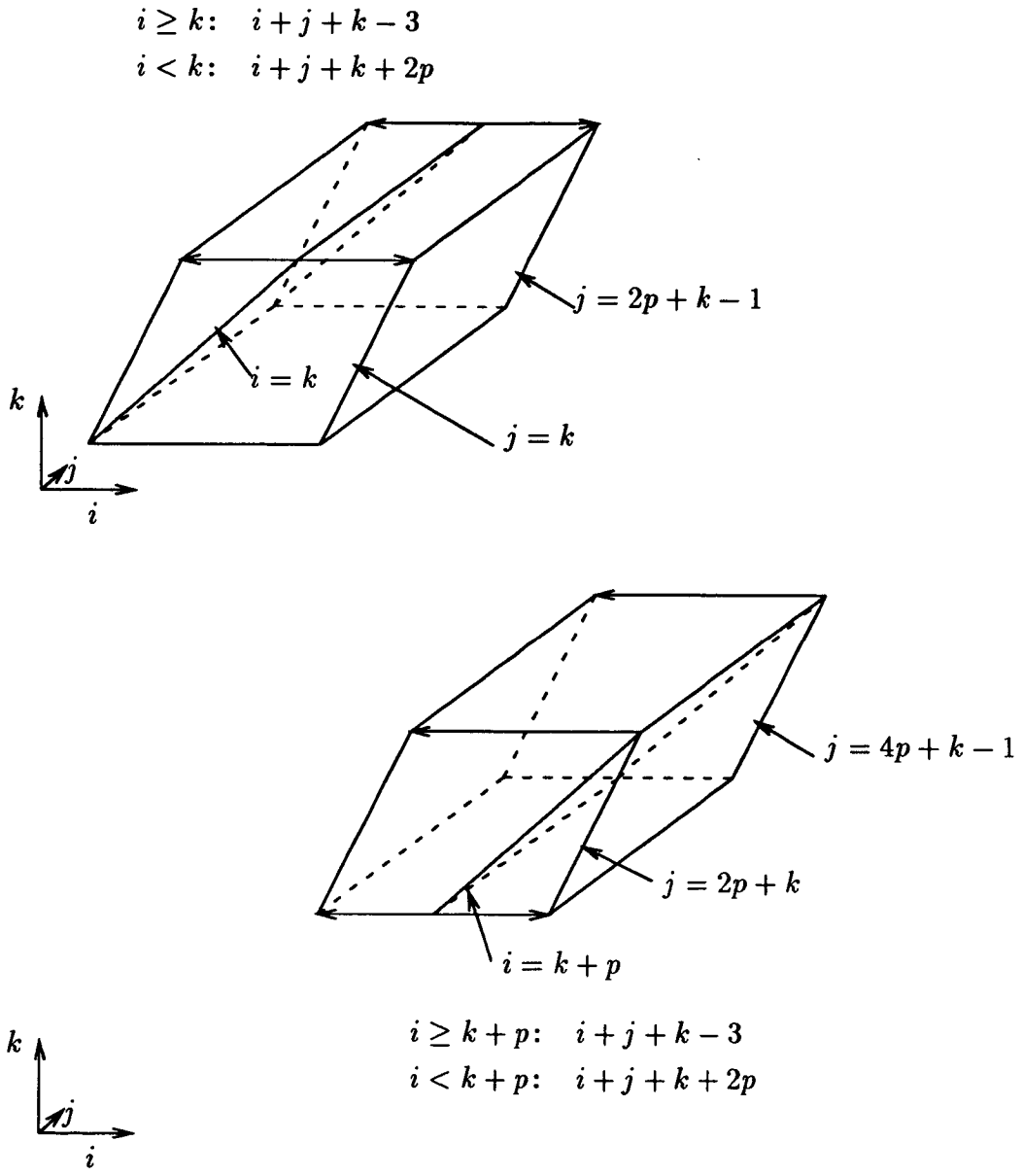


Figure 13: Timing-function

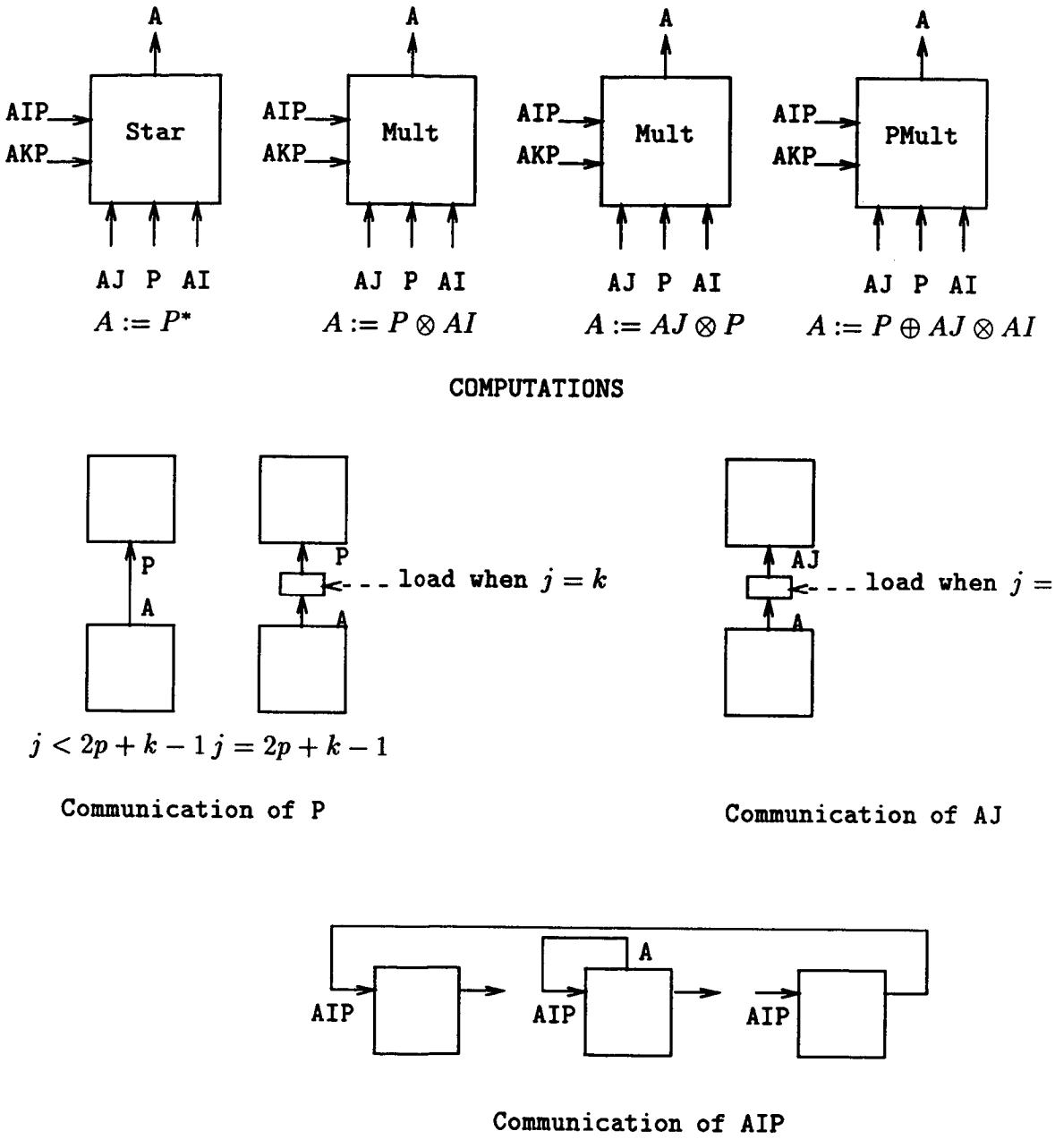
