SYSTEMATIC HARDWARE ADAPTATION OF SYSTOLIC ALGORITHMS

Miguel Valero-Garcia, Juan J. Navarro
Jose M. Llaberia and Mateo Valero

Dept. Arquitectura de Computadores
Facultad de Informática (UPC) Pau Gargallo 5
08028 BARCELONA (SPAIN)

ABSTRACT

In this paper we propose a methodology to adapt Systolic Algorithms to the hardware selected for their implementation. Systolic Algorithms obtained can be efficiently implemented using Pipelined Functional Units. The methodology is based on two transformation rules. These rules are applied to an initial Systolic Algorithm, possibly obtained through one of the design methodologies proposed by other autors. Parameters for these transformations are obtained from the specification of the hardware to be used. The methodology has been particularized in the case of one-dimensional Systolic Algorithms with data contraflow.

1. INTRODUCTION

Systolic Algorithms (SAs) exhibit some features that make them suitable for a direct hardware implementation (VLSI/WSI). Specifically, SAs are highly parallel/pipelined algorithms, specified on the basis of simple operations (fine granularity), with an high degree of homogeneity in the operations and regularity in the communication pattern. When an SA is implemented in hardware, then a *Systolic Array Processor* (SAP) is obtained [1].

The early SAs were obtained, probably, in an heuristic way [2]. They are SAs oriented to matrix problems (matrix multiplication, LU decomposition, etc). Later, automatic methodologies to design SAs have been proposed. The benefits of a design methodology are, among others, a savings in design time, the correctness of designs, and the possibility to obtain several solutions and choose the best according to a given criterium.

Any methodology uses some representation of the computation to be performed (signal flow graphs [3], [4], algorithms with loops [5], recurrences [6], [7], parallel programming languages [8], or data dependency graphs [9]). SAs are obtained through systematic manipulations of the chosen representation. In [10] a survey of proposed methodologies can be found. An up-to-date version of this paper appears in [11].

This work was supported by CAICYT under contract PA85-0314.

In general, SAs obtained through these methodologies show some features which trouble their direct and efficient hardware implementation. Among them, we point out the following:

a) SAs are *problem-size-dependent*, that is, the number of cells of the SAs depends on the size of the problem to be solved.

b) SAs have *simple synchronization*, that is, it is assumed that any cell spends the same time (a systolic cycle) to perform any operation, in spite of the fact that some operations can be more complex than others.

Feature (a) represents an evident drawback, because the number of processing elements (PEs) in an SAP is fixed and the size of the problems can be variable. This problem can be solved by *partitioning* the SA. Particular solutions to the partitioning problem have been presented in [12], [13] and [14], and more general solutions in [15], [16] and [17].

An SA with simple synchronization may exhibit two mayor drawbacks:

1) For a given implementation, some operations (square roots, divisions,etc) may require more time to be performed than others (multiplications, additions, etc), due to its complexity. If the SA is directly implemented, then the time required to perform the slowest operation becomes the cycle time. In this case, cells that perform simpler operations will be idle during a part of every cycle. We have not found, up to the present, any report dealing with this problem that we call *cycle-level unbalanced load.*

2) The SA can not be efficiently implemented using *Pipelined Functional Units* (PFUs). This kind of units can be used to increase the throughput of the system. Two-level pipelined SAs, that can be efficiently implemented using PFUs, are described in [18] and [19]. In [20] a technique is proposed to transform SAs with simple synchronization into two-level pipelined SAs. This technique was applied only to SAs without *data contraflow*.

In this paper we present a technique which permits to solve systematically, any of the above mentioned problems. Some previous results appear in [21]. This technique uses two transformations. The first one is based on the *retiming* and *slowdown* concepts [3], [22]. The second one is based on *coalescing* [23]. We propose a model to represent SAs in order to permit the formalization of these transformations. This model improves the one proposed in [22]. Algorithms are proposed to determine the transformations that allow us to obtain cycle-level balanced SAs efficiently implementable using PFUs. More precisely, the model and transformations are particularized in the case of one-dimensional (1D) band SAs with data contraflow.

1D band SAs with data contraflow are efficient for solving problems such as: band triangular systems of linear equations [1], LU decomposition [1] or QR decomposition [24]. For any of these problems, dense SAs without data contraflow can also be found.

In 1D dense SAs without contraflow for the above problems, matrices involved in the computation enter the cells by rows or columns. Every cell must perform complex operations (square roots, divisions) in some cycles, and simpler operations (multiplications and additions) in other cycles. Dense SAs are easily partitioned. Typically, matrices are partitioned into square blocks and partial results are combined to produce the final result. SAs without data contraflow can be easily modified to use efficiently a second level of pipeline if the cells are implemented using PFUs.

On the other hand, in 1D band SAs with data contraflow, matrices enter the cells by diagonals. In this case only one cell performs complex operations. So, band SAs with data contraflow require less hardware to be implemented. However, partitioning the SA is not intuitively simple and the use of PFUs is more difficult due to the existence of feedback cycles. Efficient solutions to the partitioning problem can be found in [24], [25] and [26]. In this paper, automatic techniques are given to solve the problems associated with the use of PFUs in the design of the PEs.

This paper is structured as follows: Section 2 presents the model proposed for the description of SAs. In section 3, transformations applied to SAs are enunciated. In section 4, we propose an algorithm to adapt an SA to the hardware used to implement its cells. This algorithm uses transformations described in section 3. In this section, some restrictions are imposed to the hardware in order to facilitate the use of transformations. In section 5 an improvement is proposed to obtain more efficient algorithms. In section 6, restrictions imposed in section 4 are eliminated. In section 7, a final improvement is proposed which will permit to apply our methodology to a wide set of SAs.

2. MODELLING OF SYSTOLIC ALGORITHMS

An SA is a set of cells interconnected through unidirectional links. Each cell is also possibly communicated with the outside world through I/O unidirectional links. The cells perform operations on data arriving every cycle through input links. Every cell is busy during each cycle in which it performs an operation. In the proposed model for SAs, we assign these cycles to communication links, and it is assumed that any operation is performed in zero time. Results of operations are sent to other cells, or to the outside, through output links in every cell. To synchronize the whole computation performed by the SA, a time delay is associated with each link. The value of this delay is the number of cycles required to traverse the link. We assume that every cell performs the same operation each cycle (*time_homogeneous SAs)*. Furthermore, we suppose that only one link, at most, in each direction exists between any pair of cells is the SA. These restrictions are imposed here just in order to simplify notation, and they could be easily eliminated.

Any SA, and in particular, an 1D band SA with data contraflow, can be modelled by the tuple:

$$A = (w, I, O, R, E, S, k)$$

and by the definition of operations performed in each cycle by every cell. The meaning of each element in the tuple is the following:

w is the number of cells of SA A.

I is the set $\{I_1..I_p\}$, with p being the number of links entering into the SA from the outside. I_j is the data sequence $\{I_j(1), I_j(2),..\}$ which inputs to the SA through the j-th input link.

O is the set $\{O_1..O_q\}$, with q being the number of links leaving the SA. O_j is the data sequence $\{O_j(1), O_j(2),..\}$ which outputs the SA through the j-th output link.

