IMPLEMENTATION OF SYSTOLIC ALGORITHMS USING PIPELINED FUNCTIONAL UNITS

Miguel Valero-García, Juan J. Navarro, José M. LLabería and Mateo Valero

Dept. Arquitectura de Computadores, Univ. Politècnica de Catalunya, Pau Gargallo 5, 08028 Barcelona Spain

In this paper we present a method to transform simple synchronization systolic algorithms into two-level pipelined systolic algorithms so that they can be efficiently implemented using pipelined functional units. The paper includes an example of application of the method to a one-dimensional systolic algorithm with data contraflow for QR decomposition.

1. INTRODUCTION

In this paper we present a method to implement *Systolic Algorithms* (SAs) using *Pipelined Functional Units* (PFUs). This kind of units allow to improve the throughput of a processor because of the possibility to initiate a new operation before the previous one has been completed.

In general, SAs obtained through automatic design methodologies [1] have *symple synchronization*. This means that it is assumed that every 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.

A SA with simple synchronization cannot be directly and efficiently executed using PFUs because in every cell each operation is performed only when the previous one has been completed, even if those operations are independent. This fact will not permit to achieve a good utilization of the PFU in every cell unless the original SA is transformed into a *two level pipelined* SA. This transformation is specially difficult when the SA has *feedback cycles* because in this case the operations performed by every cell are not independent.

Some proposals of two level pipelined SAs appear in [12], [3] and [6]. The problem of systematic transformation of SAs with simple sinchronization into two level pipelined SAs is studied in [4]. In that paper, the case of SAs without *feedback cycles* is studied. However, there is a number of problems with dependences between results (triangular system of equations, LU decomposition, etc) that can be efficiently solved using SAs with feedback cycles. In these cases, the SAs obtained when applying the techniques proposed in [4] do not make a good use of the hardware.

In [11] we propose an automatic method to transform SAs with simple synchronization into two level pipelined SAs. This method allows to obtain efficient results even when the original SAs have feedback cycles. The method is based on three well-known SA transformations: *slowdown, retiming* and *coalescing* [5]. The contributions of the present paped complement those in [11]. The result is a whole methodology that can serve as a basis for an automatic tool to help in the design of SAPs. In the following section we present, by means of a simple example, the problems arising when a SA must be executed using PFUs. At the end of the section we present the structure of the rest of the paper and describe briefly the main contributions in each of the remaining sections.

2. SYSTOLIC ALGORITHMS AND PIPELINED FUNCTIONAL UNITS

In this section we describe intuitively the proposed method to execute efficiently SAs using PFUs. This description will permit to present the major contributions of this paper. The method will be particularized for the case of *1D SA with data contraflow*. This kind of SA has feedback cycles. As we said before, it is difficult to transform SAs with data contraflow into two level pipelined SAs. For this reason, 1D SA with data contraflow will permit us to demonstrate the power of the method.

Work supported by the Ministry of Education of SPAIN (CICYT TIC 299/89)

CH2920-7/90/0000/0272$01.00 © 1990 IEEE

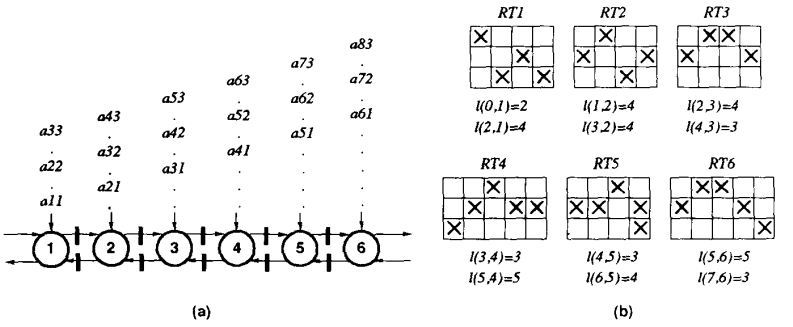

Figure 1: *(a) General aspect of a 1D SA with data contraflow. (b) An example of reservation tables to implement the operations performed by the cells of the SA shown in figure 1.a.*

Figure 1.a shows the general aspect of a 1D SA with data contraflow and simple sincronization. A delay of one cycle is associated with every link between cells. These delays are represented by black rectangles in figure 1.a. Each cell performs one valid operation every two cycles. For this reason, it is said that the SA is 2-*slow* [5]. Figure 1.a also shows the input pattern for a matix involved in the computation. The main advantage of 1D SA with data contraflow is that, for non homogeneous problems like triangular systems of equations, LU decomposition or QR decomposition, only one cell must perform complex operations (divisions or square roots). The rest of cells perform simple operation (inner product steps)

1D SAs with data contraflow have been used to solve problems such as band triangular systems of equations [2], [6], LU decomposition [7] or QR decomposition [10]. Obviously, the nature of the problem to be solved determines the operations performed in every cell of the SA and the I/O data sequences.

Suppose that the SA in figure 1.a must be executed using PFUs in such a way that the computation time for any stage of the pipeline becomes the systolic cycle. Every cell i performs always the same operation, named OP_i, and the reservation table RT_i represents the order in which the PFU stages are used to perform OP_i [9]. Let $l(i,j)$ be the number of cycles required by the PFU to compute, by performing OP_j, a value involved in OP_i.

Figure 1.b shows an example of reservation tables and values $l(i,j)$ for the SA in figure 1.a. These values indicate, for instance, that cell 1 requires 4 cycles to compute the value to be sent to cell 2. It is not very realistic to think of SAs with a number of different reservation tables and computation times. Frecuently, several cells of the SA perform the same operation and reservation tables associated with these operations are equal. However, our intention is to propose a general method that, as we will see later, benefits from the fact that different cells perform the same operation.

The SA shown in figure 1.a cannot be directly executed using the available hardware because, for instance, it is assumed that cell 2 performs each one of its operations one cycle after cell 1. The original SA can be adapted to the hardware in an automatic way, by using temporal and spatial transformations [11].

Temporal transformation

A temporal transformation permits to modify the cycle in which every cell performs its operations. In our case, this transformation is used to obtain a new SA A', in which the constraints imposed by the hardware are preserved.

The temporal transformation consists in introducing new delays between cells. These delays model the time required by every cell to perform its operations. Specifically, this transformation must introduce a delay of, at least, $l(i,j)$ cycles between cells j and i.

The required temporal transformation can be done by slowdown and retiming. Slowdown allows to introduce new delays in the SA. These delays can be adequately distributed by retiming [5].

Applying these transformations to our example, it is possible to obtain the SA A' shown in figure 2.a. In this figure, dots represent those delays introduced to model the hardware constrains. Black rectangles represent extra delays introduced to synchronize the whole SA. The value beside the North link in every cell is the number of cycles before the first valid data item is received by the cell through that link. From that cycle on, data items are received one every 9 cycles. In [11] we describe how to apply retiming and slowdown in order to obtain that SA.

As a result of applying slowdown, the slow of the SA has increased to 9. SA A' performs the same computations than the original one but requires more cycles.

At this point, it is important to note that equivalent 9-slow SAs can be obtained just moving extra delays. As an example, an equivalent SA can be obtained by moving the extra delay between cells 2 and 1 of SA in figure 2.a to the link in the opposite direction (and delaying one cycle the input data sequences to cells 2 to 6).

Spatial transformation

SA A' obtained through the temporal transformation can be directly executed using the available hardware. However, the hardware utilization will be very low because each cell performs only one valid operation every 9 cycles. The efficiency of the SA can be improved by applying a spatial transformation.

A spatial transformation permits to redistribute operations among cells. Nevertheless, each operation is performed in the same cycle than in the original SA. The result of the spatial transformation is a new SA A^*, which performs the same computation, in the same number of cycles but requiring less cells.

We propose the use of coalescing as spatial transformation. Each cell of A^* will perform those operations assigned to a set of adjacent cells of the original SA. Figure 2.a shows an example of the application of coalescing. Specifically, cells 1 and 2 of A' have been assigned to cell 1 of A^*. Figure 2.a also shows the utilization of the 3 stages of the PFU used to implement cell 1 of A^*. Stages used during the computation of OP_i have been marked with value i. Operations OP_1 and OP_2 can be executed without conflicts following the timing established by A'.

However, it is possible to increase the utilization of the PFU by selecting adequately the location of extra delays as it is shown in figure 2.b. In this case, coalescing has been applied to a SA equivalent to that in figure 2.a, obtained by moving two extra delays. This modification permits to execute OP_1, OP_2 and OP_3 in the same PFU.

There are two other ways to improve the utilization of the PFU: by increasing the slow of A' and by modifying the shape of the reservation tables associated with the operations of the SA. In our example, it is easy to prove that increasing the slow to 10 it is possible to execute OP_1, OP_2, OP_3 and OP_4 in the same PFU without conflicts. However, an increase of the slow implies always an increase of the number of cycles required to execute the SA. On the other hand, as we will see later, modifying RT_1 and RT_4 as shown in figure 3.b it is possible to execute OP_1, OP_2, OP_3 and OP_4 in the same PFU, without increasing the slow.

In [11] we propose a model for SAs and we formalize slowdown, retiming and coalescing so that these transformations can be applied automatically. Moreover, we propose an algorithm to determine the location of extra delays in order to minimize the computation time and maximize the hardware utilization. This algorithm does not consider the possibility of modifying the reservation tables.

In this paper we complement the work done in [11]. Specifically, in section 3 we describe how the delay insertion technique can be applied to our problem. In section 4 we propose an algorithm to obtain the structure and control of the PEs required to execute the SA. In section 5 we extend our algorithms to non time homogeneous SAs, in which cells perform different operations during the computation. Finally, in section 6 we present a concrete example consisting in the design of a SAP to perform the QR decomposition of a matrix by Givens rotations.

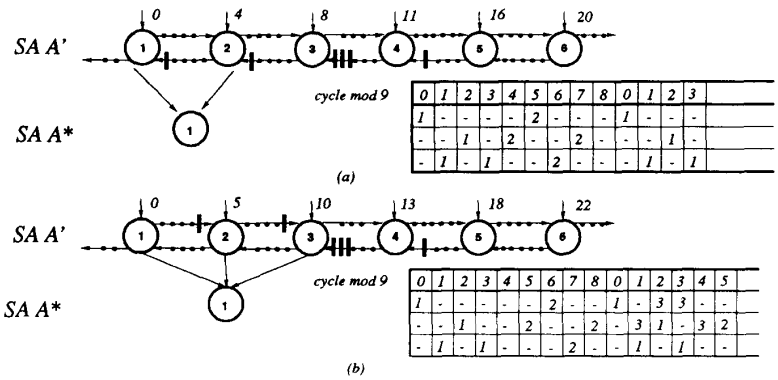

Figure 2: *(a) Example of temporal and spatial transformation of the SA shown in figure 1.a. The table shows the ocupation of the PFU stages if cells 1 and 2 of A' are implemented by the same cell of A*. (b) A new SA A' obtained by moving extra delays. This modification permits to improve the utilization of the PFU used to implement cell 1 of A*.*

3. IMPROVING THE UTILIZATION BY DELAY INSERTION

The delay insertion technique, proposed in [8], permits to modify the reservation tables of one or several operations in order to avoid conflicts in the use of the PFU stages. As a result, the throughput of the PFU is improved. In this section we describe how the delay insertion technique can be applied to our problem.

A reservation table can be modified by delay insertion. Delaying n cycles the computation performed by a given stage is represented by moving the mark associated with this computation n columns right in the reservation table. In [8] it is assumed that there is not information about dependences between stages. In this case, in order to preserve possible dependences, any mark placed on the right of the moved one must be also move n columns right.

The modification of RT_X is parametrized by values $d_{m,n}(X)$. Value $d_{m,n}(X)$ is defined as the number of columns mark in $RT_X(m,n)$ must be moved to the right relative to marks on its left side. After the modification, the new position of mark $RT_X(m,n)$ will be $RT_X(m,n')$, where:

$$n' = n + d_{m,n}(X) + \sum_{j=1}^{n-1}\left(\max_{i=1}^{p} d_{i,j}(X)\right)$$

being p the number of stages of the PFU.

In [8] it is described how to modify the reservation tables associated with a set of operations to allow these operations to be initiated in the same PFU, without conflicts and following a given initiation pattern. The considered initiation pattern is an initiation cycle. An initiation cycle is represented by means of a tuple. Each element of this tuple represents the type of operation to be initiated and the number of cycles separating this initiation from the previous one. The tuple has the following form:

$$\langle a_r, \ldots, b_s, c_t, \ldots, d_u \rangle$$

This expression indicates that OP_t will be initiated c cycles after OP_s, and OP_r will be initiated a cycles after OP_u. In the example shown in figure 2.b, the cell obtained by coalescing cells 1, 2 and 3 executes its operations following the initiation cycle described by the tuple:

$$\langle 4_1, 1_3, 4_2 \rangle$$

In order to ilustrate how the delay insertion technique can be applied to our problem, we focus on the scheduling of operations for cell 1 of SA A^*.

First, it is necessary to determine how many operations of the original SA can be assigned to cell 1 of A^*. In order to do that, we must take into account that any cell of A^* repeats its operations in a periodic way, every k' cycles, where k' is the slow of A'. So, none of the stages of the PFUs can be used more that k' cycles in each period. This fact determines an upper boundary to the number of operations that can be assigned to the same PFU. Specifically, it is possible to assign, at most, the first s operations to cell 1 of A^* if, for any of the stages of the PFU, the number of marks in the reservation tables associated to these s operations is not greater than k'. This condition can be expressed as follows:

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

where $MU(i,k)$ is the number of marks in row k of RT_i. In our example, cell 1 of A^* can execute, at most, the first 4 operations of A'.

Now, it is necessary to determine a possible initiation cycle for the selected operations. In our case, due to the fact that the location of extra delays is not unique, there is a number of possible initiation cycles. As an example, in the SA shown in figure 2.a, the initiation cycle for the first 4 operations is:

$$\langle 1_1, 2_4, 2_2, 4_3 \rangle$$

However, the initiation cycle for these operations in the equivalent SA shown in figure 2.b, is:

$$\langle 4_1, 1_3, 3_4, 1_2 \rangle$$

Both are valid initiation cycles for the first 4 operations. In general, any valid delay distribution represents a valid initiation cycle. A delay distribution is valid if the delay associated with link between cells j and i in A' is enough to model the time required by cell j to compute the value to be sent to cell i. In the case of 1D SA with data contraflow, a valid delay distribution must satisfy the following condition:

$$r'(i+1,i) \ge l(i+1,i) \qquad and \qquad r'(i,i+1) \ge l(i,i+1) \qquad i \in [1..n-1]$$

where $r'(i,j)$ is the delay associated with the link between cells j and i, and n is the number of cells.