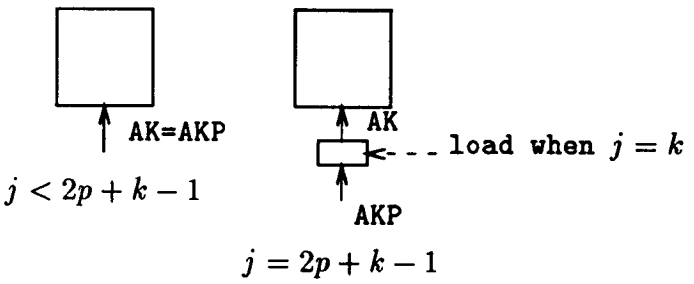
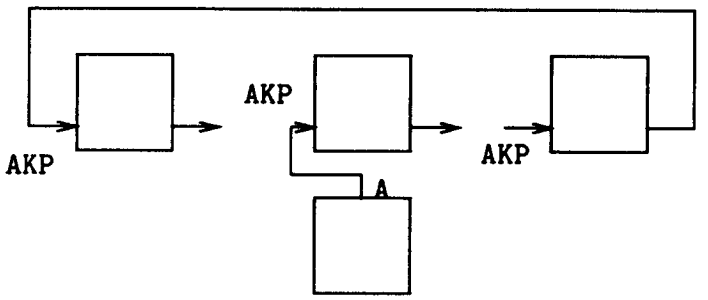