R and E are matrices with elements in the form of: $X(i,j) = Z^{-x(i,j)}$ or $X(i,j) = 0$ [22]. Matrix R has w-by-w elements. The value $r(i,j)$ is the delay associated with the link from cell j to cell i. If such link does not exist, then $R(i,j) = 0$. Matrix E has w-by-p elements. The value $e(i,j)$ is the number of cycles from the beginning of the SA operation until cell i receives the first data item in the I_j sequence. If cell i receives no data from I_j then $E(i,j) = 0$.

Matrix S has w-by-q elements in the form of $S(i,j) = Z^{s(i,j)}$ or $S(i,j) = 0$. The value $s(i,j)$ is the number of cycles until cell i produces the first data item in O_j. If cell i produces no data for O_j, then $S(i,j) = 0$.

k is the *slow* [3] of the SA. A k-slow SA can solve k equal and independent problems in an interleaved way.

In most cases, the SA has a *regular I/O structure*, that is, only data $\{I_j(1), I_j(k+1), I_j(2k+1)...\}$ are valid data (data which influence the final result). Analogously, only data $\{O_j(1), O_j(k+1), O_j(2k+1) ...\}$ are valid results.

Operations performed by every cell in each cycle can be described in several ways (algorithmically, graphically, etc). In our case, we use a graphic notation, but this decision does not affect the methodology at all.

As an example, we show the model for an 1D band SA with data contraflow to solve the band triangular system of linear equations $Lx = b$. This SA is shown in figure 1.a. The model is:

$$w = \textit{bandwidth of matrix } L \ (8 \textit{ in the example})$$

$$k=2$$

$$e(i,i) = i+6 \qquad i \in [1..8]$$

$$e(8,9) = 14$$

$$E(i,j) = 0 \qquad i \in [1..8], j \in [1..9], i \neq j \textit{ and } (i,j) \neq (8,9)$$

$$s(8,1) = 14$$

$$S(i,1) = 0 \quad i \in [1..8] \textit{ and } i \neq 8$$

$$r(i,i+1) = 1 \textit{ and } r(i+1,i) = 1 \quad i \in [1..7]$$

$$R(i,j) = 0 \quad i,j \in [1..8], i \neq j+1 \textit{ and } j \neq i+1$$

$$I_i(kt+1) = l_{t+i,t+1} \quad t \geq 0 \ ; \ I_i(t) = 0 \textit{ otherwise} \quad i \in [1..8]$$

$$I_9(kt+1) = b_{t+1} \quad t \geq 0 \ ; \ I_9(t) = 0 \textit{ otherwise}$$

$$O_1(kt+1) = x_{t+1} \quad t \geq 0 \ ; \ O_1(t) = 0 \textit{ otherwise}$$

cell 1 performs a division in every cycle

cell i (i≠1) performs an inner product step in every cycle

This SA, proposed in [1], can also be obtained through any design methodology. It exhibits the drawbacks mentioned in section 1:

a) The number of cells ($w=8$) is equal to the bandwidth of matriz L. In a practical case, the number of PEs of an SAP is fixed and, likely lesser than w.

b) It is assumed that any operation requires one cycle to be performed. In this case, divisions, performed in cell *1*, are more time consuming than inner product steps, performed in the rest of cells. In a direct implementation of the algorithm, the cycle time would be fixed by the time required to perform a division.

3. TRANSFORMATION RULES FOR SYSTOLIC ALGORITHMS

In this section we describe two transformation rules which will permit to obtain SAs efficiently implementable in hardware. The formulation of these rules is based on the model presented in the previous section. The rules have been particularized in the case of 1D SAs.

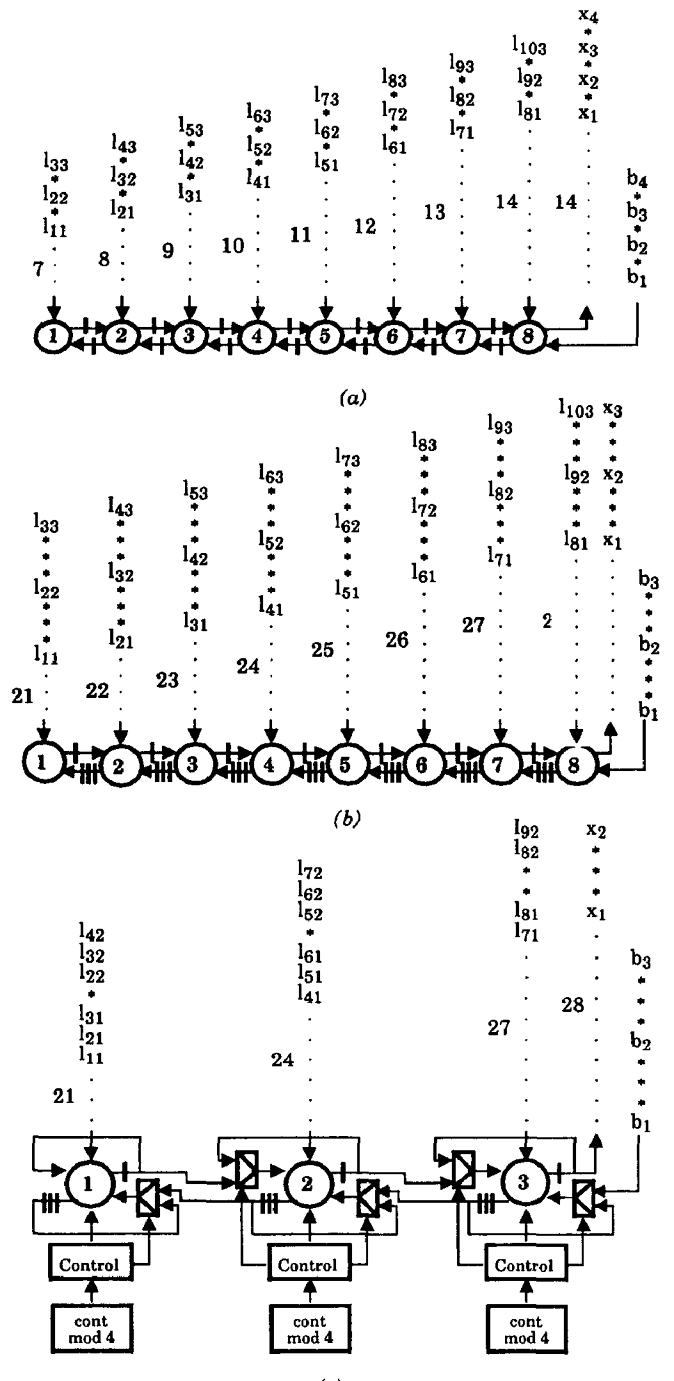


Figure 1. *(a) 1D SA for band triangular system of linear equations, (b) SA after applying rule 1, and (c) SA after applying rule 2.*

<u>Rule 1:</u>

Rule 1 is equivalent to transformations presented in [22]. It is based on the concepts of retiming and slowdown [3]. The rule allows to obtain an equivalent SA by resynchronizing the original one. In this new SA, cycles in which data arrive to cells and delays between cells have been modified. These new delays will serve to model the time needed to perform every operation for a given hardware to implement the cells. The rule is:

SA $A' = (w', I', O', R', E', S', k')$ performs the same computation as $A = (w, I, O, R, E, S, k)$ if:

A has a regular I/O structure

$$w' = w$$

$$k' = ck$$

$$E' = DE^c$$

$$S' = S^c D^{-1}$$

$$R' = DR^c D^{-1}$$

$$I'_i(k't+1) = I_i(kt+1) \quad i \in [1..p] \; and \; t \geq 0$$

$$O'_i(k't+1) = O_i(kt+1) \quad i \in [1..q] \; and \; t \geq 0$$

Any matrix of the form X^c is defined as: $X^c(i,j) = Z^{-cx(i,j)}$ if $X(i,j) = Z^{-x(i,j)}$ or $X^c(i,j) = 0$ if $X(i,j) = 0$. D is a diagonal matrix with w-by-w elements in the form of $D(i,i) = Z^{-d_i}$. The values c and d_i can be any of those belonging to the set of rationals. Values c and d_i $(i \in [1..w])$ are the parameters of rule 1.

<u>Rule 2:</u>

This rule is based on coalescing. Its objective is to eliminate the inefficiency of a k-slow SA, with a regular I/O structure, in which every cell performs one valid operation only one out of every k cycles. The rule allows to transform an SA A' (k'-slow) into an SA A^*, which performs the same computation, requiring the same number of cycles, but using a lesser number of cells (w^*). Cell q of A^* will perform operations assigned to $p(q)$ adjacent cells of A'. That will be possible if the $p(q)$ adjacent cells of A' perform their operations in different cycles. The values of $p(q)$ $(q \in [1..w^*])$ are the parameters of rule 2. These values must satisfy the following condition:

$$\sum_{q=1}^{w^*} p(q) = w'$$

Cell q of A^* will perform operations assigned to cells $a(q)..b(q)$ of A', where:

$$a(q) = 1 + \sum_{i=1}^{q-1} p(i) \quad and \quad b(q) = a(q) + p(q) - 1 \quad q \in [1..w^*]$$

Rule 2 can be formulated as follows:

Let A' be an SA represented by $A' = (w', I', O', R', E', S', k')$. We define $T(i)$ as the number of cycles a data item requires to travel from cell 1 to cell i in SA A':

$$T(i) = \sum_{j=1}^{i-1} r'(j+1,j) \quad i \in [1..w']$$

If A' satisfies condition:

$$T(i) \bmod k' \neq T(j) \bmod k' \quad i,j \in [a(q)..b(q)], i \neq j \; and \; q \in [1..w^*] \quad (1)$$

then applying rule 2, with parametres $p(q)$ to A' we obtain a new SA $A^* = (w^*, I^*, O^*, R^*, E^*, S^*, k^*)$ which performs the same computation as A'. Elements in the tuple modelling A^* are obtained by the following expressions:

$$e^*(q,q) = min_{i=a(q)}^{b(q)} e'(i,i) \quad q \in [1..w^*]$$

$$e^*(w^*, w^*+1) = e'(w', w'+1)$$

$$s^*(w^*, 1) = s'(w', 1)$$

$$r^*(q+1,q) = r'(a(q+1),b(q))\,;\; r^*(q,q+1) = r'(b(q),a(q+1))$$

$$q \in [1..w^*-1]$$

$$I^*_q(t+1) = I'_i(hk'+1) \text{ if } t+e^*(q,q)-e'(i,i) = hk'$$

$$i \in [a(q)..b(q)],\, q \in [1..w^*] \text{ and } t \geq 0$$

$$I^*_{w^*+1} = I'_{w'+1}$$

$$O^*_1 = O'_1$$

k^* is the number of different problems that can be solved by SA A^* in an interleaved way. An algorithm to obtain k^* is described in [27]. This value is not necessary to achieve our goal.

Figure 1.b shows the SA obtained by applying rule 1 with parameters $c=2$ and $d_i = -(i-1)$ $(i \in [1..8])$ to the SA shown in figure 1.a. The new SA is 4-slow. Delays between cells and the structure of input data sequences have been changed according with expressions described in this section. Now, applying rule 2 with parameters $p(1)=3$, $p(2)=3$ and $p(3)=2$ we obtain a new SA A^* shown in figure 1.c. This SA has only 3 cells. Using a counter module $k'=4$, initialized to zero, it is possible to determine, in every cycle, the operation to be performed by each cell as well as the data involved. These data can arrive from the neighbour cells or from operations performed by the cell in previous cycles. That is the reason for the feedback links. These links represent communications between cells of A', assigned by coalescing, to the same cell of A^*. The input data sequences to each cell of A^* have been obtained by interleaving data sequences in SA A'. A more detailed description of the procedure to obtain this structure can be found in [27].

Note that, in the previous example, rule 2 could have been applied with parameters $p(q) = k'=4$. For these values of the parameters $p(q)$, the number of cells of A^* is minimum $(w^* = \lceil w'/k' \rceil)$ and the utilization of every cell is maximum (each cell performs one valid operations every cycle).

4. ADAPTING SYSTOLIC ALGORITHMS TO THE HARDWARE

Transformation rules presented in the previous section can be used with different goals in mind (automatic SA partitioning, SA adaptation for its execution in Distributed Memory Multiprocessor Systems, etc). In this section we show how to use these rules to adapt SAs to the hardware selected to implement their cells. Specifically, a method is proposed to implement 1D band SAs with data contraflow using PFUs efficiently.

Figure 2 synthesizes graphically the proposed methodology. Using a description of the hardware, parameters for rules 1 and 2 are obtained. These rules are then applied to the initial SA A. Using the resulting models for A' and A^* and the description of the hardware, the structure and control of the SAP is automatically obtained. In this section we will see how to obtain an SA A^* with a minimum number of cells $(w^* = \lceil w'/k' \rceil)$. So, parameters for rule 2 are: $p(q) = k'$ $(q \in [1..\lceil w'/k' \rceil])$ and it will be necessary to find the parameters for rule 1.

We define OP_i as the operation performed by cell i of the initial SA A. This operation produces two results. One of them is sent to cell $i+1$ (or to the outside) and the other is sent to cell $i-1$ (or to the outside). We need some kind of representation of the hardware to be used in order to implement each cell. We propose a couple as a model of the hardware:

$$H = (RT, L)$$

RT is a vector of w reservation tables. Reservation table RT_i describes how the stages of a pipelined multifunctional unit (PMU) are used to perform OP_i. We suppose that the PMU is able to perform any of the w operations $OP_1 .. OP_w$.

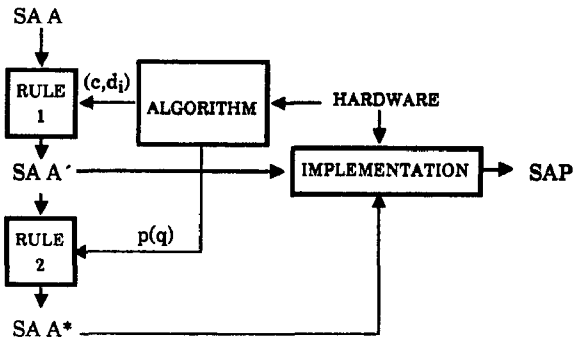


Figure 2. Use of transformation rules to adapt systolic algorithms to the hardware selected to implement their cells

L is a matrix with w-by-w elements in the form of $L(i,j) = Z^{-l(i,j)}$ or $L(i,j) = 0$. The value $l(i,j)$ is the number of cycles required by the PMU to obtain, by performing OP_j, the value to be sent to cell i. If cell j produces no values for cell i then $L(i,j) = 0$.

To apply rule 2, it is necessary to fulfil condition (1) described in section 3. But, if PMUs are used to implement each cell, this condition is not sufficient. It is necessary also to guarantee that there will not be conflicts in the use of the stages of the PMUs. In our case, $p(q) = k'$ $(q \in [1..w^*])$. So, reservation tables for the k' operations assigned to each cell of A^* must allow that any two operations can be initiated, in successive cycles, without conflicts. This restriction obviously influences the design of the PMUs to be used. Henceforth, we will call this restriction as R1.

As an example, figure 3 shows the hardware selected to implement SA in figure 1.a. To perform divisions required in cell 1, the division inversion algorithm [28] is applied, as shown in the following expression:

$$Q = \frac{A}{B} = -AR(2+RB)$$

where R is an approximation of value $-1/B$ obtained by indexing a lookup table with some bits of B. If we use two 3-stages pipelined multipliers and one 2-stages pipelined adder then a division requires 8 cycles, if we disregard the time to access the lookup table. Reservation table in figure 3.a shows how this operation is performed. Using the same kind of multipliers and adders, an inner product step can be performed in 5 cycles as shown in figure 3.b. Note that reservation tables in figure 3 satisfy restriction R1.

The model for the hardware to be used is completed by giving the following values:

$$l(2,1) = 8$$

$$l(i+1,i) = 0 \quad i \in [2..w-1]\,;\; l(i,i+1) = 5 \quad i \in [1..w-1]$$

$$L(i,j) = 0 \quad \textit{otherwise}$$

Now, we need to obtain the parameters of rule 1 which is necessary to adapt SA A to the hardware. These parameters must satisfy the following conditions:

<u>condition (a):</u>

In the resulting SA A', the delay associated with a link from a cell to any of its neighbours must be, at least, equal to the number of cycles required by the former to produce a value to be sent to the later. Moreover, this value must be, at least 1 to avoid data broadcasting. For the case of 1D band SAs with data contraflow, this condition can be enunciated as follows:

$$(a.1) \quad r'(i,i+1) = d_i + cr(i,i+1) - d_{i+1} \geq max(1, l(i,i+1)) \quad i \in [1..w-1]$$

$$(a.2) \quad r'(i+1,i) = d_{i+1} + cr(i+1,i) - d_i \geq max(1, l(i+1,i)) \quad i \in [1..w-1]$$

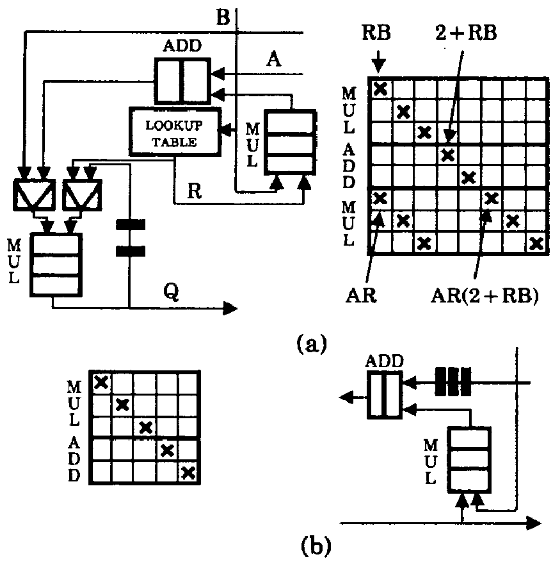


Figure 3. *A possible hardware to implement: (a) divisions, (b) inner product steps.*

<u>condition (b):</u>

The first data item in any input sequence to A' must not arrive to the cells before the SA computation starts. In our case, this condition can be expressed as follows:

$$(b.1)\quad e'(i,i) = d_i + ce(i,i) \geq 0 \quad i \in [1..w]$$

$$(b.2)\quad e'(w,w+1) = d_w + ce(w,w+1) \geq 0$$

<u>condition (c):</u>

This condition is derived from the particularization of condition (1) described in section 3, for the case of 1D band SAs with data contraflow, and assuming $p(q) = k'$. In this case:

$$T(i) = \sum_{j=1}^{i-1} r'(j+1,j) = \sum_{j=1}^{i-1} (d_{j+1} + cr(j+1,j) - d_j) \quad i \in [1..w]$$

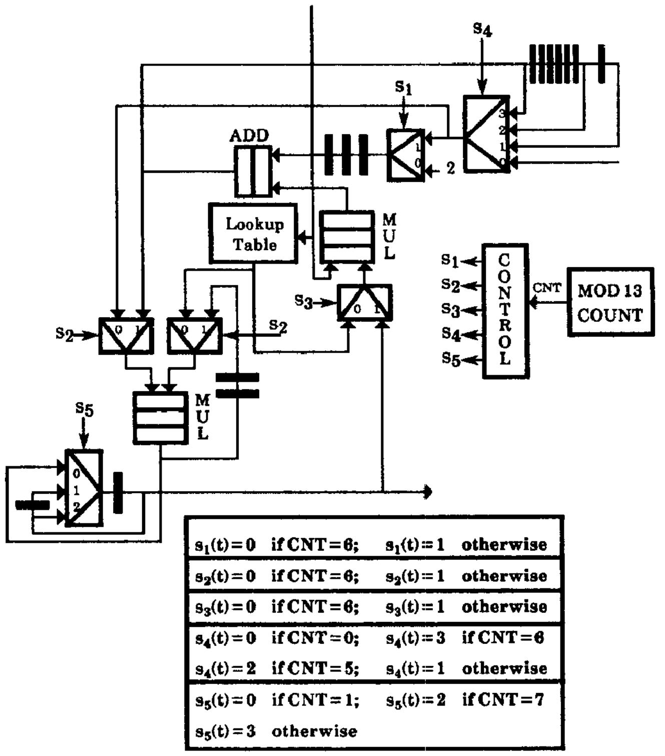
The condition is:

$$(c)\quad T(i) \bmod k' \neq T(j) \bmod k'$$

$$i,j \in [(q-1)k'+1..qk'],\ i \neq j \ and\ q \in [1..\frac{w'}{k'}]$$

Algorithm 1 can be used to find a set of values (c,d_i) which satisfy conditions (a), (b) and (c). In the obtained solution, c has the minumum possible value. There can be different valid solutions. It is possible also that no solution exists, if we use the minimum value for c. In this case, solutions must be found by increasing the value of c. This increase represents a loss of parallelism, and consequently, the resulting SA A^* will require more cycles than A' to obtain the same final result.

Once values (c,d_i) are obtained, then rule 1 and rule 2 are applied to A. From the models of A', A^* and the hardware used, a simple algorithm presented in [27] obtains the SAP structure and control. Figure 4 shows the internal structure and control for PE *1* obtained by applying the method to SA in figure 1.a and using the hardware described in figure 3.This PE initiates divisions when the *CNT* is *6* and inner product steps otherwise.



$s_1(t)=0$ if CNT=6;	$s_1(t)=1$ otherwise
$s_2(t)=0$ if CNT=6;	$s_2(t)=1$ otherwise
$s_3(t)=0$ if CNT=6;	$s_3(t)=1$ otherwise
$s_4(t)=0$ if CNT=0;	$s_4(t)=3$ if CNT=6
$s_4(t)=2$ if CNT=5;	$s_4(t)=1$ otherwise
$s_5(t)=0$ if CNT=1;	$s_5(t)=2$ if CNT=7
$s_5(t)=3$ otherwise	

Figure 4. *Internal structure and control for PE 1 of SAP to solve a band triangular system of equations.*

5. TRANSMITTENT DATA FLOWS

SAs obtained by applying rule 2 (as the one shown in figure 1.c) allow high hardware utilization, which tends to *1*, when the initiation phase has been completed (and all the cells begin to perform valid operations). However, the initiation phase requires many cycles. Most of these cycles are the result of the unnecessary retention suffered by some data items inside the cells. For instance, in figure 1.c, each element of vector x computed in cell 1 is held in this cell during $k'=4$ cycles before it is sent to cell 2. This value could be sent to cell 2 as soon as it is computed. Thus, the initiation phase could become shorter.

This innefficiency appears when, in the initial SA A, there is a transmittent data flow. Data in a transmittent data flow do not suffer any change during its travel through the cells. This problem can be solved by appropriately applying rule 1 to A.

In this section we consider the case of an 1D band SA with data contraflow in which there is a transmittent data flow. Data in this flow are generated at one end of the SA and travel to the other end without modifications. If data in the transmittent data flow go from cell *1* to cell w then, in the model of the hardware to be used, we have:

$$l(i+1,i) = 0 \quad i \in [2..w-1]$$

If data in the transmittent data flow go from cell w to cell *1* then we have:

$$l(i,i+1) = 0 \quad i \in [1..w-2]$$

We will show how to solve the first case. The second one can be solved in a similar way. We will assume also, as in the previous section, that $p(q) = k'$ $(q \in [1..w^*])$. Now, the parameters of rule 1 must satisfy the following conditions:

$$(a.1)\ d_i + cr(i,i+1) - d_{i+1} \geq max(1, l(i,i+1)) \quad i \in [1..w-1]$$

$$(a.2)\ d_{i+1} + cr(i+1,i) - d_i \geq max(1, l(i+1,i))$$

$$i \in [1..w-1] \text{ and } i \text{ is not a multiple of } k'$$

$$(a.3)\ d_{qk'+1} + \sum_{j=(q-1)k'+1}^{qk'} cr(j+1,j) - d_{(q-1)k'+1} \geq$$

$$\geq max(1, l((q-1)k'+2, (q-1)k'+1))$$

$$q \in [1..\frac{w'}{k'}-1]$$

Conditions (a.1) and (a.2) are similar to those appearing in section 4. Condition (a.3) indicates that the minimum delay associated to the path from cell $(q\text{-}1)k'+1$ to cell $qk'+1$ in $A'(q \in [1..w^*\text{-}1])$ is equal to the number of cycles needed by the former to produce a value which will be sent through this path. So, after applying, cell q-1 of A^* will send data to cell q as soon as possible. On the other hand, parameters d_i must satisfy conditions (b) and (c) as described in section 4.

The algorithm to obtain parameters for rule 1 is a slightly different version of algorithm 1. Sentences (3) and (4) in procedure *find_initial_parameters* of algorithm 1 must be replaced now by the following ones:

```
for i := 1 to w-1 do
    if i is not a multiple of k'
        then $d_{i+1} := \max(1,l(i+1,i)) - cr(i+1,i) + d_i$

for q := 1 to w'/k'-1 do
    $d_{k'q+1} := \max(1,l((q-1)k'+2,(q-1)k'+1)) + d_{(q-1)k'+1} - \sum_{j=(q-1)k'+1}^{qk'} cr(j+1,j)$
```

This set of values are obtained by replacing the sign ≥ for = in expressions (a.2) and (a.3). Sentences (16), (17) and (18) in procedure *find-new-parameters* must be replaced by:

```
if not impossible
    then for m := i to r(q)-1 do
        $d_{m+1} := d_m + \max(1,l(m+1,m)) - cr(m+1,m)$
```

As an example, applying rule 1 with parameters $c=2$, $d_1=0$, $d_2=-1$, $d_3=-2$, $d_4=-5$, $d_5=-6$, $d_6=-7$, $d_7=-10$, and $d_8=-11$, to the SA shown in figure 1.a we obtain SA in figure 5. A delay of *-1* cycle is associated with link from cell *3* to cell *4* and link from cell *6* to cell *7*. This delay is represented by means of a white rectangle. A negative delay does not make any physical sense. However, SA in figure 5 must be understood in the following way: a value sent to the right by cell *1* is received at the same time, one cycle later, by cells *2* and *4*, and 2 cycles later by cells *3*, *5* and *8*. Applying now rule 2 with parameters $p(1)=3$, $p(2)=3$ and $p(3)=2$ we obtain an SA similar to that shown in figure 1.c. However, in this case, cells 2 and 3 receive the first value from the North 2 and 3 cycles sonner respectively.

There is another source of innefficiency when applying rule 2. With reference to figure 1.c, cell *1* can not perform the first valid operation until it receives b_1. This value requires *21* cycles to reach cell *1* from cell *8* in spite of the fact that it is not modified during its travel. This problem can be solved bypassing the firsts values of *B* so that the cells can initiate operations as soon as possible.

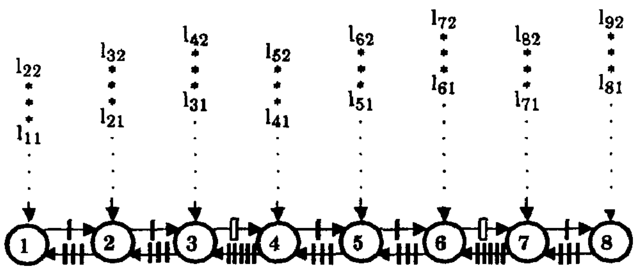


Figure 5. Applying rule 1 when there is a transmittent data flow

6. IMPLEMENTATION WITH MINIMUM HARDWARE

Restriction R1, described in section 4, can force to use more hardware than is strictly necessary to implement each operation. For example, as figure 6 shows, a division can be implemented with just one multiplier and one adder, in the same number of cycles as if two multipliers and one adder were used. However, if this implementation were used, then restriction R1 would not be satisfied and rule 2 could not be applied as described in section 4. In this section we propose an algorithm to determine the parameters of rule 1 when the used hardware does not satisfy restriction R1. In this case, it will not be generally possible to assign *k'* cells of *A'* to every *A** cell . The proposed algorithm also obtains the parameters *p(q)* for rule 2.

Solutions to this problem can be found by adapting results of previous works [29], [30]. In these works, methods are proposed to optimize the execution of a given set of operations in a PMU. In our case we know that:

a) the order in which operations must be initiated is fixed by the SA.

b) the initiation sequence must be periodically repeated, depending on the slow of the SA.

c) due to data dependencies, the initiation of operations must be separated in time by a minimum number of cycles.

On the other hand, an additional restriction is imposed to the solution of our problem. We will not allow a PMU to initiate more than one operation per cycle. The objective of this restriction is to limit the required bandwidth when implementing each cell.

Our first step is to evaluate the number of adjacent cells of *A'* that can be assigned to each *A** cell , that is, the parameters of rule 2. Then, the cycles in which every operation must be initiated are obtained. These values will serve to determine the parameters of rule 1. Without any lack of generality, we describe the method by applying it to cell *1* of *A**. This cell will execute $OP_1, OP_2, \ldots$

Suppose that *c* is evaluated as described in algorithm 1. So, $k' = kc$. Suppose also that every PMU has *p* stages and that it is able to perform any of the *w* operations of SA A. Let RT_i be the reservation table to execute OP_i, using the PMU.

We define *MU(i,m)* as the number of marks in the *m*-th row of RT_i. It is possible to assign, at most, *s* operations to cell *1* of *A** if, for any of the *p* stages of the PMU, the number of marks in reservation tables of these operations is not greater than *k'*. So, *s* satisfies the following condition:

$$max_{m=1}^{p} \sum_{i=1}^{s} MU(i,m) \leq k' \text{ and } max_{m=1}^{p} \sum_{i=1}^{s+1} MU(i,m) > k'$$

Value s is a maximum for parameter *p(1)* fixed from the reservation tables. A maximum for the MAL (minimum average latency) [30] of the PMU is *s/k'*.

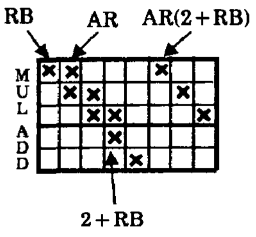


Figure 6. Reservation table to implement a division using just one multiplier and one adder

It is possible that not all of these s operations can be assigned to cell *1* of A^*. To determine the real value of $p(1)$ we use the reservation tables and the static collision matrices. Suffice it to apply the theory described in [30], taking into account features (a), (b) and (c) of our problem.

Let CM_i be the static collision matrix associated to OP_i. To obtain this matrix it is necessary to consider feature (b), that is, OP_i must be initiated every k' cycles. Moreover, all the elements in the first column of every *CM* will be TRUE to avoid the initiation of more than one operation in the same cycle.

Suppose that OP_i has been initiated in a given cycle and the dynamic collision matrix (*DCM*) of the PMU for this cycle, has been obtained. Now, we decide when OP_{i+1} can be initiated. Due to (c), this operation can be initiated x-*1* cycles after OP_i if:

$$(1)\ x-1 \geq l(i+1,i)$$

$$(2)\ x-1 + l(i,i+1) \leq k'$$

$$(3)\ DCM\ (i+1,x) = 0$$

Conditions (1) and (2) are due to data dependencies between OP_i and OP_{i+1} and between OP_{i+1} and OP_i, respectively. Condition (3) must be satisfied to avoid conflicts in the use of the stages of the PFU.

If we decide to initiate OP_{i+1} x-*1* cycles after OP_i then the new *DCM* must be obtained by ORing , bit-by-bit, CM_{i+1} with the *DCM* associated to the PMU when OP_i was initiated, rotated x-*1* columns left. This rotation in necessary to take into account feature (b).

Due to the fact that the initiation sequence is finite and periodical, the typical modified state diagram appearing in [30], which represent states of the PMU (nodes) and transitions (edges), becomes, in our case, a tree. Each node at level i in the tree represents a possible initiation of OP_i. From every node at level i there are as many edges to nodes at level $i+1$ as valid initiations exist for OP_{i+1}. The number of levels of the tree is the value of $p(1)$, the number of A' cells assigned to cell *1* of A^*. Any path in the tree with size $p(1)$ gives a possible initiation sequence for the $p(1)$ operations.

In order to improve the PMU efficiency it is possible to modify the reservation tables by using the method of delay insertion [30]. Moreover, in our case, a little increase of value c (which implies an increase of k') can result in an increase of the number of cells assigned to each A^* cell.

Figure 7 shows a simple example of this procedure. Figure 7.a shows the reservation tables to implement the *4* cells of an SA. Values of matrix L, needed to complete the hardware model, are also shown. Suppose that $k' = 8$. Then only *3* cells, at most, can be assigned to cell *1* of A^*. Figure 7.b shows the static collision matrices for these *3* operations and figure 7.c shows the resulting tree. In this case, the *3* operations can be assigned to cell *1* of A^*. OP_2 must be initiated *5* cycles after OP_1 and OP_3 *2* cycles after OP_2.

Algorithm 2 synthesizes the proposed procedure. It can be used to obtain the parameters for rules 1 and 2 that must be used to adapt SA A to the hardware.

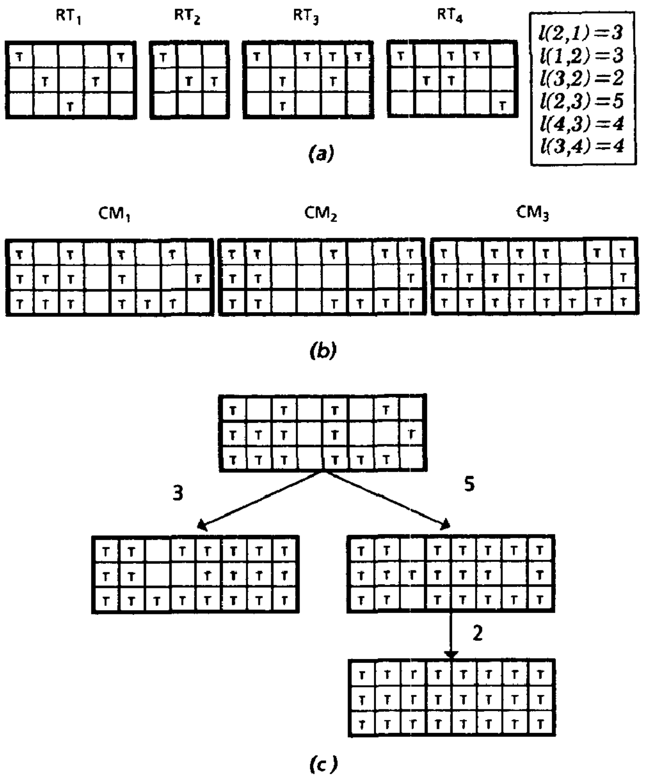


Figure 7. Example of procedure described in section 6: (a) model for the hardware to be used, (b) collision matrices, (c) state obtained.

7. NON_TIME_HOMOGENEOUS SYSTOLIC ALGORITHMS

The methodoly presented in previous sections can be applied to SAs in which each cell must perform always the same operation (time__homogeneous SAs), though different cells can perform different operations (non__spatial__ homogeneous SAs).

Non__time__homogeneous SAs appear frecuently in the literature. For instance, in the systolic ring for triangular system of equations proposed in [20], every cell must perform divisions in some cycles and inner product step in others. Non__time homogeneity appears also is most cases when applying DBT patitioning [14].

Non__time__homogeneous SAs can be easily treated by the proposed methodology. Let $OP_{i,1},..,OP_{i,n_i}$ be the n_i different operations performed by cell i during the SA computation. Let $RT_{i,j}$ be the reservation table to implement $OP_{i,j}$. We define RT_i as the reservation table obtained by ORing the n_i reservation tables associtated with cell i. Now, using RT_i as the reservation table for cell i, algorithms presented in previous sections can be used. This simple procedure can be successfully applied, specially if reservation tables associated with a cell are similar. This fact can be easily achieved for typical operations appearing in non__time__homogeneous SAs.

8. CONCLUSIONS

Generally, SAs Design Methodologies proposed in the literature start working from a specification of the problem to be solved, but do not take into account any implementation restriction.

In this paper we have proposed a technique that can be used to adapt automatically an SA to the hardware used to implement it. This technique permits to obtain SAs with any of the following features:

a) number of cells independent of the problem size,

b) different time consuming operations,

c) two_level pipelined

The technique is based on two transformation rules and a set of algorithms that use these transformations. Specifically, algorithms have been particularized to obtain efficiently implementable, two-level pipelined 1D band SAs with data contraflow. As an example, an SA to solve band triangular system of equations is presented. This kind of SAs can not be obtained through any other design methodology proposed up to now.

The methodology is generalizable to any other kind of 1D SAs as well as 2D SAs.

Transformation rules described in this paper can also be used for automatic partitioning of SAs. The basic idea consists on applying rule 1 with parameter:

$$c = \frac{1}{k} \lceil \frac{N}{w} \rceil$$

where N and k are, respectively, the number of cells and the slow of the original SA and w is the number of cells of the target one. Rule 2 is then applied with parameters:

$$p(q) = \lceil \frac{N}{w} \rceil \quad q \in [1..w]$$

At present, we are developing software tools based on the results of this work. These tools will permit to design automatically efficient SAPs.

REFERENCES

[1] H.T. Kung and C.E. Leiserson, "Systolic Arrays (for VLSI), " *Sparse Matrix Proc. 1978*, 1979, Society for Industrial and Applied Mathematics (SIAM), pp. 256-282. (A slightly different version appears in the text *Introduction to VLSI Systems*, Section 8.3. C.A. Mead and L.A. Conway, eds.,1980, Addison-Wesley, Reading, Mass.).

[2] H.T. Kung, "Why Systolic Architectures?," *Computer*, Vol. 15, No. 1, Jan. 1982, pp. 37-46.

[3] C.E. Leiserson and J.B. Saxe, "Optimizing Synchronous Systems," *Proc. 22nd Annual Symp. on Foundations of Computer Science*, Oct. 1981, pp. 23-36.

[4] S.Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors," *Proc. IEEE* Vol. 72, No. 7, July 1984.

[5] D.I. Moldovan, "On the Design of Algorithms for VLSI Systolic Arrays, " *Proc. IEEE*, Vol. 71, No. 1, pp. 113-120, 1983.

[6] P. Quinton, "Automatic Synthesis of Systolic Arrays form Uniform Recurrent Equations," *11th Int' l Symp. Computer Architecture*, pp. 208-214, IEEE & ACM, June 1984.

[7] G.J. Li, B.W. Wah, "The Design of Optimal Systolic Arrays," *IEEE Trans. on Computers*, Vol. C-34, No. 10, Jan. 1985, pp. 66-77.

[8] M. Chen, "Synthesizing VLSI Architectures: Dynamic Programming Solver," *Int' l Conf. on Parallel Processing*, 1986, pp. 776-784.

[9] I.V. Ramakrishnan and D.S. Fussell, "On Mapping Homogeneous Graphs on a Linear Array-Processor Model," *Int'l Conf. Parallel Processing 1983* pp. 440-447.

[10] J.A.B. Fortes, K.S. Fu and B.W. Wah, "Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays," *Proc. Int'l Conf. Acoustic, Speech and Signal Processing, 1985*, pp. 8.9.1-8.9.5.

[11] J.A.B. Fortes, K.S. Fu and B.W. Wah, "Systematic Design Approaches to Algorithmically Specified Systolic Arrays," *Computer Architecture Concepts and Systems*, North Holland 1988, pp. 455-494.

[12] D. Heller, "Partitioning Big Matrices for Small Systolic Arrays," *Chapiter 11 of VLSI and Modern Signal Processing*, S.Y. Kung, H.J. Whitehouse and T. Kailath eds. 1985, Prentice-Hall, Englewood Cliffs, N.J., pp. 185-199.

[13] R. Schreiber and P.J. Kucks, "Systolic Linear Algebra Machines in Digital Signal Processing," *Chapiter 22 of VLSI and Modern Signal Processing*, S.Y. Kung, H.J. Whitehouse and T. Kailath eds. 1985, Prentice-Hall, Englewood Cliffs, N.J., pp.389-405.

[14] J.J. Navarro, J.M. LLaberia and M. Valero, "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors," *Computer*, Vol. 20, No. 7, July 1987, pp. 77-89.

[15] D.I. Moldovan and J.A.B. Fortes, "Partitioning and Mapping Algorithms Into Fixed Size Systolic Arrays," *IEEE Trans. on Computers*, Vol. C-35, No. 1, Jan. 1986, pp. 1-12.

[16] H.W. Neils and E.F. Deprettere, "Automatic Design and Partitioning of Systolic/Wavefront Arrays for VLSI," *Circuits Systems Signal Process*, Vol. 7, No. 2, 1988, pp. 235-252.

[17] H. Moreno and T. Lang, "Graph-based Partitioning of Matrix Algorithms for Systolic Arrays: Application to Transitive Closure," *1988 Int'l Conf. on Parallel Processing.*

[18] H.T. Kung, L.M. Ruane and D.W.L. Yen, "Two-Level Pipelined Systolic Array for Multidimensional Convolution," *Image and Vision Computing*, Vol. 1, No. 1, Febr. 1983 pp. 30-36

[19] D.W.L. Yen and A.V. Kulkarni, "Systolic Processing and an Implementation for Signal and Image Processing," *IEEE Trans. on Computers*, Vol. C-31, No. 10, Oct. 1982, pp. 1000-1009.

[20] H.T. Kung and M.S. Lam, "Wafer-Scale Integration and Two-Level Pipelined Implementation of Systolic Arrays," *Journal of Parallel and Distributed Processing*, Vol. 1, No, 1, 1984.

[21] M. Valero-Garcia, J.J. Navarro, J.M. LLaberia and M. Valero, "Systematic Design of Two-Level Pipelined Systolic Arrays with Data Contraflow," *Proc. IEEE Int'l Conf. on Circuits and Systems 1988*, pp. 2521-2525.

[22] H.T. Kung and W.T. Lin, "An Algebra for Systolic Computation," *Elliptic Problem Solvers II*, Academic Press 1984, pp. 32-63.

[23] C.E. Leiserson, "Area-Efficient VLSI Computations," *PhD dissertation*, Departament of Computer Science, Carnegie-Mellon University, OCt. 1981, Published in book form as part of the *ACM Doctoral Dissertation Award Series* by the MIT Press, Cambridge, Massachusetts, 1983.

[24] N. Torralba and J.J. Navarro, "A One-Dimensional Systolic Arrays for Solving Arbitrarily Large Least Mean Square Problems, " *Proc. Int'l Conf. on Systolic Arrays.* May 1988 pp. 103-112.

[25] J.J. Navarro, J.M. LLaberia and M. Valero, "Computing Size-Independent Matrix Problems on Systolic Array Processors," *13th Int'l Symp. Computer Architecture*, 1986, pp. 271-279

[26] J.J. Navarro, J.M. LLaberia and M. Valero, "Solving Matrix Problems with No Size Restriction on a Systolic Array Processor," *Int'l Conf. Parallel Processing, Aug. 1986*, pp. 676-683.

[27] M. Valero-Garcia, J.M. Llaberia and J.J. Navarro, "Considering Implementation Features in the Design of Systolic Array Processors," *Internal Report*, RR 88/01, Facultad de Informatica Barcelona, Spain.

[28] Floating Point Division/ Square Root/ IEEE Arithmetic WTL 1032/1033, Application Note, *Weitek*, 1983.

[29] C.V. Ramamoorthy, "Pipeline Architecture," *Computing Surveys*, Vol. 9, No. 1, March 1977, pp 61-102.

[30] P.M. Kogge, "The Architecture of Pipelined Computers," *Hemisphere Publishing Corporation*, 1981.

ALGORITHM 1

/* Algorithm 1 finds parameters c and d_i for rule 1 assuming that parameters for rule 2 are $p(q) = k'$ $(q \in [1..w^*])$. First, the algorithm finds a set of values which satisfies condition (a) described in section 4. Then, these values are successively modified until they satisfy condition (c). Finaly, a simple correction of the values is applied to fulfil condition (b). */

```
Procedure find_initial_parameters

/* This procedure find a set of values which satisfy condition (a) */
begin
    /* Adding expressions (a.1) and (a.2) we obtain the minimum
    possible value for c */

(1)     $c := \max_{i=1}^{w-1} \left( \frac{\max(1,l(i,i+1)) + \max(1,l(i+1,i))}{r(i,i+1) + r(i+1,i)} \right)$

(2)     $d_1 := 0$

    /* We assign values to $d_i$ by replacing the sign ≥ for = in
    expression (a.2) */

(3)     for $i := 1$ to $w-1$ do
(4)         $d_{i+1} := \max(1,l(i+1,i)) - cr(i+1,i) + d_i$

/* This solution represents the assignation of the necessary delays
to the links going from right to left of the SA. Similary, delays can
be assigned to the links going from left to right */

end

Procedure check_condition (p, ok)

begin

/* This procedure, not shown here, checks condition (c). If it is not
satisfied then p takes the value of the index of the cell which causes
conflicts */

end

Procedure find_new_parameters (p,impossible)

begin

(1)     $k' := ck$
(2)     $q := (p-1)$ div $k' + 1$

    /* Cell p of A' will be assigned to cell q of A* */

(3)     $a(q) := (q-1)k + 1$
(4)     $b(q) := a(q) + k' - 1$
(5)     $i := p$
(6)     stop := FALSE
(7)     while not stop do
        begin
(8)             $d_i := d_i + 1$

            /* This increment of $d_i$ represents the transfer of one
            delay unit from the link going from cell i to cell i-1 to
            the link going from cell i-1 to cell i. It is necessary to
            check if this transfer is possible, that is, if this delay
            unit exists */

(9)             if $d_i + cr(i,i-1) - d_{i-1} + \max(1,l(i-1,i)) \le k'$
                then    begin
(10)                        stop := TRUE
(11)                        impossible := FALSE
                        end
                else    begin
(12)                        $i := i-1$
(13)                        if $i = a(q)$
                            then    begin

                                    /* It is not possible to
                                    avoid conflicts between
                                    operations assigned to
                                    cell q of A* */

(14)                                stop := TRUE
(15)                                impossible := TRUE
                                    end
                        end
        end
(16) if not impossible
(17)    then for $m := i$ to $w-1$ do
(18)            $d_{m+1} := d_m + \max(1,l(m+1,m)) - cr(m+1,m)$

end
```

```
Procedure fit_parameters

/* This procedure modifies parameters in order to satisfy condition
(b) */

begin
(1)     $d := \min(\min_{i=1}^{w} (d_i + ce(i,i)), d_w + ce(w,w+1))$

(2)     for $i := 1$ to $w$ do
(3)         $d_i := d_i - d$

end

begin   /* Algorithm 1 */

(1)     find_initial_parameters
(2)     repeat
(3)         check_condition (p, ok)
(4)         if not ok
(5)             then find_new_parameters (p, impossible)
(6)     until ok or impossible
(7)     if ok
(8)         then fit_parameters

end
```

ALGORITHM 2

```
/* This algorithm obtains parameters for rules 1 and 2 if restriction
R1 is not satisfied by the hardware. */

begin /* Algorithm 2 */

(1)     $c := \max_{i=1}^{w-1} \left( \frac{\max(1,l(i,i+1)) + \max(1,l(i+1,i))}{r(i,i+1) + r(i+1,i)} \right)$

(2)     $k' := kc$
(3)     $q := 1$
(4)     $a(q) := 1$
(5)     stop := FALSE
(6)     while not stop do
        begin
(7)             Select b(q) which satisfies

            $\max_{m=1}^{p} \sum_{i=a(q)}^{b(q)} MU(i.m) \le k'$ and
                    $(\max_{m=1}^{p} \sum_{i=a(q)}^{b(q)+1} MU(i,m) > k'$ or $b(q) = w)$
(8)             Obtain the tree for cell q of A* and select a
                maximum size path. Let p(q) be the number of
                operations in the initiation sequence associated
                with the selected path

(9)             $b(q) := a(q) + p(q) - 1$

(10)            Let $r'(a(q)+i, a(q)+i-1)$ be the label of edge from
                node at level i to node at level i+1 in the selected
                path $(i \in [1..p(q)-1])$

(11)        if $b(q) = w$
                then    begin
(12)                        stop := TRUE
(13)                        $w^* := q$
                        end
                else    begin
(14)                        $a(q+1) := b(q) + 1$
(15)                        $r'(a(q+1), b(q)) := l(a(q+1), b(q))$
(16)                        $q := q + 1$
                        end
        end
(17)    $d_1 := 0$
(18)    for $i := 1$ to $w-1$ do
(19)        $d_{i+1} := r'(i+1,i) + d_i - cr(i+1,i)$
(20)    for $i := 1$ to $w$ do
(21)        $d_i := d_i - d$

/* The value of d is defined in algorithm 1 */
/* Cell q of A* will perform operations assigned to cells a(q)..b(q) of
A */
end
```