Moreover, because A' is k'-slow, a valid delay distribution must also satisfy the following condition:

$$r'(i+1,i) + r'(i,i+1) \le k' \qquad i \in [1..n-1]$$

In our example, there are 16 valid initiation cycles for the first 4 operations.

Finaly, we must determine if it is possible to initiate the operations in the same PFU following one of the valid initiation cycles. In order to determine if it is possible to initiate the operations following a given initiation cycle, the initiation interval sets $G_{X,Y} \bmod c$ must be computed. $G_{X,Y} \bmod c$ contains all the intervals between initiations of OP_Y and initiations of OP_X, in this order, following the initiation cycle. Value c is the period, in cycles, of the initiation cycle. In our case $c=k'$.

Now, the allowable usage interval sets $H_{X,Y} \bmod c$ as obtained as follows:

$$H_{X,Y} \bmod c = Z_c - G_{X,Y} \bmod c$$

where Z_c is the set of integers module c.

This information permits to determine if the initiation cycle is possible. In order to do that, the reservation tables associated with the s operations being studied are overlapped, forming a single table RT. The initiation cycle is possible if the following condition is satisfied:

condition (a)

Any couple of marks $RT_X(a,b)$ and $RT_Y(a,c)$ from RT must be separated by a number n of columns, where:

$$n = c - b \in H_{X,Y} \bmod c$$

If the reservation tables do not fulfil this condition it is necessary to modify them. The values $d_{m,n}(X)$ that parametrize the required modification can be obtained by a branch and bound algorithm, subjected to condition (a). However, in our case, because there are dependences between operations involved in the initiation cycle, an extra condition must be satisfied.

condition (b)

As a result of the modification of RT_i, values $l(i+1,i)$ and $l(i-1,i)$ can have been changed. The new values will be noted by $l'(i+1,i)$ and $l'(i-1,i)$. This modification should not force an increase of the delay between cell i and its neighbours. So:

$$r'(i+1,i) \geq l'(i+1,i) \qquad and \qquad r'(i,i+1) \geq l'(i,i+1) \qquad\qquad i \in [1..n-1]$$

If condition (b) is not satisfied, OP_{i+1} can not be initiated $r'(i+1,i)$ cycles after OP_i, as it is established by the initiation cycle.

So, in our case, the branch and bound algorithm that finds the values $d_{m,n}(X)$ is subjected to conditions (a) and (b). If it is not possible to find a set of values which satisfy these conditions for a given initiation cycle, the procedure is repeated using one of the alternative valid initiation cycles.

In our example, it is possible to initiate OP_1, OP_2, OP_3 and OP_4 in the same PFU without conflict, as it is shown in figure 3.a, following the initiation cycle represented by the tuple:

$$\langle 3_1, 1_3, 4_2, 1_4 \rangle$$

To do that, it is necessary to modify RT_1 and RT_4 as shown in figure 3.b. The values which parametrize this modification are:

$$d_{2,3}(1) = 1 \quad and \quad d_{2,5}(4) = 1$$

This modification implies an increase in the values of $l(2,1)$ and $l(5,4)$. Due to this fact, in figure 5, some extra delays (rectangles) have been replaced by delays associated with the PFU (dots).

To conclude this section, it is worth remarking that, if several cells of the SA perform the same operation and the reservation tables associated with these operations are equal, then it is desirable that, after inserting the required delays, the reservation tables for these operations remain equal. This fact facilitates the design and control of the PFU. This additional restriction can be easily considered in the branch and bound algorithm used to find values $d_{m,n}(X)$.

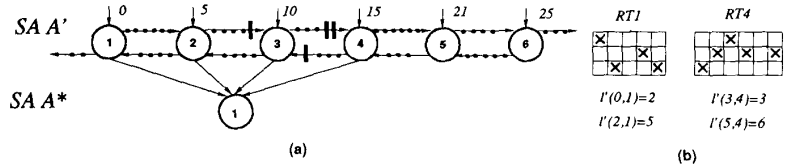

Figure 3: *Increasing the utilization of the PFU by delay insertion: (a) Cell 1 of A* obtained by coalescing cells 1, 2, 3 and 4 of SA A'. (b) Modified reservation tables for OP_1 and OP_4.*

4. AUTOMATIC DESIGN OF PROCESSING ELEMENTS

The procedure described in the previous section permits to determine how many adjacent cells of A' are assigned to every cell of A^*. The initiation cycle for each cell of A^* and the final shape of the reservation tables are also obtained. In this section we propose an algorithm that permits to obtain the structure and control of the PEs required to execute SA A^*. Again, we concentrate on the case of 1D SA with data contraflow. Similar algorithms can be derived for other kinds of SAs.

In order to ilustrate the algorithm we use the example shown in figure 3.a. Specifically, we will demonstrate how to obtain the structure and control of PE_1 responsible for executing OP_1, OP_2, OP_3 and OP_4, with the initiation cycle $\langle 3_1, 1_3, 4_2, 1_4 \rangle$

The delay $r'(i+1,i)$ which separates initiations of OP_i and OP_{i+1}, is decomposed into two parts:

$$r'(i+1,i) = l'(i+1,i) \quad + \quad re'(i+1,i)$$

Value $l'(i+1,i)$ corresponds to the delay established by the used hardware, once the reservation tables have been modified (dots in the figures). Value $re'(i+1,i)$ corresponds to the extra delays (black rectangles). Analogously:

$$r'(i-1,i) = l'(i-1,i) \quad + \quad re'(i-1,i)$$

We define $T(i)$ has the number of delays in the path from the cell which receives the first value, from the outside, to cell i. In our case, we assume that the first value enters the SA through cell 1. So:

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

In the SA shown in figure 3.a we have: $T(1)=0$, $T(2)=5$, $T(3)=10$, etc.

The general structure of PE_1 (and any of the remaining PEs required to implement the cells of A^*) is shown in figure 4.a. PFU_1 must be able to perform operations OP_1,..,OP_n assigned to cell 1 of A^*, as specified in their reservation tables. The results of OP_i are obtained through outputs $o_l^{T(i) \bmod k'}$ and $o_r^{T(i) \bmod k'}$, where k' is the slow of A'. In our example, the results of OP_3 are obtained through o_r^1 and o_l^1.

Values involved in every operation enter the PFU through N, i_r, and i_l. The last two values come from multiplexors M_r and M_l, which are controlled using a module k' counter, initialized to zero. This counter is also used to determine the operation to be initiated every cycle. Specifically, the PFU must initiate OP_i when the counter gets the value $j=T(i) \bmod k'$.

Operands for OP_i are taken from input N and from input $T(i) \bmod k'$ of M_l and M_r. The operand taken from M_l is one of the results of OP_{i-1}, that was obtained through $o_r^{T(i-1) \bmod k'}$. This value has been delayed $re'(i,i-1)$ cycles. Analogously, the value taken from M_r is one of the results of OP_{i+1}, obtained through $o_l^{T(i+1) \bmod k'}$. This value has been delayed $re'(i,i+1)$ cycles.

Finally, the link from EP_2 to EP_1 is connected to input $T(s) \bmod k'$ of M_r, with a delay of $re'(s,s+1)$ cycles. The left input of PE_1 is connected to input $T(1) \bmod k'$ of M_l.

Note that some outputs of the PFU and some inputs of the multiplexors are not used and appear just to allow the formulation of the algorithm.

Applying the algorithm proposed in this section to the SA shown in figure 3.a we obtain a PE_1 with the structure and control shown in figure 4.b.

5. NON TIME HOMOGENEOUS SYSTOLIC ALGORITHMS

The techniques proposed in the previous sections apply only to time homogeneous SAs, in which every cell performs always the same operation. However, many SAs are non time homogeneous SAs. In [6] some examples of this kind of SAs can be found. In this section we describe how the proposed techniques can be modified in order to apply them to non time homogeneous SAs.

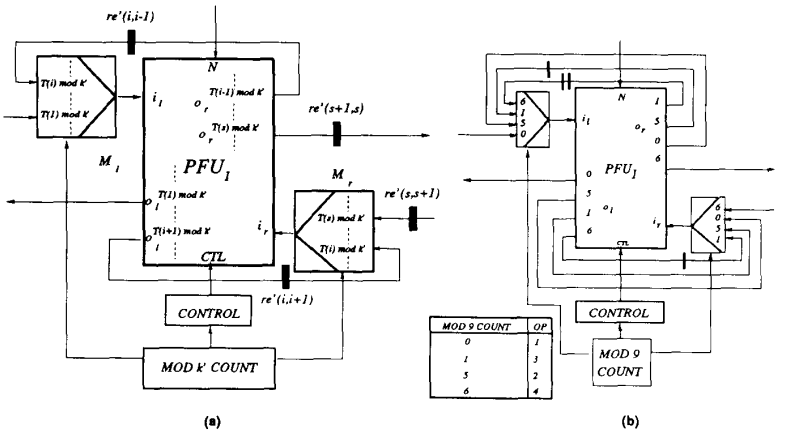

Figure 4: *(a) General structure of each cell of A* obtained by coalescing. (b) PE 1 obtained to implement cell 1 of A* (figure 3.a).*

Let $OP_{i,1}$.. $OP_{i,ni}$ be the *ni* different operations performed by cell *i* of the original SA during the computation. Each one of these cells receives a control signal that we will call c_i. This signal determines the type of operation to be performed by cell *i* in every cycle. Let $RT_{i,j}$ be the reservation table associated with $OP_{i,j}$. Let $l_k(i,j)$ be the number of cycles required to obtain, by performing $OP_{j,k}$, a value required in cell *i*.

In order to schedule operations for every PFU, we associate with each cell *i* a reservation table that covers the requirements of any of the operations performed by this cell. This table, that will be named RT_i, is obtained by ORing the *ni* reservation tables associated with cell *i*:

$$RT_i = OR(RT_{i,k}) \qquad k \in [1..ni]$$

Moreover, we use as value *l(j,i)* that associated with the slowest operation:

$$l(j,i) = \max_{k=1}^{ni} l_k(j,i)$$

Now, the techniques described in section 3 can be applied directly using reservation tables RT_i and values $l(i,j)$.

The proposed strategy is, at some extent, conservative, specially if the reservation tables associated with a given cell are significatively different. In these cases, the knowledge of the behaviour of each cell during the computation permits more efficient schedules, at the expense of an importat complication of the proposed algorithms. However, in practical cases the different operations apearing in a SA can be performed in such a way that the associated reservation tables are very similar. In these cases, the strategy proposed in this section obtains efficient results.

The algorithm proposed in section 4 to obtain the structure and control of every PE must be modified in the case of non time homogeneous SAs. The value $r'(i+1,i)$ is still the number of cycles separating the initiation of operations in cells *i* and *i+1*. However, the number of cycles required to perform an operation in cell *i* depends now on the type of operation to be performed. Therefore, the extra delay between cells *i* and *i+1* is not fixed and depends also on the operation performed in cell *i*.

In general, when cell i performs $OP_{i,k}$, the extra delays for the results of this operation are:

$$re'_k(i+1,i) = r'(i+1,i) - l'_k(i+1,i) \quad and \quad re'_k(i-1,i) = r'(i-1,i) - l'_k(i-1,i) \qquad k \in [1..ni]$$

The delay between cells i and $i+1$ is implemented by associating a fixed delay with the link between these cells and complementing this delay, in cell i, with an extra delay which depends on the type of operation performed by cell i. The fixed delay is:

$$re'(i+1,i) = \min_{k=1}^{ni} re'_k(i+1,i) \quad and \quad re'(i-1,i) = \min_{k=1}^{ni} re'_k(i-1,i) \qquad i \in [1..n]$$

In order to construct every PE we use a PFU slightly different to that shown in figure 4.a. In this new PFU, the results of $OP_{i,k}$ are obtained through outputs $o_l^{T(i) \bmod k',k}$ and $o_r^{T(i) \bmod k',k}$.

Output $o_r^{T(i) \bmod k',k}$ ($k \in [1..n_i]$) is connected to input k-1 of a multiplexor, that will be named $M_r^{T(i) \bmod k'}$. The output of this multiplexor is $o_r^{T(i) \bmod k'}$.

Output $o_l^{T(i) \bmod k',k}$ ($k \in [1..n_i]$) is connected to input k-1 of a multiplexor, that will be named $M_l^{T(i) \bmod k'}$. The output of this multiplexor is $o_l^{T(i) \bmod k'}$.

These multiplexors permit to apply an extra delay to values computed in the PFU. Specifically, a delay of $re'(i+1,i)$ cycles is associated with output $o_r^{T(i) \bmod k'}$ and a delay of $re'_k(i+1,i)$-$re'(i+1,i)$ cycles is associated with output $o_r^{T(i) \bmod k',k}$. Signal $c_r^{T(i) \bmod k'}$ must take the value k in cycle t if the PFU initiated $OP_{i,k}$ in cycle t-($r'(i+1,i)$-$re'(i+1,i)$). One of the results of this operation was obtained, $l'_k(i+1,i)$ cycles after its initiation, through $o_r^{T(i) \bmod k',k}$, it was delayed $re'_k(i+1,i)$-$re'(i+1,i)$ cycles and routed to output $o_r^{T(i) \bmod k',k}$, where it will be delayed again $re'(i+1,i)$ cycles. As a result, this value will reach its destination $r'(i+1,i)$ cycles after the initiation of $OP_{i,k}$. The delays and control for links in the opposite direction are defined in a similar way.

Signals c_l^i and c_r^i, which control multiplexors M_l^i and M_r^i are generated, together with signal *CTL*, from a module k' counter and from signals c'_i, $i \in [1..s]$. Remember that signal c_i is a control sequence that indicates the type of operation to be performed by cell i of the original SA in every cycle. Signal c'_i is the equivalent control sequence for A', obtained by applying slowdown and retiming.

6. IMPLEMENTATION OF A SYSTOLIC ALGORITHM FOR QR DECOMPOSITION

In this section we present a specific example of application of the techniques described in this paper. Specifically, we describe an efficient implementation in hardware on a non time homogeneous 1D SA with data contraflow for QR decomposition proposed in [10]. The decomposition is done by Givens rotations. Figure 5 shows the structure of this SA as well as the operations performed by every cell. Cell 1 performs OP_a when signal c_1 takes value a and OP_b when c_1 takes value b. The rest of cells perform always OP_b. So, for this SA we have: $OP_{11}=OP_a$, $OP_{12}=OP_b$ and $OP_i = OP_b$ $i \in [2..n]$.

In this SA every cell sends to its right neighbour a pair of values (c', s'). We consider that there is only one link between two cells in each direction and that values c' and s' are sent as a single data item. On the other hand, besides the values sent to the neighbour cells, every cell computes also a value x' that is sent to the outside. This difference in comparison to structure shown in figure 1 does not affect the application of the techniques described in previous sections.

SA in figure 5 has an arbitrary number of cells. In [10], a possible partitioning scheme is also proposed to obtain the QR decomposition of a matrix with any size. The I/O data sequences are not shown in figure 5 because their structure does not affect the design procedure for the cells of the new SA.

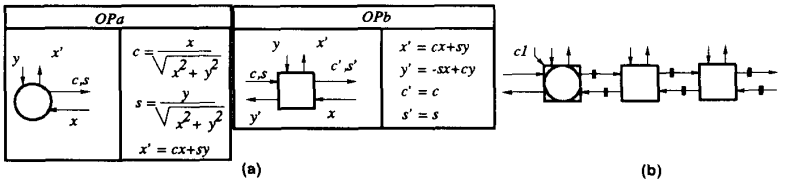

Figure 5: *(a) Operations and (b) structure of a 1D SA with data contraflow for QR decomposition.*

Suppose that pipelined multipliers and adders are used to execute OP_a and OP_b. Specifically, an inverse square root can be performed using only one multiplier and one adder as follows:

$$R1 = 0.5*R0*(3.0 - A*R0*R0) \qquad and \qquad \frac{1}{\sqrt{A}} \approx 0.5*R1*(3.0 - A*R1*R1)$$

where $R0$ is an approximation of value $1/\sqrt{A}$ obtained by indexing a lookup table with some bits of A [13].

Figure 6 shows two reservation tables to perform OP_a and OP_b using one multiplier and one adder, both pipelined into 3 stages. Therefore, in this case, we have:

$$l_a(2,1) = 31,\ l_b(2,1) = 1,\ l_b(0,1) = 7$$
$$l(i+1,i) = 1\ ,\ l(i-1,i) = 7 \qquad i > 1$$

For the sake of simplicity, we assume that the time required to access the table of approximations can be neglected. Note that, when a cell performs OP_b, the value received from the left is directly sent to the right, without modification. In order to avoid global communications, we associate a delay of 1 cycle with this operation. Using the notation proposed in section 5, $RT_{11} = RT_a$, $RT_{12} = RT_b$ and $RT_i = RT_b$ *(i>1)*.

ORing reservation tables RT_{11} and RT_{12} we obtain RT_1. In this example, the minimum possible slow for A' is $k' = 38$. This value is determined by the time required by cells 1 and 2 to compute the values to be sent to each other.

For a slow of $k' = 38$, the maximum number of operations that can be assigned to cell 1 of A^* is 7. However, after applying the techniques proposed in section 3, only the first 4 operations can be assigned to cell 1 of A^*. Moreover, in order to do that it is necessary to modify RT_1 by delaying 4 cycles the last multiplication and addition. This modification does not affect values $l(i,j)$. OP_2 must be initiated 31 cycles after OP_1 (any of the operations performed by cell 1, OP_{11} or OP_{12}). OP_3 must be initiated 19 cycles after OP_2, and OP_4 must be initiated 10 cycles after OP_3. So, the delays between initiations are:

$$r'(2,1) = 31 \quad r'(3,2) = 19 \quad r'(4,3) = 10 \quad r'(1,2) = 7 \quad r'(2,3) = 19 \quad r'(3,4) = 28$$

and the corresponding initiation cycle is $\langle 7_1, 12_3, 10_4, 9_2 \rangle$

A maximum of 9 operations can be assigned to each one of the rest of cells. In this case, it is possible to execute these operations without any modification of the reservation table RT_b. The initiation cycle for cell 2 of A^* is $\langle 6_5, 4_6, 4_7, 4_8, 4_9, 4_{10}, 4_{11}, 4_{12}, 4_{13} \rangle$. The rest of cells have the same initiation cycle with different operation numbers (all these operations are OP_b). We focus now on the design of PE_1 required to execute cell 1 of A^*.

The first valid operation is performed in cell 1 of SA A'. So, in this case we have: $T(1) = 0$, $T(2) = 31$, $T(3) = 50$, $T(4) = 60$. In order to obtain the structure and control of PE_1, we have to design a PFU able to execute OP_a and OP_b as indicated in the associated reservation tables. The external aspect of that PFU can be seen in figure 7. When signal *CTL* takes values *a* the PFU initiates OP_a, and when *CTL* takes value *b* the PFU initiales OP_b. Due to the presence of non time homogeneity, we must distinguish between signals c'_a and s'_a, used to output values c' and s' obtained by OP_a, and signals c'_b and s'_b, used to output values c' and s' obtained by OP_b.

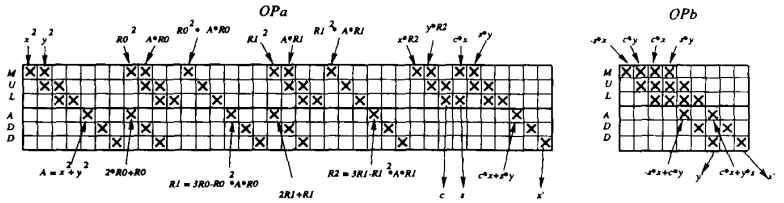

***Figure 6:** Reservation tables to implement OP_a and OP_b using one multiplier and one adder, both pipelined into 3 stages.*

Applying expresions presented in section 5, we obtain the delays to be associated with the feedback links appearing in PE_1:

$$re'_a(2,1) = r'(2,1) - l'_a(2,1) = 31 - 31 = 0$$

$$re'_b(2,1) = 31, re'(3,2) = 19, re'(4,3) = 10, re'(5,4) = 1, re'(1,2) = 0, re'(2,3) = 12, re'(3,4) = 21$$

Figure 7 shows the internal structure and control of PE_1. This PE is controlled by a module 38 counter, initialized to zero, and signal c'_1. When signal *CNT* takes value 0 then the PFU must initiate one of the operations that, in the original SA, performs cell 1 ($T(1) \bmod 38 = 0$). This operation is OP_a or OP_b depending on the value of signal c'_1. The PFU receives the values involved in the computation through inputs 0 of the multiplexors and input *N*.

When *CNT* takes value 31 the PFU initiates one operation OP_b. This operation corresponds to one of the operations performed by cell 2 of the original SA ($T(2) \bmod 38 = 31$). In this case, one of the values involved in the computation is a result of an operation assigned to cell 1 of the original SA. This value has been previously computed by the PFU, and multiplexor $M_{r,0}$ is used to determine the source of this value, depening on the type of operation performed to obtain it. Analogously, OP_b is initiated when *CNT* takes value 12 or value 22. The design of the remaining PEs is quite easy and it is not described here.

7. CONCLUSIONS

During the last ten years, a lot of attention has been paid to the SA automatic design problem. A number of methodologies have been proposed, although in general, the SAs obtained cannot be directly and efficiently executed. Aspects including limitations in the number of PEs, fault tolerance of communication bandwidth limitations can hardly be taken into account if the SAs are obtained through automatic synthesis techniques. For this reason, SAs transformation techniques can play a role of major importance in the design of algorithmically specialized processors.

The method proposed in this paper permits to transform a SA so that it can be efficiently executed using PFUs. The method is based on two temporal transformation (slowdown and retiming) and one spatial transformation (coalescing). The temporal transformations permit to modify the SA in such a way that dependences established by the PFU are preserved. The spatial transformation permits to improve the hardware utilization. The method has been applied to 1D SAs with data contraflow. This type of SAs are specially suitable for hardware implementation. However, the use of PFU is difficult due to the presence of feedback cycles.

To demonstrate the effectiveness of the method, we have described an efficient implementation of a non time homogeneous SA with data contraflow for QR decomposition. The structure and control of each one of the PEs is obtained automatically. Similar results can not be obtained through any other design methodology. The proposed method can be applied to any kind of 1D SA as well as 2D SA and it can serve as the basis for an automatic tool oriented to the design of SAPs.

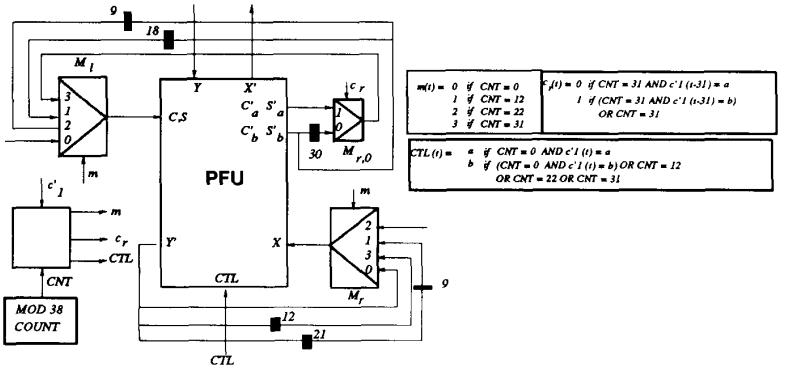

Figure 7: *Internal structure and control for PE 1 which executes cells 1, 2, 3 and 4 of the original SA.*

8. REFERENCES

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

[2] H.T. Kung y 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.).

[3] 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

[4] 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.

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

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

[7] J.J. Navarro et al. "LU Decomposition with No Size-Restriction Using a One-Dimensional Systolic Array Processor," *Proc. Second Int'l Conf. Supercomputing*, May 1987, Vol. 3, p. 218.

[8] J.H. Patel and E.S. Davidson, "Improving the Throughput of a Pipeline by Insertion of Delays," *Proc. 3th Annual Int'l Symp. on Computer Architecture*, 1976, pp. 159-164.

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

[10] 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.

[11] M. Valero-Garcia, J.J. Navarro, J.M. LLaberia y M. Valero, "Systematic Hardware Adaptation of Systolic Algorithms," *Proc. 16th Annual Int' l Symp. on Computer Architecture* 1989, pp. 96-104.

[12] 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.

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