Figure 14: Structure of the cells of the array



Communication of AK



Communication of AKP

Figure 15: Structure of the cells of the array (cont'd)

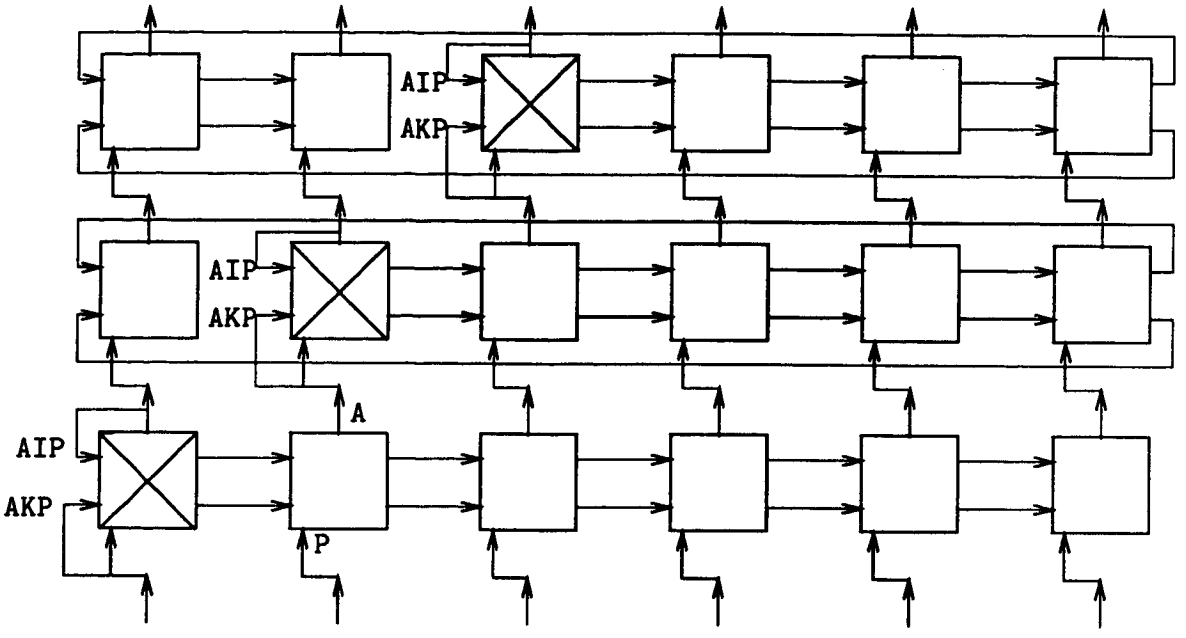


Figure 16: Systolic array for the first phase

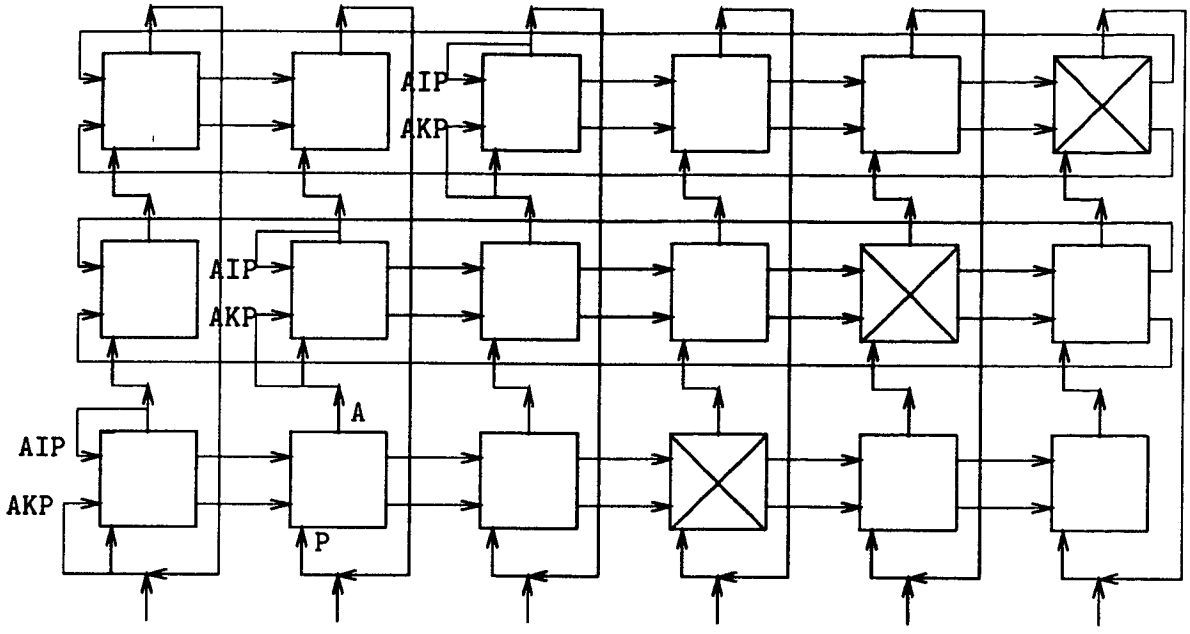


Figure 17: Systolic array for the second phase

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 480 **THE MODELLING SYSTEM PYRAMIDE AS AN INTERACTIVE HELP FOR THE GUIDANCE OF THE INSPECTION VEHICLE CENTAURE**
Philippe EVEN, Lionel MARCE
22 Pages, Juin 1989.

PI 481 **VERS UNE INTERPRETATION QUALITATIVE DE COMPORTEMENTS CINEMATIQUES DANS LA SCENE A PARTIR DU MOUVEMENT APPARENT**
Edouard FRANCOIS, Patrick BOUTHEMY
40 Pages, Juin 1989.

PI 482 **DEFINITION DE ALPHA : UN LANGAGE POUR LA PROGRAMMATION SYSTOLIQUE**
Christophe MAURAS
18 Pages, Juin 1989.

