Abstract—The solution of tridiagonal systems is a topic of great interest in many areas of numerical analysis. Several algorithms have recently been proposed for solving tridiagonal systems based on the Divide and Conquer (DC) strategy. In this work we propose an unified parallel architecture for DC algorithms which present the data flows of the Successive Doubling, Recursive Doubling and Parallel Cyclic Reduction methods. The architecture is based in the perfect unshuffle permutation, which transforms these data flows into a constant geometry one. The partition of the data arises in a natural manner, giving way to a systolic data flow with a wired control section. We conclude that the constant geometry Cyclic Reduction architecture is the most appropriate one for solving tridiagonal systems and, from the point of view of integration in VLSI technology, is the one which uses the least amount of area and the smallest number of pins.

Index Terms—Divide and conquer, tridiagonal system solver, parallel architecture, perfect unshuffle permutation, recursive doubling, parallel cyclic reduction and VLSI design.

I. INTRODUCTION

TRIDIAGONAL SYSTEMS (TS) are used in several areas of science and engineering. They arise when solving partial differential equation systems using FACR (Fourier Analysis-Cyclic Reduction), Successive Line methods, Over-Relaxation methods, etc. [1], [9], [6]. The most efficient algorithm for solving TS systems in single processor is Gaussian Elimination (GE). Unfortunately, the GE algorithm cannot be parallelized efficiently. The alternative is to define algorithms which exploit the inherent parallelism of the TS systems. The classic parallel algorithms for solving TS systems are Stone’s [10] recursive doubling method (RD) and Hockney and Jesshope’s [7] cyclic reduction method (CR). Both algorithms are based on the Divide and Conquer strategy (DC).

The basic idea underlying the DC strategy is to solve a problem of a given size by dividing it into similar problems of smaller sizes, solving them and combine their solutions for obtaining the solution of the original problem. Usually the same technique is applied to each subproblem, making the procedure recursive. The DC strategy has been the most used in the formulation of TS algorithms because the data flow analysis has an implicit parallel architectural model, i.e., distributed memory system with one processor per equation. In general, the algorithms proposed consist of three phases. In the first phase, (elimination), each processor performs some transformations over the equations it is assigned and obtains one or more equations. In the second phase (reduction), the tridiagonal system made up of these equations is solved by all the processors using an interconnection network. In the third phase (substitution), the solution obtained is sent to each processor which completes the solution of the problem.

There has been much recent effort dedicated to the development of parallel TS algorithms which exploit the inherent parallelism of the DC strategy. We can establish three directions in the design of parallel TS algorithms based on the DC strategy: extension of the parallelism of the RD and CR methods by means of the Parallel Prefix (PP) [5] and Parallel CR (PARACR) [7] algorithms, respectively; design of new DC algorithms [15], [18]; parallelization and or removal of the three phases mentioned above. Algorithms based on the RD method are those proposed by Egecioglu et al. [5] (hypercube network) and Lin and Chung [10] (exchange-perfect unshuffle network). The TS algorithm proposed by Wang and Mou [18] presents the Successive Doubling (SD) data flow, which is similar to that of the radix 2 decimation in time FFT algorithm. They also propose the GEDC algorithm [18] implementing the Gaussian Elimination (GE) in phase one, using the DC algorithm in phase two. Recently, Spaletta and Evans [15] have presented a recursive decoupling method based TS algorithm which uses Sherman-Morrison formula to calculate the inverse of coefficients matrix.

From the point of view of communications, the TS algorithms referenced can be classified into the following groups: all to all broadcast [4], [17]; binary tree [8], [12], [15]; RD data flow [2], [5], [10]; SD data flow [11], [18]; CR data flow [6], [8]; PARACR data flow [17]; ring [3] and linear data flow [14], [17]. Krechel et al. [8] use an exchange network (CR in phase one) and a binary tree (phase two) so that phase three is carried out without communications. Reale [14] proposes a variant of the CR algorithm that uses a linear array in phases one and three. Müller and Scheerer [12] propose a method whose phase two is solved in parallel using a binary tree. The algorithm presented by Spaletta and Evans [15] has only two phases, the second one having a binary tree pattern. Sun et al. [17] propose three algorithms: Parallel Partition LU (PPT), which uses the LU decomposition in phases one and two, requiring an all to all broadcast among phases; Parallel Partition Hybrid (PPH), substitutes phases two and three by the RD algorithm; Parallel Diagonal Dominant (PDD), which only needs nearest neighbor communications (linear array). A parallel TS algorithm similar to PPH [17] was proposed by...
Bondeli [2], Cox and Knisely [4] perform a redundant partition of the TS system, where the processors implement a local CR, although the complete solution requires an all to all broadcast. Brugnano [3] presents a new parallel TS algorithm based on the sweep method, using a ring network. Finally, Johnson [9] has recently carried out an analysis of CR and GECR algorithms over different interconnection networks.

In this work we present an unified parallel architecture for TS algorithms with SD [18], RD [5] and PARACR [7] data flows. These algorithms are only, in general, stable (reliable) when applied to systems that can be solved without pivoting. We have unified these three flows of data by means of their transformation into a single constant geometry flow. This constant geometry is obtained by performing a perfect unshuffle permutation of the data generated in each of the log$_2 N$ phases of the algorithms ($N = 2^n$ is the number of equations in the TS system). The action carry out by the unshuffling of the data is to transform the original TS algorithm into a new one with the same data flow in all its stages. The resulting architecture is a column of $Q$ processors ($Q = 2^q$, with $q \leq n - 1$), where the partition of the data arises in a natural manner.

The rest of the work has been structured as follows. In Section II, we present three new constant geometry DC algorithms for solving TS systems. The design of the processor for the evaluation of these algorithms is presented in Section III. The processor column, based on the perfect unshuffle permutation is introduced in Section IV. Finally, in Section V, we evaluate the unified parallel/pipelined architecture proposed.

II. TRIDIAGONAL SYSTEM ALGORITHMS WITH CONSTANT GEOMETRY

A tridiagonal system is made up of equations of the type

$$E_i^{(0)} = a_i^{(0)} x_{i-1} + b_i^{(0)} x_i + c_i^{(0)} x_{i+1} = d_i^{(0)}$$

for $i = 0, \ldots, N - 1$, where $x_{-1} = x_N = 0$.

We consider systems with $N = 2^n$ equations and we will assume that the sequence $S$ of equations is distributed as a matrix $D$ of dimensions $2 \times 2^n$ so that the equation of index $i = [i_1, \ldots, i_2]$ is located in column $x = [i_1, \ldots, i_2]$, row $y = [i_1]$ of matrix $D$, where $i_1, \ldots, i_2$ means the binary representation for index $i$. We will call this distribution criterion $d$.

$$d[i_1, \ldots, i_2] = [i_1, \ldots, i_2][i_1].$$

The rows of $D$ are numbered 0 and 1 from top to bottom and the columns from right to left are numbered from 0 to $2^{n-1} - 1$.

In this section, we will briefly describe the SD [18], PP [5] and PARACR [7] algorithms and we will formulate a constant geometry version for each one of them. We now give the definitions of two operators which are particularized for the needs of this work.

**Definition 1:** A butterfly operator $B$ over a column of a matrix of dimension $a \times b$ is a function with $a$ inputs and $b$ outputs.

**Definition 2:** The perfect unshuffle operator $B$ transforms matrix $D$ into another matrix $D'$ of the same dimension. The element located in column $x = [i_1, \ldots, i_2]$, row $y = [i_1]$ of matrix $D$ is placed at location (column, row) $= ([i_1], [i_1, \ldots, i_2], [i_2])$ of $D'$.

$$\Gamma(x, y) = [i_1, i_1, \ldots, i_2][i_2].$$

The operator $\Gamma$ can also be applied to a sequence $S$. In this case, $\Gamma$ performs a cyclic rotation to the right, of order 1, in the binary representation of the index of each element in the sequence $S$.

**Lemma 1:** The distribution $d$ defined in (2) "commutes" with operator $\Gamma$.

**Proof:** In this paper, we will assume that all operators strings act from left to right

$$\Gamma d[i_1, \ldots, i_2] = d[i_1, \ldots, i_2] = [i_1, i_1, \ldots, i_2][i_2]$$

$$= \Gamma[i_1, \ldots, i_2][i_1] = d[i_1, \ldots, i_2].$$

As a consequence of Lemma 1, it is equivalent to apply operator $\Gamma$ to sequence $S$ and to matrix $D$. Furthermore, any distribution resulting from a cyclic shift of $d$ also commutes.

A. Successive Doubling Algorithm

Recently, Wang and Mou [18] have proposed a new algorithm based on the successive doubling method, that we will denote as SD, for solving TS systems. The SD algorithm is carried out in $n$ stages. In the $t$th stage ($1 \leq t \leq n = \log_2 N$), we start from equations $E_{st}^{(t-1)}$, $0 \leq t \leq N - 1$, whose unknown terms, that we will indicate by means of their subscripts, are $(s2^{t-1} - 1, i, (s + 1)2^{t-1})$, where $s = [i/2^{t-1}]$. For the sake of brevity, when we refer to equation $E_{st}^{(t-1)}$, we will use the underlined subscript, discarding the superscript of the stage when there is no danger of misinterpretation. The algorithm is applied to groups of three equations (trios). Each trio is associated to one equation $E_{st}^{(t-1)}$, we will refer to it by means subscript $i$ in bold face and in brackets $[i]^{(t-1)}$ and it is made up of the following equations

$$[i]^{(t-1)} = [s2^{t-1} - 1, i, (s + 1)2^{t-1} - 1].$$

Notice that not all the equations on a trio are necessarily distinct. Each trio $[i]^{(t-1)}$ of stage $t$ for which $s$ is even is combined with the trio $[i + 2^{t-1}]^{(t-1)}$ in order to produce the trios $[i]^{(t)}$ and $[i + 2^{t-1}]^{(t)}$ of the next stage. We will indicate as $B_{sd}$ the operator which performs this transformation:

$$\begin{bmatrix} \left[\frac{i}{i} \right]^{(t)} \\
\left[\frac{i}{i + 2^{t-1}} \right]^{(t)} \end{bmatrix} = B_{sd} \begin{bmatrix} [i]^{(t-1)} \\
[i + 2^{t-1}]^{(t-1)} \end{bmatrix}.$$
the variable \((s + 1)2^{t-1} - 1\) in the three equations of the trio \([t + 2^{t-1}]_{(t-1)}\). This process introduces the variables \(s2^{t-1} - 1\) and \((s + 1)2^{t-1}\) in these equations. We write as \( (s + 1)2^{t-1}' \), \((i + 2^{t-1})'\) and \((s + 2)2^{t-1} - 1)' the modified equations of the trio \([i + 2^{t-1}]_{(t-1)}\) after the first phase, by means of 16 arithmetic operations.

In the second phase, we use equation \((s + 1)2^{(t-1)}'\), whose unknowns are \(s2^{t-1}, (s + 1)2^{t-1}\) and \((s + 2)2^{t-1}\), in order to eliminate \((s + 1)2^{t-1}\) in equations \(s2^{t-1}, i\), \(i + 2^{t-1}\) and \((s + 2)2^{t-1} - 1\)'. This phase produces four equations with identical first and third unknowns, \(s2^{t-1} - 1\) and \((s + 2)2^{t-1}\) respectively and whose central unknowns are \(s2^{t-1}, i, i + 2^{t-1}\) and \((s + 2)2^{t-1} - 1\). As \(s\) is even, these four equations are \(s2^{t-1} - 1\), \(i\), \(i + 2^{t-1}\) and \((s + 2)2^{t-1} - 1\)' identical to the equations which make up the trios \(i\)' and \([i + 2^{t-1}]\). 25 arithmetic operations are carried out in this phase.

In the last stage of the algorithm, each trio \([i]^{0}\) contains equation \(E_{t}^{0}\) with the unknown terms \(-2^{n}, i, 2^{n}\). As \(x_{N} = x_{N} = 0\), a simple division permits the calculation of each unknown term \(x_{t}\). Fig. 1(a) shows the data flow of the algorithm for \(N = 8\). Each node (□) means a trio (see (4)), being \([i]^{0}, 0 \leq i \leq 7\), the input data trios. In each stage, four butterflies are computed. At the \(t\)th stage \((t = 1, 2, 3)\) the operator \(B_{sd}\) (see (5)) is applied to trios at distance of \(2^{t-1}\). Summarizing, the SD algorithm proposed by Wang and Mou [18] for solving TS systems presents a very regular data flow. Each data item consists in a trio of equations (twelve coefficients). Each butterfly produced by operator \(B_{sd}\) implies 41 arithmetic operations.

**Constant Geometry:** The SD algorithm admits a constant geometry version. It is based on the use of the perfect unshuffle operator for exploiting the fact that in the \(t\)th stage of the SD, pairs of terms at a distance of \(2^{t-1}\) are computed.

Theorem 1: The SD algorithm is equivalent to the following constant geometry algorithm, which we will name CGSD.

\[
\begin{bmatrix}
    y^{(0)}(\Gamma(2i)) \\
    y^{(0)}(\Gamma(2i + 1))
\end{bmatrix} = B_{sd} \begin{bmatrix}
    y^{(t-1)}(2i) \\
    y^{(t-1)}(2i + 1)
\end{bmatrix}
\]

\((t = 1, \ldots, n; i = 0, \ldots, N/2 - 1)\)

where \(y^{(0)}(i) = [i]^{0}\).

The proof of Theorem 1, given in the Appendix, consists in verifying that in each stage of both algorithms identical results are obtained, although in a different order. The CGSD algorithm is similar to the one developed by Pease [13] for the radix 2 time decimation FFT.

We will assume that in each stage \(t\) of the CGSD algorithm, the trios \(y^{t-1}(i)\) flow towards the processor as a matrix \(D_{t-1}\) of \(2^{n-1}\) columns with 2 elements each, following the distribution defined in (2):

\[
D_{t-1} = \begin{bmatrix}
    y(2^{n} - 2) \cdots y(2i) \cdots y(2) y(0) \\
    y(2^{n} - 1) \cdots y(2i + 1) \cdots y(3) y(1)
\end{bmatrix}
\]

In \(D_{t-1}\), we have suppressed the subscripts \(t - 1\) of each element. The trio with index \(i\) is located in column \([t_{a} \ldots t_{z}]\), row \([t_{a}]\) of the matrix. From the computational viewpoint, we can consider that the column indicates the cycle in which the trio will be processed and the row the bus through which the trio accesses the processing section. We will say that the index admits a (cycle, bus) interpretation.

In the execution of the \(t\)th stage of the CGSD algorithm we can differentiate the following two actions.

1) The butterfly operator \(B_{sd}\) is applied to each column of \(D_{t-1}\)

\[
B_{sd} \begin{bmatrix}
    y(2i) \\
    y(2i + 1)
\end{bmatrix} = \begin{bmatrix}
    z(2i) \\
    z(2i + 1)
\end{bmatrix}
\]

And thus, matrix \(D_{t-1}\) is transformed into matrix \(G_{t-1}\):

\[
G_{t-1} = \begin{bmatrix}
    z(2^{n} - 2) \cdots z(2i) \cdots z(2) z(0) \\
    z(2^{n} - 1) \cdots z(2i + 1) \cdots z(3) z(1)
\end{bmatrix}
\]

2) The perfect unshuffle operator is applied to \(G_{t-1}\) in order to obtain matrix \(D_{t}\).
Thus, the tth stage can be expressed as the application of the operator string $B_{sd}$ to matrix $D_{t-1}$. Given the fact that all the stages are identical, the algorithm CGSD consists in the application of $B_{sd} \Gamma^n$ to matrix $D_0$, that is,

$$ \text{CGSD} = (B_{sd} \Gamma)^n. $$

Fig. 1(b) shows the data flow of the CGSD algorithm for $N = 8$. It can be observed that all computation stages have the same flow because the output trios are unshuffled. In each stage we compute pairs of trios at a distance of one and the results are located in the same position for every stage. Each data item is a trio $[i](t-1)$ defined in (4) and in each node of the graph the butterfly operator $B_{sd}$ defined in (5) is applied. Observe that $y^{(i)}(\Gamma^t(i)) = [i]^t$, where $\Gamma^t$ means which $\Gamma$ is applied $t$ times.

In the case (b) all the stages are similar: $B_{sd}$ is applied to trios $y^{(i)}(i)$ at distance of 1 and the perfect unshuffle operator $\Pi$ permutes the resulting sequence to obtain trios $y^{(i)}(i)$.

### B. Recursive Doubling Algorithm

The recursive doubling method (RD) was proposed by Stone in order to introduce parallelism in solving linear recurrence equations [16]. Several algorithms exist for solving tridiagonal systems based on the RD method [5], [10]. Here, we will use the version by Egecioglu et al. [5], which is called Parallel Prefix (PP), and which we will now describe.

If we assume that $c_i \neq 0$, $0 \leq i \leq N - 1$, we can isolate $x_{i+1}$ in equation (1) and, by defining $\mu_i = -b_i/c_i$, $\rho_i = -a_i/c_i$ and $\sigma_i = d_i/c_i$, the following matrix expression is obtained

$$
\begin{bmatrix}
  x_i + 1 \\
  x_i \\
  1
\end{bmatrix} =
\begin{bmatrix}
  \mu_i & \rho_i & \sigma_i \\
  1 & 0 & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x_i \\
  x_i - 1 \\
  1
\end{bmatrix}
$$

or, more briefly, $X_{i+1} = P_iX_i$. Consequently,

$$ X_{i+1} = P_iP_{i-1} \cdots P_0X_0. $$

Therefore, solving the system is reduced to finding all the partial products of the matrices $P_i$, calculating $x_0$ using the last equation and substituting in order to obtain the rest of the unknown quantities. It has been observed that some numerical stability problems can arise in the use of recursive doubling algorithm when the size of the system is large.

The RD method is the most widely used one for performing the parallel calculation of all the partial products or prefixes of a succession of $N$ elements. It permits the calculation of all the prefixes in $[\log_2 N]$ stages and presents the data flow of Fig. 2(a). As can be observed, the data flow is very regular. Each node consists in a $3 \times 3$ matrix and the operation to be carried out in each node is the matrix product. The initial matrices $P_i$ (see (11)) have as last row $\{0, 0, 1\}$, a feature which is maintained throughout the product of them. Thus, in order to obtain the product matrix only six elements must be computed. That is, 20 arithmetic operations are carried out in each node. A node identified as $\square$ at $t$th stage means which its matrix is the same that in the previous stage. No computation is carried out in these nodes. In the other hand, a node identified as $\Box$ represents the product of its input matrices. At the $t$th stage ($t = 1, 2, 3$) a total of $8 - 2^{t-1}$ matrix products are computed and each node participates in two products. Input matrices to each node are at distance of $2^{t-1}$.

**Constant Geometry:** The RD algorithm admits a constant geometry formulation with characteristics similar to those of the CGSD algorithm.

**Theorem 2:** The RD algorithm is equivalent to the following algorithm with constant geometry and which we will name CGRD:

$$ y^{(i)}(\Gamma^t(i)) = \begin{cases} y^{(i-1)}(i), & \text{if } i = k \\ y^{(i-1)}(i-1) - y^{(i-1)}(i), & \text{otherwise}, \end{cases} $$

where $0 \leq k < 2^{t-1}$ and $y(i-1)y(i)$ is the product of two matrices, being $y^{(0)}(i) = P_i$.

The proof of Theorem 2 is given in the Appendix.

As in the case of the CGSD algorithm, we assume that in the $t$th stage the data flows towards the processor as a matrix $D_{t-1}$ (see (7)). The phases we must now consider in the execution of each stage of the CGRD algorithm are the following.
1) Matrix $D_{t-1}$ is transformed into matrix $F_{t-1}$:

$$
F_{t-1} = \begin{bmatrix}
    y(2^n - 3) \cdots y(2t - 1) \cdots y(1) & I \\
    y(2^n - 2) \cdots y(2) \cdots y(2) & y(0) \\
    y(2^n - 1) \cdots y(2t + 1) \cdots y(3) & y(1)
\end{bmatrix}
$$

where $I$ is the unit matrix for the matrix product. We will denote as $\xi_{rd}$ the operator which performs this first phase.

2) The butterfly operator $B_{rd}$ is carried out over each column of $F_{t-1}$:

$$
B_{rd} \begin{bmatrix}
    y(2i - 1) \\
    y(2i) \\
    y(2i + 1)
\end{bmatrix} = \left( \begin{array}{c}
    z(2i) \\
    z(2i + 1)
\end{array} \right)
$$

where $z(i)$ is equal to the right hand side of expression (13), i.e., $z(i) = y(i - 1)y(i)$ except for $i = k2^{n-t+1}$ in which $z(i) = y(i)$. Notice that operator $B_{rd}$ performs two matrix products over each column of matrix $F_{t-1}$, carrying out a total of 40 arithmetic operations. Matrix $F_{t-1}$ is transformed into matrix $G_{t-1}$ defined in (9).

3) We apply the perfect unshuffle operator to $G_{t-1}$ in order to obtain $D_t$.

Summarizing, each stage consists in the application to matrix $D_{t-1}$ of the operator string $\xi_{rd}B_{rd} \Gamma$ and the CCR algorithm consists in the application of $\xi_{rd}B_{rd} \Gamma$ $n$ times to the initial matrix $D_0$, that is

$$
\text{CCR} = (\xi_{rd}B_{rd} \Gamma)^n.
$$

Expression (13) evidences the geometry of the algorithm. Fig. 2(b) shows the data flow of the CCR algorithm for $N = 8$. In this case all stages have the same data flow, as a consequence of the perfect unshuffle permutation applied to the results $z^t(i)$ (see (15)) to obtain $y^t(i)$ (see (13)). This permutation forces the nodes to be located at distance one. The nodes of Fig. 2(b) have the same meaning that the nodes of Fig. 2(a). Note that a node of Fig. 2(a) and (b) represents an equation (see (12)) while a node of Fig. 1 (a) and (b) represents a trio of equations (see (4)).

C. Parallel Cyclic Reduction Algorithm

The PARACR algorithm by Hockney and Jesshope [7] consists of two stages: elimination and substitution. The elimination stage consists in obtaining in the $t$th stage equations

$$
E_i^t = a_i^t x_{i-2^t} + b_i^t x_i + c_i^t x_{i+2^t} = d_i^t
$$

for $i = 0, \ldots , N - 1, t = 1, \ldots , n$. We assume that if $i \pm 2^t$ is not in the $[0, N - 1]$ range, any unknown term with that subscript is 0 and the whole equation with that subscript would have $\{0, 0, 0, 0\}$ as coefficients (we will indicate those equations as $I$).

The $t$th stage consists in obtaining equations $E_i^t$ by performing over the equations with indices $i - 2^t$, $i$, and $i + 2^t$ the operation

$$
E_i^t = c_i^{(t-1)} E_{i-2^t}^{(t-1)} + E_i^{(t-1)} + a_i^{(t-1)} E_{i+2^t}^{(t-1)}
$$

where $c_i^{(t-1)} = (-a_i/b_{i-2^t-1}$), $a_i^{(t-1)} = (-c_i/b_{i+2^t+1}$). We have eliminated the superscripts $(t - 1)$ from the coefficients.

As the unknown terms in $E_{i-2^t}^{(t-1)}$ have subscripts $(i - 2^t, i - 2^t - 1, i)$, the first term in the right hand side of (18) eliminates $i - 2^t$ in $E_i^{(t-1)}$ and introduces the unknown term $i - 2^t$ in it. In the same way, the last term eliminates $i + 2^t - 1$ and introduces $i + 2^t$ in the equation $E_i^{(t-1)}$, from which $E_i^{(t)}$ results.

In the $n$th stage, last of the algorithm, we obtain equations $E_i^{(n)}$ with variables $i - 2^n$, $i$, $i + 2^n$ that, except for the central one, are 0. This permits the calculation of $x_i$ in each equation $E_i^{(n)}$.

The right-hand side of (18) will be abbreviated as $[E_{i-2^t}^{(t-1)}, E_i^{(t-1)} E_{i+2^t}^{(t-1)}]^{(t-1)}$.

The PARACR algorithm presents a regular data flow, shown in Fig. 3(a) for $N = 8$. Note that it is similar to the RD algorithm, but with two address flow. Each node is an equation which has four coefficients. The number of stages is $n$. There are four classes of nodes in Fig. 3(a). Nodes identified as $\square$ are initial equations $E_i^{(0)}$. Nodes marked as $\blacksquare$ represent the result of computing expression (18) for its three input equations, where 12 arithmetic operations are carried out. A
circle (shadowed triangle) node also computes equation (18) but forcing their upper (lower) input to the identity $I$ equation. Input equations $E_{t-1}^{t-1}(i)$ to each operation (18) are at distance of $2^{t-1}$.

An advantage of the parallel cyclic reduction algorithm is that if the TS system is sufficiently diagonally dominant, the elimination phase can be stopped before completion without loss of accuracy [7].

**Constant Geometry** As in the previous cases, a version with constant geometry can be formulated for the PARACR algorithm.

**Theorem 3:** The PARACR algorithm is equivalent to the constant geometry algorithm we will name CGCR:

$$y^{(t)}(\Gamma(i)) =
\begin{cases}
    [y[i+1], y[i]], & \text{if } i = k2^{n-t}+1, \\
    [y[i-1], y[i], y[i+1]], & \text{otherwise}, \\
    [y[i-1], y[i]], & \text{if } i = (k+1)2^{n-t}+1 - 1,
\end{cases}
$$

where $0 \leq k < 2^{t-1}$ and $y^{(0)}(i)$ is equation $E_{t}(0)$.

The proof of Theorem 3 is analogous to that of Theorem 2 (see Appendix).

As in previous cases, we will assume that in the $t$th stage, the input data is organized as a matrix $D_{t-1}$ defined in (7). Each stage consists of three actions.

1) Matrix $D_{t-1}$ is converted into matrix $F_{t-1}$:

$$F_{t-1} =
\begin{bmatrix}
    y(2^{n-3}) & \cdots & y(2i-i) & \cdots & y(1) & I \\
    y(2^{n-2}) & \cdots & y(2i) & \cdots & y(2) & y(0) \\
    y(2^{n-1}) & \cdots & y(2i+1) & \cdots & y(3) & y(1) \\
    I & \cdots & y(2i+2) & \cdots & y(4) & y(2).
\end{bmatrix}
$$

We will indicate as $\xi_{ct}$ the operator performing this first action.

2) The following butterfly operator will act on each column of $F_{t-1}$:

$$B_{ct} =
\begin{bmatrix}
    y(2i-1) \\
    y(2i) \\
    y(2i+1) \\
    y(2i+2)
\end{bmatrix} =
\begin{bmatrix}
    z(2i) \\
    z(2i+1)
\end{bmatrix}
$$

where $z(i)$ is the right hand side of expression (20), that is, if we use notation (19), $z(i) = [y(i-1), y(i), y(i+1)]$, except for the values of $i$ indicated in (20). Note that operator $B_{ct}$ acts over each column of matrix $F_{t-1}$, performing 24 arithmetic operations. The result is matrix $G_{t-1}$ defined in (9).

3) We apply the perfect unshuffle operator to $G_{t-1}$ in order to obtain $D_{t}$.

Summarizing, each stage can be expressed as the operator string $\xi_{ct}B_{ct}\Gamma$ applied to matrix $D_{t-1}$ and the CGCR algorithm as $(\xi_{ct}B_{ct}\Gamma)^n$ applied $n$ times to the initial matrix $D_0$. That is,

$$\text{CGCR} = (\xi_{ct}B_{ct}\Gamma)^n.$$
another one implementing the butterfly operator \( B_{be} \). Fig. 4 shows the design of the processor (solid lines). The RS section is made up of two FIFO queues. In each stage, one queue acts as output buffer (writing the data generated in the current stage), whereas the other acts as input buffer where the data generated in the previous stage is read. This function is exchanged in the next stage under the control of several multiplexors (1 to 4). The inputs to the FIFO queue are provided by the concatenation operator. The writing process is carried out in parallel over each one of the segments of the queue. In each operation cycle the data is written in the FIFO queue, although the displacement of the queue is only one position each cycle. The operation of writing all the data is completed in \( 2^{n-1} \) cycles.

The decimation operator provides the outputs of the FIFO queue. The reading process consists in directing each of the data items located in the last two cells to one of the output buses \((s_0, s_1)\). The FIFO queue has to perform a 2 cell shift operation in order to position the next two data items in the reading location. The reading of all the data produced in the stage also uses \( 2^{n-1} \) cycles. Each output bus is connected to the corresponding input bus of the processing section \((s_0, s_1, s_2)\) to allow for the sequencing of the stages. On the other hand, in order to consider the processing section PS as a segment of the processor, the length of the FIFO queues will be \( N - 1 \) cells.

B. Design of the Processor for the CGRD Algorithm

Taking expression (16) into account, the design of the processor for the CGRD algorithm is obtained by adding to the previous design the necessary logic to make operator \( \xi_{ed} \) provide the first row of matrix \( F_{i-1} \) (see (14)). Observe that the elements of this first row are in the last row, preceding column. Therefore, the element of the third row which is going to be processed in the \( i \)th cycle must be directed to a latch in order to be used in the next cycle. Fig. 4 shows the situation of latch \( L \) connected to input \( a \) of the PS section, which has three inputs. Inputs \( b \) and \( c \) are provided by the output buses \( s_0 \) and \( s_1 \) respectively.

Operator \( B_{Be} \) performs two \( 3 \times 3 \) matrix products each cycle. According to definition (13), if \( i = k \cdot 2^{n-t} - 1 \), (i.e., in the cycles \( k2^{n-t} \) of the \( t \)th stage) the output \( b' \) is equal to the input value in \( b \), so we introduce a multiplexor which determines the output \( b' \) of the PS section, as Fig. 5(a) shows.

C. Design of the Processor for the CGCR Algorithm

According to expression (23), the design of the processor for the CGCR algorithm is obtained by adding to the CGSD processor design the necessary logic for the implementation of operator \( \xi_{ed} \) which provides the first and last rows of matrix \( F_{i-1} \) (see expression (21)). The first row is provided by latch \( L \) and input \( a \), as in the case of the CGRD processor. The elements of the fourth row are obtained from the second row with unit delay. Therefore, an output in cells 2 of the FIFO queue which provides each cycle the \( d \) input for the PS section will be necessary as shown in Fig. 4 (drawn with dashed line).

In the \( i \)th cycle, the PS section must compute equations \( z(2i) \) and \( z(2i+1) \) (see (22)). According to (20), in cycles \( i = k \cdot 2^{n-t} \) of the \( t \)th stage (column \( i \) of matrix \( F_{i-1} \) is processed) \( y(2i - 1) \) does not participate, it is substituted by equation \( I \) with coefficients \((0,1,0,0)\) and in cycles \((k + 1) \cdot 2^{n-t} - 1 \), equation \( I \) substitutes \( y(2i + 2) \). Consequently, in the design of the PS section we must introduce two multiplexors for making the appropriate selection each cycle. Fig. 5(b) shows the corresponding design.

Summarizing, the architecture of the CGCR processor encompasses as particular case those of the CGSD and CGRD processors. Obviously, the type of cells of the routing section and the PS sections of each processor will be different. The duration of the operation cycle can be reduced by pipelining the PS sections. In this case, the architecture must be modified in order to eliminate waiting cycles between stages (see [20]).

IV. PROCESSOR COLUMN

Processor pipeline or array are the two most widely employed methods for projecting the inherent parallelism of the data flow of the DC algorithms onto an architecture which can be implemented using VLSI or WSI integration. The design of a processor pipeline has the drawback of its limited I/O bandwidth. With a processor column constituting a constant geometry architecture we can exploit the spatial parallelism
found in each stage. In this section we will design a parallel architecture for the CGSD, CGRD and CGCR algorithms based on a perfect unshuffle interconnection network which makes the partitioning of the algorithms possible, permits wired control and can be considered systolic.

In order to define the processor column by means of operators, we generalize to three dimensional structures $T$ of dimensions $2^x \times 2^y \times 2^z$ the operators used in Section II and we define the restrictions of the operators to two of the dimensions. For avoiding redundancies, in this section we will only introduce the basic notation for expressing the algorithms. We note $T = b + c$ and $n = a + b + c$.

Definition 5:

$$
\Gamma(x, y, z) = \left[ i_1 i_2 \cdots i_{m+2} \right] \left[ i_{m+1} i_{m+1} \cdots i_{m+2} \right] \left[ i_{m+1} i_{m+1} \cdots i_{m+2} \right] \left[ i_{m+1} i_{m+1} \cdots i_{m+2} \right] \left[ i_{m+1} i_{m+1} \cdots i_{m+2} \right].
$$

We assume that the sequence $S$ of equations at stage $t$ is distributed as a tridimensional matrix $T_{t-1}$ of dimensions $2^x, 2^y, 2^z$ among the $Q = 2^d$ processors of the column. We consider two distributions: cyclic ($d_{cy}$) and consecutive ($d_{co}$).

Definition 6:

$$
d_{cy}[i_1, \cdots, i_l] = \left[ i_1, \cdots, i_{q+2}, i_{q+1}, \cdots, i_2 \right] t_1.
$$

Using cyclic distribution makes that consecutive columns in matrix $T_{t-1}$ (see (7)) are processed in consecutive processors. For example, if $N = 64$, and $Q = 4$, the element $y(47)$ will access to processor 3 by bus 1 and will be processed at the fifth cycle since $d_{cy}(47) = [10][11][1]$. On the other hand, consecutive columns of matrix (7) are assigned to the same processor if $d_{co}$ is used. For this data distribution, since $d_{co}(47) = [10][11][1]$, $y(47)$ would access to processor 2 by bus 1 and would be processed at the last cycle of the current stage.

Lemma 3: Operator $\Gamma$ commutes with distributions $d_{co}$ and $d_{cy}$.

Proof follows easily from Lemma 2 and Definition 5. With this lemma we decompose the perfect unshuffle operator into two partial perfect unshuffles, one over coordinates $(x, z)$ and the other over $(y, z)$.

A. Processor Column for the CGSD Algorithm

The expression of the CGSD algorithm as a composition of three dimensional operators is obtained following the same steps as in Section II-A. The matrix of trios to be computed by processor PE in the $t$th stage, written as $DFPE_t$, is similar to the one defined in (7) but has dimension $2^x \times 2^y \times 2^z$ and the trio located in row $y$ column $x$ is $y^{(t-1)}$ (PE) [PE] [y]. The algorithm consists in three steps which are analogous to those described in Section II-A and the algorithm can be expressed by means of equation (10). The design will be the same as in the case of each PE, each stage of the algorithm consists in three steps which are analogous to those described in Section II-A and the algorithm can be expressed by means of equation (10). Fig. 6 shows the data trios evolution of the algorithm for each of the distributions aforementioned.

Theorem 3: The CGSD algorithm can be carried out in a column of $Q = 2^d$ processors which perform the following operator string in each stage

$$
B_{ad}^{PE, bus} \Gamma^{cycle, bus, cy} \Gamma^{cycle, bus, cy} \Gamma^{cycle, bus, cy}.
$$

The proof is immediate from (10) and (31). According to (32), operator $B_{ad}$ sequentially computes $2^t-1$ butterflies in each processor. The operator string $\Gamma^{cycle, bus, cy}$ internaly unshuffles the results in each PE. Therefore, each processor is analogous to the one designed in Section III-A (Fig. 4), with the only difference that here, the length of the FIFO queues will be $2^t-1$ cells if we consider the PS section as an additional segment of the internal pipeline of the processor. Operator $\Gamma^{PE, bus}$ performs the external unshuffle and defines the interconnection network of the processors in the column. All the trios computed in processor PE = $[i_{t+1} \cdots i_2]$ using the same output bus $i_2$ if $i_1 = 0$, $s_1$ if $i_1 = 1$) is positioned by operator $\Gamma^{PE, bus}$ in the input bus $i_2$ if $i_1 = 0$, $s_1$ if $i_2 = 1)$. The solid lines correspond to the connections between processors. In the case of $d_{co}$ distribution, taking into account interstage overlaps, a cyclic shift of the operator string (33) can be considered in order to obtain

$$
\Gamma^{cycle, bus, cy} \Gamma^{cycle, bus, cy} \Gamma^{cycle, bus, cy} B_{ad}^{PE, bus}.
$$

The design will be the same as in the case of the $d_{cy}$ case, but now the PS section of each PE must be located behind the FIFO queues.

Summarizing, both distributions require the same external interconnection network between processors. The designs of the processors differ for each type of distribution in the location of the PS section with respect to the FIFO queues.
Fig. 6. Initial trio distributions (stage 1) for the algorithm CGSD: (a) consecutive; (b) cyclic. Moreover, these figures show the evolution of the trios for each stage in CGSD.

B. Processor Column for the CGRD Algorithm

The expression of the CGRD algorithm as a composition of three dimensional operators is obtained in a similar manner to the two dimensional case. The computation of the $t$th stage consists in three steps which are analogous to those described in Section II-B. However, we must extend operator $\xi_{id}$ which carries out step 1 of Section II-B to three-dimensional structures $T_{t-1}$. The extension is immediate if the distribution of the sequence of input matrices in the $t$th stage is consecutive: $T_{t-1}$ is transformed into $F_{t-1}$ so that in processor PE matrix $D_{t-1}^{PE}$ is transformed into the matrix (35) found at the bottom of the page.

We will maintain the notation $\xi_{id}$ for the operator performing this transformation. If the distribution is cyclic, the transformation is different. Matrix $D_{t-1}^{PE}$ ($d_{cy}$ distribution) is transformed in each PE into the matrix (36) found at the bottom of the page and the corresponding operator will be named as $\eta_{id}$.

Steps 2 and 3 are analogous to those of the single processor case and the expression of the algorithm analogous to (16).

Theorem 5: The CGRD algorithm can be carried out in a column of $Q = 2^q$ processors which perform the following string of operators in each step

$$\eta_{id}B_{id}(\text{cycle, bus cycle, bus PE bus})^{-1} \cdot \text{(cy)} \quad \text{(37)}$$

$$\xi_{id}B_{id}(\text{PE bus, cycle bus cycle, bus PE})^{-1} \cdot \text{(cy)} \quad \text{(38)}$$

The proof is immediate from (31) and (16). If we compare the operator string (37) with (32) ($d_{cy}$ distribution) we observe that the architecture of the column for the CGRD algorithm is obtained by adding to the design of the processor column of the CGSD algorithm the logic necessary for the implementation of
operator \( \eta_{rd} \), which must provide in the \( i \)th cycle the data item \( y(2Qi + 2PE - 1) \), (see (35)) whose location is the following:

- **a)** If \( PE \neq 0 \) then (cycle, PE, bus) = (1, PE - 1, 1).
- **b)** If \( PE = 0 \) and \( i \neq 0 \) then \( 2Qi - 1 = 2Q(i - 1) + 2(2^{q-1}) + 1 \), and (cycle, PE, bus) = (1, PE - 1, 1). If \( i = 0 \) the complementary data item is not necessary.

Consequently, input bus \( c \) in each PE is also input bus \( a \) of the PS section of processor \( PE + 1 \). Input bus \( c \) of processor \( Q - 1 \) is connected to a latch that provides the input \( a \) of the PS section of processor 0 in the next cycle. Therefore, in Fig. 4 latch \( L \) must be removed for all the processors except the first one. In Fig. 7, the shadowed lines show the connections implementing operator \( \eta_{rd} \).

A similar study can be carried out for the implementation of operator \( \xi_{rd} \) (\( d_{cv} \) distribution). However, the resulting design does not admit pipelining. Observe in Fig. 6(a) that at any stage, in the first cycle of each PE a data item from the last cycle of the PE - 1 processor is needed. For example, at stage 2 the first cycle of PE = 1 needs data 14 which would be into the pipeline yet. Therefore, the consecutive distribution is not adequate if we pipeline the PS section of the CGRD processor. This problem also appears in the CGCR algorithm and, consequently, we will not consider it in the rest of this work.

**C. Processor Column for the CGCR Algorithm**

The CGCR algorithm can be expressed as a string of three dimensional operators. As in the previous cases, the corresponding expression is obtained by considering three steps in the execution of each stage of the algorithm, analogous to those considered in section II C. In the first step, we must extend operator \( \xi_{cr} \) so that matrix \( D_{PE}^{(i)} \) (\( d_{cy} \) distribution) is transformed in each PE into the matrix (39) found at the bottom of the page. We will name as \( \eta_{cr} \) the operator that carries out this transformation.

Steps 2 and 3 are analogous to those of the single processor case and the expression of the algorithm analogous to (23). We will not consider the case of the consecutive distribution for the same reasons as in section B.

**Theorem 6**: The CGCR algorithm can be carried out in a column with \( Q = 2^q \) processors which perform the following operator string in each step

\[
\eta_{cr} B_{cy} B_{cycle,bus} B_{cycle,bus}^{-1} PE_{bus} \ (d_{cy}). \quad (40)
\]

The proof is immediate from (31) and (23). If we compare operator string (40) to (32) (\( d_{cv} \) distribution) we observe that the architecture of the PE column for the CGCR algorithm is obtained by adding to the design of the column for the CGSD algorithm the logic needed for the implementation of operator \( \eta_{cr} \), which must provide in the \( i \)th cycle equations \( y(2Qi + 2PE - 1) \) and \( y(2Qi + 2PE + 2) \). The necessary connections are deduced in the same way as was done for the CGRD column. It is sufficient to add to the design of the CGCR column a bus connecting the \( b \) input to each processor PE with the \( d \) input bus to the PS section of PE - 1. Input \( d \) of processor \( Q - 1 \) is provided by the output \( s_y \) of the cells number 2 of the FIFO queues of the processor PE = 0. Therefore, processor 0 of the column must be a complete CGCR processor. The latch located at input \( a \) and the logic providing output \( s_y \) (MUX5 of Fig. 4) can be removed in the rest. Fig. 7 shows in dashed line the connections that must be added in order to complete the design of the CGCR column.

**V. EVALUATION**

In this article, we have proposed three new constant geometry algorithms for solving TS systems based on the SD [18], RD [5] and PARACR [7], respectively. We have also proved that the data flow of CGSD, CGRD, and CGCR algorithms can be implemented in a unified parallel architecture based on the
perfect unshuffle permutation. In this section we will present a comparative analysis of the complexity of the implementation of the CGSD, CGRD, and CGCR algorithms in a message passing type multiprocessor system and their integration in VLSI technology.

Table I summarizes the basic characteristics of the three algorithms. The first column specifies the number of floating point arithmetic operations performed (d: division, m: multiplication, a: addition) by the butterfly operators of each algorithm, as well as an enumeration of these operations. We observe that the CGRD algorithm does not require divisions. The second column specifies the data size (in bytes) required in each algorithm (we consider simple precision 32 bit data, so data size = 4c, being c the number of coefficients). This value will determine, among other things, the size of the memory cells. The number of input (I) and output (O) buses of the PE’s associated to each algorithm is specified in the third column. In the design of the PE column of the CGCR algorithm only PE = 0 requires three output buses. Observe that even though the CGCR architecture requires four input buses, its bandwidth (4×16 bytes) is smaller than those of the CGRD (3×24 bytes) and CGSD (2×48 bytes) designs. The fourth column shows the additional hardware elements the PE columns of the CGRD and CGCR algorithms need. The last two columns specify the memory and message sizes. The CGCR algorithm saves 50% and 75% memory if compared with CGRD and CGSD algorithms, respectively. The three algorithms produce the same number of messages (2 log$_2$N) as indicated in section II.C, the algorithm can be stopped if some useful properties of the perfect shuffle permutation are given in the following.

**Lemma 5:** Consider 0 ≤ k < 2$^{t-1}$. Then

(a) if i ≠ k2$^{n-t+1}$, \[ \delta^{-1}(i) = \delta^{-1}(i+1) > 2^{t-1} \]  

(b) if i ≠ (k + 1)2$^{n-t+1}$, \[ \delta^{-1}(i) = \delta^{-1}(i+1) > 2^{t-1} = \delta^{-1}(i+1) \]

**Proof:** a) The hypothesis implies that some of the n - t + 1 least significants bits of i is not 0, and then $\delta^{-1}(i) > 2^{t-1}$. Furthermore

\[ \delta^{-1}(i) + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} = \delta^{-1}(i) \]

where $i_{n-t+1} \cdots i_{t} = i_{n-t+1} \cdots i_{t} = 1$. b) The hypothesis implies that some of the n - t + 1 least significants bits of i is 0 and then $\delta^{-1}(i) < N - 2^{t-1}$. 

The structure of the RS section of the general PE presented in Fig. 4 is based on FIFO queues, permitting the partitioned systolic design of a PE column or array of columns. As the addressing of the data is implicit in the architecture of the RS section and the interconnection network, the design of the control section of the PE is simple and can be wired in. The partitioning of the data arises in a natural way by means of the decomposition of their index into three fields: (cycle, PE, bus), cyclic distribution; (PE, cycle, bus), consecutive distribution. Given the arithmetic complexity of the butterfly operators of the three algorithms, a high speed design in VLSI technology requires the pipelining of the PS section. Pipelining which is only possible if we adopt a cyclic distribution of the data in the CGRD and CGCR architectures (see Section IV). An analysis of the impact of the pipelining of the PS section on the design of the RS section of a constant geometry architecture was carried out in [20]. Finally, we will point out that from the point of view of integration in VLSI technology, the CGCR architecture is the one using the least area and the smallest number of pins.

**APPENDIX**

Let’s $\phi$ be the perfect shuffle permutation.

\[ \phi[i_0 i_1 \cdots i_l] = [i_{n-1} \cdots i_1 i_0]. \]  

Some useful properties of the perfect shuffle permutation are given in the following.

**Lemma 5:** Consider 0 ≤ k < 2$^{t-1}$. Then

(a) if i ≠ k2$^{n-t+1}$, \[ \delta^{-1}(i) > 2^{t-1} \]  

(b) if i ≠ (k + 1)2$^{n-t+1}$, \[ \delta^{-1}(i) = \delta^{-1}(i+1) > 2^{t-1} \]

**Proof:** a) The hypothesis implies that some of the n - t + 1 least significants bits of i is not 0, and then $\delta^{-1}(i) > 2^{t-1}$. Furthermore

\[ \delta^{-1}(i) + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} = \delta^{-1}(i) \]

where $i_{n-t+1} \cdots i_{t} = i_{n-t+1} \cdots i_{t} = 1$. b) The hypothesis implies that some of the n - t + 1 least significants bits of i is 0 and then $\delta^{-1}(i) < N - 2^{t-1}$. 

The structure of the RS section of the general PE presented in Fig. 4 is based on FIFO queues, permitting the partitioned systolic design of a PE column or array of columns. As the addressing of the data is implicit in the architecture of the RS section and the interconnection network, the design of the control section of the PE is simple and can be wired in. The partitioning of the data arises in a natural way by means of the decomposition of their index into three fields: (cycle, PE, bus), cyclic distribution; (PE, cycle, bus), consecutive distribution. Given the arithmetic complexity of the butterfly operators of the three algorithms, a high speed design in VLSI technology requires the pipelining of the PS section. Pipelining which is only possible if we adopt a cyclic distribution of the data in the CGRD and CGCR architectures (see Section IV). An analysis of the impact of the pipelining of the PS section on the design of the RS section of a constant geometry architecture was carried out in [20]. Finally, we will point out that from the point of view of integration in VLSI technology, the CGCR architecture is the one using the least area and the smallest number of pins.

**APPENDIX**

Let’s $\phi$ be the perfect shuffle permutation.

\[ \phi[i_0 i_1 \cdots i_l] = [i_{n-1} \cdots i_1 i_0]. \]  

Some useful properties of the perfect shuffle permutation are given in the following.

**Lemma 5:** Consider 0 ≤ k < 2$^{t-1}$. Then

(a) if i ≠ k2$^{n-t+1}$, \[ \delta^{-1}(i) > 2^{t-1} \]  

(b) if i ≠ (k + 1)2$^{n-t+1}$, \[ \delta^{-1}(i) = \delta^{-1}(i+1) > 2^{t-1} \]

**Proof:** a) The hypothesis implies that some of the n - t + 1 least significants bits of i is not 0, and then $\delta^{-1}(i) > 2^{t-1}$. Furthermore

\[ \delta^{-1}(i) + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} + 2^{t-1} = i_{n-t+1} \cdots i_{t} i_{n-t+2} = \delta^{-1}(i) \]

where $i_{n-t+1} \cdots i_{t} = i_{n-t+1} \cdots i_{t} = 1$. b) The hypothesis implies that some of the n - t + 1 least significants bits of i is 0 and then $\delta^{-1}(i) < N - 2^{t-1}$. 

### Table I

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Arithmetic operations</th>
<th>Data size</th>
<th>Buses I/O</th>
<th>Extra hard.</th>
<th>Memory size</th>
<th>Message size</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGDC</td>
<td>41 // 5d,21m,14a</td>
<td>48</td>
<td>2/2</td>
<td></td>
<td>96 N/Q</td>
<td>48 N/Q</td>
</tr>
<tr>
<td>CGRD</td>
<td>40 // 24m,16a</td>
<td>24</td>
<td>3/2</td>
<td>Latch</td>
<td>48 N/Q</td>
<td>24 N/Q</td>
</tr>
<tr>
<td>CGCR</td>
<td>24 // 4d,12m,8a</td>
<td>16</td>
<td>4/2(3)</td>
<td>Latch, mux</td>
<td>24 N/Q</td>
<td>16 N/Q</td>
</tr>
</tbody>
</table>
The proof is completed if we observe that \( i + 1 \) verifies the hypothesis of statement a).

**Proof of Theorem 1:** The SD algorithm is given by

\[
\begin{bmatrix}
 x(t+1)(i) \\
 x(t+2)(i)
\end{bmatrix} = B_{sd} \begin{bmatrix}
 x(t)(i) \\
 x(t+1)(i)
\end{bmatrix}
\]  

(A.4)

where \( i/2^{t+1} \) is even and \( 0 \leq t \leq n = \log_2 N \). Induction in t will prove that \( y^t(i) = x^t(\phi^t(i)) \), where \( y^t(i) \) is defined in (6). The expression to be proved is obviously true for \( t = 0 \).

Let’s suppose that it is true for the value \( t = k \). It can be easily demonstrated that \( \phi^t+1(2j+1) = \phi^t+1(2j) + 2 \) and also that \( \phi^t+1(2j)/2^t \) is even. From expression (A.4) and the induction hypothesis we deduce

\[
B_{sd} \begin{bmatrix}
 y(t+1)(2j) \\
 y(t+1)(2j+1)
\end{bmatrix} = B_{sd} \begin{bmatrix}
 x(t+1)(\phi^t(2j)) \\
 x(t+1)(\phi^t(2j+1))
\end{bmatrix} = \begin{bmatrix}
 x(t)(\phi^t(2j)) \\
 x(t)(\phi^t(2j+1))
\end{bmatrix}
\]

If compared with (6), last expression permits conclude that \( y(t+1)(i) = x^t(\phi^t(i)) \). \( \square \)

**Proof of Theorem 2:** The RD algorithm is established as

\[
x(t) = \begin{cases} 
 x(t-1)(i) & \text{if } i < 2^t-1 \\
 x(t-1)(i-2^t) & \text{otherwise}
\end{cases}
\]  

(A.5)

being \( 0 \leq i \leq N - 1 \), \( 0 \leq t \leq n \). Again, we use induction in \( t \) to show that \( y^t(i) = x^t(\phi^t(i)) \). Now \( y^t(i) \) is given by equation (13). Our claim is obvious for \( t = 0 \). Let’s suppose that it is also true for \( t = k \). It follows from both algorithms ((A.5) and (13)) and the induction hypothesis, the Theorem is proved in this case

\[
y^t(i) = y^{t-1}(i)
\]

\[
= x^{t-1}(\phi^{t-1}(i))
\]

\[
= x^t(\phi^t(i))
\]

\[
\square \]

b) If \( i \neq 2^{t-1} \), using induction and (A.2), it follows that

\[
y^t(i) = x^{t-1}(\phi^{t-1}(i))
\]

\[
= x^t(\phi^t(i))
\]

\[
\square \]

**Proof of Theorem 3:** PARACR algorithm is as follows

\[
x(t)(i) = \begin{cases} 
 l(i), x(i + 2^t - 1)(t-l) & \text{if } i < 2^t-1 \\
 x(i - 2^t-1), x(i), t-l & \text{otherwise}
\end{cases}
\]  

(A.6)

As before, the equivalence between both PARACR and COGR algorithms is obtained by induction to prove that \( y^t(i) = x^t(\phi^t(i)) \), where \( y^t(i) \) was defined in (20).

a) if \( k = 2^n - 1 \), the proof is similar to the same case of Theorem 2.

b) If \( k = (k+1)2^n - 1 \), it follows that \( \phi^t+1(i) + 2^t - 1 > N \), and \( \phi^t+1(i-1) + 2^t - 1 = \phi^t(i) \).

So, from induction hypothesis we obtain

\[
y^t(i) = [y(i-1), y(i), t-l]
\]

\[
= [x(\phi^t(i-1)), x(\phi^t(i)), t-l]
\]

\[
= [x(\phi^t(i-1) - 2^t-1), x(\phi^t(i)), t-l]
\]

\[
= x^t(\phi^t(i))
\]

c) If \( i \) has a value different than in two cases considered before, using induction and the expression (A.3) we get

\[
y^t(i) = [y(i-1), y(i), y(i+1)]
\]

\[
= [x(\phi^t(i-1)), x(\phi^t(i)), x(\phi^t(i+1))]
\]

\[
= [x(\phi^t(i) - 2^t-1), x(\phi^t(i)), x(\phi^t(i + 2^t-1))]
\]

\[
= x^t(\phi^t(i))
\]

\[
\square \]

**REFERENCES**


Juan López received the B.S. degree in mathematics from the University of Granada in 1968 and the Ph.D. degree in computer science, University of Málaga.

During 1968–1976, he was an Assistant Professor with the Department of Mathematics at the University of Granada. During 1977–1989, he was teaching at different Secondary Schools at Sevilla. His research interests include parallel numerical algorithms, and parallel architectures.

Emilio L. Zapata received the B.S. degree in physics from the University of Granada in 1978 and the Ph.D. degree in physics from the University of Santiago de Compostela in 1983.

During 1978–1981, he was an Assistant Professor in the University of Granada, and during 1982–1991, he successively was Assistant, Associate, and Full Professor in the University of Santiago de Compostela. Currently, he is a Professor with the Department of Computer Architecture at the University of Málaga. His research interests are in the area of parallel computer architecture, parallel algorithms, numerical algorithms for dense and sparse matrices, and VLSI digital signal processing.

In these areas, he has published more than 35 papers in refereed International Journals and about 50 refereed International Conference Proceedings.