PI 483 **POLYNOMIAL IDEAL THEORETIC METHODS IN DISCRETE EVENT, AND HYBRID DYNAMICAL SYSTEMS**
Michel LE BORGNE, Albert BENVENISTE, Paul LE GUERNIC,
20 Pages, Juillet 1989.

PI 484 **IMPLEMENTING ATOMIC RENDEZVOUS WITHIN A TRANSACTIONAL FRAMEWORK**
Jean-Pierre BANATRE, Michel BANATRE, Christine MORIN
22 Pages, Juillet 1989.

PI 485 **THE MAPPING OF LINEAR RECURRENCE EQUATIONS ON REGULAR ARRAYS**
Patrice QUINTON, Vincent VAN DONGEN
40 Pages, Juillet 1989.

PI 486 **SYNTHESIS OF A NEW SYSTOLIC ARCHITECTURE FOR THE ALGEBRAIC PATH PROBLEM**
Abdelhamid BENAINI, Patrice QUINTON, Yves ROBERT,
Yannick SAOUTER, Bernard TOURANCHEAU
34 Pages, Juillet 1989.

PI 487 **PLANS SIMULATION USING TEMPORAL LOGICS**
Eric RUTTEN, Lionel MARCE
40 Pages, Juillet 1989.

PI 488 **ON FINITE LOOPS IN LOGIC PROGRAMMING**
Philippe BESNARD
20 Pages, Septembre 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique